CHALMERS

Diabetes care on smart phones running the Android platform

Design and implementation of a system to help self monitoring and managing

Master's Thesis in Integrated Electronic System Design

ANDERS WIDÉN
MAIL@ANDERSWIDEN.COM

Department of Comptuter Science and Engineering
Division of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gőteborg, Sweden 2010
Master's Thesis 2010:NN

Examiner: Sally A. McKee

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gőteborg, Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Gőteborg, Sweden June 2010

Abstract

The purpose of this thesis was to investigate the possibility and implementation of a logging application on a smart-phone to help diabetics in their daily lives. Diabetics have to keep track of measured blood glucose values as well as daily routines that affect their condition. Having this functionality in the users cell phone gives: easy access, mobility and communication capabilities. Research was done, including feedback from doctors, to provide a list of features that would help the patients the most. Effort was put into providing tools allowing the patients to understand what affects their condition and simplifying data entry. The results show that it possible to create an alternative to conventional logging with features not present in today's systems. Graphs can be sent by email, parents can receive reports of their children's values. Integration between exercise applications provides an easier way to track activities with GPS. Automatic upload of values from meter to phone is implemented as a proof of concept. All these concepts should make logging: easier, consistent and help diabetics keeping track of their condition. Feedback from doctors was positive, but an evaluation still has been done by users.

keywords: diabetics, android, e-health, log-book, mobile health, embedded, application, bluetooth.

Acknowledgements

First I would like to thank my project adviser Sally A. McKee for giving me the opportunity to work on this master thesis and for the valuable assistance along the road. A special thanks goes out to Lotte Waller and the diabetics team at Mölndals hospital for taking their time, giving me another perspective and feedback on the project. I also would like to include Jacob Lidman, Bhavishya Goel, and Chen Guancheng for valuable advice on design decisions and many good laughs in the office. Finally I would like to thank my friends and family for being there.

Foreword

This thesis started with a meeting with Sally A. McKee, where I explained that I wanted to work within embedded mobile computing. She gave me the opportunity to proceed even though my ideas were very loose. After doing some research, medicine sounded like a field where this thesis might provide actual benefit to people. Having an acquaintance with diabetes I had some knowledge about the condition and the challenges to overcome. A proposal was written with a lot of ideas that could be implemented, based on my loose knowledge of smart-phones at the time. In early 2010 Android had just started to gain momentum, and it was still unclear how much Google and the hardware manufactures would push the platform. During this time Android has been adopted by consumers and is now one of the largest platforms for smart-phones. I've learned a lot and hope to continue working on this as a side project after my thesis is over.

Contents

List of Figures

Abbreviations

ADB	**A**ndroid **D**ebug **B**ridge
API	**A**pplication **P**rogramming **I**nterface
DDMS	**D**alvik **D**ebug **M**onitor **S**erver
IDE	**I**ntegrated **D**evelopment **E**nvironment
GPS	**G**lobal **P**ositioning **S**ystem
SMS	**S**hort **M**essage **S**ervice
SPP	**S**erial **P**ort **P**rofile
UUID	**U**niversally **U**nique **ID**entifier
USB	**U**niversal **S**erial **B**us
XML	e**X**tensive **M**arkup **L**anguange

Chapter 1

Introduction

Diabetes is a chronic disease with no cure as of 2010, leaving sufferers to manage their disease through self-monitoring for their whole life. Patients have problems controlling their blood glucose level because of problems with either producing or absorbing insulin. To be able to measure the level during the day, most carry a blood glucose meter to avoid acute and chronic complications.

Both type-1 and type-2 diabetes has to be treated through a combination of anti-diabetic drugs, regular exercise, watching your weight and eating right. All these aspects affect the condition differently, so to be able to track cause and effect patients are advised to use a log-book for their daily routines.

Together with a blood-glucose meter, blood-measurement-sticks and medicines this is a lot to carry around so many patients don't use their books regularly. By providing a better platform for keeping track of the above mentioned categories, I hope to raise the life quality of people with diabetes.

1.1 Current managing systems

- **Paper**
  The free solutions come in electronic form that you print-out and fill in but there are also journals that you can buy. The pros are than they're easy to understand and the learning curve is small. But the big disadvantage is that the patient has to carry another thing with them and there's very little room for customization.
  - Jean-Franois Yale, M.D., Associate Professor of Medicine, Crabtree Nutrition Laboratories
    http://www.mendosa.com/logsheet.pdf
  - Weekly Track - book - 21$
    http://www.osspublishing.com/weeklytrack/wt2.html
- **Software/Online**
  There have been some development in online journals in recent years and most of them

provide an easy interface and good graphing capabilities. The problem is that the patient needs access to an computer and can't input the values at time of measurement. GlucoseBuddy has solved this problem by having an iPhone application that interfaces with the users online profile, but no Android version is planned.

  - GlucoseBuddy + iPhone
    http://beta.glucosebuddy.com/
  - SI Diary - multiple platforms - $50 or $20/year
    http://www.sidiary.org
  - Carelogger
    http://carelogger.com/
  - Diabetes Pilot (iPhone) - $12
    http://www.diabetespilot.com

- **Android**
  DiabetesPro isn't available on the swedish market yet and has therefor not been tested. Ontrack Diabetes is available, free and does a lot of thing right. But there's no support for automatic uploads.

  - DiabetesPro - $2
    http://www.androlib.com/android.application.com-physiosensing-ediabetes_pro-jApB.aspx
  - Ontrack Diabetes
    http://www.androlib.com/android.application.com-gexperts-ontrack-jnim.aspx

- **Monitoring device + software (package)**
  Most glucose measurement companies provide some kind of software when you buy their monitor. If they support automatic upload you have to connect the device to your computer in some way. The advantage is that you can use the software to view your whole measurement history without forgetting measurements you've taken. The downside is that it's not very mobile, and adds extra work for the user.

  - Everymed
    http://www.everymed.se
  - Bayer Contour - GlucoFacts
    http://www.bayercontourusb.us

1.2 Statement of question

The question to be answered will be if it's possible to build an application on Android which is both easy to use and with enough features that it can replace other logging alternatives for patients. Another important sub-question is if the proposed application can help the patient understand how different aspects of their life-style affect their condition. Weight will also be put on finding new ways to help doctor-patient communication.

1.2.1 Worth

Although there has been application like these developed, none has all the features proposed. This thesis will try to create new functionality that can help patients. Android is still a fairly new operating system and developers are still learning to work with Android. There are a lot of questions about best practices, and what can and can't be done. This thesis will also serve as an evaluation on what challenges still face developers wanting to start working with Android.

1.2.2 Goal

The goal is to help patients with diabetes by providing a tool for better understanding of their condition. Secondary objectives are to give insight into Android development for the writing as well as the reader. It's the writers intent to try to implement as many different techniques for solving a problem as possible within the operating system.

1.2.3 Scope

The purpose of this thesis is to first investigate the needs of diabetics and diabetic care, and then create an application-platform with functionality based on that research. Focus will be put on log book functionality with the strive to automate data upload from the wide range of medical devices diabetics use. Effort will also be put into creating base functionality first and polish will come secondary.

Chapter 2

Hardware Platform

To provide the best experience the application should be developed on a platform with: Strong communicative support for sharing and uploading data (3G, WiFi, GPRS, Bluetooth). High ease of use in input and presentation and be future proof for the developer and user. Most cell phones of today do well in a few of these categories but only smart-phones do well in every field.

In the smart-phone field there are two main competitors the Apple iPhone and Android compatible devices. iPhone although popular is closed as a development platform and development can only be done on Apples terms which has been changed in the past[1][2]. The Android platform is open source and can be modified to suit the needs of the application. Android also has a more open policy toward application developers giving more freedom in the system. These facts together with strong hardware support from major hardware manufactures like: Motorola, Samsung, Sony-Ericsson, HTC, LG, Intel, ARM and Nvidia, makes the Android platform a future-proof device with strong potential to be powering most mobile devices in the future.

Developing on a larger platform like Android gives the benefit integrating exciting software with the product to produce a richer experience. There are exercise application to track how far you walked/ran/cycled using GPS with estimations in calories burned. There are applications for watching your weight and calculating nutritional values. All these could interact with the proposed application to add additional information.

Chapter 3

Background Information

3.1 Glucose Monitoring

Diabetics are divided into type 1 and type 2. Patients with type 2 diabetes are usually not treated with insulin and doesn't have to take measurements as frequently[1]. They are instead recommended to have a healthier life-style. A type 1 sufferer has no insulin production and has to inject insulin during the day.

Before each meal a glucose measurement is taken by drawing a droplet of blood by using a small lancet and put on a test strip. The test strip is put into a blood glucose meter that will analyze the glucose level and give a value. Then the carbohydrates of the meal is estimated; based on the current glucose and the carbohydrates the amount of insulin to be injected is calculated. This is then injected by a kind of syringe or an insulin pump. A couple of hours after the meal a measurement is taken again for verification. This procedure is then repeated and in total a diabetic might have 6 blood sticks and 4 injections, or even more, during a day.

3.2 Android Platform

Android is an open source platform developed by the Open Handset Alliance (OHA). The main contributor of the project is Google and that's why the perceived notion is that they own the platform although that's not the case.

The first appearance of Android running, was with the launch of the HTC G1 phone in the last quarter of 2008. Since then there has been several major updates to include more functionality with the most recent being the 2.2 version. Android's main application has been as a smart-phone OS, which also is the focus of the project, but there have been recent demos of Android running on TVs and tablet devices.

[1] This is a simplification of the disease and treatment, not all patients are the same and the treatment is varied. More information about diabetes can be found at the American Diabetes Association website (www.diabetes.org).

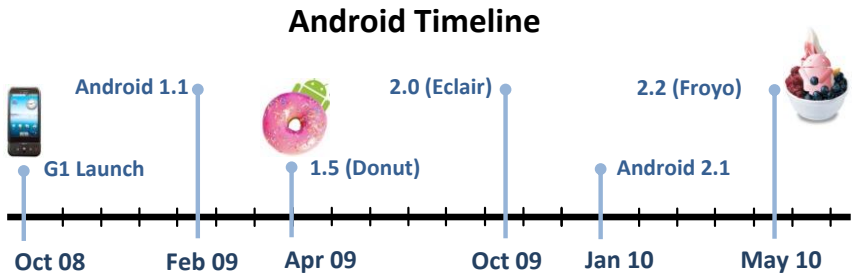

FIGURE 3.1: Releases of the platform over the past one and a half years.

Android is based around a heavily modified Linux kernel with most core libraries written in a mix of C and C++. Although C/C++ is the main language for the framework, almost all application development including the home-screen and virtual keyboards are written in Java. This is possible through the use of the Dalvik Virtual Machine which converts Java class files into its own format by merging classes and converting byte code to match its own instruction set. Conversions are also made of calls to common Java methods like toString() to equivalent implementations made in the native language[3]. If high performance is needed for a specific calculation, functions can be implemented in C and then be called from Java. Experiments show that this can give a 10x performance increase[4]. Although the development language is Java it's important to note that the version of Java used doesn't support all common Java libraries like J2ME, AWT and Swing[5]; Which makes porting exciting applications harder.

3.3 Development and SDK

The Android system is all based around applications, most of the things shown on the screen is applications that can be replaced if the user chooses to. Custom keyboards and desktops are common and separate Android from other smart-phone systems.

The SDK contains a rich API, optional plugins for Eclipse, a set of development tools including an emulator for easier testing. The emulator is a good way to verify cross compatibility between the various versions of Android because different versions of the OS can be loaded. Android strives to be backward compatible, giving support to most applications that can run on older platforms.

For testing on hardware there different methods: Google released the G1 as a developer phone meaning that it has a custom bootloader where developers can load their own system images. An consumer phone, can in most cases, be used when installed with the correct driver and loaded through the Android Debug Bridge (ADB). Using Eclipse: development, compilation, install and launch can be done straight through the IDE. Eclipse also provides log output and debug through DDMS on both hardware and the emulator.

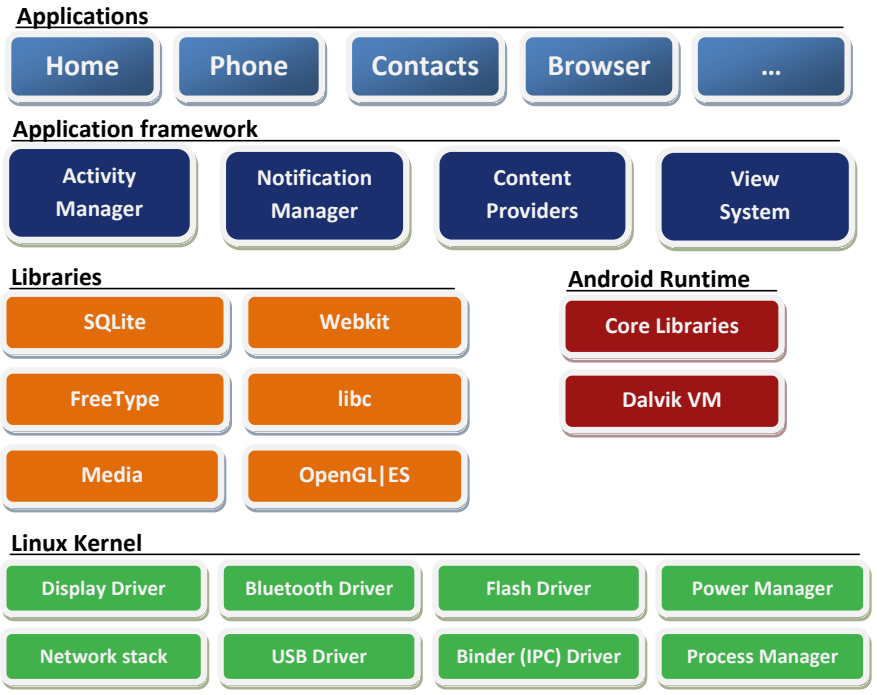

FIGURE 3.2: The android architecture.

3.4 Bluetooth

Bluetooth (BT) is a wireless technology for sending and receiving data through short distances. It was first developed by Ericsson in 1994, but has since then been taken over by the Bluetooth Special Interests Group and become an open standard. The idea behind Bluetooth is to unite all previous communication standards and make communication between devices easier. Bluetooth transmits data in the unlicensed 2.4GHz band by using frequency hopping spread spectrum at rates between 1-3Mbit/s.

3.4.1 Profiles

BT uses a packet based communication method where multiple clients can be connected to a host in a small network. Each device has to have one or several BT profiles to be able to identify what kind of communication the device accepts. For instance a wireless headset want to transmit and receive audio data so it will tell the host that it can communicate using the Headset Profile (HSP) and if the host supports that, a successful connection can be made between the two.

3.4.2 Pairing

Before communication can be started between devices pairing must be made. This is a security requirement so that two devices can't share information without both parties knowing about it. This is usually done when two devices first come in contact with each other. Depending on

the Bluetooth version supported and the input capabilities of the devices, this can take different shape, but in general they share a link-key that has to be verified by the user. When pairing has been made the key is saved and communication can then be started without user interaction when devices are in range of each other.

3.4.3 Bluetooth on Android

Although Bluetooth has been fairly common on most mobile devices for a while now, Android have had it's fair share of problem for anyone wanting to use Bluetooth on their device. Android 1.5 (April 2009) was the first version to support Bluetooth but only had profiles for headsets (A2DP) and remote control (AVRCP) on a framework level. There was no support for writing applications that used these profiles for developers. Bluetooth was essentially glued on so hardware manufactures could support headsets on their phones.

There was demand for applications that could share files and contacts using Bluetooth. This lead to a couple of developers hacking the framework by rooting their phones and trying to implement their own versions of Bluetooth support. A couple of applications were developed that could do these things but required either rooting or had other complications. When Android 2.0 was released in December 2009 one of the major updates was official Bluetooth support (with API) and most of the hacked applications don't work on Android 2.0+ because of the new implementation. This means that most developers have to reinvent the wheel when using Bluetooth and there isn't much prior information on the subject.

3.5 Application Concepts

Application handling is based around a sandbox design where each application run on its own Linux process and each process has its own virtual machine[6]. This ensures that each application can run code isolated from all other applications. Each application also has an unique ID to ensure that its files only can be accessed by itself. Applications usually save information in a database structure implemented by SQLite.

The coming sections will describe core concepts of Android development relevant to understanding the implementation chapter 5. Concepts not relevant to the thesis application will be left out.

3.5.1 Activities

An Activity is a part of the application; designed to do one specific task by interacting with the user. You could think of an Activity as a specific window inside the application. Activities contain code for setting up the GUI and methods that are associated with that part of the application. Most applications contain different activities for each part of the program. For example an email application might have one activity that shows a list of all emails, another one for writing an email, another one when the user wants to attach something etc. Using Activities is something that separates Android from conventional programming, with a main entry point.

Each Activity is its own sub-application and can be started without running the rest of the application.

3.5.2 Services

Android can handle multitasking by creation of Services. A Service is a part of the application that isn't linked to the application life-cycle. It also doesn't run on the GUI-thread and can therefore be active when user is doing something different without stalling the user interface. A good example of a Service is music-playback that can be running at the same time as the user is doing something completely different.

Services should not be confused with Activity pausing which is when another application needs the attention of the user. The running Activity then gets paused and its current state saved until the interrupting application is finished and the paused Activity is resumed. The difference is that the thread in the paused Activity isn't active although it might be partially visible.

3.5.3 Intents

The Activity model gives a great advantage in that specific parts of another application can be started to do a service for your own application if permitted. As the name implies you can program an intention to do something, without knowing the applications the user has installed to handle these types of requests. For example you can send an email by sending an Intent to the system to look for an Activity that can handle sending emails. Any application that is installed for that Intent-type will then say that it can handle the request and the user can decide which one to start. Intents can also include extra information as the email-address and subject for the started Activity to handle. Intents are also used for linking an applications Activities together by providing a way for new Activities to be created within your application.

3.5.4 Databases

Most applications create one or many databases to save and access information using SQLite. To access the database a query is created that specifies from which tables and rows the data should be selected and which conditions have to be fulfilled for that row to be selected. All rows that matches this query description is returned as Cursor (a type of pointer). The Cursor is a list of the selected data and points to a specific row in the dataset. To access information the Cursor is moved to the correct row and then columns can be read by supplying their id. Most applications also have an Adapter class that the developer creates to handle specific calls to the database instead of rewriting queries in each part of the application.

3.5.5 Content Providers

As described above in section 3.5 applications have their own sandbox and databases are private to the application itself. To be able to access information between applications Content Providers can be used. The Content Provider is implemented by the application developer of the sharing

application to deliver *specific* parts of the database. Examples include the contacts application which has a Content Provider, giving access to the users phone contacts. Accessing a Content Provider is done in a similar matter as database queries and will return a Cursor. For security reasons the receiving application will have to declare the access to sensitive information in the manifest.

3.5.6 Broadcasts and Receivers

Whenever an event happens on the system: for example a reboot or when a SMS has been received, the system will send a Broadcast. This is done to ensure that any application that is interested in these type of events can handle them. You can also program your own Broadcasts to be sent.

Broadcasts are handled by a Broadcast Receiver which is a part of the application that handles a subset of the available Broadcasts. For example an application might have a Service for downloading news for the Internet. This should ideally be started after a reboot of the device so an Broadcast Receiver is then implemented.

3.5.7 WebView

A WebView is as the name implies a way to view web-content. A WebView can load and view local or online HTML-content, but much of the GUI and functionality of a real browser is removed. Thus the WebView doesn't have to be as large and can be placed as a small windows in the layout.

3.5.8 Application Manifest

Each application has a file called the manifest. This is a security implementation that makes sure that an application doesn't have parts or functions that isn't publicly known. For example if the application wants to access the Internet this has to be specified in the manifest otherwise this application won't get this access from the system. Other examples are reading the SD-Card, reading Contacts and the Broadcasts this application will intercept. All these permissions are displayed to the user each time you install an application so that the user can make an informed decision about the program.

The manifest also contains information about all the Activities and Services that the application can start. As well as information on what version of system the application can be installed and run on.

Chapter 4

Solution and Design choices

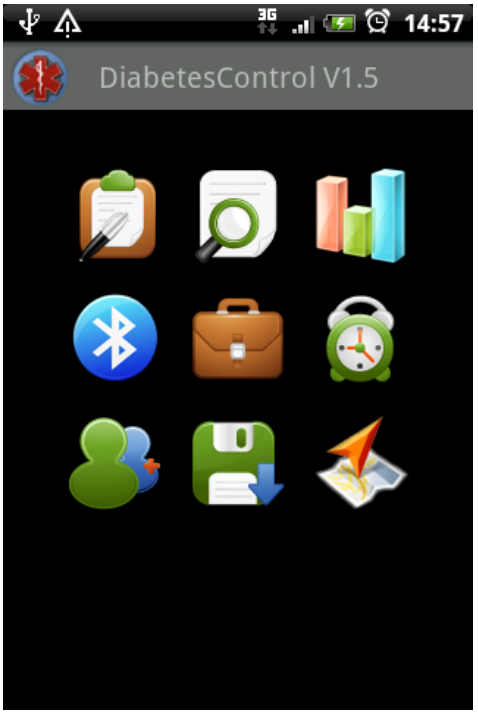

FIGURE 4.1: Starting screen of the designed application.

An application has been developed that can be installed on any phone running Android 1.5+ or above which is 99,9 percent of the devices in the market[7]. In the following section, design choices and the solution will be presented to give a overview before reading the implementation5.

4.1 Information Entry

The main functionality of the application is to log information. Many of the design choices have been around how to serve a large audience where every age and life-style is represented. Logging too much information will make each entry to time-consuming and complicated for some users,

where others might want more functionality. All entries have been designed to be as general as possible so that the user can decide what works for them.

The application supports manual input of four main categories: measurements, medicine, exercise and meals. All designed only to take a couple of seconds for a user to complete. All entries have input for notes, where the user can write optional information important to them; for example: being sick, high carbohydrate meal before measurement or how the exercise felt. When logging medicine the users can input their own medicines; which will be saved for easy access later through a drop down menu. Meals are counted as kilo calories (kcal), which is the most relevant information of each meal for diabetics[8]. Exercise can be added as a duration together with intensity and the type of activity done. Activities can be customized just as medicines to suit the user. There's also support for importing exercises from the popular GPS tracking application My Tracks[9].

Automatic input is supported in the form of Bluetooth connectivity. There are very few glucose meters sold right now that support Bluetooth and unfortunately getting access to one during the thesis was unsuccessful. Therefore automatic upload is made as a proof of concept where the device connects to a PC which then sends values that are saved into the database. Modifying this to work with an actual device should, in theory, be a small matter.

4.2 Presentation of Data

Presentation of data is supported as a list of entries and through plotting. The list supports all four major categories of data and is designed to provide as much relevant information to the user in a way that's easy to understand. Entries are chronically ordered with headings showing dates and each category has its own icon. The user can also edit or delete the data by pressing on the entry displayed.

Graphing is made to provide a better way to see how the data-points change over time. There's also functionality for plotting different data sets side by side which enables the user to see how data can relate to each other (measurements vs. medicine for example). During the research phase a meeting was held with DiaSend where pie charts were suggested as an easy way for the user to see average blood glucose values over a longer time period. Plotting these are now a feature and will shown how many values are too high, too low or in the normal range for a given time period. Setting the range of *normal* values can be done in settings by recommendation of the patients doctor.

4.3 Sharing Data

The major driving force behind adding this functionality was the first meeting held with Lotte Waller at Mölndals Hospital where phone consultation was discussed. Trying to describe your situation as a patient over the phone isn't easy, instead a graph of the values can be sent before a meeting or phone consultation. This can be done from the graphing part of the application where the user finds the time-period and then the graph is sent by email to the doctor. Using email

adds no third party program to be installed for the hospital and is an easy and understandable technology for both parties.

Another part of the application is the option to send reports of your blood glucose values during the day. Daily reports is aimed toward concerned parents of newly diagnosed children with diabetics, but can also be used by a care taker of an elderly diabetic using the application. If enabled the system is automated and will assemble an SMS once a day and send it to the number specified by the user.

4.4 Reminders

One of the problems brought up at the meeting with Lotte Waller is that patients, especially children, often forget to take their measurements during the day. Therefore reminders can be set to be repeating at lunch each day for example. Each reminder will alert the user by sound and vibration and provides an easy way to open the application through a notification at the top of the phones menu.

4.5 Backup

The user shouldn't have to abandon all their values when buying a new phone, instead the user can do a backup of the database. The database is then saved on the SD-card and can be imported on the new device. Export of the database can also be done as an XML-file which then can be used to import the data to any other service that can parse the information.

Chapter 5

Implementation

5.1 Information Entry

The application has support for input of four main parameters: blood glucose, medicine dosage, exercise and meals. Of these the most important point of data is the blood glucose value. All four inputs are based on a similar system where the user inputs: the date and time of the measurement, the value and some optional notes. These are saved into a database structure and can then later be accessed. When using the application the only thing the user has to input is the actual value, as time and date by default is set to the current time and day (but can be changed). This is of course to make the interaction as quick and easy as possible when using the application. When the user wants to edit an existing entry the Activity for that type of entry is started and all the fields are populated using information from the database.

5.1.1 Blood Glucose

After the patient has measured their blood glucose, the value can be inputted using manual input as well as automatic input by using Bluetooth to connect to the device. Manual input is pretty straight forward using either the on-screen keyboard or the NumberPicker talked about in section 6.2.1 to choose a value. Automatic input is done in the same way but the values are sent over Bluetooth which will be discussed in section 5.6.

5.1.2 Medicine Dosage

Medicine is almost inputted the same way as blood glucose with the addition that the patient can fill out the medicine and dosage taken. Selection of medicine is implemented as a drop down list with a few standard choices, with the functionality to add and delete medicines from the list. When the user selects the medicine taken, only the unique id of that medicine is saved with the entry. This is to conserve space and eliminate the need of redundant database information. The medicines themselves are thus saved in their own database and only linked by their id.

Although this approach has a lot of benefits, if the patient deletes a medicine that has been used in previous entries these entries will be shown as missing a medicine name.

5.1.3 Exercise

Input of exercise is done as the duration of your exercise, the intensity and the exercise activity that you made. The same drop down list functionality as in medicines is used for the activities.

Most daily exercise activities like walking, running and cycling center around moving along a route and are perfect candidates for using the on board GPS fitted to most phones. There's a wide range of exercise applications out there for the Android platform, but at the time of writing the most popular is My Tracks[9] developed by Google. My Tracks accesses the GPS during your exercise and plots your course on a map along with relevant information about speed, duration, distance, elevation gains etc. The users can record their exercises with My Tracks and then import the information to be stored with all your other exercise activities. To accomplish this a link between the application and My Tracks Content Provider has been made. It will list all the recorded tracks so that the user can choose the one desired, as shown in figure 5.1.

FIGURE 5.1: Screenshot showing how import from My Tracks is presented to the user for selection.

The application will first query the Content Provider for information about all tracks saved by the user. The Content Provider will then return a Cursor object with the rows and columns available. The relevant information is then mapped into an ArrayAdapter that populates a ListView on the screen (more about ListViews in section 5.2.1. Some formatting has to take place to be able to correctly present the values since most of the queries return primitive values. As an example the time stamp is the number of milliseconds since 1st of Jan, 1970. When the user selects a track another query will be made to fetch more information about the selected track and save these to the local database. Fields that can't be directly mapped like average speeds, distance and description are put into the note area so that they are still presented. The import will then return to the exercise Activity with the newly created entry's rowid which will use this to populate the fields and present the import to the user, who can make changes if needed.

5.1.4 Meals

Meals are entered with the number of kcal the meal consisted of together with a selection of what kind of meal it was (breakfast, lunch, dinner or a snack).

5.2 Presentation of data

Presentation can be done in two ways in the developed application. Either by showing all the entries in a list or by using the graphing capabilities. Focus was put on presenting the data in a way that was easily readable by the user and gave the opportunity to compare values across a time period.

5.2.1 List Presentation

List presentation is implemented using a custom ListView. A ListView is a way to present a large series of data, separated by rows, in a scrollable list. A ListView contains three major components. The row-data object (usually a hash table of keys and values), a sorted list of the hash tables and the row-layout. To create the ListView you input the list, and map the row-layout objects with each of the hash table keys. This will create a ListView where each row is matched to a specific hash table and each of the keys are mapped to a specific layout object in the row as seen in figure 5.2.

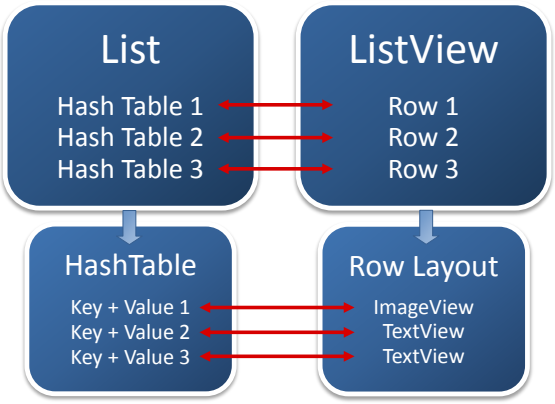

FIGURE 5.2: Structure of how a ListView maps data from lists to rows.

The reason why ListViews can handle so much data is because only the number of rows visible, actually exist at one given moment. When the users scrolls the list, the row (View) that disappears is reused for the one that appears at the bottom. This reusing of Views are essential to resource management as creating View objects are an expensive process[10].

The implementation breaks down into two major steps. Dividing the entries in the database by date and then formatting each row specifically based on what type of entry it consists of. To be able to divide by dates a special SeperatedListAdapter is made that consists of a parent adapter, header divider for each new date and then a IconSimpleAdapter as can be seen in figure 5.3.

The IconSimpleAdapter will take the generic icon-object for each row and replace it with an icon suited for that type of entry. This is done by sending an extra key+value for the master-rowid in the database. Whenever a new row-view is created it will do a lookup in the database to determine what type of entry this is and change the icon. To increase performance this is done without allocating any memory for a variable which was a problem in early testing. To differentiate between entry types further, exercise will show the activity and medicine will show

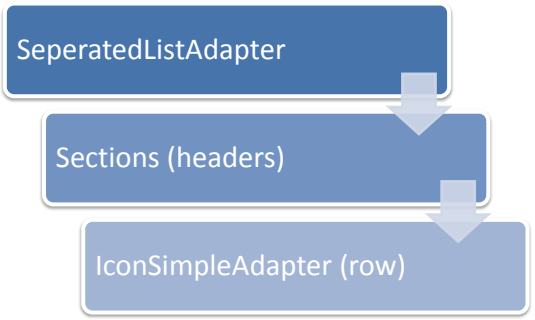

FIGURE 5.3: Structure of how ListViews are linked together.

the medicine, etc. This is accomplished by doing lookups in the database for special fields which changes for each data-type.

In Android each row in a ListView has a position and a id; whenever a press is made on an entry the ListView will ask the adapter for the id of that row. In this implementation the parent will always ask the child adapter for an answer until it comes to IconSimpleAdapter. It will do a lookup in the hash table for the master-rowid and return. This is used when the user wants to edit an exciting entry by tapping the row. The master-rowid is returned and is passed to the Activity that can edit that type of entry, which will populate all the fields with values from the database.

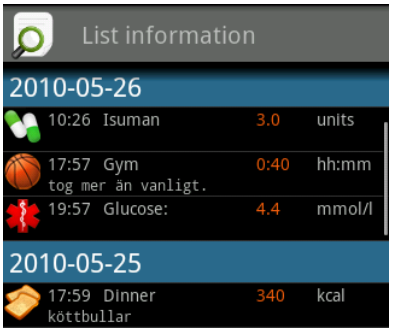

FIGURE 5.4: List presentation of entries (cropped).

5.2.2 Graphs

The intension of the application where always to give the user control over his/her situation by providing an easy way to draw conclusions based on the available data. To provide as much freedom as possible over data graphing were implemented with this in mind. Graphing had to be very customizable, give good overview with an easy interface.

5.2.2.1 Choosing a Library

Creating graphs on Android is still a problem with no good native libraries out there. aiCharts is available, but a single license costs $150. There has been some development done to create a free alternative, but they're still immature and lack functionality. Using a native Java library is also hard because most of them use the java.awt package which isn't included in Android's Java implementation.

There seemed the only way would be to develop a whole charting library from scratch. This would contribute to learning as well as providing all the features needed, but would be a huge undertaking. With so many different features still needed to be implemented another solution was thought out.

Sharing data was always a goal of the application and a move over to a doctor oriented web-portal would benefit of having the same charting library as the one running on the phone. It was known that Android had the capability of displaying a custom browser-window called a WebView as talked about in 3.5.7. Using a web-based charting library opened up the project for a whole new range of options see table 5.1.

Most web-based charting libraries use the fact that they run on a platform with constant connection to the Internet. Popular alternatives like Google Charts and Emprise Charts generate charts on their servers and send the result back. These where out because of privacy issues of handling medical data, Internet connection required and load times. A library that was free, offline, fast, looked good, and with good documentation was needed. A study of interesting alternatives was done and the result can be seen in table 5.1.

Name	**Technology**	**Online**	**Other Requirements**	**Look (1-5)**
achartengine	Android	No	Closed Jar	2
ChartDroid	Android	No	Lacks functionality	4
aiCharts	Android	No	$150 licence	5
JFreeChart	Java	No	java.awt.*	3
EasyCharts	Java	No	$650 licence	4
Google Charts	JavaScript	Yes	?	5
Emprise Charts	JavaScript	Yes	?	5
Bluff Graphs	JavaScript	No	Lacks functionality	5
Raphael	JavaScript	No	SVG (Vector drawing)	5
Flot	JavaScript	No	jQuery Library	5
Flotr	JavaScript	No	Prototype Library	5

Table 5.1: Charting alternatives

Implementations with Bluff, Raphael and Flotr[11] were made. Bluff was used a long while until it came to a point where functionality was needed that wasn't supported. Rapheal is based around building SVG vector graphics that should be supported by the WebKit version used in Android. Unfortunately this has been cut out of the Android implementation, probably because of performance issues. There was still doubt if using a JavaScript library would be fast enough to redraw the graph on a physical device but tests using Flotr seamed promising to the extent that it would be the basis of graph generation going forward.

5.2.2.2 Overview Implementation

The grapher is implemented in three parts, first it's the Java class that holds all control of what's being drawn, calls to database, dates, GUI and menu. It also holds all information about the WebView and the page that is loaded into it. The second part is the HTML-page that has all the JavaScript methods to ask for information and interface to the third part which is the Flotr library. A simplified description of the data-flow is described in figure 5.5.

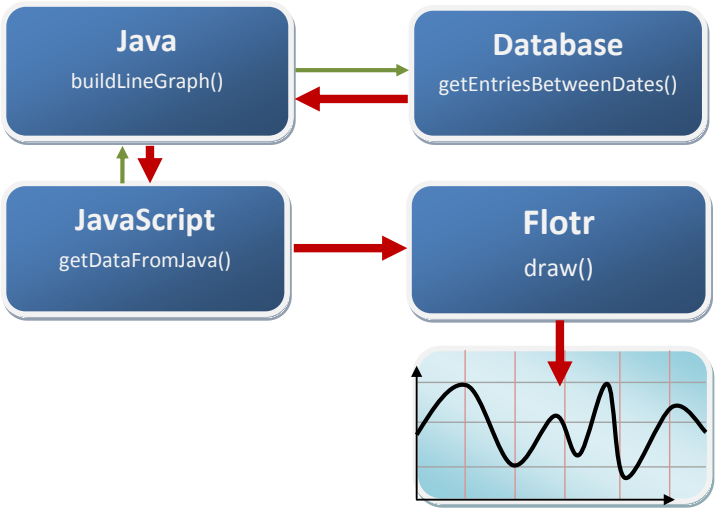

FIGURE 5.5: Structure of graphing calls and data flow.

To clarify there's a Java Class, a WebView, HTML+JavaScript and Flotr (JavaScript). Communication from the Java Class to JavaScript is done by calling functions through the WebViews address bar just as can be done on a normal desktop browser; but here it's completely invisible to the user. Communication from JavaScript to Java is done through a JavaScript interface[12] which is a part of the WebView implementation. One challenge with this integration between Java and JavaScript is that all methods called by JavaScript are run on its own thread and synchronization between the both can be tricky. Which is why the redraw has a seemingly complicated data and control path, more on that in section 5.2.2.5.

5.2.2.3 Date System

Graphing is based around a dynamic two date system where the grapher always knows the start and end date of the current plot. Whenever a redraw of the graph is made a new start and end -date is calculated. All data points that are in the database between these dates are fetched. Because the JavaScript has no knowledge of dates and times each entry has to be matched against an index system, where each entry's date+time are recalculated to an index. The system has a resolution of 24-points a day which gives a resolution of one hour in the graph for each day. This seemed like a reasonable tradeoff between speed and accuracy, and can be easily changed in the future.

The grapher also supports different levels of zoom (week, month, quarter) these are also based around setting a different offset between the start and end -dates. The user simple taps the screen to zoom in, the application will zoom one level and center the zoom on that location. To zoom out the user double taps and the application redraws with the previous dataset centered. The user can also move around by selecting the previous or next week/month. This will again recalculate the start and end -dates and redraw the graph. Together this means that the grapher can show any time period and the user is free to zoom and center any time frame that he/she is interested in. The drawback of this freedom is that labels can't be shown indicating that it's "April" because usually it's a subset of April and another month.

5.2.2.4 Data selection

Being able to plot different data-types (measurements, medicine, exercise) in the same graph was important because this would enable the user to compare how medicine dosage affected measured blood glucose for example. But because plotting data of different ranges and different units, and still make it presentable is hard, several compromises had to be made. Line charts are available for blood glucose and medicines, and exercise and meals are plotted as dots with a label indicating the value beside it.

5.2.2.5 Redrawing

When a redraw is executed and the Java knows which dates to fetch, it will start JavaScript functions to start the redraw as seen in figure 5.6. JavaScript will then do initializing to start the redraw with Flotr and then call methods inside the Java class through a JavaScript interface to retrieve data to plot. Java will then fetch data from the database between the dates specified, convert it and return the result to JavaScript. Three types of data is fetched: data points for each line to show, labels for the X-axis and the levels indicating a high/low glucose value as set by the user in settings.

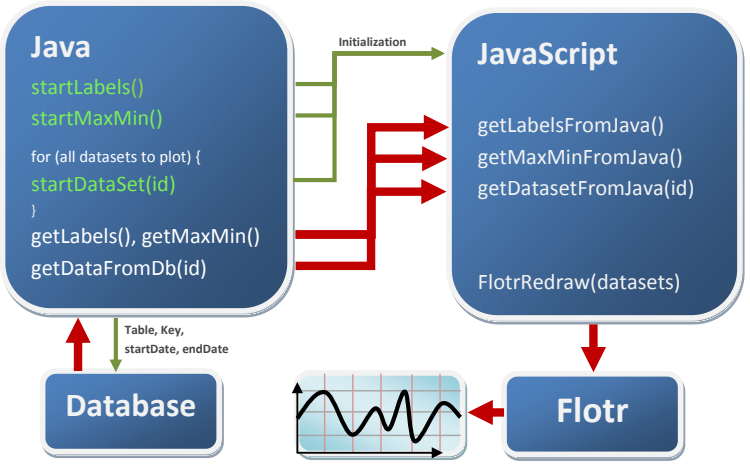

FIGURE 5.6: Detailed structure of data and control flow during a redraw.

All these methods return data as a set values (index,value,index...) in a long string. Which will then be placed into an array that Flotr can interpret as set a of data. Concerns with performance using this method was raised but testing has shown that this process is very fast when compared to other parts of the redraw process, where Flotr drawing the graph takes the longest time. All the data is then bundled together and sent to Flotr along with the settings and Flotr takes care of the drawing and updates the HTML which is shown in the WebView.

5.2.2.6 Performance and Pie Charts

Performance is mainly hindered by reading and parsing Flotr and Prototype, the library it uses. This process takes a couple of seconds when starting the graphing application and the web-page is first loaded. After that each redraw then takes less than a second on a HTC Legend running Android 2.1 and should be even faster on Android 2.2 with the new JavaScript engine.

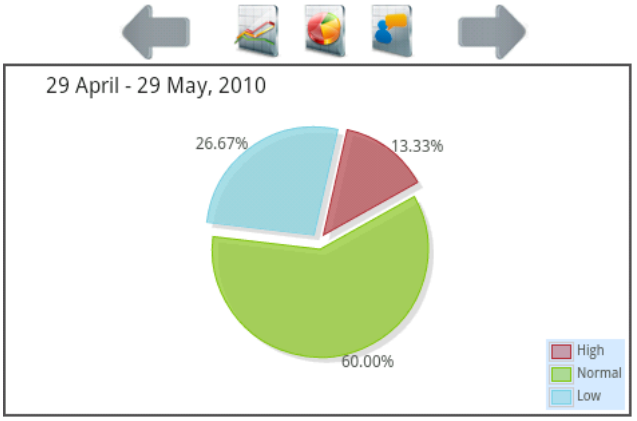

FIGURE 5.7: Plotting glucose distribution as a pie-chart.

Pie-charts showing the number of data-points within the optimal blood glucose range was implemented as an easy tool for the user to see how well he/she is doing. The user can easily see if most readings are below or above the specific threshold. This functionality was first implemented as separate web-page to keep things clean. But because of the mentioned bottleneck of reading in the libraries it's advisable to use one HTML-page and instead create multiple fetch and draw functions, when dealing with large libraries.

5.3 Gesture Detection

Interaction with the application is usually done using touching buttons on the screen, but there's also support for doing touch gestures in a frame. To be able to interpret these gestures a OnTouchListener has to be created and registered together with a GestureDetector. Whenever the system registers that the user touches the frame the OnTouchListener is called with an MotionEvent. In this implementation this is then forwarded to a GestureDetector which has support for a couple of standard gestures like: single/double tap and a fling motion. These were implemented to give a richer experience when using the grapher. Fling (which is a fast swipe over

the screen) is implemented to show the next/previous time period depending on the direction of the fling. And taps are used for zoom as described in section 5.2.2.3.

5.4 Sharing Data

5.4.1 Sending Graphs

Being able to send information to your doctor before a phone consultation or a meeting was a priority. Various options were looked into and synchronization with a web-portal where the doctor can see all patients last values would have been ideal. But because of limitations in time, privacy concerns, hospitals journals and priorities this is not feasible.

Sending data can be done with graphs generated by the grapher and then sent to your doctor by email. The graphs themselves are static images of what is displayed in the grapher when the export is started. This gives the user a familiar way of locating and presenting what he/she has questions about to their doctor. As well as an easy way for the doctor to interpret the result. By using email no extra system has to be installed on the hospitals side and it's a technology that it fairly common to doctors.

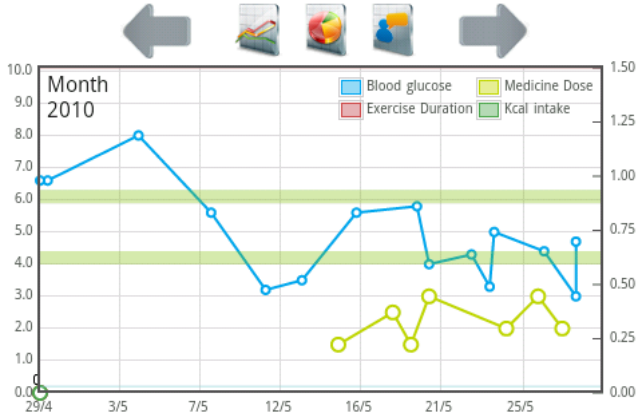

FIGURE 5.8: Plotting glucose and medicine in same graph.

When the user decides to send a graph; an email is created automatically with basic information about the doctor and the user taken from the settings the user has provided. All this information are bundled into an Intent together with a URI with information about the graph to attach. The Intent is fired and will start the default email application installed on the users device.

5.4.2 Daily Reports

Daily reports are a way to send glucose values by SMS to a care-taker. When enabled, reports are assembled automatically at a customizable time during the day, preferable the evening, by fetching all the values from this date and assembling them into a message. The message are then sent by SMS to a specified number using the SmsManager[12]. All this is done without user interaction.

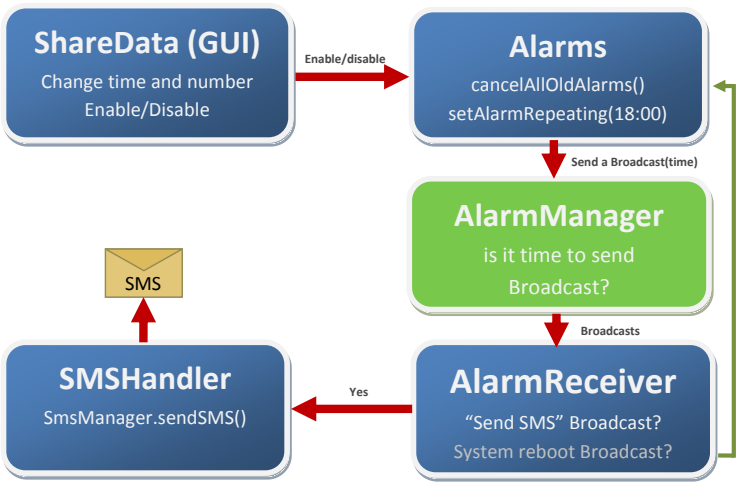

FIGURE 5.9: Simplified block diagram to describe how system Broadcasts are used to be able to schedule and trigger events in Daily Reports.

To be able to schedule an event for later processing a Service could be implemented that runs in the background but that will consume unnecessary resources. Instead it's recommended that the AlarmManager[12] is used, which is a system service where events can be scheduled to happen at a certain time using Broadcasts. How this is implemented is shown in figure 5.9: ShareData is the GUI part of the application where the user can specify; if the service should be enabled, the time and phone number to send the reports to. These settings are then saved to the database and a call to Alarms is triggered. Alarms is a class that sets all the Alarms in the application and first figures out if the time specified is today or tomorrow and then sets a calendar to next alarm time. Alarms then calls the system AlarmManager (green) to send a special Broadcast at the time set by the calendar. Everything is now set for the alarm to go off, and when that happens the AlarmManager will send this Broadcast to all Broadcast Receivers on the device. Among them is my implementation that listens to the custom Broadcast and then starts SMSHandler which will create and send the actual SMS using information from the database. One complication with the AlarmManager is that it's cleared if you reboot the device, fortunately a BroadCast is also send when the device has been rebooted. AlarmReceiver listens for this BroadCast and then calls Alarms which will then set the alarms again.

5.5 Reminders

The implementation of Reminders is closely related to how daily reports (5.4.2) work. The user can specify a large number of reminders which all will be saved in a database. Reminders can either be one shot or repeating each day. Each reminder is then handled by the Alarm class and set in the AlarmManager, but when the AlarmManager fires the BroadCast it's handled a bit differently. It's still received by the AlarmReceiver but instead of sending an SMS the reminder is displayed through a notification (same principle as showing that and SMS has been received, or that an event in the schedule is due). The phone will play a sound, vibrate and a small icon

will be displayed with the text of the reminder in the notification menu at the top of the GUI. Tapping the notification will open the application so that the user can do the task.

5.6 Bluetooth

Since getting hold of a Bluetooth enabled glucose meter proved more difficult than anticipated emulation is done using a PC. The PC has a Bluetooth dongle and a terminal program running for sending commands to the phone. This provides proof of concept of the functionality and can easily be modified to support real meters as they become available.

5.6.1 Functionality

The Bluetooth application uses the SPP profile, with a well known UUID for connections to a PC (this can easily be changed, to fit the device the patient is using). The SPP profile is one of the earliest implemented and is supported by most Bluetooth devices. SPP encapsulates the information as an emulated serial RS-232 connection. Using a more advanced profile was considered, but for sending small parts of information SPP works just as good and there's nothing to be gained by using object exchange (OBEX) for example.

The first thing the application does is to check that the device actually has Bluetooth support and if it's currently enabled. If not it will prompt the user to enable Bluetooth by showing a dialog. If this goes well the user can bring up a list of devices to connect to, if there is paired devices currently in range these are shown first, but the user can connect to any device capable of communicating with the phone. If there are no paired devices a scan can be made and pairing will be done with the device the user selects. After a connection has been made threads are started to handle reading and writing to the channel. This is required because both reading and writing to the channel are blocking call and will stall the GUI-thread and make the application unresponsive. If the connected device sends a special string */input YYYY-MM-DD, HH:MM, VALUE* this will be added into the database of blood glucose values. Values can also be sent without specifying a time stamp and in that case the current date and time will be added. The device can also send a command saying that it has finished with */exit* which will exit the program and return Bluetooth to its previous state (before the application was started). This solves the bug with closing the BluetoothSocket and leaving the BluetoothStack unusable talked about in the next section.

5.6.2 Problems with 2.1 Bluetooth

The emulator doesn't support Bluetooth so development has to be done on hardware which limits the version of Android used to the one installed on the device. The main test platform was the HTC Legend[13] running Android 2.1 connected to the computer through USB to be able to read log output during run-time. It was soon obvious that things didn't work as described in the documentation and a lengthy debug-process were done to figure out where the problems were coming from and some of the results are shown here:

- Closing the BluetoothSocket during a read (reading has to be done all the time, not to miss any messages coming from the connected device) as shown in the documentation will leave the Bluetooth stack in a unusable state, this means that any application that uses Bluetooth can't be closed without leaving Bluetooth on the device unusable. This can only be fixed by turning off and starting Bluetooth again.
- When the application starts it should check if Bluetooth is enabled, if not it should prompt the user to enable Bluetooth. This is done using an Intent to the BluetoothAdapter class (BluetoothAdapter.ACTION_REQUEST_ENABLE) this request can be denied or accepted by the user by a simple yes/no -dialog. If approved the class enables Bluetooth and sends back a message with either Activity.RESULT_OK or Activity.RESULT_CANCELED constants. This type of behavior is standard for all Android Activities that ask the user for permission. These are then read by the program that sent the Intent to see what the user has done. Testing has shown that these are mixed up on the HTC Legend. The BluetoothAdapter sends RESULT_OK when the user pressed "No" and Bluetooth wasn't enabled, a small bug that makes most Bluetooth applications that asks the user for permission to fail on the HTC Legend.

After talking to a Google developer, whom were very surprised about these issues, it was confirmed that the second bug were present in an earlier build of Bluetooth that were in the repository before the 2.1 release. Therefore testing were also carried out on the HTC Desire (2.1) and similar bugs but also some new issues were found. Testing was also done together with Serge Sozonoff who had access to the Nexus One and the Motorola Milestone/Droid. Both these devices use the standard 2.1 framework and seem to work as intended. Serge also confirmed that rooting a Desire and flashing it with a new ROM would relive these issues, once for all making sure that it's a software problem.

This leads me to believe that HTC grabbed an early build and didn't do proper testing. HTC were informed about these issues with an intent to try to solve this issues with one of their engineers. A request they replied to with little interest. Advertising a device as 2.1 when that's clearly not the case breaks the whole idea of having standard versions and breaks compatibility between devices.

5.7 Export/Import

Export and import of the database is a important feature if the user wants to do a backup or move to another device. Much of this code was reused from a tutorial[14] that Charlie Collins wrote. Support for XML export and illegal character replacement has been added and can later be used as a base to export data to an online service.

5.8 Database

All data saved by the application goes into a local SQLite[15] database only accessible by the application itself. SQLite is a part of the framework and not something that has to be implemented

separately.

5.8.1 Structure

Several design of the database were considered and implemented. All centered on a good way to save multiple entry types (measurement, medication, exercise) with a mix of common and distinct data-columns.

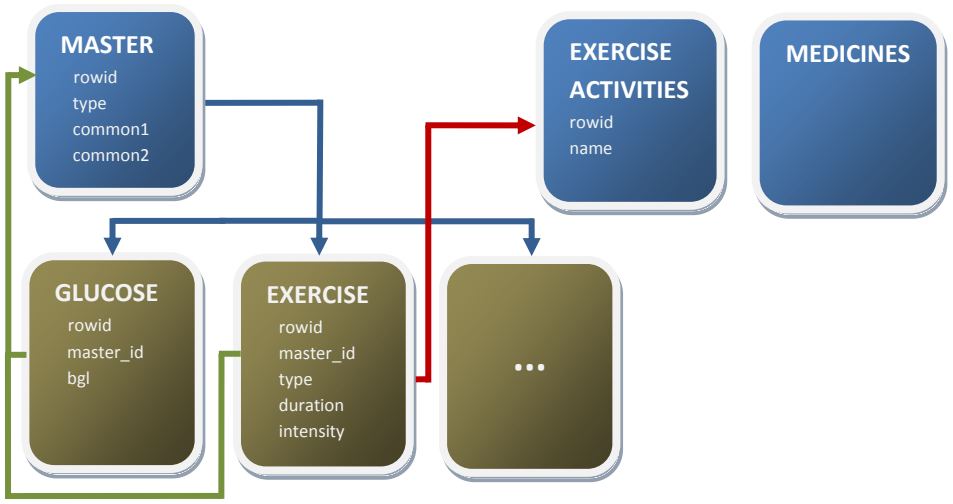

FIGURE 5.10: Block diagram of the database structure. All blocks represent tabels, and arrows show how data is related between tables.

The database uses a linked foreign key system where each entry has a main entry in a table specifying the common parameters and a separate table for all the distinct values for that particular entry type. These are then linked together by the foreign key as can be seen in figure 5.10. Compared with using a separate table for each entry type, this enables the application to easily list all the added entries without doing multiple table sorting. Tables containing exercise entries also holds a reference to a third table holding all the exercise activities (gym, cycling, golf etc.). This enables editing one value and directly updating all entries and also minimizes the information that has to be saved for each entry. The same implementation is used for all the medicines the user can select from.

The first time the application is started, the database is created and populated with default values for medicines/exercise types and user settings. These are all saved into the resources as lists of strings to allow for easy editing without having to dive into the program code.

5.8.2 MyDbAdapter

To be able to fetch data from the database, SQLite queries are made. Instead of directly using these queries from all parts of the application that needs access to the database a convenience class has been created that holds methods for these procedures. Whenever a part of the application needs access it creates an object of this class and opens a reference to the database. It then

has full access to all public methods. Using this method it's important to close the reference to the database when it's not needed anymore. Otherwise exceptions will be sent for holding a reference to a database in an Activity that's closed. For more information about this see [16].

5.9 Moving to Hardware

Testing the application were mostly done using the emulator as access to hardware came later in the project. Things should work the same way on emulator as in hardware, but a serious bug with the grapher was found on hardware and couldn't be reproduced on the emulator. The problem was rather random and hard to reproduce but left JavaScript turned off and redrawing of the graph impossible. Sometimes during a redraw of the graph it would just not redraw and logs showed nothing out of the ordinary.

There seems there can be problems with handling touch events in a WebView. When a touch event was recognized a redraw would be started and then the method would return true (because the event was handled). It seems that the system touch handler sometimes would see this as a timeout and exit the thread leaving JavaScript to crash because it was in the middle of execution. The solution is based around always returning true, and design so that there's only one TouchListener for that frame.

Chapter 6

Look and Feel

"To design is to communicate clearly, by whatever means you can control or master."

Milton Glaser

Interaction design takes a long time but is a necessary part of any application oriented toward consumers. Even if the functionality is there, the interaction has to make the user feel connected to what happens behind the screen.

6.1 Resources

Android has tried to separate layout from functionality by handling layout in separate XML files. Layouts are split into files that can be reused and modified through code so that the object themselves change but retain their layout properties. Usually a layout is defined to be used for each separate Activity the application contains. Layout are only one of several resources[17] that an application contains that's not part of the code:

- Drawables is a group of objects, usually images that the application uses. But drawables can also be XML files that define a shape (cube, box, etc.) that can be defined and drawn when loaded.
- There's also support for defining reusable styles. For example a style can be created for a TextView that defines how large the text will be and reused across the whole application.
- Instead of defining GUI text in code, a reference can be held to objects inside strings.xml. This is recommended because it lets the application to support different locales by creating different translations for each language. Android will then use the one that matches the users selection locale.

All these resource objects are then converted into memory addresses and accessed by calling the auto-generated R.java file with the resource name.

6.2 GUI Implementation

Surprisingly few widgets are standard to use. There's support for buttons, images, text, check boxes and some miscellaneous items, but that's about it. Therefore it's encouraged to build your own widgets and graphics to be used.

6.2.1 NumberPicker

There's a widget for selecting dates built into Android, compromised of smaller widgets where a number can be selected by a plus and minus button. Unfortunately it's not part of the public libraries, but was found after downloading the framework source. Because it's not public it's prone to be modified in a future release of Android and shouldn't be used in any application from its original position. Therefore the base code was copied and is now a part of the application package. Changes were also made to the appearance and functionality to add support for decimal numbers and a way to show the unit of the number the user inputs. Units were used for displaying mmol/l or mg/dL for glucose measurements.

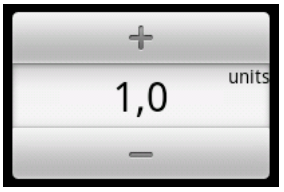

FIGURE 6.1: Screen capture of the NumberPicker used for input of glucose values.

6.2.2 Roller

Although the NumberPicker works great for inputting certain values, there was a need for a type of roller where you scroll through numbers in a faster manner. No previous documentation was found on how to create this so a roller was implemented by using a ListView with a custom background and behavior. The background together with fading at the top and bottom creates the illusion of a round object when the list is being scrolled. The real challenge was creating the functionality to be able to save the value at the row were the roller stopped, but was handled by creating a custom ListView with methods to calculate the offset of the list depending on the total number of rows and row height. Creating a roller object is done by specifying the start and end -values together with the incrementation.

FIGURE 6.2: Two rollers side by side, for setting time.

6.2.3 Animations

The grapher felt a bit stale during a redraw because it was hard to understand what was happening. Because of that simple animations were added to give the user some feedback. When you zoom in an animations locks to the position the user tapped and magnifies the area. When zooming out the reverse happens; the view shown will animate and become smaller as the CPU is doing the redraw. Similar animations were made when doing a transition to the next/previous month. The current view will slide in the direction indicated by the fling and then redraw.

Each animation is based on a set of transition points with an interpolate function between them. For example zooming in is based around the starting point (user tap) with normal zoom, and the end position has the same center but a smaller view. Between these positions all images are interpolated by an accelerating function. All together this gives a zoom animation that will start off slow but increase in speed.

6.2.4 Icons

Some icons/graphics were made from scratch or modified, but most were created by other authors and released for free. Icons were used from *milky* and *bright* from IconEden.com, *Coquette* from dryIcons.com, *medical* from KlukeArt.com and *Bluetooth* from Seanau.com. Credit goes to them.

Chapter 7

Conclusion

An application has been developed for the Android system that can log relevant data in an easy manner. Most of the logging is still done manually but features like My Tracks import and Bluetooth upload strive to make it easier for the user. There's still a problem of finding glucose meters on the market that support Bluetooth but applications like these might push patients to demand support from hardware manufactures.

Presentation is done in a way that enables users to see trends in their condition and draw their own conclusions. Features have been designed and implemented to make doctor and patient communication easier (sending graphs and emails) as well as giving security to parents of children with diabetes. Features which has never been implemented in any previous systems.

By being on the patients cell-phone access is always available, and the need of carrying an extra log has been eliminated. The system doesn't require an Internet connection to function and there's no need to send any patient information out from the system for functionality. Something that is required for online services.

A last meeting with Lotte Waller and colleagues at Mölndals hospital was held to evaluate the application from their point of view. Presentation and demonstration was met with positive feedback. But the real evaluation will be in the hand of the users themselves.

7.1 Discussion and Reflections

7.1.1 Android Development

As stated in the beginning Android is a fairly new system with a lot of momentum around it as can be seen in figure 7.1. This makes things exciting for application developers but also creates complications. Many device manufacturers make such large modifications to the basic framework that they can't support new releases of the system to their phones without investing large resources. This has lead to a current market where there are many different versions of the system and backward compatibility isn't always perfect. A more severe complication can be

that device manufactures advertise that they're running a version of the system when they're not as in the HTC Bluetooth case talked about in section 5.6.2.

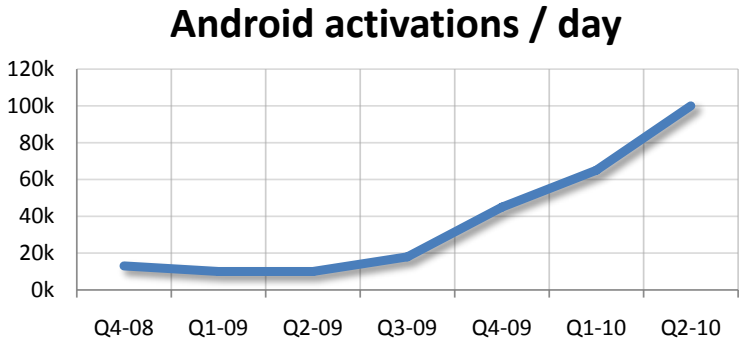

FIGURE 7.1: Daily activation of devices running the Android platform in the US. Source: Google I/O, 2010 via NPD Group (www.npd.com).

To help developers Google has put out a suit of example applications that demonstrate how accomplish different things within the system and many developers refer to these small examples for best practices. Although many are excellent some applications have questionable solutions in them. As is the case of the *"Notepad Tutorial"* where database management is handled in a way that leaves unclosed database references that will leak memory and cast exceptions when viewing the application log. Mistakes like these together with a surge of new developers has created a situation where a lot of people are trying to figure out to do things on the system and unnecessary mistakes are being made.

This also ties into the problem of getting exactly what you want. The framework designers have put in good documentations for most common things and as long as developments is done within that frame things come together rather easy. But the moment the invisible line is passed things get very complicated.

These are all problems that probably will be fixed as the platform matures. It should be clarified despite this, the guiding principles around Android make it an exciting platform to develop for. First: all applications are created equal, even core functions like the home-screen, virtual keyboard and contact viewer can all be exchanged for the users own versions. Secondly the openness of the project, the whole framework can be downloaded as open source from a public git. Third: no bias to what can and can't be installed; you can send application files and install them without going through the Google Market. This also means there's no censorship of any kind. Forth: There's a great development community around Android where people help each other out.

7.1.2 Glucose Meters with Bluetooth

A major feature was automatic uploads through Bluetooth; a technique that was believed to be supported already or soon by glucose meters used. After talking with DiaSend it was found out that there's a valid reason for this lack of technology adaptation. Glucose meter companies don't

make any money on the devices themselves but on the one-time strips that patients use for each measurement, with a cost of around 5 SEK each. The industry recently went through a shift where low-cost strips were manufactured, that forced the larger companies to lower their prices on strips and thus profits. For these reasons devices are designed to be as cheap as possible and Bluetooth hasn't been a priority.

7.1.3 Fall detection

A serious issue when having diabetes is the problems with hypoglycemia (low blood sugar) and is sometimes unpreventable, although the patient is doing everything right[18]. It's treatable by eating something containing sugar, but isn't always easy to detect. Hypoglycemia can among other things lead to shakiness, dizziness and seizure. If left untreated can make the patient pass out and go into diabetic coma which needs immediate attention from someone close by. A recent study[19] found out that around 7.3% of diabetics suffer severe[1] hypoglycemia in a 12 month period.

To prevent serious complications for patients that pass out a fall detection feature was designed that could alert friends/medical personal together with the GPS to give a position of the patient. This idea was first though out by Frank Sposaro and Gary Tyson in their article[20] but for elderly care. To be able to accomplish this the phone has to read the accelerometer and then detect a fall. Unfortunately this isn't supported when the device is in stand-by (power saving) even if a wake lock is initiated. This bug has been addressed with the newest release of Android 2.2 which is just a couple of days old at the time of writing and therefore wasn't implemented in the developed application but should be investigated in future work.

7.2 Future work

7.2.1 Application Specific

Effort should be put into acquiring the glucose meters out there with Bluetooth support and research how they send their data. After that support should be added to the application for these devices.

Another approach to do automatic uploading is by USB-cable. This was investigated as a possibility but both Android phones and glucose devices operate as USB-slaves and can only connect to a USB-host, a computer for example. Support for USB-host has been discussed[21], if added in a future release opens up a range of possibilities for communication. The most direct approach is connecting the glucose meter directly to the phone with a USB cable. But there could also a possibility to connect a radio-frequency (RF) adapter to the Android device and connect to the glucose meters out there that support RF which is more common that Bluetooth.

A web-platform could be developed that will sync with the device. The platform could serve as a base where doctors could log in and see all their patients information. Effort should be put into

[1] Severe hypoglycemia was described as when the patient have to have someone else help them. Either a friend or ambulance personal.

making the system as easy to understand as possible for the doctor for fast adaptation. There's a lot of laws governing how patient information has to be handled so research should start here. Encryption should be put on patients records during a sync as well as on the web-database itself. The solution should be tested for SQL-injections, brute-force login etc.

An evaluation has to be done by users to see how the application could be improved further. Questions about practicality, daily routines and additional features should be addressed. Only user themselves can be the judges of success.

7.2.2 E-health

The open source nature of Android makes for an interesting opportunity of customization. A version of the system could be downloaded and functionality customized to the point that it's no longer a smart-phone OS and instead a general purpose platform for the specific application developed. By cutting support for various features battery-life and performance could be improved vastly. This could then be deployed on any device that supports Android. The benefits is that mass produced relatively cheap hardware (older models) could be used for a very specific application. This could be an alternative where hardware costs are high because of low volume production to make a cheaper more flexible alternative.

Similar systems could be developed for remote-monitoring of patients with a wide range of problems. Many patients live in rural areas or are unable to visit the doctor because of complications but still require long time monitoring and E-health could improve quality of life for many of them.

Bibliography

[1] Erica Sadun, ArsTechnia. Apple changes to App Store review policy worry developers, 05 2009. URL http://arstechnica.com/apple/news/2009/05/developers-worried-by-apple-change-to-app-store-review-policy.ars.

[2] JR Raphael, PC World. Apple's New iPhone App Policy: Unreasonable and Unjustifiable, 04 2010. URL http://www.pcworld.com/article/193916/apples_new_iphone_app_policy_unreasonable_and_unjustifiable.html.

[3] Google. Designing for Performance, 2010. URL http://developer.android.com/guide/practices/design/performance.html.

[4] Marko Gargenta. Using NDK for Performance - Dalvik Versus Native, 2010. URL http://marakana.com/forums/android/android_examples/96.html.

[5] Scott Delap. Google's Android SDK Bypasses Java ME in Favor of Java Lite and Apache Harmony, 2007. URL http://www.infoq.com/news/2007/11/android-java.

[6] Google. Application Fundamentals, 05 2010. URL http://developer.android.com/guide/topics/fundamentals.html.

[7] Google. Platform Versions, 05 2010. URL http://developer.android.com/resources/dashboard/platform-versions.html.

[8] J. I. Mann. Diet and diabetes. *Diabetologia*, 18(2):89–95, 2 1980. URL http://www.springerlink.com/content/h0x533216610301v/.

[9] Google. My Tracks Website, 2010. URL http://mytracks.appspot.com/.

[10] Romain Guy Google. Make your Android UI Fast and Efficient. Google I/O 2009 Conference, 03 2009. URL http://www.youtube.com/watch?v=N6YdwzAvwOA.

[11] Bas Wenneker. Flotr JavaScript Plotting Library, 2009. URL http://solutoire.com/flotr/.

[12] Google. Android API, 2010. URL http://developer.android.com/reference/packages.html.

[13] HTC Corporation. HTC Legend Specifications, 2010. URL http://www.htc.com/europe/product/legend/specification.html.

[14] Charlie Collins. Backing up your Android SQLite database to the SD card, 2010. URL http://www.screaming-penguin.com/node/7749.

[15] D. Richard Hipp. Sqlite home page, 2010. URL http://www.sqlite.org/.

[16] Anders Widen. Database mangement and the Activity life-cycle, 2010. URL http://awiden.wordpress.com/2010/03/26/database-mangement-and-the-activity-lifecycle/.

[17] Google. Application Resources, 2010. URL http://developer.android.com/guide/topics/resources/index.html.

[18] American Diabetes Association. Hypoglycemia (Low blood glucose). URL http://www.diabetes.org/living-with-diabetes/treatment-and-care/blood-glucose-control/hypoglycemia-low-blood.html.

[19] Graham P. Leese, Jixian Wang, Janice Broomhall, Paul Kelly, Andrew Marsden, William Morrison, Brian M. Frier, and Andrew D. Morris. Frequency of Severe Hypoglycemia Requiring Emergency Treatment in Type 1 and Type 2 Diabetes. *Diabetes Care*, 26(4): 1176–1180, 2003. doi: 10.2337/diacare.26.4.1176. URL http://care.diabetesjournals.org/content/26/4/1176.abstract.

[20] Frank Sposaro and Gary Tyson. Geriatric Medical Application Suite on a Sweet Phone. Technical report, Florida State University, 2010.

[21] Various authors. Android on Google Code - USB Host feature, 2010. URL http://code.google.com/p/android/issues/detail?id=738.

Appendix A

Screenshots

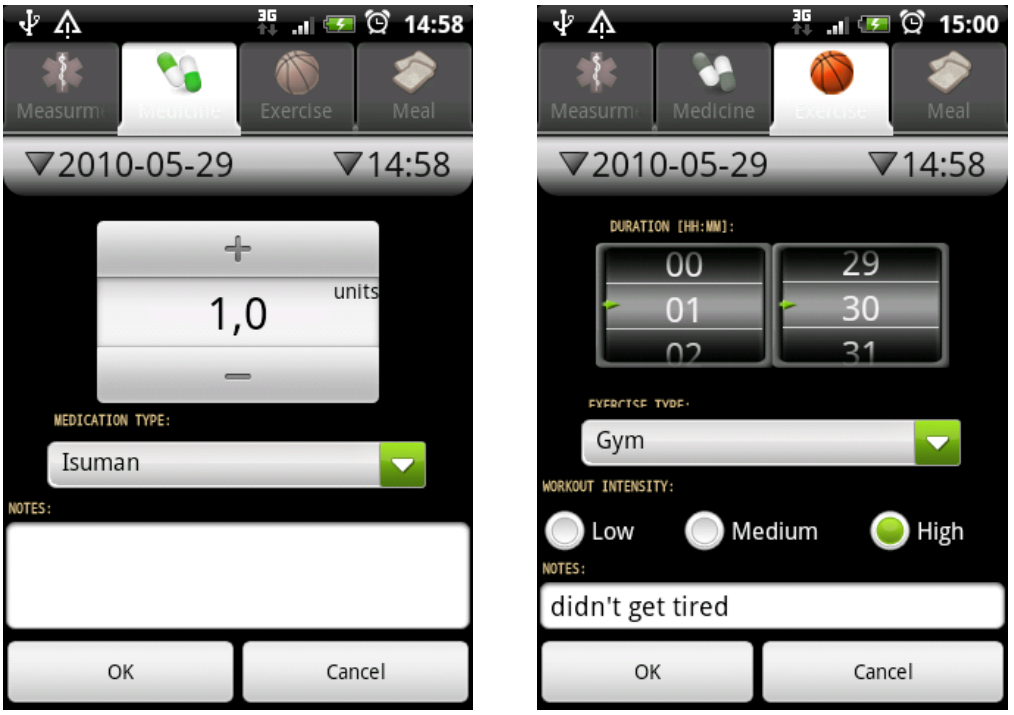

FIGURE A.1: Screenshots showing input of medicines (left) and exercise (right).

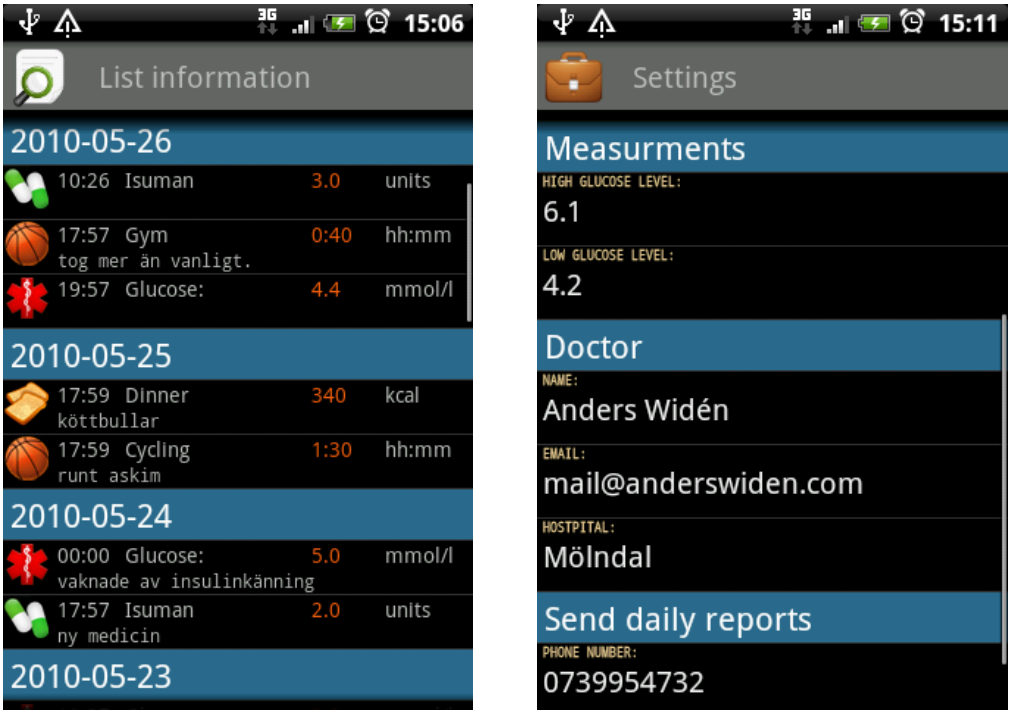

FIGURE A.2: List presentation of recent entries and settings.

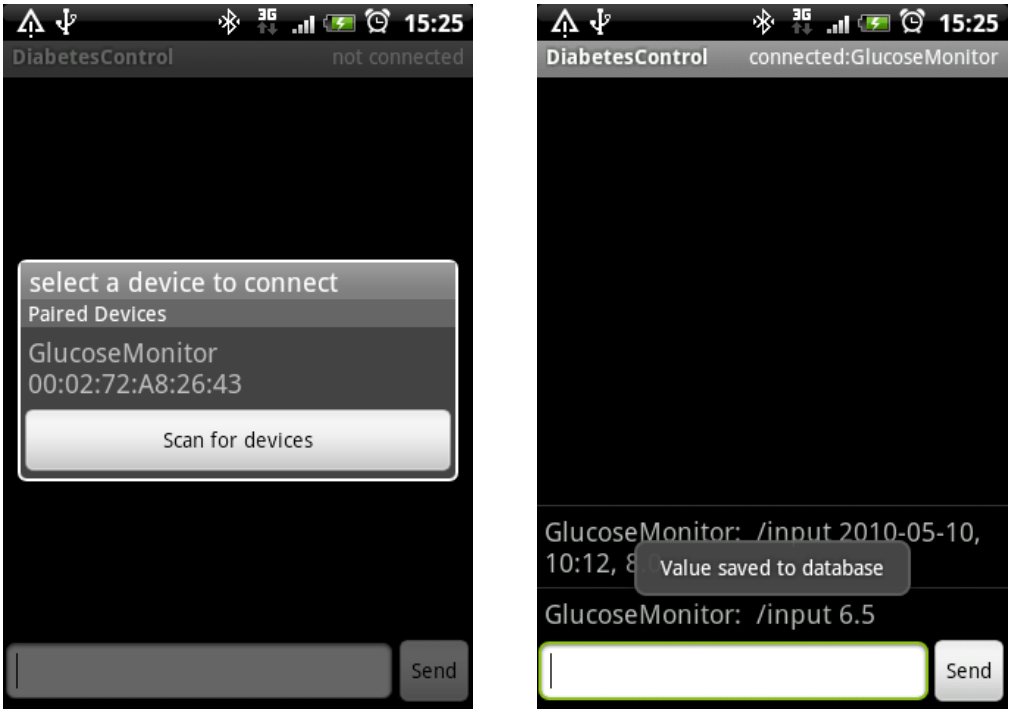

FIGURE A.3: Connecting to a glucose meter using Bluetooth and saving values from the device.

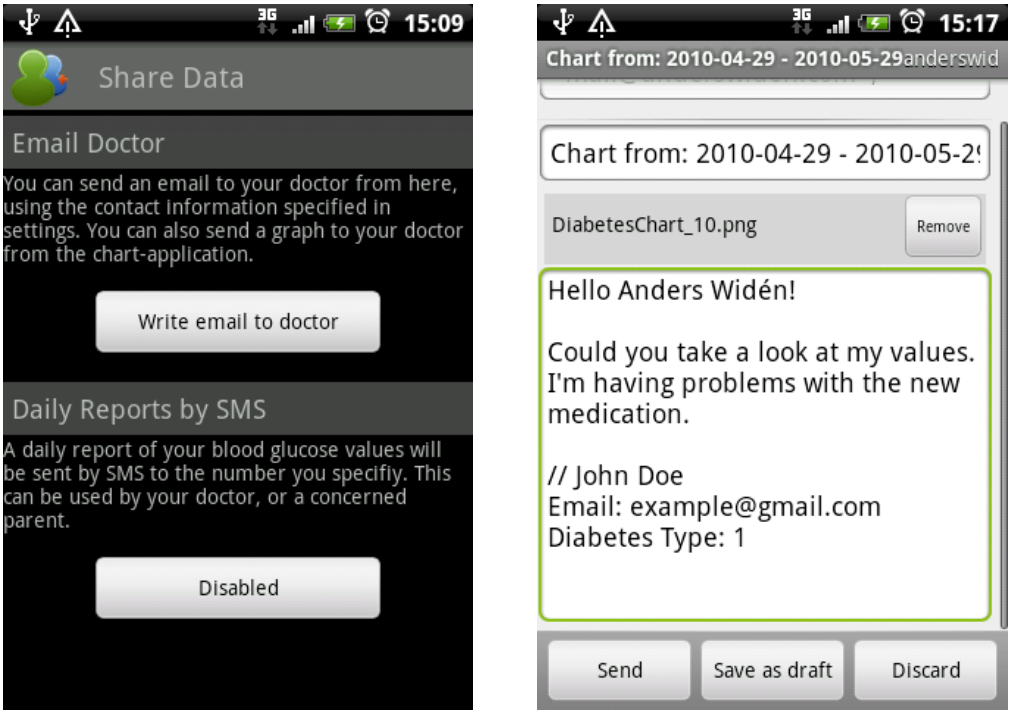

FIGURE A.4: Left: Sharing data by email or daily reports. Right: Sending a graph to your doctor by email.

FIGURE A.5: Left: Sharing data by email or daily reports. Right: Sending a graph to your doctor by email.

Appendix B

Requirements Document

Must...

- Provide logging for measurements, medicines, exercise and meals
- Run on a device with Android 1.5+.
- Run on a device with Android 2.0+ for Bluetooth implementation to work.
- Function when phone is in *airplane* mode.

Should...

- Add features not seen before.
- Be fast and responsive.
- Look professional.
- Test different techniques inside Android.
- Have function to send graphs to doctor.
- Integrate with GPS exercise solutions.
- Provide a proof of concept of what can be done.

Must not...

- Send user data outside the device for functionality.
- Require an Internet connection.

Appendix C

Functionality Mindmap

- Log - Medical
  Blood glucose level (before, after meal), choose medicines: (insulin, other, automatic pump, needles).
- Log - Diet
  Meals (what has been eaten, carbohydrate-counting), list of food (nutritional information), Weight.
- Log - Exercise
  Walking, running, cycling (measure, distance, speed, maps, GPS, keep track of results, pedometer).
- Add data
  Manually by user (has to be easy and fast), automatic (bluetooth, usb adapters), widget on desktop.
- Present data
  Graphs (color coded, show different regions, plot vs weight, exercise, drugs, help user to understand how things interact), lists of data.
- Share data
  Doctors, parents (before checkup, remote monitoring, parents can keep track of their child's values, web portal).
- User protection
  Send critical information to parents, fall detection (long lie, diabetic collapse, gps, phone, sms, alert surroundings).