The Report committee for Vaidehee Padgaonkar Gokhale
Certifies that this is the approved version of the following report:

Enabling Telemedicine with Smartphones

APPROVED BY

SUPERVISING COMMITTEE:

Adnan Aziz, Supervisor

Mark McDermott

Enabling Telemedicine with Smartphones

by

Vaidehee Padgaonkar Gokhale, B.S.E.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2011

I would like to dedicate this report to my family...

My parents, Dr. Arvind Padgaonkar and Dr. Vanita Padgaonkar, for all their love, encouragement, and guidance throughout the years.

My sister, Dr. Vaishalee Kenkre, for being an amazing role model all my life.

My brother-in law, Dr. Prabhav Kenkre, for all the medical expertise he provided for this report.

My husband, Samir, for his love, support, and incredible patience as I went through the Masters program while working full-time.

Acknowledgments

I would like to acknowledge my supervisor, Dr. Adnan Aziz, for the support and guidance that he provided to make this project possible. I would also like to thank Mark McDermott for reading and providing feedback on this report.

Enabling Telemedicine with Smartphones

Vaidehee Padgaonkar Gokhale, M.S.E.
The University of Texas at Austin, 2011

Supervisor: Adnan Aziz

As smartphone technology continues to mature, one of the many areas it can help enhance is telemedicine: the concept of using telecommunications to provide health information from a distance. A new medical condition or disease can require frequent visits to the doctor for simple biometric monitoring. These frequent visits are time-consuming and can be extremely inconvenient for the patient. This report describes how a smartphone can be the optimal platform to communicate critical biometric measurements to one's physician, reduce in-person hospital visits, and still allow for the patient to receive feedback from the doctor. A proof-of-concept infrastructure for enabling telemedicine is demonstrated by interfacing a glucose meter with an Android device that uploads that data to the Cloud to be viewed by the doctor.

Table of Contents

List of Tables

List of Figures

Chapter 1

A New Vision for Telemedicine

The concept of telemedicine has been around for decades and in recent years laptops have served as the platform to communicate a patient's biometric data to doctors. This project expands that concept by using smartphones to provide more mobility than a laptop and communicate biometric data to doctors through the cloud. The goal of such a system is to reduce in-patient visits and provide flexibility for the patient to collect and communicate biometrics from anywhere.

The use model envisioned involves a patient collecting a biometric reading with an off-the-shelf monitoring device and then interfacing that device with a smartphone. An application on the smartphone, also referred to as app, can be launched to automatically extract the reading from the device and upload that data to the cloud for the doctor to view. This infrastructure is depicted in Figure 1.1.

While there are many conditions that would be good candidates for telemedicine, diabetes was the one selected for the proof-of-concept infrastructure detailed in this paper. The medical community widely accepts at-home monitoring for diabetes using off-the-shelf glucose meters, also referred to as

glucometers. The medical use model is already in place, which was the main motivation for starting with this condition and biometric device.

The American Diabetes Association reports that in the United States alone, 28.5 million people have diabetes and 79 million people are in a pre-diabetes phase. Uncontrolled diabetes can lead to more severe conditions like diabetic nephropathy, heart disease, and retinopathy [1].

According to Dr. Prabhav Kenkre, an internal medicine doctor at the University of Wisconsin, the medications and treatment methods for diabetes are in place, but the greatest challenge with keeping diabetes under control is patient compliance. Glucose meters are available for self-testing, but patients are not keen on drawing their own blood five times a day and logging the data. Dr. Kenkre believes that improving the comfort or convenience of this process would increase patient compliance and potentially avoid more severe conditions from developing. The following is a typical scenario that he encounters with some recently diagnosed diabetic patients.

1.1 Case Study of a Diabetes Patient

John Doe, age 42, has been experiencing symptoms of fatigue and frequently wakes up in the middle of the night to urinate. He currently weighs 203lb, which is considered to be slightly overweight for his height of 5ft 9in. John has not had a physical examination in a few years, so his wife urges him to see a doctor.

The doctor checks John's blood pressure and pulse, and both of the measurements are normal. The doctor then decides to gives him a physical and everything is normal except that he is slightly dehydrated. When the lab work comes back, his serum glucose level is 331mg/dL, which his doctor considers to be extremely high. The patient did not come in that day expecting a physical, so this was a non-fasting reading. The doctor asks John to return the next day after fasting for another testing. When those results come back, his glucose level is now 219mg/dL. Normal levels after fasting should be under 126mg/dL. The doctor informs the patient that he has type II diabetes. John is surprised, and slightly in denial, because no one in his family is diabetic and he is relatively young.

The doctor starts him on a medication called Metformin and asks him to come back two weeks later. On that third visit, his fasting serum glucose level is 197mg/dL. The doctor now prescribes another oral medication, Glucotrol, in addition to Metformin. Another two weeks pass, John returns, and this time his glucose level is 168mg/dL.

The standard medications do not seem to be rectifying his condition enough, so the doctor informs John that he will require insulin injections and will need to keep track of his blood sugar levels. As the weeks pass, John's work schedule gets busier and he finds it harder to keep up with the measurements and journal entries. After years of this intermittent self-monitoring and treatment, John develops nephropathy (kidney disease) and requires three-hour dialysis sessions in the hospital three times per week. He is now unable

to work full time because of the amount of time required for his treatments.

1.2 Current State of Biometrics on Smartphones

In the scenario presented, John had four hospital visits just for the initial diagnosis and medication plan. These visits could have been consolidated to one visit if a telemedicine infrastructure were in place. Moreover, if the logging process had been automated and frequently communicated to the doctor, John may have stayed on track with his medication plan and could have avoided developing a second condition that completely changed his life.

There are some biometric applications available on Android today, but they are limited to manual logging, which provides nothing more than an electronic notebook for patients to document their readings. There are a few apps that attempt to use the smartphone built-in features to collect biometrics. An app called "Instant Heart Rate Monitor" is an example of this. It uses the smartphone's camera flash on a human finger to determine heart rate, but the readings are very inaccurate and this is not an FDA approved method. For a true telemedicine infrastructure to be successful, doctors must feel comfortable that the devices being used for data collection are accurate. The Bayer USB Glucose meter used for this project is FDA approved, which was a major consideration in the selection process.

1.3 Project Development Overview

The selection of components was the first step. A Bayer USB Glucose meter played the role of the biometric device. The Motorola Droid A855 running Android OS 2.2 was the initial candidate smartphone. However, USB APIs are not available on versions before Android OS 3.1. Since smartphones today cannot run Android OS 3.1, a Motorola Xoom tablet was used instead to emulate capabilities that smartphones will have within a year. The Xoom was then used to download the readings and upload them to the Google AppEngine Cloud service.

After deciding on the hardware and software infrastructure, the approach was to focus on highest risk items first, which was the hardware development since it did not have as clear of a path as software. There is also a lead time to order and receive additional or replacement components. For these reasons, reverse engineering the glucometer became the first focus. Once communication with the glucometer had been established, development work on the smartphone began. Challenges with USB host support on current smartphones led to the use of a tablet instead and will be discussed in more detail in Section 2.3. In order to enable USB host mode on the Xoom, administrative privileges had to be enabled, a process also referred to as rooting. Also, a USB OTG cable was required to interface the glucometer with the Xoom. The final milestone was the software development of an app that extracted the glucose readings onto the Xoom and uploaded them to the app engine datastore. The results are displayed on a website for the doctor to view.

1.4 Report Outline

This report will first detail the hardware engineering and then the software engineering. The hardware development of reverse engineering the glucometer and enabling USB host mode on the Xoom had to take place first in order to understand the software requirements for the App and AppEngine solutions. The final chapters will summarize the development costs and timeline, hardware and software best practices, and future work that would be required to commercialize this proposed infrastructure. The key contributions that will be described are:

1. Reverse engineering the Bayer USB Glucometer
2. Reverse engineering the Contec Pulse Oximeter
3. Hardware enabling for USB host mode on Motorola Droid A855
4. Hardware enabling for USB host mode on Motorola Xoom
5. Software development for the Android Biometric App
6. Software development for the cloud solution

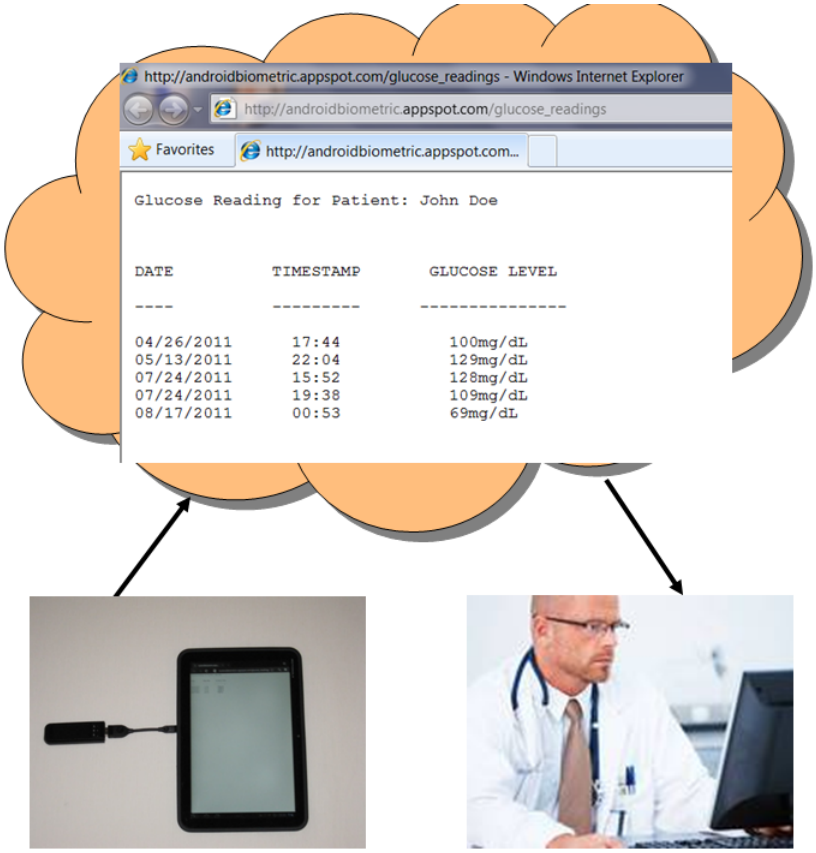

Figure 1.1: Infrastructure for telemedicine using smartphones

Figure 1.2: Summary of the development process

Chapter 2

Hardware Engineering

Hardware engineering was required for the biometric device as well as the host platform. The Bayer USB Glucose Meter was an ideal choice for the biometric device because of its low cost, FDA approved status, and small form factor. Integrating this device to a smartphone was intended to provide a true mobile infrastructure. Most people today may not always have their laptops or Wi-Fi access. Most people today do, however, have their smartphones connected to a 3G or 4G network with them at all times and locations.

This chapter discusses the hardware development, challenges, and results of interfacing the Bayer USB Glucometer with an Android device.

2.1 Reverse Engineering the Bayer USB Glucometer

The Bayer USB Glucometer comes with Glucofacts software that is Java-based and supported on Windows and Mac OS. When it is plugged into a PC or Mac, the device lights up and indicates it is in a USB charging mode. Upon interfacing the glucometer with a PC, two devices are mounted, enabling access to the device through the computer's filesystem. One of the devices represents the Glucofacts software and the other is for USB storage

of result snapshots in Adobe PDF. The readings were not readily available on either of the devices mounted. It was reasonable to believe that the readings are stored in flash memory inside the device and the software launch initiated a USB transaction triggering the data download.

There is some native C-code, which prevents it from being plug-n-play on Linux-based platforms without a recompile of the source code. This, along with the lack of technical documentation and Bayer's refusal to provide more information on this patented device, presented an interesting challenge, namely how to extract data from the glucometer without Glucofacts. With the help of an open-source USB snooping software available online, the process of reverse engineering the glucometer became a relatively straightforward task.

Each USB device has a vendor identification number and product identification number. These identification numbers are required for the USB snoop tool to attach to the correct device. In Ubuntu, the *lsusb* command provides this information. For the Bayer USB Glucose meter, the vendor ID was 0x1a79 and the product ID was 0x6002. *Usbsniffer* is a Windows program available online [8] and is similar to the Linux usbmon program in that it reports the USB traffic until the recording is terminated by the user. The *usbsniffer* traffic captured while Glucofacts was launching was in ASCII hex format. While the glucose readings of interest were detectable in the usbsniffer output, converting this into programmable commands was the next challenge. Fortunately, a usbsnoop2libusb Perl script was available online [6].

The conversion script translated the usbsniffer log to C-code that uses

libusb, a Linux-based USB library. The C-code replays the transactions recorded by the sniffer tool. Once this code was compiled and executed it became clear which transaction and snippet of code corresponded to the glucose reading download. Not only was this a crucial first milestone, but it also highlighted a general reverse engineering process for any standard USB device.

Some software development forums have discussions about the inflexibility of the Bayer USB Glucofacts software [2]. Users complain that the Glucofacts GUI is too complex and does not offer an option to save readings in a simple csv format to be filtered and analyzed in excel. Other comments point out that Glucofacts is limited to Windows and Mac OS, so for Linux users it is inconvenient to switch OS just to use the Glucofacts software. Furthermore, since most smartphones are Linux-based, current Glucofacts users won't be able to directly port the software to their phones. The reverse engineering process just outlined offers a solution to create a more configurable software.

2.2 Reverse Engineering the Contec Pulse Oximeter

Although the glucometer was the primary biometric device used to demonstrate this infrastructure, some time was spent examining other uses for telemedicine and what it would take to enable another biometric device. In a situation where a patient is having intermittent symptoms such as palpitations, sweating, dizziness, which do not present at the time of the in-patient visit, it becomes difficult for a doctor to accurately diagnose the condition. It would

be ideal to send the patient home with a portable electrocardiogram (ECG) device to collect data when the symptoms do arise and communicate those recordings to the doctor immediately through a smartphone.

In attempt to reuse the reverse engineering method, the Contec Pulse Oximeter with a USB interface was selected. However, when applying the same reverse engineering methodology the USB traffic logged did not correspond to the readings on the device's LED interface. Upon further research, I discovered that this device had a CP210x device and could simply be read as a character device, so I wrote C-code to extract the pulse and oxygen readings real-time in Ubuntu. The intent of selecting a second device was to demonstrate that there is little incremental effort to enable more biometric devices once the infrastructure was in place for one device. Although the pulse oximeter did not require the same reverse engineering process, it is still a good candidate for telemedicine because of its small size and potential application.

2.3 Smartphone Challenges

The smartphone was intended to be the USB host that powers up and initiates transactions with the USB slave device, in this case the glucometer. Very few Android smartphones today have USB host mode hardware support built-in, and none of those that do are in USB host mode by default in order to avoid unnecessary power consumption. One of the few phones that has the needed hardware support today is the Motorola Droid A855.

2.3.1 Enabling USB Host Mode on the Motorola Droid

A micro-dongle was needed to boot the Droid into USB host mode. The micro-dongle pictured in Figure 2.1 that I built by detaching the micro-USB end of a car phone charger, removing the resistor, and creating a solder bridge to short the USB mode ID to ground [7]. Figure 2.2 shows the micro-dongle device that was inserted into the Motorola Droid's USB port on boot-up. It was removed before the OS home screen appeared. At that point, the Motorola Droid was in USB host mode and the glucometer went into charging mode when connected to the Droid via a standard USB to micro-USB adapter (see Figure 2.3). The caveat to this is once the device was removed, the Droid defaulted back to USB slave mode. To configure it into host mode, a reboot with the micro-dongle was required.

2.3.2 The Motivation for Android 3.1

Aside from the non-ideal reboot, USB APIs required to program an app that could communicate with the glucometer became available with the Android 3.1 release, which is not currently available on smartphones. Google has reported that Smartphones running Android 3.1 will become available by the end of 2011 [4], which motivated a shift for this proof-of-concept project toward a tablet running Android 3.1 rather than a smartphone running Android 2.2.

2.4 Enabling an Android Tablet

Selecting of the tablet model was arbitrary choice, but the slight inclination toward the Xoom over others was due to an article that made specific mention of its USB host features [4].

The Xoom had to be rooted for superuser permissions to be enabled. To do this, I unlocked the Xoom by connecting it to my PC via USB and use the Android adb tool to run a reboot command (*adb reboot bootloader*). After the Xoom rebooted, I had to flash the recovery image and install a bootloader file [5]. A USB OTG cable was required to interface the glucometer with the tablet. A standard USB to micro-USB adapter that was used for the Motorola Droid USB host configuration was not sufficient because it did not provide the host and peripheral role swapping that the OTG cable does. Upon connection through the OTG cable, the glucometer powered into USB charging mode (see Figure 2.4). This setup did not require a reconfiguration reboot for USB host mode like the Motorola Droid did.

An Android app called *Terminal Emulator* provides an xterm-like functionality and supports the *lsusb* command. It was a quick and easy way to gain confidence that the Xoom was able to detect the glucometer and determine what device it was mounted to. The next challenge was to start communicating with the device through an Android app.

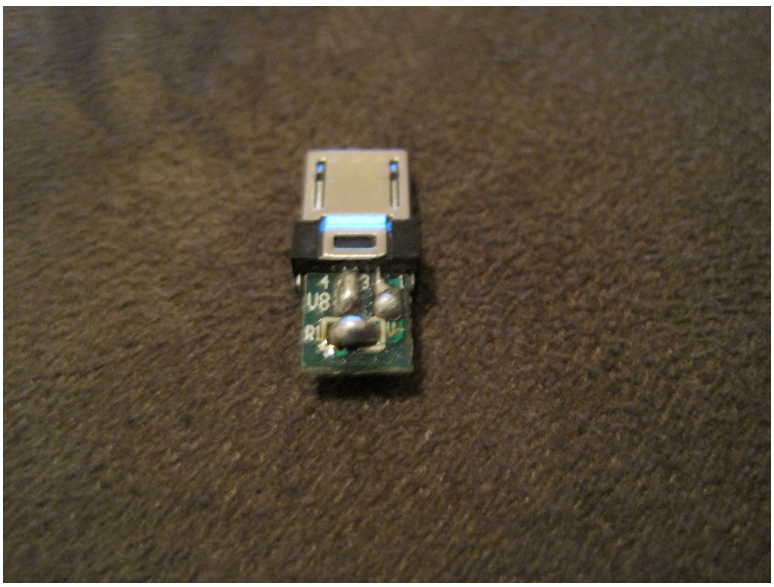

Figure 2.1: Micro-dongle for Motorola Droid's USB host mode

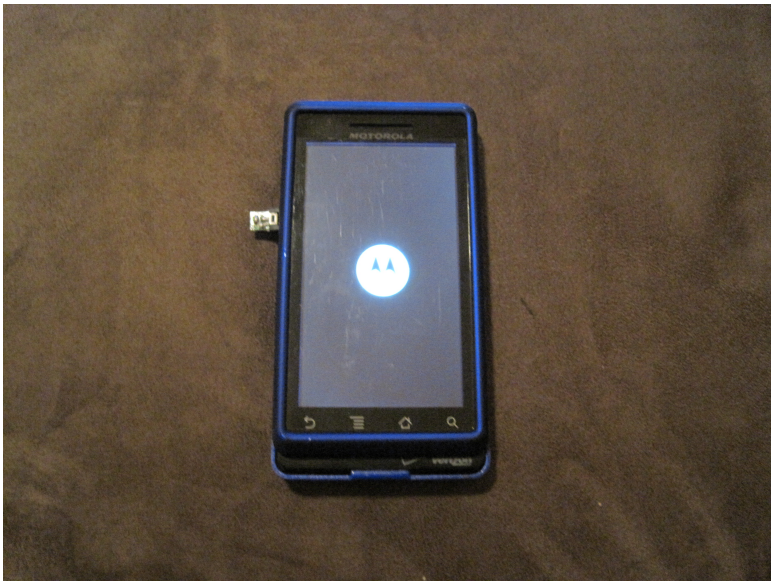

Figure 2.2: Motorola Droid booting into USB host mode

Figure 2.3: Glucometer powered by Motorola Droid in USB host mode

Figure 2.4: Glucometer interfaced with the Motorola Xoom

Chapter 3

Software Engineering

Logging glucose measurements into a journal is a task that diabetes patients find to be a deterrent for self-monitoring, according to Dr. Kenkre. Software that automates the administrative details and enables remote communication with the doctor make patients more accountable and could potentially increase compliance. This project attempted to address that through the development of an Android app, which downloads the readings from the glucometer and then uploads them to the cloud, demonstrated through Google App Engine.

Figure 3.1 shows the software flow and interaction that will be discussed in the next few sections.

3.1 Android App Development

When the glucometer was reverse engineered, C-code was generated to mimic the USB bus transfers in Ubuntu 10.10. This was a good starting point for understanding what commands needed to be issued from the USB host to the glucometer in order to get the desired readings. I wrote a Perl script to parse the write and read commands from the C-code and translate them

to Java USB host API calls. The complete USB API features were released with Android 3.1 in May 2011 and are essential for an app that needs to communicate with a USB device. In the Android SDK, API level 12 or higher should be used since USB host APIs were not available before that.

Before the bulk transfers are issued, some setup is needed including requesting and receiving permission to communicate with the USB device, getting the interface and write/read endpoints of the device, and claiming the interface. It is worth noting that there are approximately 300 write/read bulk transfers in this app. The commands that actually handle requesting and receiving the glucose readings are only a few of those 300 transfers. The additional overhead is due to the fact that the C-code that this Java app is based on was generated from USB traffic snooped while Glucofacts was being launched. In the absence of engineering specs, it was unknown how to configure the device more efficiently using less commands. In the read routine, there is a check for whether the data just read back is a glucose reading. It can be identified by the "Glucose" that precedes the glucose value. The reading is immediately sent though the HTTP GET method to the App Engine, but is not stored on the Android device itself. The details of the cloud solution through App Engine will be discussed in Section 3.3.

3.2 Software Reuse for Pulse Oximeter

Although C and Java code was developed to extract values from the pulse oximeter in Ubuntu, complete integration with the Android app was

infeasible due to lack of time. Integration of the pulse oximeter or any other USB biometric devices with the Android device would require a reverse engineering step and a new set of bus transactions in the Java App. However, the USB configuration in the app and the communication with the cloud could potentially be reused.

3.3 Cloud Solution

Google App Engine is a cloud computing infrastructure for public development and hosting of web applications using Google's data centers. It provided a free and simple solution to demonstrate the cloud aspect of this project. There is a dedicated servlet for the glucometer measurements that queries for a key ("Glucose"). Although each new entry is passed in with this key, the reading is stored with a different key ("StoredGlucose"). In order to avoid duplicate entries, the entire datastore (i.e. App Engine database) is traversed and an entry is only added if it does not already exist in the datastore. The app running on the Android device is also programmed to launch the app engine site, via a browser, and sit on top of the app interface.

The data download and upload takes place within ten seconds. Most of this time is due to the overhead of about 230 write/read transfers between the app and the glucometer before the readings are downloaded. These were transactions that snooped while the Glucofacts software was being launched. Removing all of them prevented any data download. While some of the transactions likely could have been eliminated, isolating those would have required

time-consuming guess-and-check work. Since this project was meant to serve as a proof-of-concept, I decided to accept the ten second data download delay.

The text on the glucometer display flashes off and back on to indicate the readings have been extracted. The results are then immediately available for the doctor to view at `http://androidbiometric.appspot.com/`. In the event that a Wi-Fi connection is unavailable, the readings downloaded will be displayed on the app interface but will not be uploaded to the App Engine until the next app launch when a network connection is present. Figure 3.2 shows the main app engine site representing one patient's biometric data communicated from a smartphone. Each device would have its own servlet link.

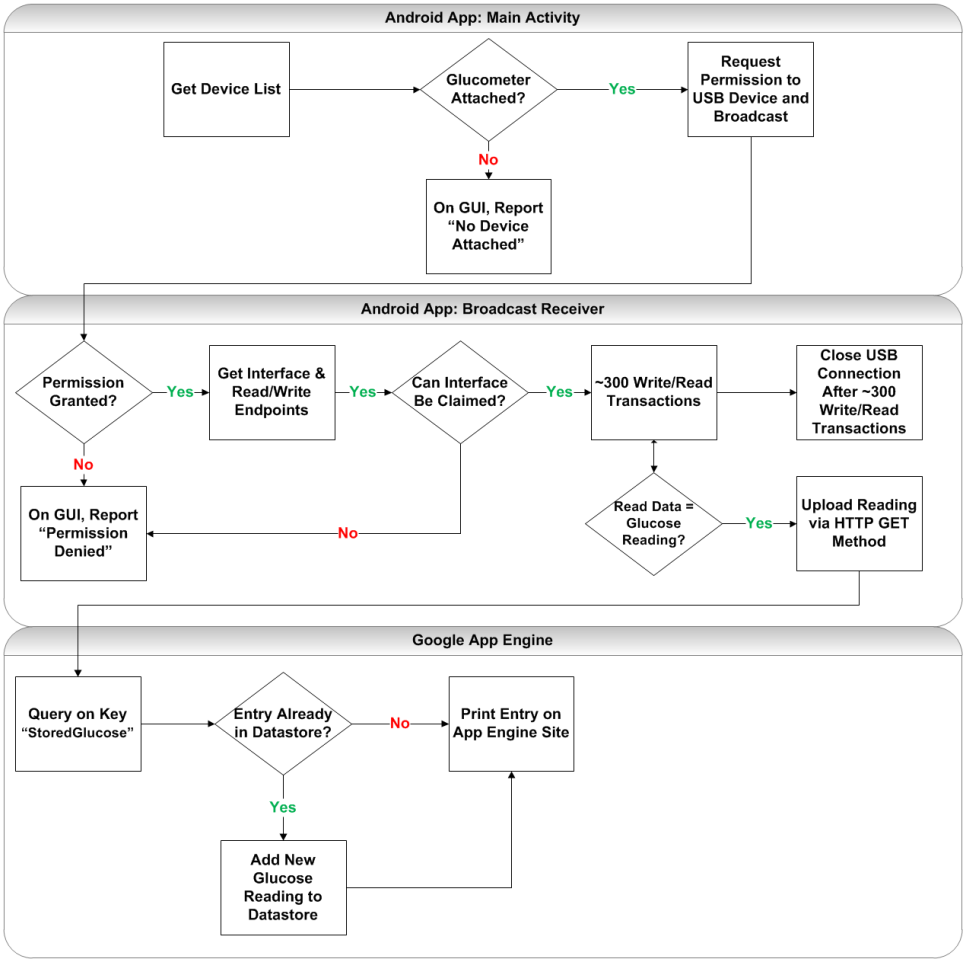

Figure 3.1: Android app and app engine software flow

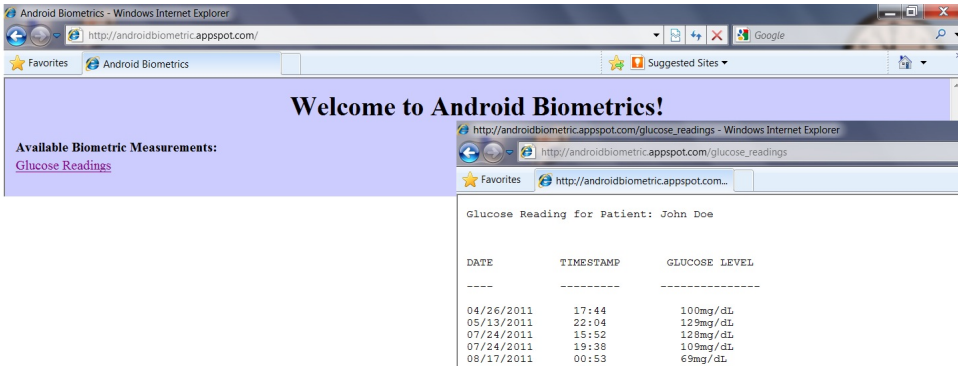

Figure 3.2: Cloud interface powered by Google App Engine

Chapter 4

Summary of Results

The hardware and software engineering required to build a telemedicine system with smartphones as the communication medium has been demonstrated. This chapter will provide an analysis on that development.

4.1 Cost of Infrastructure

Table 4.1 provides a cost breakdown for the system developed. The only incremental cost for a diabetic patient to use this infrastructure, however, would be the $20 for the USB OTG cable. This assumes the patient already has a smartphone, can download the application for free, and will purchase a glucometer and test strips anyways.

Table 4.1: Summary of Development Costs

Item	Purchased From	Cost
Bayer USB Glucose Meter	http://www.amazon.com/	$30
Bayer Glucose Test Strips (25 count)	Walgreens Pharmacy	$25
Contec Pulse Oximeter CMS50E	http://www.amazon.com/	$100
USB OTG Cable	http://www.amazon.com/	$20
Motorola Xoom	http://www.bhphotovideo.com	$500
Software Development	NA	$0
Total		$675

4.2 Development Timeline

The project duration was approximately four months at 25% effort and Table 4.2 summarizes the amount of time required to reach each major milestone. The reverse engineering method used for the glucometer took only a few minutes, but arriving at that point took about a month. Attempting to enable USB host on a smartphone also took about a month and this work could not be used in the final product. The tablet enabling took approximately a week and the app development took three weeks. About two weeks was spent on trying to upload data to a simple Google online spreadsheet, but I was unable to write to it. This motivated a shift to use App Engine for the cloud solution and it took just a few days since the online tutorials and examples were thorough.

Table 4.2: Summary of Project Timeline

Milestone	Duration
Reverse Engineering Bayer USB Glucose Meter	4 weeks
Reverse Engineering Pulse Oximeter	2 weeks
Enabling USB Host for Smartphone	4 weeks
Enabling USB Host for Xoom Tablet	1 week
Android App Development	3 weeks
Google Spreadsheet for Cloud Solution	2 weeks
Google App Engine for Cloud Solution	1 week
Total	17 weeks

4.3 Hardware and Software Best Practices

Some practices worked out very well and some that could have been done better. This section will explain the following key learnings:

- Assess hardware and software interaction early in the project
- Outline a hardware and software solution that will be optimal for both
- Prioritize based on risk
- Always opt for officially supported software over hack solutions

The interaction of the hardware and software in this project was key to the success of this project. Although the selection of the biometric device seemed like a purely hardware decision initially, it actually set the course of the entire project. A USB device was cheaper than a bluetooth device, but bluetooth API support is available on the Android version that runs on today's smartphones. The use of a USB device with Android was an interesting and unexpected challenge that required more time than originally anticipated. It left less time for enabling a second device.

It is important to assess early on that there will be a suitable hardware and software solution when going down a particular path. I should have considered software options for USB host mode on a smartphone before trying to enable it. There are no officially supported USB APIs for the Android version running on phones today. Had I considered this, I could have made the decision to use a tablet, in place of a smartphone, much sooner.

Another approach that worked well was prioritizing based on risk. Communication with the glucometer was the highest risk because without this baseline, there was no project. It also had the least amount of documentation and support forums. The remaining hardware development was the next point of focus because if additional components were needed then there is some lead time to order and receive it. Software development along the way was needed to gain confidence in the success of the end result, but the initial development and validation in Ubuntu was a more familiar route for me and was a reasonable platform since Android is Linux-based.

The decision to use supported software allowed me to focus on the actual development instead of wasting time on getting hacked code to work. There was a hacked javalibusb wrapper for Android available online, but this would have been an obsolete solution once Android 3.1 becomes available on phones. The advantage of using officially supported USB APIs is that now there is a portable solution for future android smartphones.

When it came to the cloud solution, I was trying to find a "easier" way by attempting to write to a Google spreadsheet. I spent two weeks on this with no promising results, at which point I tried to use Google App Engine and was successful in enabling that within a matter of days. This reinforced the idea of using supported software rather than work-around solutions.

Chapter 5

Future for Proposed Infrastructure

I have provided a proof of concept for telemedicine through smartphones, but more development work would be required if this concept were to be commercialized. This final chapter will describe the improvements and enhancements that could be made to this system.

5.1 Software Enhancements

While the app and App Engine developed for the Xoom can be easily ported to a smartphone running Android 3.1, there are certain limitations that should be addressed.

Currently, the biometric device has to be attached before launching the app. The app code could be modified to listen for the device in the event that it is attached while the app is already running.

Another feature that could be added is storing the data on the Android device in case there is no network connection and the data needs to be uploaded at a later time. Currently, if a network connection is unavailable when the glucometer is connected to the android device, the data will just be displayed on the app interface and downloaded the next time the glucometer is docked

to the Android device in presence of a network connection.

This project did not account for the doctor-patient confidentiality in the sense that the app engine currently has no authentication, so anyone can view and edit those glucose readings. Therefore, a secure login feature would need to be added. Also, the information on the site is very raw and visually unappealing; trend charts would be a good visual aid to add.

The original motivations for this telemedicine infrastructure was to (1) improve doctor-patient communication without excessive in-person office visits, and (2) increase compliance by adding a level of accountability. To further support these goals, alert features can be added to the app. For example, if the glucose readings start trending abnormally, the app could send a text or page to the doctor. The app could also support alerts notifying the patient's family members the glucometer has not been docked to the smartphone for an extended period of time, indicating a lack of compliance.

5.2 Enabling Additional Biometric Devices

An important aspect when selecting a biometric device intended for telemedicine is that the device itself must have a small form factor. This can be limited by the actual function of the device. For example, a blood pressure tool has to wrap around a person's arm, so its compactness is limited by the physical size of the body part that it is servicing.

The glucometer and pulse oximeter were good candidates for this project

because they were pocket-sized and low cost. Through this effort, the process of reverse engineering the devices and the pieces for hardware and software development are available. If the concept of telemedicine via smartphones comes to fruition, the next focus should be how to make biometric devices in a smaller, low-cost form factor.

This new vision of telemedicine can also be expanded to rural areas that cannot afford large hospitals and do not have the medical expertise. Moreover, for a relatively low cost to the patient, health care and doctor-patient communication could be revolutionized by a telemedicine infrastructure enabled through smartphones.

Bibliography

[1] American Diabetes Association. American Diabetes Association 2011 Statistics, January 2011. http://www.diabetes.org/diabetes-basics/diabetes-statistics/.

[2] Ubuntu Forums. Bayer Contour / Glucofacts, 2009-2011. http://bit.ly/nQnEv4.

[3] Google. Android Developer, May 2011. http://developer.android.com/index.html.

[4] Matt Hamblen. Google Rolls Out Android 3.1, May 2011. http://bit.ly/lO3SuX.

[5] Steady Hawkin. Getting Root on your 3G Xoom, May 2011. http://bit.ly/n2qhzh.

[6] Timo Lindfors. How to Capture USB Traffic with Usbsnoop 1.8, October 2010. http://lindi.iki.fi/lindi/darcs/usbsnoop2libusb/.

[7] Chris Paget. USB Host mode on Motorola Droid, February 2010. http://bit.ly/9lFGiU.

[8] Benoit Papillault. USB Sniffer for Windows 98, 98SE, 2000 and Windows XP, January 2003. http://benoit.papillault.free.fr/usbsnoop/.

[9] Zayed Rehman. How To Root Motorola XOOM Honeycomb Tablet On, May 2011. http://bit.ly/odLydr.

Vita

Vaidehee Padgaonkar Gokhale attended the University of Michigan, Ann Arbor, where she obtained a Bachelor of Science in Electrical Engineering and graduated Magna Cum Laude in April 2005. Since then, she joined Intel and did post-Si debug work for the Core i7 and Atom products. She is currently doing circuit design work for the next generation Atom product.

Permanent address: vaideheep@gmail.com
7805 Tusman Dr.
Austin, Texas 78735

This report was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.