CHALMERS

Smartphone Connection to Husqvarna Products

Master of Science Thesis in Embedded Electronics System Design

SIGURSTEINN HAUKUR REYNISSON

Chalmers University of Technology
Department of Computer Science and Engineering
Gothenburg, Sweden, August 2013

Smartphone Connection to Husqvarna Products

SIGURSTEINN HAUKUR REYNISSON

Examiner: LARS SVENSSON

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

Department of Computer Science and Engineering
Gothenburg, Sweden, August 2013

Abstract

Embedded systems are increasingly finding their way into various equipment and machines. Many embedded systems need to be monitored, adjusted and updated at some point during their lifetime.

Smartphones are already dominating the mobile market in some countries and are expected to dominate the mobile market worldwide in a few years. The evolution of smartphones has been rapid and they already possess several different wireless communication technologies. Wireless communication between embedded systems and smartphones could enable monitoring, adjustment and servicing of embedded systems in a simple and cost effective manner.

The main task of this project was to create a smartphone application for work tracking and servicing of a battery powered ride-on mower from Husqvarna. The system is to be used as a demonstrator to illustrate some of the possibilities that smartphone connectivity can provide. This project can also be used as a platform for further development of machine to machine solutions for Husqvarna products.

A study of suitable wireless technologies was conducted and different development platforms were evaluated. The proposed system uses Bluetooth technology and is built for Android devices. It allows monitoring of all the major system variables in a graphical manner. The system includes fault diagnostics features for the machine as well as a test interface that can be used to test different actuators and indicators. Several other service related features were also implemented.

Safety and security aspects of the proposed solution have to be studied further but the system has sufficient quality for in-house use and demonstration.

Keywords: Smartphone, Embedded System, Wireless Connectivity, Remote Servicing, Android, Bluetooth, WiFi, Machine to Machine

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank everyone working in the Electronics Technology department at Husqvarna AB, especially Anders Mattsson and Mikael Larsson Alexiusson for their help and patience. I would also like to thank my examiner, Lars Svensson, for his friendly attitude and support during my studies at Chalmers.

I want to thank my fiancée, Sigrún Inga, for her immense support, patience and help over the years. And finally I want to thank my son, Elías Ari, for providing me with high quality study breaks at regular intervals.

Gothenburg, August 2013
Sigursteinn Haukur Reynisson

List of Acronyms

8DPSK Differential Encoded 8-ary Phase Shift Keying

A2DP Advanced Audio Distribution Profile

AMP Alternate MAC/PHY

AP Access Point

API Application Programming Interface

AVRCP Audio/Video Remote Control Profile

BLE Bluetooth Low Energy

BR Basic Rate

DSSS Direct-Sequence Spread Spectrum

EDR Enhanced Data Rate

EEPROM Electrically Erasable Programmable Read-Only Memory

FH-CDMA Frequency Hopping Code-division Multiple Access

GFSK Gaussian Frequency Shift Keying

GPS Global Positioning System

GUI Graphical User Interface

HDP Health Device Profile

HID Human Interface Device Profile

HFP Hands-Free Profile

HS High Speed

HSP Headset Profile

ISM Industrial, Scientific and Medical

ISP In System Programming

MAC Media Access Control

MAP Message Access Profile

MFi Made for iPhone/iPod/iPad

NDA Non-Disclosure Agreement

NFC Near Field Communication

OBD On Board Diagnostics

OFDM Orthogonal Frequency-Division Multiplexing

PHY Physical

PAN Personal Area Network Profile

PCB Printed Circuit Board

PBAP Phone Book Access Profile

RCU Rider Control Unit

RPM Revolutions Per Minute

RTOS Real-time Operating System

SDK Software Development Kit

SIG Special Interest Group

UART Universal Asynchronous Receiver/Transmitter

U-NII Unlicensed National Information Infrastructure

XML Extensible Markup Language

$\pi/4$**-DQPSK** $\pi/4$ rotated Differential Encoded Quaternary Phase Shift Keying

1 Introduction

It is appropriate to begin this report by putting the master thesis work into perspective. The project background as well as the main purpose and goals are outlined in this section. A short overview of similar solutions that are currently available is presented as well. The selected design method is introduced and the section ends with an overview of the report structure.

1.1 Project background

Smartphones are expected to dominate the worldwide mobile phone market in the coming years. Over 50% of mobile users in North America and Western Europe are currently using smartphones [1].

With ever increasing amount of embedded electronics in various tools and machines comes an increased demand for easy monitoring, servicing and fault diagnosis. Enabling communication between machines, with embedded systems inside, and smartphones seems to be a promising approach to improve the user experience. It may also reduce servicing cost and improve the overall quality of the product. Husqvarna AB has recognized this need and wants to demonstrate this in its products for forestry and gardening.

The most basic embedded systems may rely on mechanical switches and potentiometers to control the system behavior. Analog meters and light indicators are sometimes used for monitoring and possibly diagnosing and troubleshooting the system. More sophisticated systems may need displays and keyboards of various sizes to allow proper operation and maintenance.

In some cases it is not feasible to add displays and keyboards to a system. There may be several reasons for this. Limiting factors could for example be the small physical size of the machine or the additional cost that comes with increased complexity. Furthermore, the machine may be operating under harsh environmental conditions that a cost effective solution would not be able to withstand.

One solution to this challenge is to provide the embedded system with some kind of a communication interface. This interface may be wired (e.g. USB or RS232 cables) or wireless (e.g. Bluetooth, WiFi or ANT). The wired solutions have the obvious drawback of physically tying the user and the machine together, which limits the movement of the user. Personal computers, laptops or specialized equipment can then be used to communicate with the embedded system over this interface.

Personal computers and laptops are bulky and unlikely to be found in the pocket or backpack of a regular worker in the area of forestry and gardening. Specialized service equipment on the other hand is usually not distributed or made available to the end-user.

Today's smartphones possess many of the qualities that are needed to build an easily available, low cost solution for servicing and monitoring of machines that contain embedded systems. Furthermore it is possible to use the capabilities of the smartphones to add new features to the product. It is for example possible to use the Global Positioning System (GPS) and/or the gyroscope in the smartphone to monitor the movement and current location of the machine. This could lead to improved efficiency and safety as well as creating opportunities for fleet management of a wide variety of machines.

One of the major challenges of designing a smartphone based solution of this kind is the huge diversity of mobile devices on the market. Selection of a platform (iOS, Android, Windows Phone etc.) and compatibility between different hardware is a central question in a project of this kind. It is also important to try to realize what the future trends on the mobile market might be.

> *Prediction is very difficult, especially if it's about the future.*
>
> (Niels Bohr)

1.2 Purpose and goals

The main objective of this master thesis is to develop and build a demonstrator that showcases the possibilities that a connection between a machine, with an embedded system inside, and a smartphone can provide. A battery powered ride-on mower from Husqvarna (referred to as the Battery Rider in this report) was selected as the platform for this project.

The main goal is to provide the Battery Rider with work tracking features and a servicing interface using a smartphone. The final product should include all the necessary features listed under the requirements section (Section 3) and have acceptable quality for in-house use and demonstration.

1.3 Related designs

Bluetooth enabled robotic mowers are already on the market, one example is the LawnBott LB3510. This mower is said to be Bluetooth ready, but the remote control has not been made available. An application is available for a small set of mobile phones [2]. Almost all of the supported phones are Java based feature phones. They can therefore not be considered to be modern smartphones.

A good place to look for similar solutions would be the car industry. Bluetooth has been used for years in the car industry to connect phones to the built-in sound system and microphones (headset features) in cars. Volvo provides some of their cars with smartphone connectivity option called "Volvo on Call". This mobile application enables the user to view basic statistics from the car, start the car heater and check fuel status to name few examples.

When it comes to servicing and monitoring cars using smartphones there are several options available. The reason for this is that the diagnostics connector and protocol have been standardized. This creates a large market for a solution of this kind. These solutions all rely on a Bluetooth enabled module that is inserted into the On Board Diagnostics (OBD) connector in the car. This opens a door to detailed information about the state of the car and can be used for various purposes. Fleet management solutions that are based on this information are already available.

Finally, it is important to emphasize that the task of designing a smartphone connected product for the forest and gardening industry can obviously be translated to almost any other equipment. The challenges will be more or less the same although the requirements may change.

1.4 Development method

This project followed the V-model design methodology. The design process started with a system specification and a high-level design of the system. After that, the low-level design and the actual implementation was started. When the main implementation and debugging was finished, unit and component testing was performed. System integration test was performed towards the end of the project.

1.5 Report structure

The structure of the report is as follows:

- Section 2 presents the Husqvarna Battery Rider that was used to demonstrate the proposed solution.
- The requirements for the system are presented in Section 3.
- A study of suitable communication technologies is found in Section 4.
- Section 5 outlines the Android and iOS operating systems with respect to their support for different wireless technologies.

- Modification of the Battery Rider is described in Section 6.
- Description of the Android software design is presented in Section 7.
- Section 8 contains discussions and Section 9 contains the conclusions.
- The references and appendices are found at the end of this report.

2 Husqvarna Battery Rider

One important factor in this project was the selection of a machine that could be used as a test platform for the design. The designers at Husqvarna AB had already realized that the Battery Rider (seen in Figure 1) could be a good candidate for a project of this kind.

Figure 1: Husqvarna Battery Rider.

Some of the important features of the Battery Rider are listed below.

- The machine contains a fair amount of sensors and actuators that can be observed or controlled.
  - Sixteen digital inputs (e.g. buttons and control signals).
  - Fifteen digital outputs (e.g. lights and relays).
  - Five analog inputs from sensors (e.g. throttle).
- The machine is already equipped with a wired communication interface.
  - USB connectivity using a virtual COM port operating at 115.2 kbps.
  - Decently documented protocol.
  - Built in logging features that can be modified.
- The microcontroller in the Rider Control Unit (RCU) is equipped with an unused Universal Asynchronous Receiver/Transmitter (UART) port that has its pins drawn out to an unused connector on the Printed Circuit Board (PCB).
- Test bench for the RCU is already available.

In addition to the above it is clear that a machine of this kind could benefit from a connection to a smartphone (e.g. location and speed information).

3 Requirements

A specification for the system was presented in a special specification report early in this project. This specification report was the result of a discussion with the project supervisor at Husqvarna. The requirements from the specification report are listed in this section with minor rearrangements and corrections.

3.1 Hardware requirements

The only hardware requirement that this project has is the requirement of the communication module. Its requirements depend heavily on the data that will be transmitted from the Battery Rider to the smartphone and vice versa.

The communication link will mainly be used under two different scenarios. The first scenario is when the link is used to provide constant flow of information to/from the user. This will be referred to as live data. This could for example be when the user wants to constantly monitor the status of the machine or use applications that rely on continuous data transmission. The other scenario is when the link is used to transmit larger amount of data that has been stored, or should be stored, on the machine. This will be referred to as a file transfer. File transfer could for example be used to access historical data about the temperature of the machine, error logs of the embedded system or to upload new firmware to the machine.

3.1.1 Data rates

Up-link: live data
The requirement for a link of this type depends on the amount of relevant information that is collected on the machine and the rate of changes in the system. No more than 128 bits of data are needed to represent all sensor values in the system that is currently found in the Battery Rider. Although the current system does not allow a simple transmission of this data at once it is considered to be a good measure for the maximum amount of data that would be transmitted at a given instance. Assuming that some other variables in the system might be of interest it is reasonable to add another 256 bits (eight 32-bit variables) to this data, giving a total of 384 bits.

The other major factor that needs to be considered for this type of data transmission is how often this data has to be transmitted. The minimum rate of updates is different for different measurements. Values like Revolutions Per Minute (RPM) need frequent updates while temperature measurements could safely be sent with long intervals.

One way of looking at this is that we want the user to perceive the observations as they were continuous and the movement of indicators should be smooth. To achieve this, a rapidly changing value (e.g. RPM) has to be refreshed around 24 times every second. This refresh rate (24 Hz) is widely accepted as sufficient to display smooth animations. Lower rates are likely to be sufficient since the illusion of continuous movement can be achieved with other methods (bridging, interpolation etc.) on the smartphone. Using very high rates is not considered to be reasonable for a system of this kind since its purpose is not to provide high resolution debugging information.

Based on the discussion above, it is possible to estimate the required data rate has to be no less than 9600 bps ($384\,\text{bits} \cdot 25\,\text{s}^{-1}$). It is considered feasible to leave room for improvement and transmission of higher data rates. Quadrupling the data rate gives a data rate of 38400 bps which will be used as the minimum requirement on the up-link for live data.

Up-link: file transfer
As for the link for live data, the requirement on this link relies on the amount of information that needs to be transferred. Although the internal storage of embedded systems may be increasing, it is usually relatively small. One way of estimating the requirement on this link is

to assume that the entire contents of the system memory can be transferred within a reasonable time. The current system has less than 1 Mb of memory and we consider ten seconds to be acceptable for a complete transfer. From this it is clear that the data rate has to be at least 100 kbps. Quadrupling the data rate gives data rate of 400 kbps which will be used as the minimum requirement on the up-link for file transfers.

Down-link: live data
This form of communication will be used to transmit commands from the user via the smartphone to the machine. These commands could be fairly simple and easily represented with two 8-bit words. It is obvious that this link is not the limiting one with respect to data rates but it requires relatively fast response time to ensure proper user experience.

Down-link: file transfer
This form of communication could for example be used when the user wants to upload firmware (or any larger files) from the smartphone to the machine. This link is subject to the same requirements as the up-link for file transfers. That is, filling the entire memory of the embedded system should be completed within a reasonable time. This link cannot be considered likely to become a bottleneck since file transfers of this kind will not be performed very often. One could imagine that the intervals between file transfers of this kind might be months or even years.

3.1.2 Energy efficiency

Since the link for the live data is continuously active it is important that the communication technology selected is energy efficient. Although the machine may have access to large batteries or power sources, the smartphones usually carry small batteries. This could of course be circumvented by providing the user with a power source (e.g. USB port) but it is not considered to be a preferable solution in this case.

Since file transfers are not going to be continuous, the energy efficiency of that part is not considered to be of high importance.

3.1.3 Response times

Basic limits for response times in human-computer interaction are discussed in [3] and [4]. According to [3], the response time has to be less than 100 ms for any action that the user should experience as having an immediate effect. Other sources state that this value could be as high as 200 ms [4]. For this reason, the preferred response time is limited to 100-200 ms for all commands that are sent to the machine using a smartphone. If this response time is for some reasons unachievable, the absolute maximum response time of 1 s should be achieved.

3.1.4 Range of wireless connection

The minimum range of the wireless connectivity should be at least 5 m and preferably as long as 100 m.

3.1.5 Other design considerations

The communication module is to be added to an already existing system. For this reason, the module has to support a 2-wire UART interface.

Although the system developed during this project will only be compatible with a small subset of all available smartphones, it is feasible to select a communication module that could be used with as many smartphones as possible. In other words, it should only require software development to make this solution compatible with other smartphones.

3.2 Software requirements

The main purpose of the software is to proof the concept of smartphone connected products by implementing several work tracking and servicing features. The requirements for the Android software are listed as Req1 to Req11 in Section 3.2.1. The requirements for the embedded software for the Battery Rider control unit are listed as ReqA and ReqB in Section 3.2.2.

3.2.1 Android application

The specification for the Android application is listed below with minor changes and rearrangements from the original specification report.

Req1: Device pairing

The software should enable the user to find and initiate a connection to a smartphone enabled machine. The initiation of the connection may require user interaction but once it has been established it should be stable enough to allow continuous operation.

Req2: Two-way communication

The software should be capable of using two-way communication; hence it should be able to receive data from the machine as well as sending commands to it. This requires the embedded system to be capable of responding to requests/commands (see ReqA).

Req3: Basic work timer

The software should be capable of measuring the time that the machine has been actively working. This timer should measure the time that the machine is running with the cutter down and rotating. It should pause as soon as the cutter is raised and/or turned off. Options for resetting the timer should also be implemented.

Req4: Measure the area covered

The software should be capable of using GPS data or any other available information to measure the distance traveled by the machine. This information should then be used to calculate the area that the machine has covered in active mode (cutter lowered and rotating).

Req5: Estimate and store the size of a lawn

By using any available information about the distance traveled, the software should be capable of measuring the size of a lawn and storing the data for later use (e.g. estimation of time to completion).

Req6: Estimation of time to completion

By using information about the size of the area to be cut (input from the user or saved value), the software should be able to predict how long it takes to cut what is still remaining.

Req7: Sensor values and complete state

The software should be capable of displaying all sensor values to the user in a meaningful manner. It should also be capable of displaying the state of all levers and pedals to the user. This should include the battery level.

Req8: Fault diagnosis

The software should guide the user through a simple test of all the levers, pedals and buttons of the machine. This feature should be equivalent to the feature found in other service applications from Husqvarna.

Req9: Service guidance

The software should react to information about the work hour counter of the machine and remind the user to perform preventive and periodic maintenance when applicable.

Req10: User manual

The software should enable the user to access the user manual (could also be service manual and part list) of the machine with little or no effort.

Req11: Service finder

The software should use Google Maps together with information about all Husqvarna service providers to locate the nearest service provider. It should display all relevant contact information to the user and even enable the user to get driving directions using other smartphone software (e.g. Google Maps).

3.2.2 Battery Rider embedded software

The specification report did not specifically mention any software requirements for the embedded system on board the Battery Rider. The following requirements could never the less be deduced from the other requirements listed in the specification report and are listed here for completeness.

ReqA: Two-way communication

The embedded system must be capable of responding to serial request/commands.

ReqB: Timer

The embedded system must be able to keep track of the active work time of the machine.

4 Wireless connectivity

When it comes to designing wireless solutions there are numerous wireless communication standards to choose from. The requirements for this project quickly narrowed this selection down to Bluetooth and WiFi. The reason is that they are the only ones that are widely supported on today's smartphones (see Section 5). Other technologies such as ANT+ and ZigBee are emerging but are not widely available in smartphones at the time this is written.

The Near Field Communication (NFC) technology is an interesting option that could be used for device pairing in conjunction with Bluetooth or WiFi but is not suitable as the main communication method. The focus will be on WiFi Direct instead of the regular WiFi standard. This is because the main objective is to connect smartphones and machines directly to each other without any external equipment such as an Access Point (AP).

This section outlines the Bluetooth and WiFi Direct technologies. Historical and technological background of the two technologies is presented. The energy efficiency of the two technologies is compared and finally, security issues and future trends are considered.

4.1 Background information

This section should provide a brief overview and a historical perspective of the communication technologies that are considered for this project.

4.1.1 A brief history of Bluetooth

Bluetooth was originally created by Ericsson in 1994 with the main purpose of replacing RS-232 cables. It is designed to provide robust, low power, low cost connections that work over relatively short distances [5]. Bluetooth uses the Industrial, Scientific and Medical (ISM) radio band in the range of 2400-2485 MHz [6].

The core specification of Bluetooth is maintained by the Bluetooth Special Interest Group (SIG) that was formed in 1998 [5]. The first version (v1.0) of the Bluetooth specification was released in 1999, this version is often referred to as Bluetooth Basic Rate (BR). Updated version (v2.0) of the core specification was released in 2004, this version is also known as Bluetooth Enhanced Data Rate (EDR). Yet another version of the Bluetooth core specification was released in 2009, this version is called Bluetooth High Speed (HS).

The most recent version of the core specification is Bluetooth 4.0 which was announced in 2010. This specification differs from the previous ones since it introduces a new concept into the Bluetooth specification. This new option has had several different names throughout its short lifetime but Bluetooth Low Energy (BLE) and Bluetooth Smart are the most common names for it.

This new part of the Bluetooth specification defines a new variant of Bluetooth that is intended for low energy applications. Here, low energy application means that it should be possible to power the device on a coin-cell battery for many months or even years [6].

The BLE technology differs significantly from previous versions and is therefore not compatible with older versions. To increase the confusion even more the Bluetooth SIG has introduced the concept of Bluetooth Smart Ready devices. Bluetooth Smart Ready devices implement both the previously known versions of the Bluetooth specification (BR, EDR, HS) as well as the new BLE technology. Bluetooth Smart Ready smartphones have been on the market since late 2011.

The Physical (PHY) and Media Access Control (MAC) parts of Bluetooth technology were standardized in the IEEE 802.15.1 standard in 2002 and then in an updated version in 2005 [7]. IEEE list this standard as active but it should be noted that the Bluetooth SIG has issued several new core specifications since then.

4.1.2 A brief history of WiFi Direct

The WiFi Alliance officially announced WiFi Direct in October 2010 [8]. WiFi Direct is built on the solid foundation of previously known WiFi technologies (IEEE 802.11 a/g/n) and should in most cases be interoperable with older devices although it does not support the IEEE 802.11b standard [9]. It is worth mentioning that WiFi Direct is not directly connected to any IEEE 802.11 standard.

The main purpose of this technology is to allow WiFi devices to communicate directly with each other without having to use an AP. It should enable quick, reliable and secure connections between devices that need to sync or exchange information with each other.

Since WiFi Direct is so closely related to previous WiFi technology it can in some cases be enough to perform a software upgrade on older devices to allow it to operate as a WiFi Direct device. This means that although the technology is relatively new, it is already widely available.

4.2 Technical Overview

Bluetooth and WiFi Direct both operate in the ISM band but that is about all they have in common from a technical point of view. This section outlines the basic techniques used in Bluetooth and WiFi Direct communication.

4.2.1 Bluetooth

As mentioned earlier, Bluetooth operates in frequencies in the range of 2400-2485 MHz. When Bluetooth devices connect they form a so called piconet. Each piconet can only have one master device and up to seven slaves. All the devices in one piconet share the physical channel and therefore its capacity. A master in one piconet can serve as a slave in another piconet. Piconets that share one or more slaves form a scatternet. A piconet can be thought of as a network with star topology. Each slave is directly connected to the corresponding master but not to any of the other slaves.

Bluetooth uses Frequency Hopping Code-division Multiple Access (FH-CDMA) scheme. It uses 79 different carriers with 1 MHz spacing for the channel hopping and the dwell time for each hop is 625 μs (1600 hops/second). The FH-CDMA is suitable for use in the ISM band since on average it uses relatively broad spectrum but only a small piece at a time [10]. This helps it to avoid interference. It is also possible to adjust the hopping sequence so that it avoids unusable frequencies. The hopping sequence is pseudo-random and is determined by the master of each piconet. When a slave connects to a master device it synchronizes its clock to the clock of the master device and selects the appropriate hopping sequence.

The first version of Bluetooth (BR) uses a binary modulation scheme, a Gaussian Frequency Shift Keying (GFSK) to be more specific. Signal bandwidth for a frequency hopping system in the ISM band is limited to 1 MHz which results in a maximum data rate of 1 Mb/s using this modulation scheme [10].

Later versions of Bluetooth modified the modulation scheme to allow higher data rates. Bluetooth EDR uses a mixture of GFSK and $\pi/4$ rotated Differential Encoded Quaternary Phase Shift Keying ($\pi/4$-DQPSK) to achieve data rates of up to 2 Mbps. Furthermore, Bluetooth EDR uses a mixture of Differential Encoded 8-ary Phase Shift Keying (8DPSK) and GFSK to achieve data rates of up to 3 Mbps [11].

Bluetooth HS introduced a feature called Alternate MAC/PHY (AMP) to further improve data rates up to a maximum of 24 Mbps. This method uses the regular Bluetooth channel to negotiate communication between the devices. If both of the devices support HS operation and a higher data rate is needed, the transmission moves to an 802.11 channel.

The low energy version of Bluetooth (BLE) uses 40 carriers with 2 MHz spacing between carriers and it uses GFSK, just as the BR version. The main difference between BLE and the other version is the protocol stack which is tailored to low-power operation.

Bluetooth defines various profiles to ensure interoperability between Bluetooth devices from different manufacturers. This is a major benefit since it ensured that devices that implement the same profile are able to communicate with each other. WiFi Direct currently lacks this feature

4.2.2 WiFi Direct

WiFi Direct (and WiFi in general) operates in the ISM band from 2400 MHz 2500 MHz and in some cases the 5 GHz Unlicensed National Information Infrastructure (U-NII) band. The modulation used is either Direct-Sequence Spread Spectrum (DSSS) or Orthogonal Frequency-Division Multiplexing (OFDM) and the channel bandwidth is either 20 MHz or 40 MHz. Data rates are somewhere in the range of 11 Mbps to 450 Mbps for current standards (IEEE 802.11 a/g/n) [12].

4.3 Energy efficiency

WiFi Direct and BLE are both relatively new technologies and they have been evolving rapidly since they were announced in 2010. For this reason, accurate and non-biased information about their efficiencies are not easily accessible.

To obtain some insight into the energy efficiency aspects of these two technologies it is possible to start by looking at the older versions and then look at the newer versions and their benefits compared to the previous ones.

Attempts to compare the efficiency of different wireless technologies are found in [13] and [14]. Attempts to evaluate the energy consumption and throughput of wireless technologies in smartphones are found in [15].

The articles mentioned above state that the power efficiency of different wireless technologies is more or less determined by the required data rates and latency restrictions. Table 1 (based on information in [14]) shows some typical parameters for WiFi and Bluetooth.

Table 1: Wireless protocols - typical parameters [14].

	Bluetooth BR	WiFi (ad-hoc mode)
Max. Packet Size (bytes)	339	2312
Max. Data Rate (Mbps)	.72	54
Energy per 1Kb (mJ)	0.034	0.013

This information allows us to draw some basic conclusions. Bluetooth seems to be more suitable for applications that frequently send small amount of data. This is because of its small packet size. The packet size of WiFi is much larger which makes it inefficient for sending small chunks of data. The exception to this would be when it is possible to collect data until a larger packet has been created, this would obviously increase the latency. If we consider the amount of energy spent on transmitting each bit it is clear that WiFi is more efficient for higher data rates.

Table 2: Typical low power operation characteristics [14].

	Bluetooth	WiFi
TX Current	57 mA	219 mA
RX Current	47 mA	215 mA
Sleep Current	15 μA	10 mA
V_{dd}	3.3 V	3.3 V

Table 2 shows some basic low power characteristics for Bluetooth and WiFi while Table 3 shows typical power-up values (both from [14]). The values shown in these tables will obviously

vary between different Bluetooth and WiFi modules, but this information gives some idea of the energy efficiency of the two technologies. Information of this kind also allows a designer to estimate when it is efficient to turn a wireless interface off. The long startup time and high sleep current for the WiFi module is noteworthy. These values show that the Bluetooth technology is suitable for low data rate, low latency and low energy applications such as this one.

Table 3: Typical power-up characteristics [14].

	Bluetooth	WiFi
Startup Current	5.5 mA	37.9 μA
Startup V_{dd}	2 V	3.3 V
Startup time	120 μs	2 s

One of the major drawbacks of regular WiFi is that it does not make use of power saving techniques when operating in ad-hoc mode [15]. Enabling WiFi Direct to use the power saving features that a regular client-AP connection is likely to improve the energy efficiency [16].

When it comes to BLE the Bluetooth SIG states that BLE should be able to provide power consumption that is somewhere in the range of 1/2 - 1/100 of the classical versions [17].

4.4 Initiating a connection and security issues

There is not a significant difference in the complexity of pairing devices with Bluetooth (BR/EDR/HS/BLE) and WiFi Direct. Both technologies offer different methods for establishing a connection. Since the Battery Rider is not equipped with a display or a keyboard we are limited to very simple pairing methods. That is, methods that don't require password entries or confirmation from the machine. This obviously creates several security issues. A general rule of thumb is that the simpler the process of initiating a connection, the less secure the connection will be. An interesting option here is to use NFC for pairing. This would enable out-of band pairing which is much more secure than the simple pairing methods.

5 Developing for smartphones

This section summarizes some of the important aspects of application development for smartphones. An overview of the mobile market and a short analysis of the connectivity options available in the Android and iOS operating systems are presented.

5.1 Overview of the smartphone market

Reports from mobile market research companies ([18], [19]) show that the smartphone market is dominated by the Android operating system. Android has almost 80 % market share when it comes to shipped units (worldwide) under the second quarter of 2013. The iOS operating system has around 13 % market share [18]. This means that around 93 % of smartphones shipped under this period ran either Android or iOS. This leaves little room for the competing operating systems like Windows, BlackBerry OS and Symbian.

These numbers do not tell the whole story since the market share varies significantly between different markets. Reports show that around 50 % of mobile subscribers in the US are using Android smartphones while around 40 % of them have iOS based devices [20].

5.2 Smartphone connectivity options by operating systems

Following is a short summary of the Bluetooth and WiFi connectivity options that exist in Android and iOS.

5.2.1 iOS

Bluetooth connectivity
When it comes to developing Bluetooth connected accessories for iPhone, or any other iOS device, things tend to get a little complicated. Apple states that anyone that wants to develop accessories for iPhone/iPad/iPod using standard Bluetooth (BR/EDR/HS) profiles can do so freely [21]. To be able to design products that rely on Apple's licensed components or software, one has to join Apple's Made for iPhone/iPod/iPad (MFi) program which is a fairly complicated process. It includes a strict Non-Disclosure Agreement (NDA) and a credit review of the applying company has to be completed.

Table 4 shows the Bluetooth profiles supported by devices running Apple iOS 6 [22]. According to Apple it should be possible to use these profiles freely to develop applications or accessories but there is one major limitation to this. This is only true (for most of the profiles) when developing for iOS and MFi compliant devices. All Bluetooth connections have to go through Apple's publicly available frameworks. To my best knowledge there are only two of them. One is the External Accessory framework and the other one is the GameKit framework both are limited to iOS and MFi compliant devices [23].

It is tempting to try to find ways to get around these limitations. One can for example make the connected device mimic a keyboard and use the HID profile to send simulated keyboard strokes to the application or use modulated audio to transfer the data as shown in [24]. One major concern is that any application that takes advantage of this may not be accepted by Apple for distribution in the Apple Store and therefore rendered useless.

An exception to this limitation is that iPhone 5 is Bluetooth Smart Ready. To refresh the memory of the reader, it means that it implements both the classic versions (BR/EDR/HS) and the new Low Energy (BLE) versions of Bluetooth. Apple Inc. has been a little more liberal when it comes to providing development tools for this technology. Apple's Core Bluetooth framework is public and developers are free to use it in connection with any BLE device.

WiFi Direct connectivity
Apple iOS does not support WiFi Direct at the time this is written but has its own variants instead, they are called AirPlay and AirDrop. As with the classical Bluetooth connections, Apple limits this technology to Apple and MFi compliant devices.

Table 4: iOS 6: Supported Bluetooth profiles [22].

	Hands-Free Profile (HFP) 1.6	Phone Book Access Profile (PBAP)	Advanced Audio Distribution Profile (A2DP)	Audio/Video Remote Control Profile (AVRCP) 1.4	Personal Area Network Profile (PAN)	Human Interface Device Profile (HID)	Message Access Profile (MAP)
iPhone 4 and later	✓	✓	✓	✓	✓	✓	✓
iPhone 3GS	✓	✓	✓	✓	✓	✓	✗
iPhone 3G	✓	✓	✓	✓	✓	✗	✗
Original iPhone	✓	✓	✗	✗	✗	✗	✗

5.2.2 Android

Bluetooth connectivity

The Android OS supports the classical Bluetooth versions (BR/EDR/HS) and provides implementations of four different Bluetooth profiles. Android devices running Application Programming Interface (API) 14 or later support the HFP (v1.5), Headset Profile (HSP), A2DP and the Health Device Profile (HDP) profiles. It is also possible to implement any other profile that might be needed using the BluetoothProfile interface that is available in Android's Bluetooth API.

Android added official support for BLE in API 18 which was released in July 2013. Some third party implementations were available before that time. They were designed by different chipmakers like Motorola [25] and Broadcom [26]. This caused compatibility issues and using BLE could not be considered to be a reasonable option for Android before API 18 was announced.

WiFi Direct connectivity

Android devices running API 14 or later support WiFi Direct, given that they have the necessary hardware [27].

6 Modification of the Battery Rider

The analysis presented in Section 4 and Section 5 lead to the conclusion that adding a Bluetooth classic module to the Battery Rider would be suitable for this project. The process of adding the module to the Battery Rider and modification of the embedded software on the RCU is described in this section. Few notes on testing can be found at the end of this section.

6.1 Bluegiga WT12 module

The Bluetooth module that was selected for this design was the Bluegiga WT12 module with iWRAP firmware. This module was widely available, well supported and came with a complete Bluetooth protocol stack. The module also fulfilled the data rate requirements that were set in Section 3.

Some of the main features of the module are [28]:

- Bluetooth v2.1 + EDR qualified.
- Integrated chip antenna.
- 30 meters range (line-of-sight).
- USB version 2.0 compatible.
- UART with bypass mode.
- iWRAP protocol stack.

6.2 Development process

The task of adding the module directly to the RCU of the Battery Rider was considered to be too complex to complete in a single step. Intermediate steps were added to the design process, these steps are shown below.

1. **Connect the WT12 module to a Raspberry Pi computer using UART.**
   A Raspberry Pi computer was used to perform initial tests on the WT12 module. The connections are shown in Figure 2. Wires were soldered directly to the WT12 module and connected to the GPIO pins on the Raspberry Pi, see Figure 3.

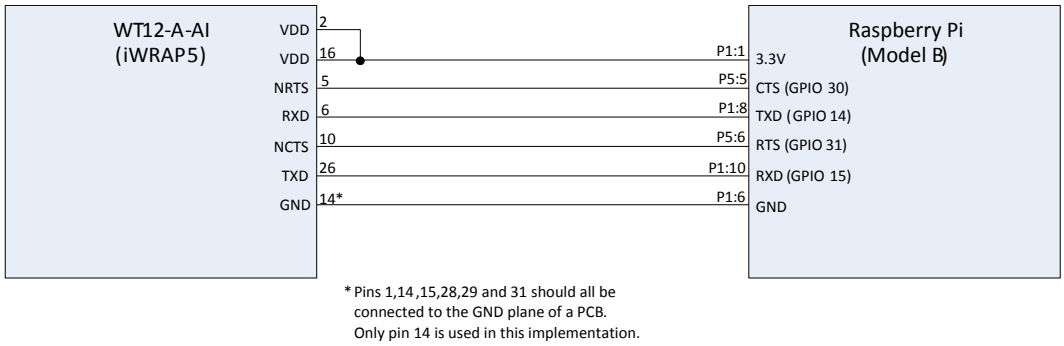


Figure 2: Wiring WT12 to the Raspberry Pi computer.

   The Raspberry Pi computer was running a Linux based operating system called Raspbian. Python scripts were used to send serial commands over the UART interface to the WT12 module.

2. **Establish a Bluetooth connection between the Raspberry Pi and a PC.**
   By using Python scripts on both the Raspberry Pi and a regular PC it was possible to configure and test the pairing process with the WT12 module. After a successful pairing, data could be transmitted back and forth over the Bluetooth interface.

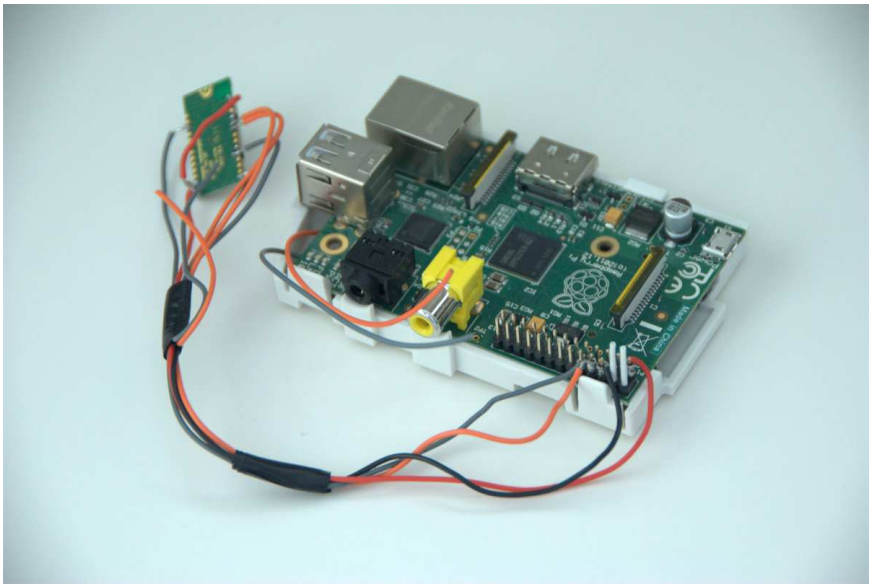


Figure 3: WT12 connected to a Raspberry Pi computer.

3. **Implement the serial protocol of the Battery Rider with a Python script.**
   This enabled testing of the serial protocol and helped with the debugging process of the Android software as well as the embedded software on the RCU.

4. **Connect the module to the RCU of the Battery Rider.**
   Having completed the steps above it was possible to connect the module directly to the RCU. Minor modifications had to be made to the embedded software running on the RCU, see Section 6.3. This enabled Bluetooth communication between Battery Rider and either a PC computer or an Android device. The connections between the WT12 module and the RCU are shown in Figure 4

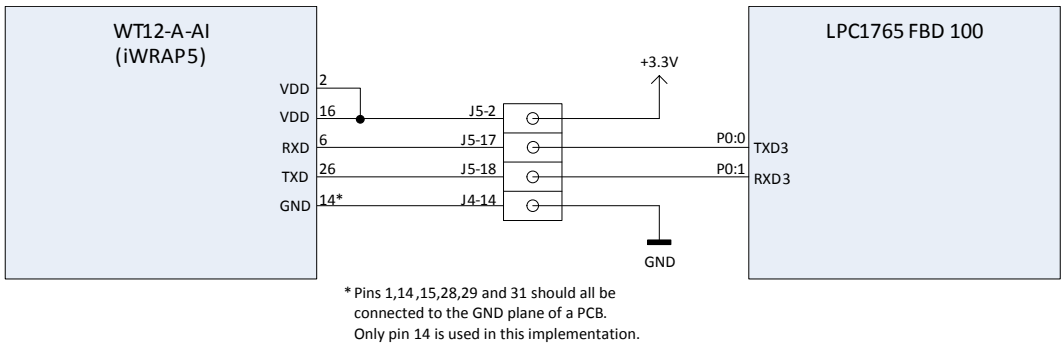


Figure 4: WT12 connection to the RCU.

5. **Firmware upgrade and reconfiguration of WT12 module.**
   The pairing process was complicated and inconsistent with the iWRAP3 firmware that the WT12 module was shipped with. It was therefore necessary to upgrade the firmware to iWRAP5 that has improved pairing options. This upgrade was performed from a Windows 7 computer using software that Bluegiga provides. The serial port of the Raspberry Pi was forwarded over an Ethernet connection to a virtual serial port on a PC running Windows 7. Upgrading of the module can be performed with various other methods but this method was used since all the hardware and connections were already in place.

   The modified RCU is shown in Figure 5.

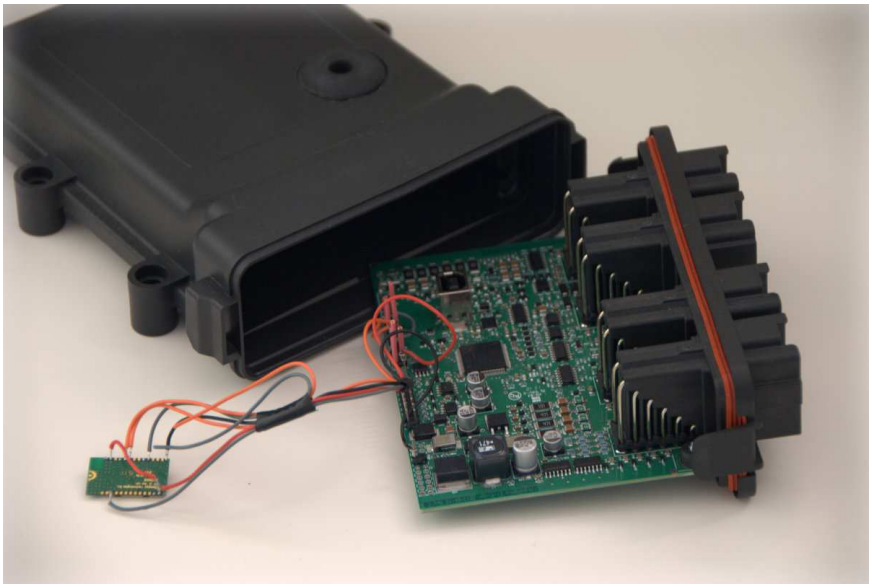


Figure 5: Modified Rider Control Unit.

6.3 Embedded software modification

The modification of the embedded software on the RCU proved to be simple since the system already had a working implementation of a serial interface via USB. Following is a short description of the software changes that led to the fulfillment of ReqA and ReqB listed in Section 3.

6.3.1 Software requirement ReqA

The RCU runs on a CooCox Real-time Operating System (RTOS). The RTOS runs several tasks, one of them is the task of providing serial communication via the USB. The Bluegiga WT12 module was connected to an available UART port of the LPC1765 microcontroller. The appropriate pins were reconfigured and a new serial communication task that used this port was added. This solution allows the USB and the Bluetooth interfaces to coexist.

6.3.2 Software requirement ReqB

Another modification of the embedded software was the addition of a timer. This timer should measure the time that the Battery Rider is active. Built in functions of the RTOS are used to measure the time. The value of the timer is regularly saved to an Electrically Erasable Programmable Read-Only Memory (EEPROM) to preserve the value when the machine is turned off. The value is then read from the EEPROM when the machine is turned on again.

6.4 Range, response time and data rate

Simple tests were performed to check the range of the Bluetooth communication. Line-of-sight range proved to be at least 20 m. Communication was also possible through concrete walls with a range of around 5 m.

Scientific testing of response times and data rates for the modified system was not conducted. Nevertheless, the overall impression was that the response time was sufficiently short to give the user the feeling of immediate change. No data on maximum connection throughput was gathered since the main emphasis was on the transmission of live data rather than transmission of large files,

7 Software design - Android application

There were several reasons for selecting Android as a platform for the smartphone application. The fact that Android is open source plays a major role in this decision. Another factor is that developing applications with the classical Bluetooth versions is well supported.

This section describes the design of the Android application. Overview of the design is given in Section 7.1 and a description of the system architecture is found in Section 7.2. Descriptions of the system components can be found in Section 7.3. Short description on the testing of the software can be found at the end of this section.

7.1 Design overview

The Android application was designed using the development tools that Google provides with the Android Software Development Kit (SDK). Version 21.0.1 of the SDK tools was used throughout the project. The target API is Google API 17 with the minimum API level of 14. Please refer to `http://developer.android.com/` for further information regarding the development tools and the API levels.

Google SDK comes with several sample projects that can be used as templates or reference designs for new projects. This application uses two of these sample projects. The first one is named "HoneycombGallery" and was used as a template for the Graphical User Interface (GUI) of the application. The other sample project that was used is named "BluetoothChat". It was used as a reference design for the Bluetooth communication part of this application. Both of these projects are distributed under the Apache 2.0 license.

It is possible to split the resources behind this application into two main categories, one is the actual Java code and the other is the resources that the Java code relies on. These resources are mostly images and various Extensible Markup Language (XML) files that are used to store layouts for different views, text strings and other constants.

Describing the details of software development for Android devices is not considered to be within the scope of this report. It is assumed that the user is familiar with Java programming and the basic concepts of Android software development.

This application uses fragments. Fragments are recommended to separate and reuse content and code in Android applications. Fragments also simplify the process of making the application run on devices with different screen sizes.

Detailed discussion about the system architecture and the system components is found in Section 7.2 and 7.3 respectively. Each component is a separate Java class.

A requirement matrix for the Android application is shown in Table 5. It lists the most important classes of the application and their relation to the software requirements that were listed in Section 3.2.1. The other classes that the application is based on are mainly for basic functionalities such as menus and other visualizations that cannot be related directly to any of the requirements.

Table 5: Requirements matrix for the Android application.

Class name	**Req1**: Device pairing	**Req2**: Two-way communication	**Req3**: Basic work timer	**Req4**: Measure area covered	**Req5**: Estimate and store size of a lawn	**Req6**: Estimation of time to completion	**Req7**: Sensor values and complete state	**Req8**: Fault diagnosis	**Req9**: Service Guidance	**Req10**: User manual	**Req11**: Service finder
	Software requirements										
AreaMeasure			✓	✓	✓	✓					
BTService	✓	✓	✓	✓	✓	✓	✓	✓	✓		
ContentFragment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Dashboard							✓				
DeviceListActivity	✓										
FaultDiagProcedure								✓			
GPSData				✓	✓	✓					✓
JSONParser											✓
JSONParserDetails											✓
MachineState			✓	✓	✓	✓	✓	✓	✓		
MainActivity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MaintenanceSchedule									✓		
RawData							✓				
ReceiveThread		✓	✓	✓	✓	✓	✓	✓	✓		
SerialMessages		✓	✓	✓	✓	✓	✓	✓	✓		
Tests		✓					✓				

7.2 System architecture

This section focuses on the interaction of classes. Please refer to Section 7.3 for a description of the inner workings of each class.

Time pressure and late design decisions have complicated the architecture of this software to some extent. This does not affect the functionality of the software but it is likely to complicate the code maintenance and/or reuse in the future.

There are two Java classes that play a central role in this software, the MainActivity class and the ContentFragment class. Both of them are involved in nearly all the activities within the program. In an attempt to avoid too cluttered diagrams the system architecture is visualized using the three different diagrams shown in Figures 6, 7 and 8.

7.2.1 Bluetooth functionality

Figure 6 shows how the MainActivity class interacts with other classes to provide the Bluetooth connectivity. The interaction is best described by going through a single life cycle of a Bluetooth connection.

1. The user starts the application, this will start the MainActivity. During startup, MainActivity performs some initialization, ask the user to enable the Bluetooth interface if it is disabled and starts a service called BTService. BTService will later on handle the actual establishment and management of the Bluetooth connection.

2. When the user begins the process of initiating a Bluetooth connection, the DeviceListActivity is started. The user will search for a Battery Rider to connect to and select the appropriate one from a list. The DeviceListActivity is then closed and the address of the selected device returned to MainActivity.

3. MainActivity receives a value from DeviceListActivity, passes it to BTService and requests a connection.

4. BTService starts a ConnectThread which in turn tries to initiate a connection. If a connection is established, BTService will start a ConnectedThread. It then notifies MainActivity that a connection has been established and stops the ConnectThread. The ConnectedThread will run as long as the Bluetooth connection is active. It will continuously forward all incoming messages to MainActivity and forward all messages that it receives from MainActivity to the Battery Rider.

5. When MainActivity is notified about a connection being established, it starts a ReceiveThread that parses the incoming messages. ReceiveThread updates the information in MachineState with the latest information received over the Bluetooth connection. ReceiveThread will also check what view is currently displayed to the user (in ContentFragment) and refreshes it to make sure that the screen is updated with the new information.

6. All commands that are sent to the Battery Rider are stored in the SerialMessages class. When the user presses a command button, the appropriate message from SerialMessages is sent via MainActivity to BTService and then to the Battery Rider.

7. When the Bluetooth connection is closed, ReceiveThread and ConnectedThread are both stopped. After that, the software is ready to begin the process again.

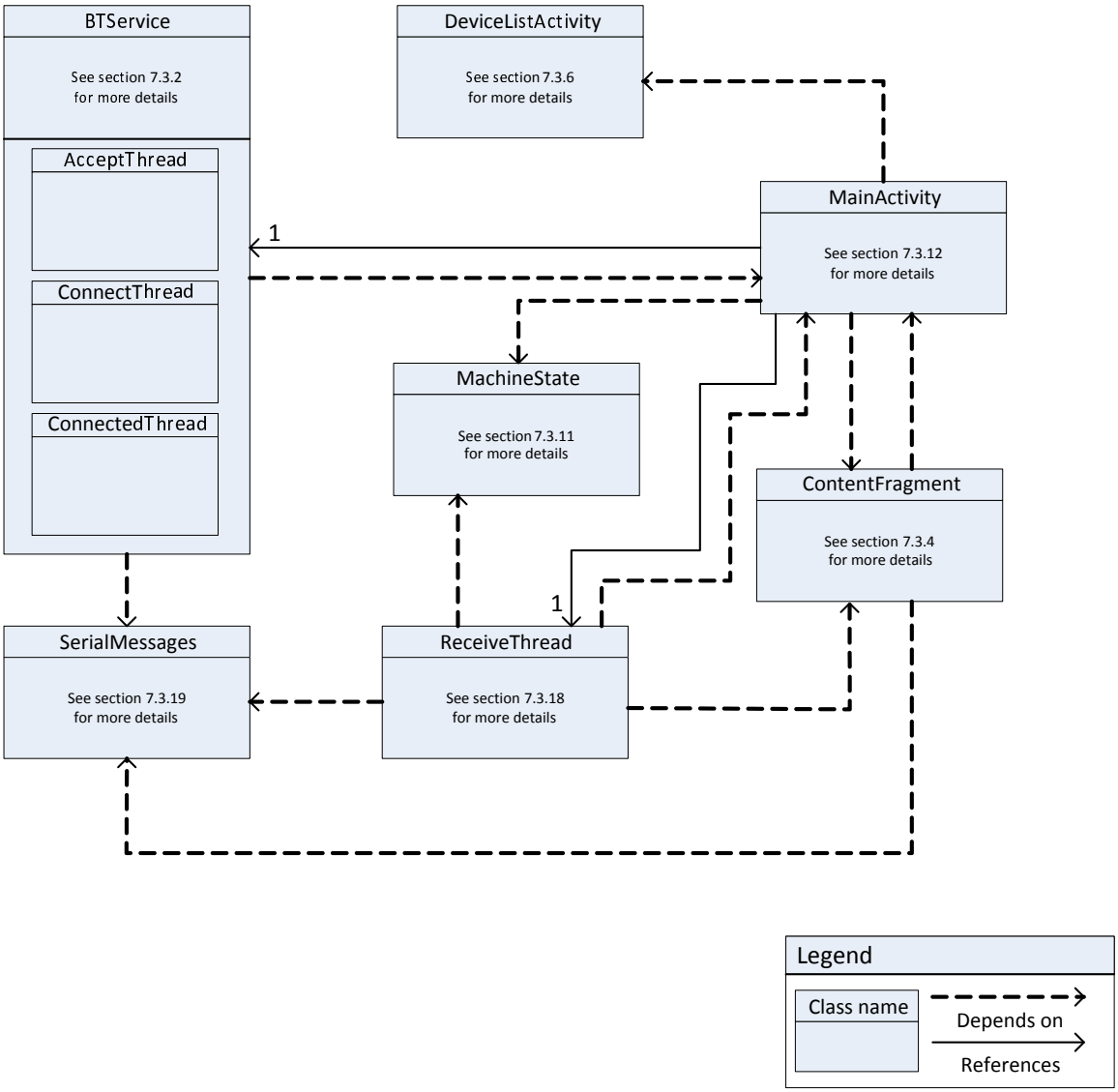


Figure 6: Architecture of the components related to Bluetooth connectivity.

7.2.2 Basic GUI functionality

Figure 7 shows the interaction of MainActivity and ContentFragment with other classes to provide the basic GUI functionality. As before, MainActivity plays a central role. It is responsible for managing the background features of the GUI, such as detecting screen sizes and selecting proper layouts. If the screen size is large enough (tablet) it will display both the MenuFragment and the ContentFragment, otherwise it will only display one of them at a time. The program menus and the application preferences are also managed by the MainActivity class.

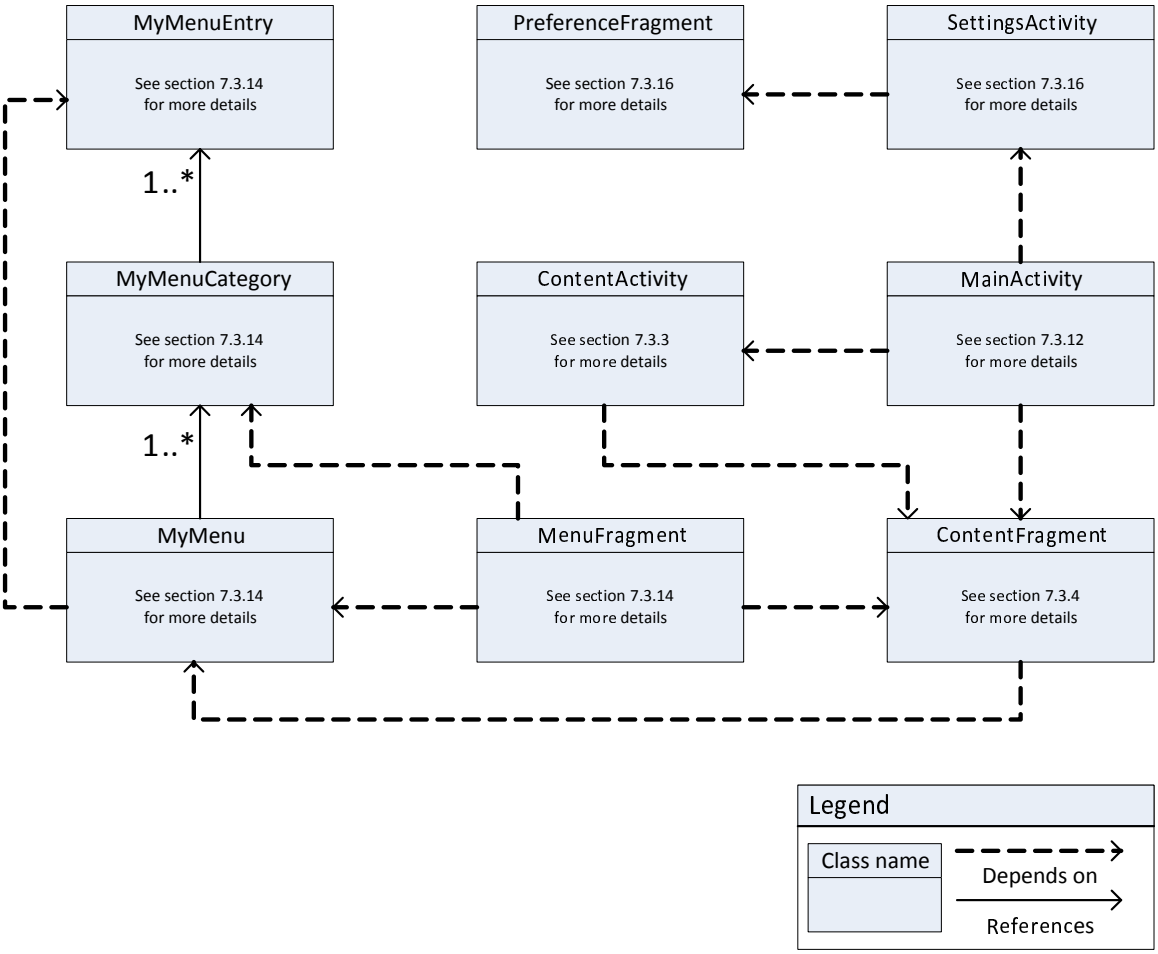


Figure 7: Architecture of the basic GUI components.

7.2.3 Work tracking and servicing functionalities

Figure 8 shows the interaction of MainActivity and ContentFragment with other classes to provide the work tracking and servicing functionalities.

As soon as the application has finished the Bluetooth initialization mentioned earlier it will try to initialize the GPS interface. If the GPS is not enabled on the device the user is offered to open the settings where it is possible to enable the GPS. Some features of the application will be limited if the device is not equipped with a GPS or the user decides to leave the GPS disabled.

Tasks that check the maintenance schedule and service times are started as soon as the application starts, these tasks are a part of the MaintenanceSchedule class. The checks will run in the background for as long as the application is running. They will notify the user about any maintenance that is due. After the initialization completes, the program is ready for use and the device will try to acquire a GPS location.

ContentFragment is the central point for this part of the application. The major role of ContentFragment is to respond to the user's menu selection and display the corresponding view to the user. All of the views are connected to XML layouts except the "User Manual" view which is completely handled by the ContentFragment. A selection of a certain program feature will make ContentFragment expand one XML layout file. Each XML layout file is connected to one Java class, this Java class is then responsible for providing the specific program features.

The MachineState class is important to many of the other classes since it contains the current state of the machine. One exception is the RawData class which displays raw sensor data from the Battery Rider. This view is updated directly from the ReceiveThread for the

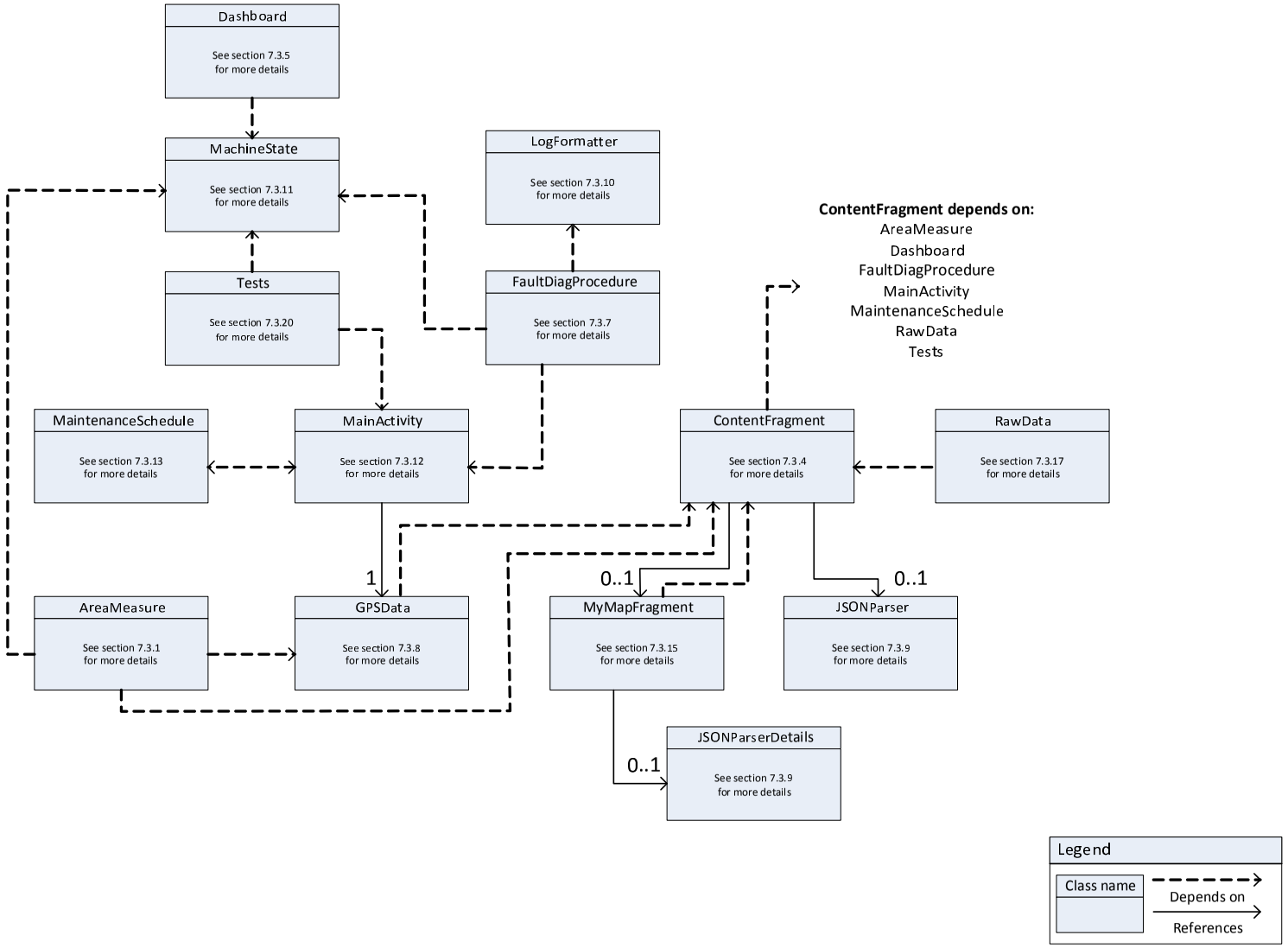


Figure 8: Architecture of the work tracking and servicing components.

sake of simplicity.

The Tests class uses MainActivity to send messages to the Battery Rider. This is also true for the FaulDiagProcedure class which also uses the LogFormatter class to format the log messages during the test procedure.

Classes named JSONParser and JSONParserDetails are both used to retrieve and parse data about approved Husqvarna service providers from the company website. The information that they retrieve is used by the MyMapFragment class and the ContentFragment class to provide the service map features of the application.

7.3 Component description

The components of the application are described in this section. Each component corresponds to a single Java class.

7.3.1 AreaMeasure

This class is responsible for all the work tracking features that the application has. The class depends on information about the state of the machine and the GPS position.

The GUI that this class presents to the user is shown in Figure 9. The information is presented in a scrolling view that is partially hidden in this figure. The users is able to observe the time, distance and area that has been cut since the application started. It is possible to save current area measurement for future reference. If the user has saved work areas, he/she can select that area as the current work area and get estimations of the remaining area and time to completion. The user can also delete previously saved areas from this view.

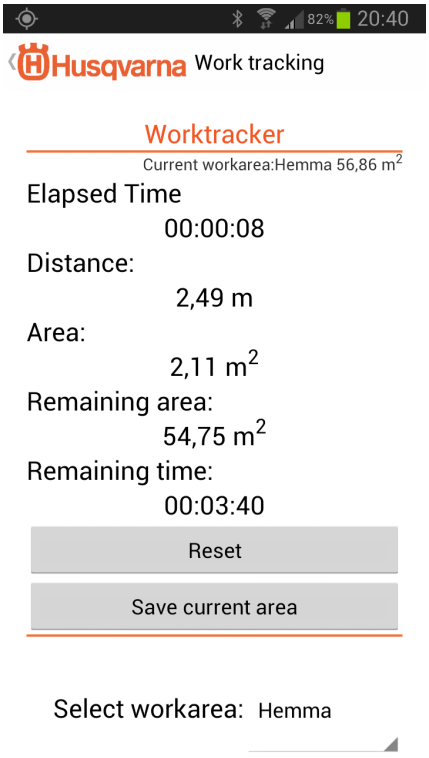


Figure 9: Work tracking GUI.

The GPSData service uses a certain function (add()) to update the total distance value every time the GPS position is updated.

The estimation of remaining time is done by calculating the average speed of the machine (current distance divided by current time). The remaining time is then calculated using that average value.

The class specifies an asynchronous task called BackgroundCheck. This task is started by the MainActivity when the work tracking feature is first started. The purpose of this task is to perform necessary status checks on the machine and make sure that all values related to work tracking are updated, even when the user has changed to another view.

The current implementation requires that the forward pedal is being pressed down to a certain threshold level and that the cutting unit is rotating. This is done to increase the accuracy of the area measurements since the user only wants to measure the actual area that is being cut. The application preferences allow some modification to these requirements but are there for development purposes.

Please refer to Figure A.1 in the appendix (pg. 42) for a class diagram of this class. The purpose of each function should be clear from its name.

7.3.2 BTService

The functionality of this class was discussed in detail in Section 7.2.1. This class contains definition of a thread that has not been mentioned earlier. That is the AcceptThread which is run when an incoming connection is expected. This thread is not used in the current implementation since it is always the Android device that initiates the connection.

Please refer to Figure A.2 in the appendix (pg. 43) for a class diagram of this class.

7.3.3 ContentActivity

This activity is merely a wrapper for the ContentFragment when the application is running on a device with a small screen, such as a smartphone.

7.3.4 ContentFragment

The functionality of this class was discussed in detail in Section 7.2. This is the fragment that shows the content selected in the MenuFragment. Its main purpose is to remove and add content to the main view of the application. It contains one view for each function of the program with two exceptions. The "About" screen is inflated directly from an XML layout file and the service map is contained in a special fragment (MyMapFragment).

Please refer to Figure A.3 in the appendix (pg. 44) for a class diagram of this class.

7.3.5 Dashboard

This class provides a graphical representation of the current state of the Battery Rider. An example of this is shown in Figure 10.

All of the elements in this view are drawn at run time using the Android drawing API. The view can be drawn in two different ways. If the view is shown in portrait mode, the view is made narrower and longer to increase readability. Landscape mode draws a wider view to minimize scrolling.

Please refer to Figure A.4 in the appendix (pg. 45) for a class diagram of this class.

7.3.6 DeviceListActivity

The purpose of this activity is to search for nearby Bluetooth devices and display a list for the user to select from. An example of this is shown in Figure 11. When the user selects a device from the list the address of that device is returned to MainActivity which in turn will request BTService to connect to it.

Figure 10: Dashboard for the Battery Rider.

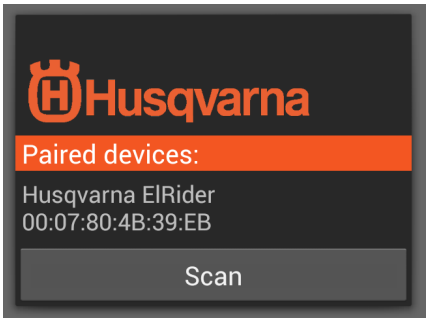


Figure 11: Bluetooth devices search results.

7.3.7 FaultDiagProcedure

The fault diagnostics part of the application is managed by this class. The user is guided through several test of the Battery Rider. Instructions regarding the test procedure are given at the beginning of each step, this may include information about the location of individual parts.

The testing of different parts of the system is performed in the following order:

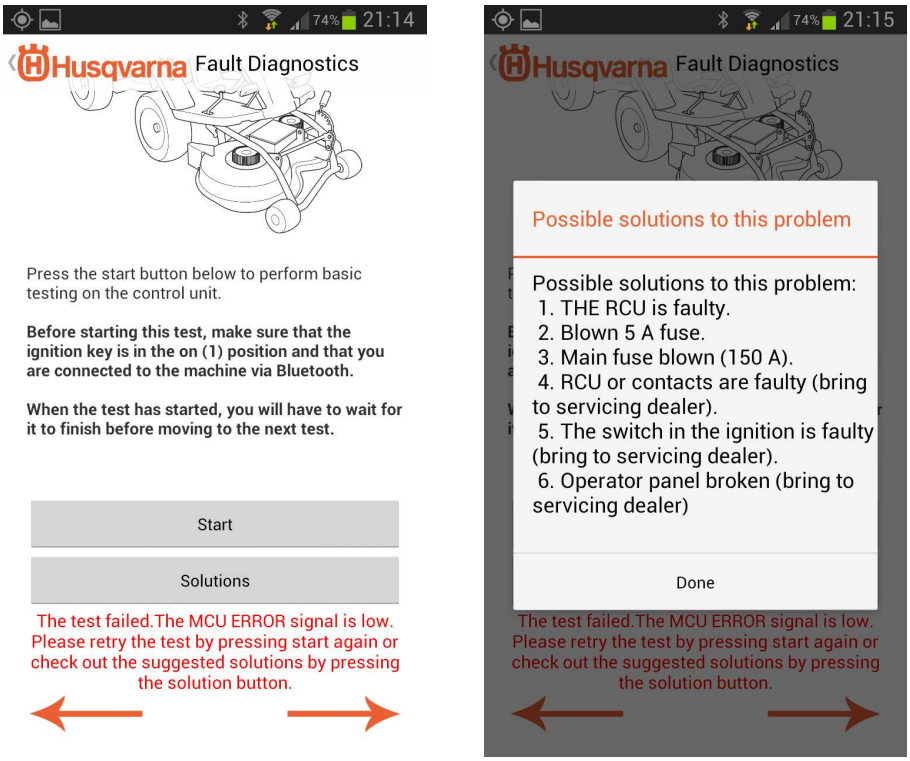
1. Test RCU Bluetooth connectivity.
2. Test RCU external environment such as voltages and control signals from other parts of the system.
3. Forward pedal test. Tests throttle potentiometer together with the forward pedal switch.
4. Reverse pedal test. Tests throttle potentiometer together with the reverse pedal switch.
5. Brake pedal switch test.
6. Lift lever switch test.
7. Deck lock switch test.

8. Seat switch test.

9. Start button test.

10. ECO button test.

All of these tests require an active Bluetooth connection to the Battery Rider. Before starting each test the status of the Bluetooth connection is checked and the user notified if the connection is not active.



(a) (b)

Figure 12: Examples of the fault diagnostics GUI.

When a test is started, an asynchronous timeout task is started. This task performs testing in the background and counts down a timer. Each test is given ten seconds to complete, otherwise it will fail and display the results to the user. An example of this is shown in Figure 12(a).

When a test fails, the program presents the user with possible solutions, see Figure 12(b). A summary of the test results is given at the end of the test procedure. During the tests a detailed log is kept, collecting the user actions and the state changes of the Battery Rider. This log can be sent via email at the end of the procedure for further analysis.

Please refer to Figure A.5 in the appendix (pg. 46) for a class diagram of this class.

7.3.8 GPSData

This class handles all the GPS related functions. It serves both the work tracking features found in the AreaMeasure class as well as the service map feature in MyMapFragment. It includes a function that calculates the distance between two points on a sphere using the Haversine formula.

Please refer to Figure A.6 in the appendix (pg. 47) for a class diagram of this class.

7.3.9 JSONParser and JSONParserDetails

The purpose of these classes is to download and parse data from Husqvarna's website. Information about the names and locations of the dealers is gathered with JSONParser. A more detailed set of information is gathered with JSONParserDetails. The detailed information includes phone numbers as well as web and email addresses. This information is used by the service map fragment. It should be noted that these classes are very dependent on the service map on the official website of Husqvarna. This solution is therefore very vulnerable to any changes that might be made on the website.

7.3.10 LogFormatter

This class is used to format the log entries that are gathered during the fault diagnostic procedure.

7.3.11 MachineState

This class is used to store the current state of the Battery Rider when a Bluetooth connection is active. ReceiveThread is the only class that updates the machine state, other classes only read the state.

Please refer to Figure A.7 in the appendix (pg. 47) for a class diagram of this class.

7.3.12 MainActivity

Many of the features of this class are covered in Section 7.2.

This class is responsible for coordinating all the program features and determines the display layout. An example of the layout on a large screen is given in Figure 13. The layout on a smaller screen is shown in Figures 14(a) and 14(b).

Please refer to Figure A.8 in the appendix (pg. 48) for a class diagram of this class.

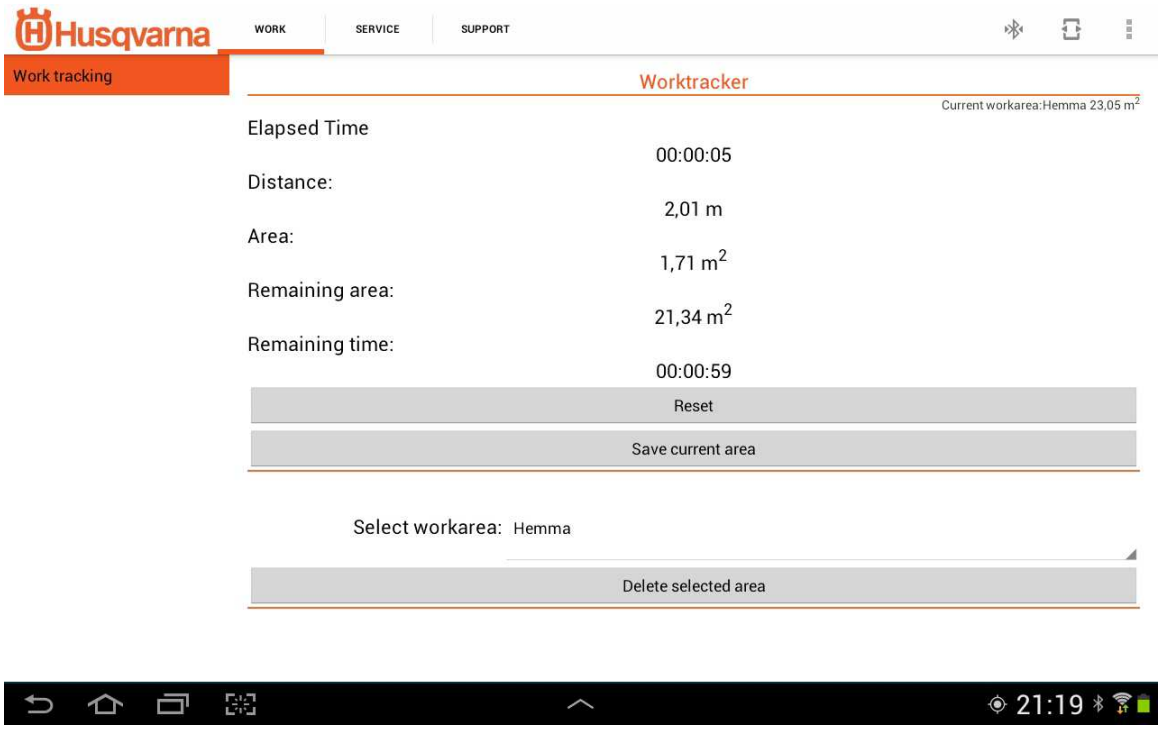
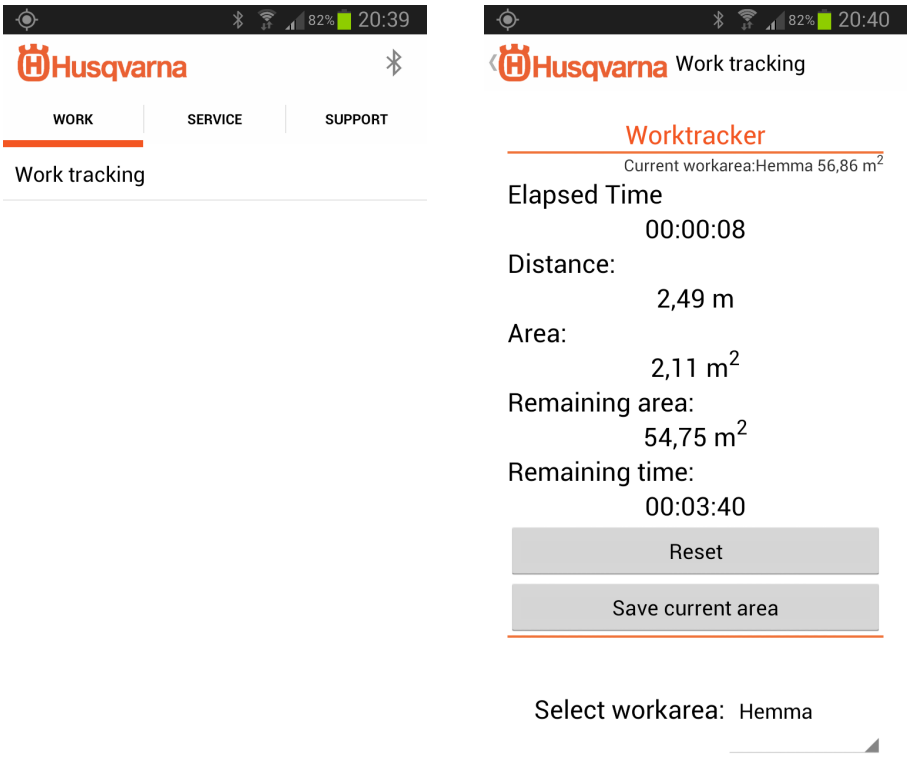


Figure 13: Application running on a tablet (large screen).



(a) Menu displayed at startup

(b) Work tracker displayed in a separate fragment

Figure 14: Application running on a phone (small screen).

7.3.13 MaintenanceSchedule

The maintenance schedule feature is handled by this class. The maintenance intervals are taken from the user manual for the Battery Rider and are hard-coded into the application. This could easily be changed so that the intervals could be adjusted in the application settings.

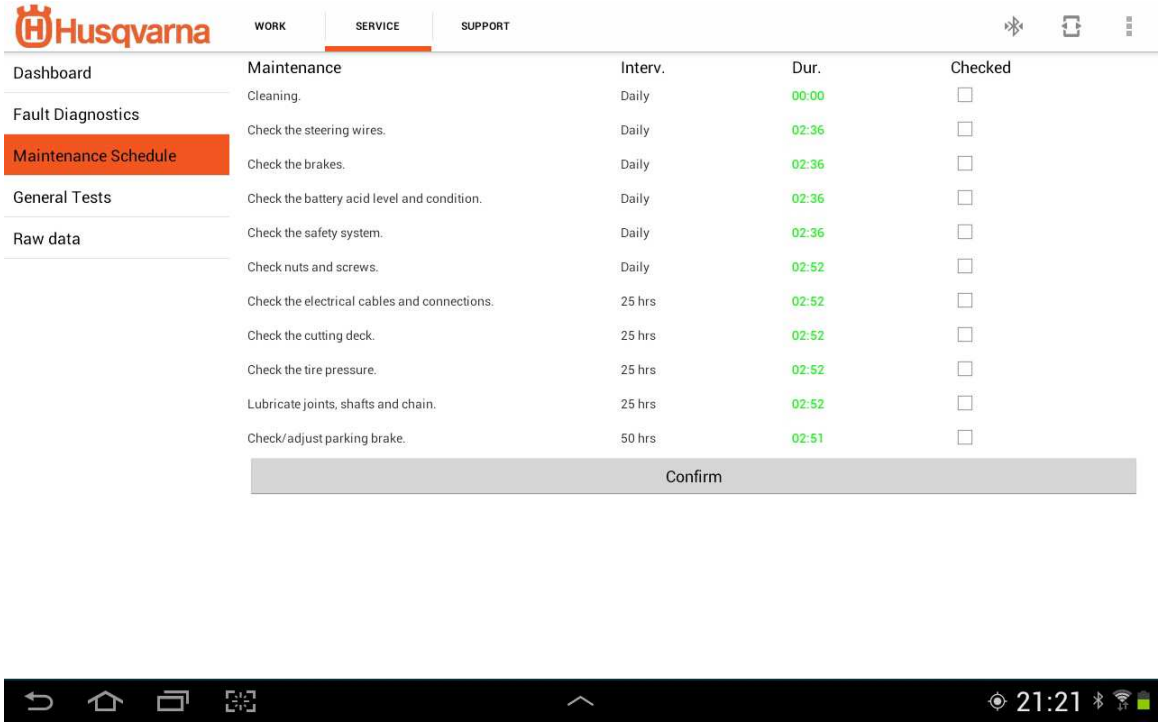


Figure 15: Maintenance Schedule GUI (tablet view).

The MaintenanceSchedule class depends on two asynchronous tasks. One is for fetching the latest value of the timer and updating the display when the user has the maintenance schedule display open, see Figure 15. The user is able to see how long it is since the maintenance was performed and check the maintenance that has just been completed. Tasks that are due are displayed in red color.

The other tasks runs in the background, no matter what is being displayed, and notifies the user if any maintenance is needed. This is done with a regular Android notification (see Figure 16) and an alarm sound.

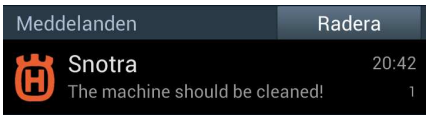


Figure 16: Maintenance notification.

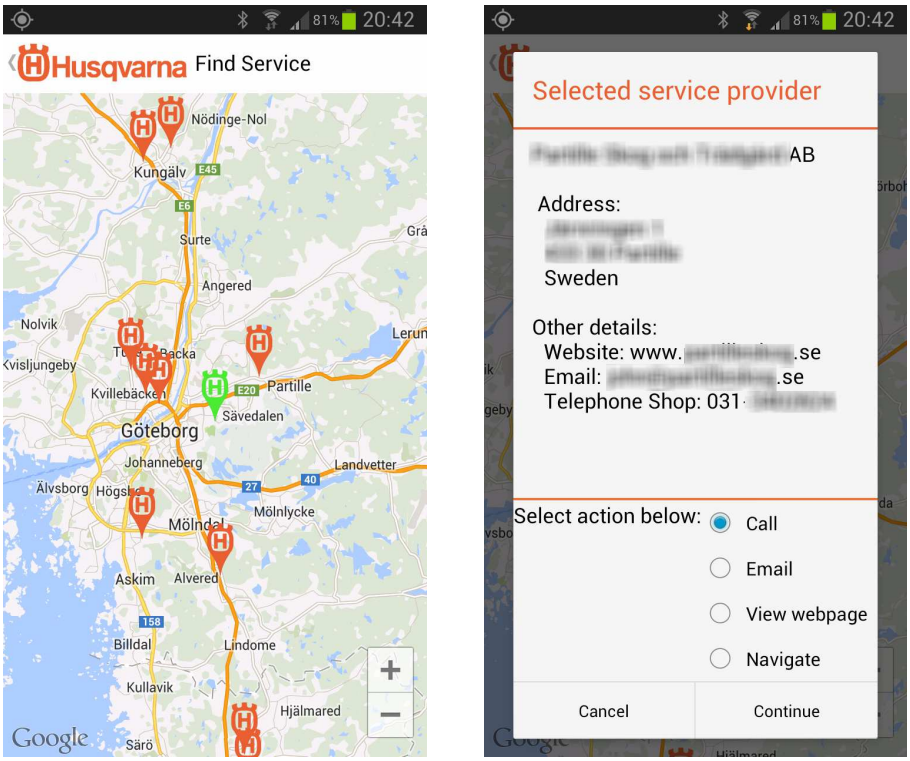
Please refer to Figure A.9 in the appendix (pg. 49) for a class diagram of this class.

7.3.14 MenuFragment, MyMenu, MyMenuCategory and MyMenuEntry

The functions of these classes was discussed in Section 7.2.2.

7.3.15 MyMapFragment

MyMapFragment contains the service map features. An example of the service map view is shown in Figures 17(a). It depends on JSONParser and JSONParserDetails since they gather all the information that is displayed. When the user selects a specific service provider a detailed view of that service provider is displayed, see Figure 17(b). The user can choose to use the information displayed to contact the service provider by phone or email, browse the company's website or use the navigational features of the smartphone to get directions to the given address.



(a) Service map at startup.

(b) Detailed information about a service provider.

Figure 17: Service map GUI (small screen).

7.3.16 PreferenceFragment and SettingsActivity

SettingsActivity is a wrapper for the PreferenceFragment. The program preferences are shown in Figure 18.

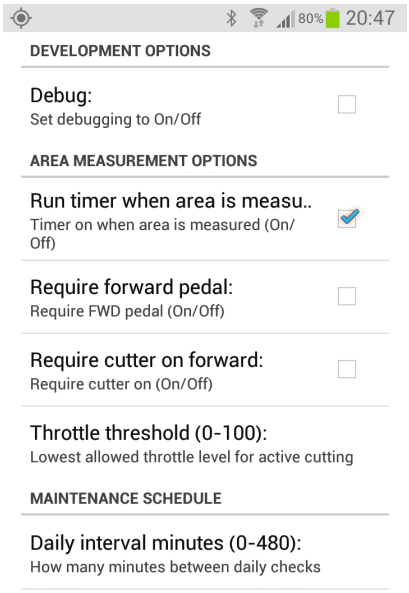


Figure 18: Preferences for the Android application.

7.3.17 RawData

The purpose of this class is to display the current state of the Battery Rider in a simple text form, see Figure 19. The views are updated directly from ReceiveThread.

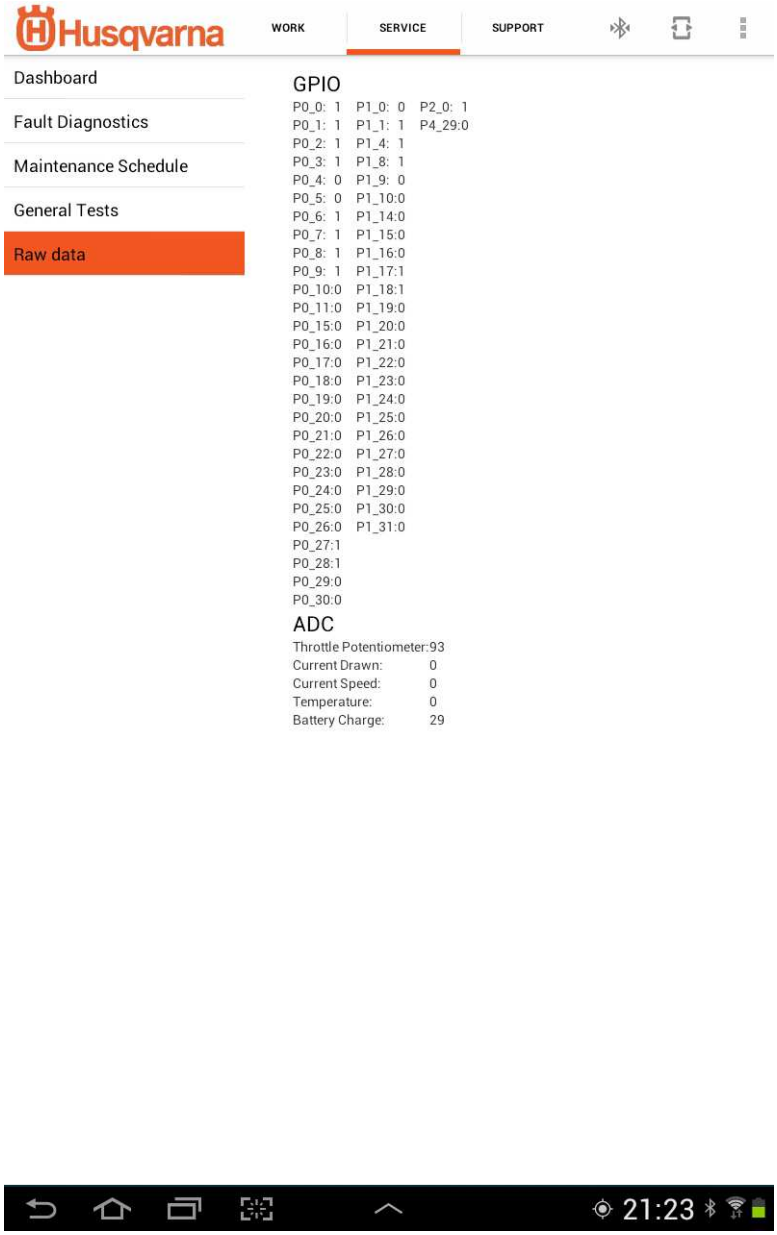


Figure 19: Raw Data GUI (tablet view).

7.3.18 ReceiveThread

The functionality if this class has been described shortly in Section 7.2.1. This thread is started as soon as a Bluetooth connection is established. It reads from a list of received messages that MainActivity stores. It parses the received messages and updates the machine state according to the information that the messages contain. After a successful reception of a message it invalidates the view that is currently displayed to the user.

7.3.19 SerialMessages

This class holds constants with all the serial messages of the Battery Rider serial communication protocol.

7.3.20 Tests

The purpose of this class is to enable the user to control the outputs of the RCU. The user interface is shown in Figure 20.

At startup, the current state of the Battery Rider is checked and the initial position of the switches moved to the correct position. Whenever a switch is moved a serial message is sent via Bluetooth to the Battery Rider. This view also displays the serial number, product number and software version of the connected machine.

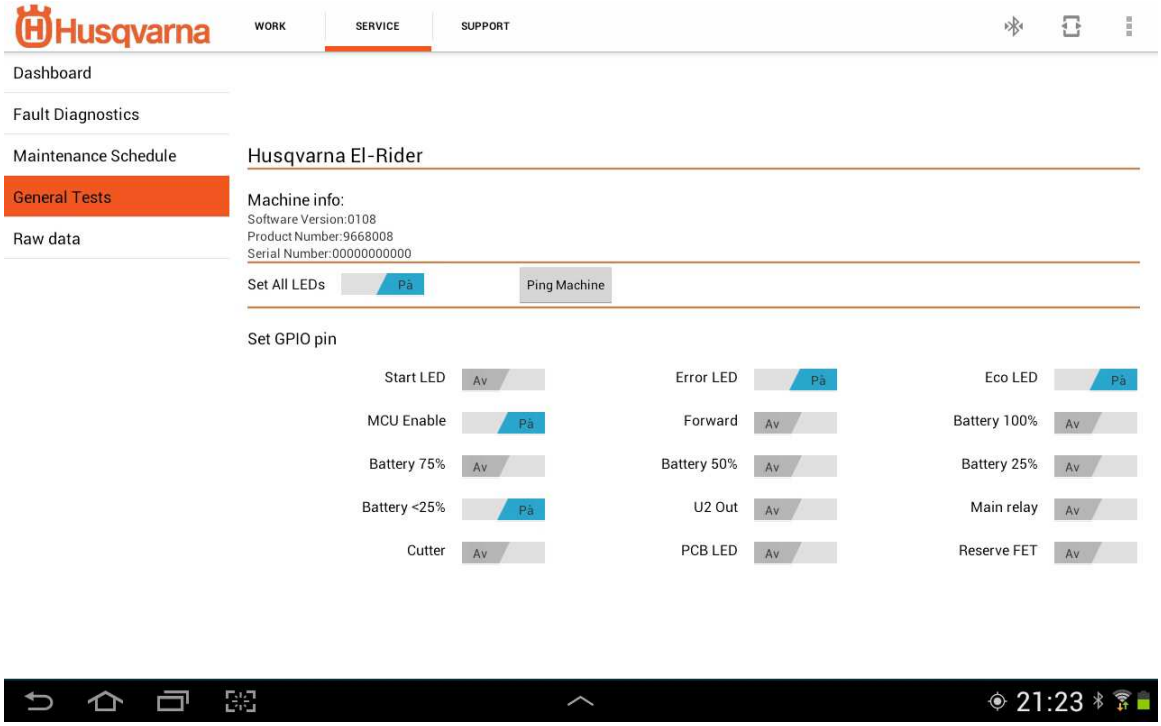


Figure 20: General Tests GUI (tablet view).

7.4 Compatibility issues and testing

The Android software was tested on two devices. The smartphone that was used was Samsung Galaxy S3 (GT-I9300) running Android version 4.1.2. The tablet that was used for testing was Samsung Galaxy Tab 10.1 (GT-P7510) running Android version 4.0.4.

The testing was performed continuously throughout the design phase. A system test was performed towards the end of the project. It was verified that the software was able to meet all the requirements on both of the devices. That is, the software provides the user with all the required features. The behavior of the software also proved to be consistent between the two devices.

Thorough testing was not performed on other devices but quick checks showed that the software could be improved with respect to compatibility. Inconsistent rendering of text on different devices was one of the compatibility issues that were noticed during these tests.

The accuracy of the distance measurement used in the work tracking features depends heavily on the quality of the GPS receiver and the signal quality. Simple tests imply that the accuracy is within $\pm 10\,\%$. This may be improved by tweaking system parameters but that requires more extensive testing.

8 Discussion and future work

Previous sections describe the design of a system that provides smartphone connectivity to a Battery Rider from Husqvarna. The major challenges and observations will be discussed in this section. Ideas for future work and improvements are also presented.

Diversity is a major challenge in a project of this kind. On one side there are mobile devices that are rapidly changing, both in terms of hardware and software. On the other side are the machines and equipment that Husqvarna builds, and they are far from being homogeneous.

One way to deal with the issue of diversity is to keep the design as modular as possible. If one software module is responsible for managing the wireless connection then it would be possible to have many different connection types (Bluetooth, WiFi, ANT+ or ZigBee) and the only thing that had to be changed would be one hardware module and one software module. Another module would be responsible for implementing the serial protocol since it may vary between machines. Each specific program feature could also be implemented as modules and could be removed or added to the software to create different software solutions for different machines.

The process of adding the connectivity option to existing systems is a challenge on its own. Some systems, such as the Battery Rider, only require a firmware update and a simple hardware module to enable this option. The hardware could be sold as an additional feature or included as a standard with every new machine. The embedded systems of other machines could require extensive redesign to enable this feature. Systems that are currently equipped with a USB service port could possibly be modified with a USB to Bluetooth module.

Husqvarna might also want to consider the option of moving to purely wireless solutions for their servicing equipment.

When it comes to future work, adding remote connectivity options to the software would be interesting since it allows a whole new set of features to be added. Fleet management is one example of such a feature. Another important feature to implement is firmware upgrading via smartphone. This option together with remote access to the system could enable remote updating of the firmware. A quick investigation on the topic of firmware upgrading shows that there should be no fundamental difference between upgrading the firmware with a wireless solution compared to the existing USB solution. The major obstacle is the software implementation of the In System Programming (ISP) protocol. Information about the protocol and source code for existing solutions may be available. Time pressure prevented further analysis of this topic.

There are several things that need to be taken into consideration when a commercial solution of this kind is designed. Security and safety of the system are probably the most important factors although market research and user behavior might also be of great interest.

Current solution uses a very simple pairing method that provides no security at all. Anyone with the Android application installed can connect to the machine and therefore view and alter its state. Different methods of device pairing exist and could be used for this system. Using NFC tags to establish the connection is one option that could improve security.

The safety issues that a wireless connectivity brings to a system has to be studied in detail and individually for each system. Having wireless connectivity on a machine that only allows reading of system parameters cannot be considered to be a safety risk. In the case of the Battery Rider, it is possible to alter the state of the machine at any time. This is an obvious safety hazard. A solution to this might be to use interlocks of some kind. The user could for example enable service mode by inserting and turning a special key.

Another issue that has to be addressed in future development of the design is the privacy aspect. Gathering and possibly distributing data about the movement and usage of a certain equipment could violate the privacy of the user. In any case, the user must be made aware of and agree to the use of this data.

This project may be used, in part or as a whole, for further investigation of smartphone connectivity solutions for Husqvarna products. This master thesis was run in parallel with another master thesis at Husqvarna AB. The other thesis focused on implementing different

features. The work of that thesis relies on the hardware implementation that is described in this report. Despite this dependency the work of the two students was completely independent.

It has already been mentioned that the task of adding wireless connectivity to the Battery Rider from Husqvarna could be generalized and applied to other embedded systems. Many embedded systems could benefit from improved monitoring and maintenance options. Improved servicing features and user guidance could extend the lifetime of the equipment. This could also improve the efficiency of the equipment. The end result would be more efficient use of resources. Smartphone connectivity to embedded systems could therefore advance sustainable development in the field of embedded systems design.

9 Conclusion

Smartphone connectivity has successfully been added to a battery powered ride-on mower from Husqvarna.

Analysis of a suitable communication technique lead to the conclusion that energy efficient solutions, such as Bluetooth, should be used. The fact that Android is both widespread and open source made it ideal for building a demonstrator of this kind.

The existing system was analyzed and modified to allow wireless two-way Bluetooth communication to an Android device running a custom application. The result is a system that demonstrates various new features and can be added to an existing system with minor software and hardware changes.

All of the software requirements that were set at the beginning of the project were met. Some of the hardware requirements that were set could not be measured to confirm their fulfillment. More time is needed to implement proper tests for maximum throughput of the current wireless solution.

Adding features like firmware upgrading over a smartphone connection was identified as a time consuming but interesting task for the future.

References

[1] eMarketer Inc., "Half of uk smartphones run on android," May 2013, accessed 7. August 2013. [Online]. Available: http://www.emarketer.com/Article/Smartphone-Adoption-Tips-Past-50-Major-Markets-Worldwide/1009923

[2] "Lawnbotts.com," October 2011, accessed 08. August 2013. [Online]. Available: http://blog.lawnbotts.com/bluetooth-compatible-phones-remote-for-lawnbott-lb3510/

[3] J. Nielsen, "Response times: The 3 important limits," January 1993, accessed 16. April 2013. [Online]. Available: http://www.nngroup.com/articles/response-times-3-important-limits/

[4] S. Seow, "User interface timing cheatsheet — revision 0.2.0," 2009, accessed 16. April 2013. [Online]. Available: http://www.stevenseow.com/papers/UI%20Timing%20Cheatsheet.pdf

[5] "Fast Facts | Bluetooth Technology Website," 2011, accessed 10. April 2013. [Online]. Available: http://www.bluetooth.com/Pages/fast-facts.aspx

[6] "Core Specifications | Bluetooth Technology Website," 2011, accessed 10. April 2013. [Online]. Available: http://www.bluetooth.org/Building/HowTechnologyWorks/CoreSpecifications.htm

[7] "IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. - part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)," *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, 2005.

[8] "Wi-Fi CERTIFIED Wi-Fi Direct: Personal, portable Wi-Fi technology (2010)," October 2010, accessed 11. April 2013. [Online]. Available: http://www.wi-fi.org/knowledge-center/white-papers/wi-fi-certified-wi-fi-direct%E2%84%A2-personal-portable-wi-fi%C2%AE-technology-2010

[9] "Wi-Fi CERTIFIED Wi-Fi Direct: Frequently Asked Questions," October 2010, accessed 11. April 2013. [Online]. Available: http://www.wi-fi.org/sites/default/files/uploads/files/faq_20101021_Wi-Fi_Direct_FAQ.pdf

[10] J. Haartsen, "The bluetooth radio system," *Personal Communications, IEEE*, vol. 7, no. 1, pp. 28–36, 2000.

[11] B. SIG, "Architecture & Terminology Overview - covered core package version: 4.0," in *Specification of the Bluetooth system.* Bluetooth SIG, 2010, vol. 1.

[12] "Discover and Learn | Wi-Fi Alliance," 2013, accessed 9. August 2013. [Online]. Available: http://www.wi-fi.org/discover-and-learn

[13] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, 2007, pp. 46–51.

[14] V. Loseu, H. Ghasemzadeh, and R. Jafari, "A wireless communication selection approach to minimize energy-per-bit for wearable computing applications," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, 2011, pp. 1–8.

[15] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of wifi and bluetooth in smartphones," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 900–908.

[16] A. G. D. Camps Mur and P. Serrano, "Device to device communications with wifi direct: overview and experimentation," 2013, accessed 18. April 2013. [Online]. Available: http://www.campsmur.cat/files/camps_ssavedra_serrano_WCM_11_00112_final_version.pdf

[17] "Basics | Bluetooth Technology Website," 2011, accessed 18. April 2013. [Online]. Available: http://www.bluetooth.com/Pages/Basics.aspx

[18] International Data Corporation, "Apple cedes market share in smartphone operating system market as android surges and windows phone gains, according to idc," August 2013, accessed 9. August 2013. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[19] R. van der Meulen and J. Rivera, "Gartner says worldwide mobile phone sales declined 1.7 percent in 2012," February 2013, accessed 3. April 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2335616

[20] comScore Inc., "comscore reports april 2013 u.s. smartphone subscriber market share," June 2013, accessed 9. August 2013. [Online]. Available: http://www.comscore.com/Insights/Press_Releases/2013/6/comScore_Reports_April_2013_U.S._Smartphone_Subscriber_Market_Share

[21] A. Inc., "Mfi program enrollment: Frequently asked questions," 2013, accessed 13. April 2013. [Online]. Available: http://mfi.apple.com/faqs

[22] ——, "ios: Supported bluetooth profiles," Oct 2012, accessed 12. April 2013. [Online]. Available: http://support.apple.com/kb/ht3647

[23] ——, "Technical q&a qa1657 using external accessory framework with bluetooth devices," October 2012, accessed 13. April 2013. [Online]. Available: https://developer.apple.com/library/ios/#qa/qa1657/_index.html#//apple_ref/doc/uid/DTS40010232

[24] C. De Dominicis, D. Mazzotti, M. Piccinelli, S. Rinaldi, A. Vezzoli, and A. Depari, "Evaluation of bluetooth hands-free profile for sensors applications in smartphone platforms," in *Sensors Applications Symposium (SAS), 2012 IEEE*, 2012, pp. 1–6.

[25] Motorola, "Motorola Bluetooth low energy API," 2011, accessed 18. April 2013. [Online]. Available: http://www.motorola.com/sites/motodev/library/bluetooth_apis.html

[26] "Android Open Bluetooth Low Energy API," 2013, accessed 18. April 2013. [Online]. Available: http://android-btle.github.io/framework/

[27] Google, "Wi-fi direct | Android Developers," 2013, accessed 18. April 2013. [Online]. Available: http://developer.android.com/guide/topics/connectivity/wifip2p.html

[28] "WT12 data sheet, version 2.95," Bluegiga Technologies, January 2012.

Appendix

Class diagrams

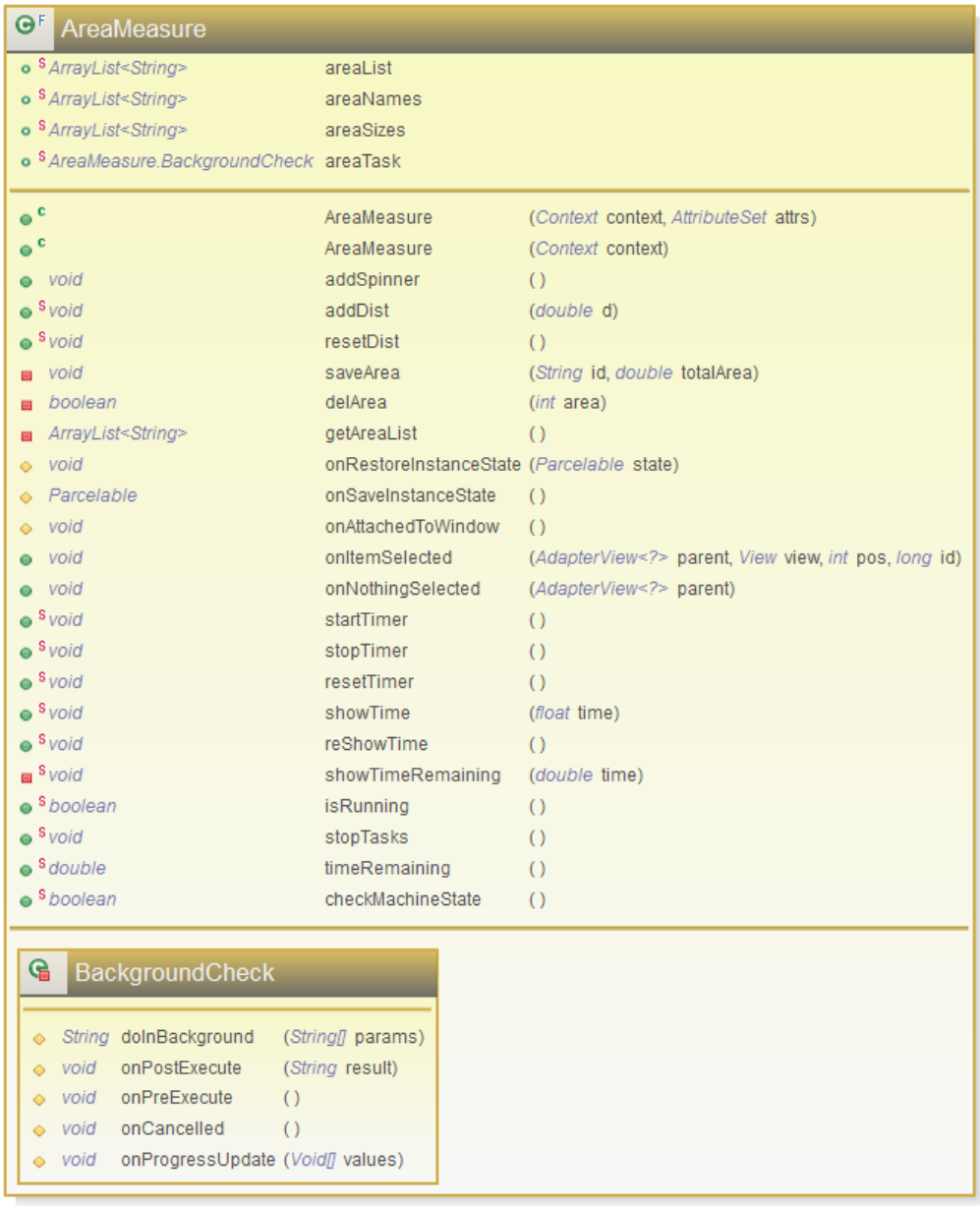


Figure A.1: Class diagram for AreaMeasure.

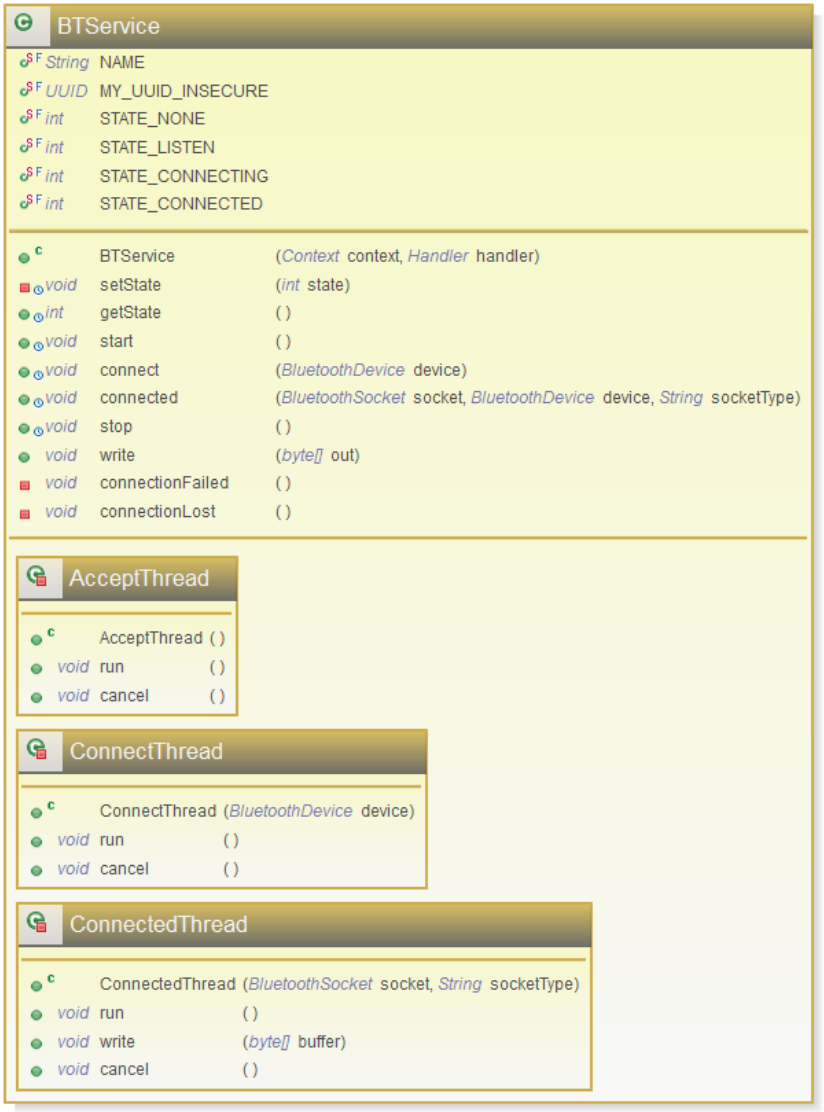


Figure A.2: Class diagram for BTService.

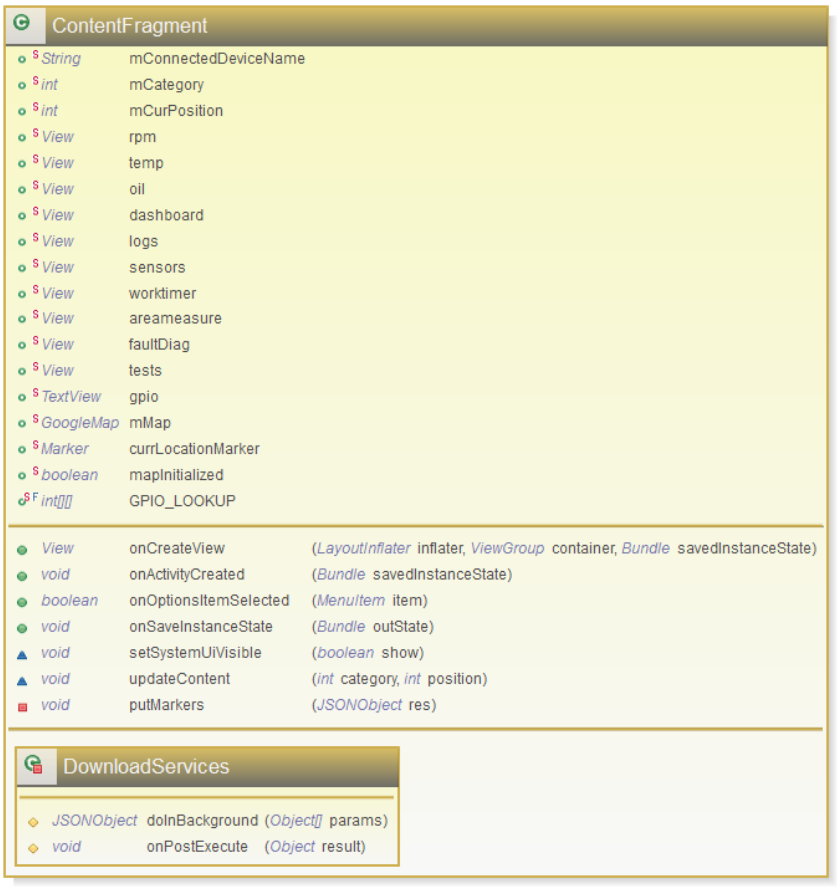


Figure A.3: Class diagram for ContentFragment.

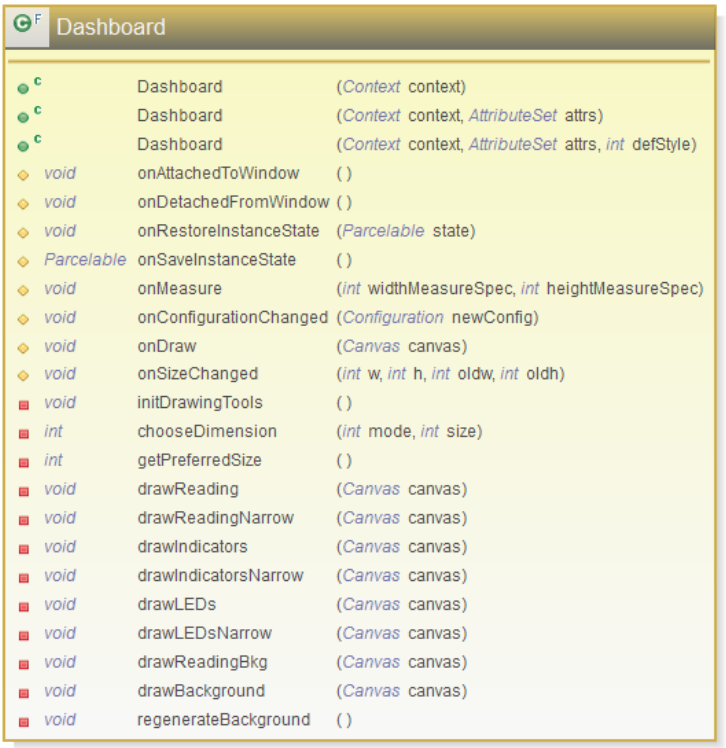


Figure A.4: Class diagram for Dashboard.

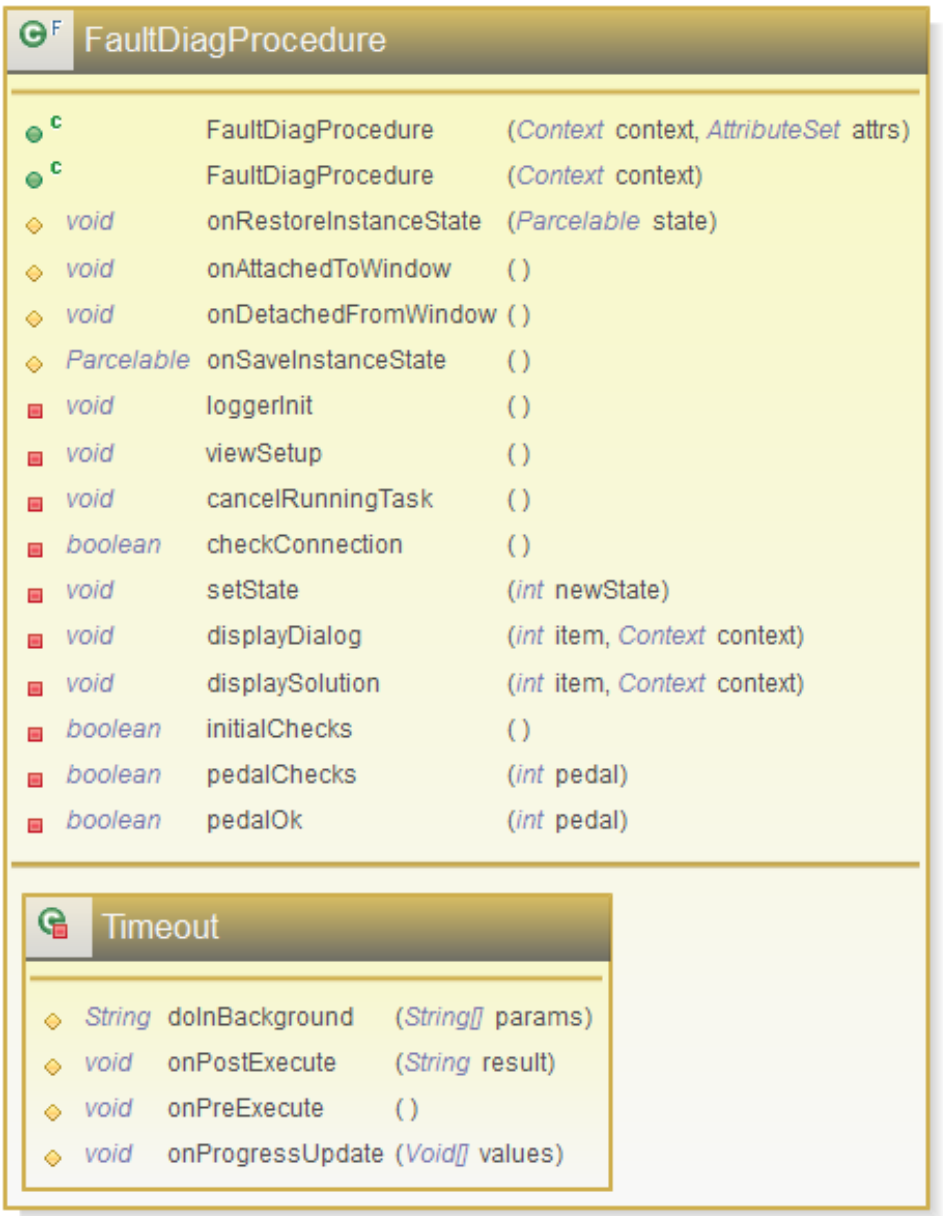


Figure A.5: Class diagram for FaultDiagProcedure.

Figure A.6: Class diagram for GPSData.

Figure A.7: Class diagram for MachineState.

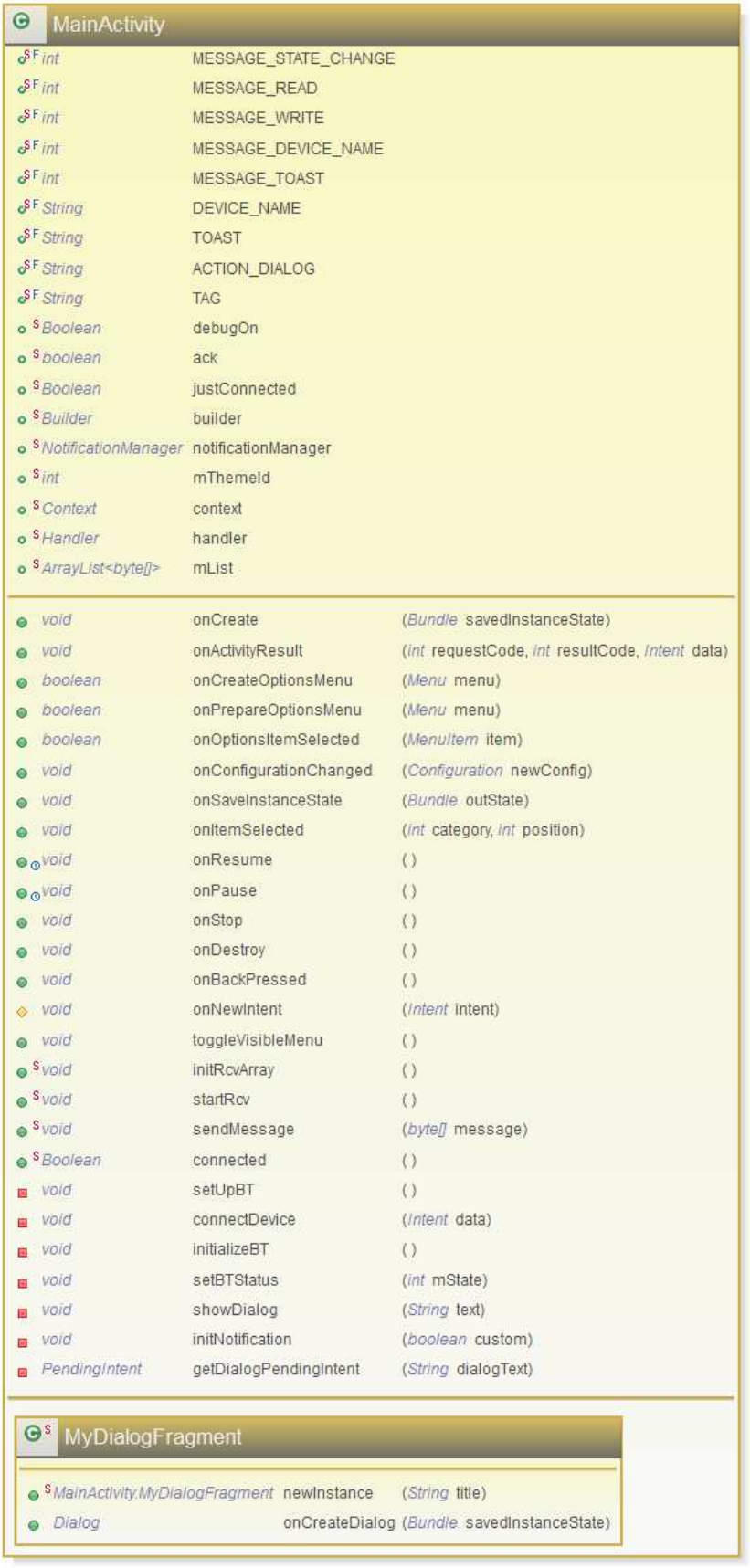


Figure A.8: Class diagram for MainActivity.

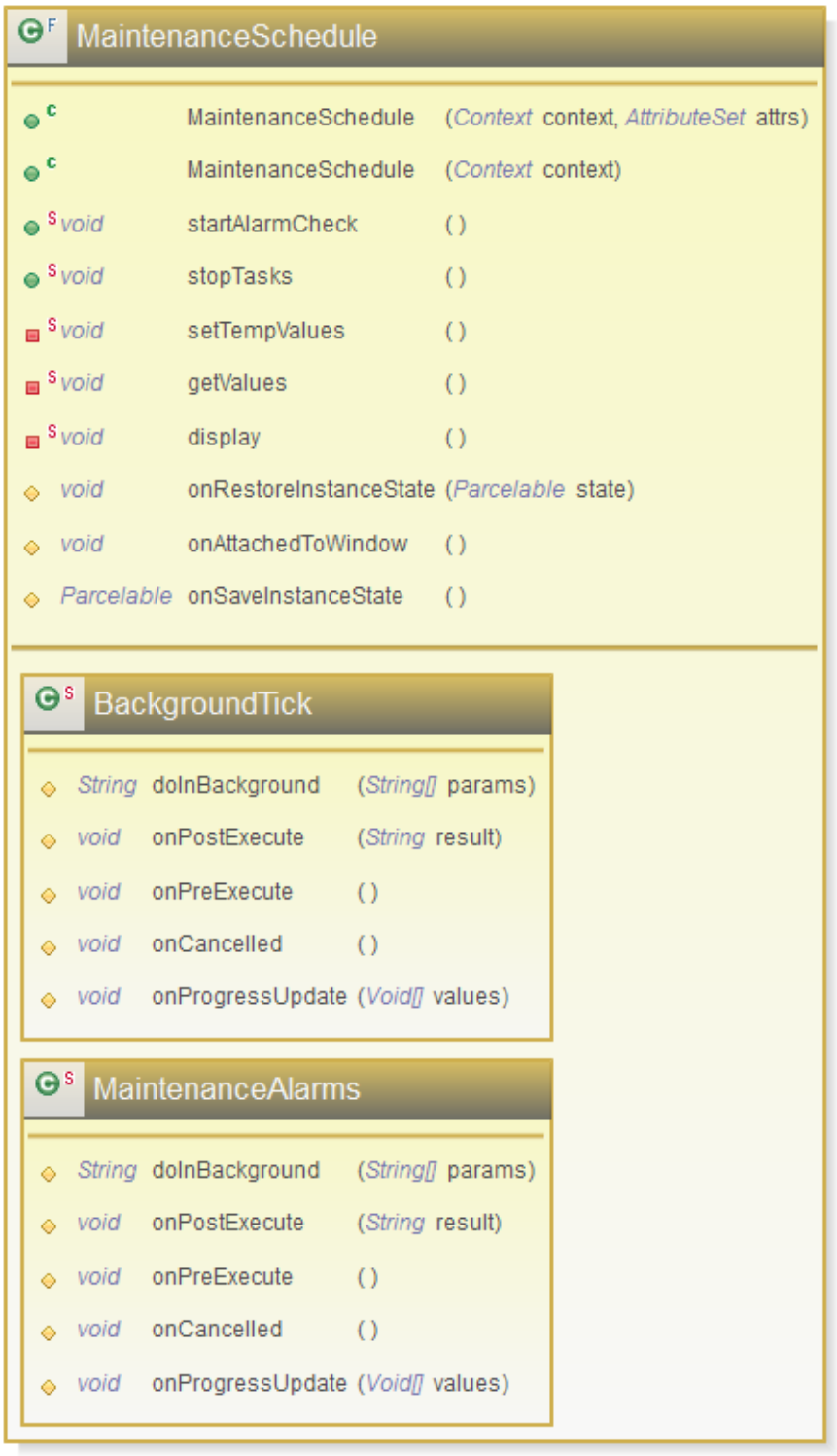


Figure A.9: Class diagram for MaintenanceSchedule.

Figure A.10: Class diagram for Tests.