Vendor-Neutral Software Application for Fitness Wearables

Kamil Wojtczyk - 40128893

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Software Engineering

School of Computing

May 2017

Authorship Declaration

I, Kamil Wojtczyk, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: May 2017

Matriculation no: 40128893

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

With the constant increase in popularity of fitness lifestyle, many people decide to buy a fitness tracker to be able to track their progress in achieving fitness goals. Major manufacturers offer a broad range of devices starting from budget options to more expensive trackers targeting fitness hobbyists. All those devices seem to share similar functionality, however, officially they are only compatible with the vendors' software. On the other hand, the specifications for each device describe that most fitness trackers are using Bluetooth 4.0 Low Energy, which provides standards for implementing specific services. This project attempts to build a vendor-neutral solution that would unify the functionality of multiple fitness trackers. It does so, by analysing the shared specification of fitness trackers and their Bluetooth interfaces, to create an abstracted model of a device. The model is then used to design a proposed architecture of a system that would offer a flexibility of providing support to individual trackers. The project also attempts to implement the design by utilising modern web technologies of using Bluetooth in the browser. The resulting application can connect to a single fitness tracker and utilise its functionality in the form of a web application. Additionally, the results of the research reveal that Bluetooth specification is lacking specific support for fitness wearables, which results in manufacturers implementing their standards.

Contents

1 Introduction 12

1.1 Project Background 12

1.2 Aims & Objectives 12

2 Literature and Technology Review 14

2.1 Introduction 14

2.2 Fitness trackers 14

2.2.1 History 14

2.2.2 Overview 16

2.2.3 Specification 17

2.2.4 Official Software Development Kits 21

2.3 Bluetooth Low Energy 22

2.3.1 Overview 22

2.3.2 Comparison of Bluetooth Classic and Bluetooth Low Energy 23

2.3.3 GAP 24

2.3.4 GATT 26

2.4 Reverse engineering 27

2.4.1 Introduction 27

2.4.2 Legal aspect 27

2.4.3 Process 28

2.4.4 Conclusion 28

2.5 Existing open-source projects 29

2.5.1 Introduction 29

2.5.2 Mibanda + Mibui 29

2.5.3 OpenYou Project 30

2.5.4 Galileo 31

2.5.5 Gadgetbridge 31

2.5.6 Conclusion 32

2.6 Ontology and OWL 33

2.7 Web Bluetooth API 33

2.8 Conclusion 35

3 Requirements Analysis 36
3.1 Introduction 36
3.2 Programming environment 37
3.2.1 JavaScript 37
3.2.2 Web Bluetooth API 38
3.2.3 Text Editor 38
3.2.4 Source Control 38
3.3 Hardware 38
3.3.1 Xiaomi Mi Band Pulse 39
4 System Design 41
4.1 Introduction 41
4.2 Ontology of a Fitness Tracker 41
4.3 Software Application Model 45
4.3.1 Activity Diagram 46
4.3.2 Class Diagram 47
5 Implementation 48
5.1 Bluetooth Service Discovery 48
5.2 Pairing problems 52
5.3 Step Count 53
5.4 Final design 55
6 Evaluation 59
6.1 Fulfilment of requirements 59
6.2 Vendor-neutral approach 60
6.3 Cross-platform support 62
7 Conclusions 64
7.1 Achievement of aims 64
7.2 Retrospective 64
7.3 Further Work 65
8 References 66

List of Tables

Table 1: Side-by-side comparison of fitness trackers from Xiaomi, Fitbit and Garmin 21

Table 2: Comparison between BLE and Bluetooth Classic (Bluegiga Technologies, 2011) 24

Table 3: Xiaomi Mi Band Pulse specification (Xiaomi, 2017) 39

Table 4: Requirements evaluation 60

Table 5: Jawbone Up Move specification (Jawbone, 2017) 61

List of Figures

Figure 1: Manpo-kei pedometer (Axworthy, 2016) 15

Figure 2: Sports Tester PE2000 (POLAR, 2013) 15

Figure 3: Fitbit tracker (Savov, 2009) 16

Figure 4: Xiaomi Mi Band (Xiaomi, 2016) 18

Figure 5: Fitbit One (Fitbit, 2017a) 19

Figure 6: Garmin vívosmart 3 (Garmin, 2017c) 20

Figure 7: (a) BLE protocol stack; (b) structure of a BLE data unit (Gomez et al., 2012) 23

Figure 8: GAP request exchanges between the Advertising and Scanning Processes (Townsend, 2014) 25

Figure 9: Broadcast topology (Townsend, 2014) 25

Figure 10: GATT Profile Hierarchy (Bluetooth SIG Inc, 2017a) 26

Figure 11: Mibui - Mibanda graphical user interface (Acena, 2015) 30

Figure 12: Gadgetbridge Android application 32

Figure 13: Web Bluetooth API device selection prompt (Yasskin, 2016) 34

Figure 14: Xiaomi Mi Band Pulse fitness tracker (Xiaomi, 2017) 40

Figure 15: Ontology of a Fitness Tracker 41

Figure 16: Visualised ontology of a Fitness Tracker using the Protégé program 42

Figure 17: Fitness Tracker class without heart rate monitor 43

Figure 18: Fitness Tracker class with heart rate monitor 43

Figure 19: List of individual fitness trackers categorised in the ontology 44

Figure 20: Theoretical data properties offered by a fitness tracker 44

Figure 21: Activity diagram 46

Figure 22: Class diagram 47

Figure 23: nRF Connect used to discover Bluetooth Services of Mi Band 1S 48

Figure 24: Reverse-engineered interface of a Mi Band 1S (Freeyourgadget, 2017) 49

Figure 25: Mi Band 1S Bluetooth interface in JavaScript 50

Figure 26: JavaScript function for storing Bluetooth characteristics..... 50

Figure 27: JavaScript code used to establish connection with the fitness band . 51

Figure 28: Java code used to authenticate with the Mi Band 1S (Freeyourgadget, 2017) 52

Figure 29: JavaScript code used to authenticate with Mi Band 1S 53

Figure 30: Reading the step count data from the fitness tracker using JavaScript 54

Figure 31: Reading additional data from the fitness tracker..... 54

Figure 32: Live streaming data using JavaScript's notification system 55

Figure 33: Interface of the implemented JavaScript application..... 56

Figure 34: Connecting to the fitness tracker..... 56

Figure 35: Main dashboard of the application 57

Figure 36: Increasing step count when live streaming the data 57

Figure 37: Debugging console of the JavaScript application 58

Figure 38: Jawbone Up Move fitness tracker (Jawbone, 2017) 61

Figure 39: Testing web application on the Google Nexus 6p..... 62

Figure 40: Fitness tracker successfully connected to the Google Nexus 6p 63

Acknowledgements

First and foremost, I would like to thank my supervisor, Brian Davison, for all his support and enthusiasm through the completion of the project. There were many situations where I felt stuck, but thanks with the help of discussing the problem together, I managed to move on.

I would also like to thank my partner and family for supporting me through the journey of discovering Bluetooth Low Energy devices.

1 Introduction

1.1 Project Background

With the increase in popularity of fit lifestyle over the last years, more companies are offering ways to track progress in fitness achievements. From basic step trackers to precise activity monitors, each customer may find the best device for their needs. Wearables from biggest brands seem to differ mostly in design, as the functionality offered very similar to the competitors. Nevertheless, the included software application is unlikely to support any additional device outside the manufacturer's range.

Imagine a software solution being able to unify all fitness wearables and synchronise the fitness activities from them. It would be able to process the data and visualise it, while also offering the user freedom to use it anyhow – from exporting to a single CSV file to uploading the fitness logs to third party solutions. This solution would create a centralised hub for fitness trackers that could serve as an alternative software, giving the user freedom to choose between the application provided by the manufacturer and a vendor-neutral approach. It could be updated over time to provide support to additional wearables after they are released.

1.2 Aims & Objectives

The aim of this project is to design, implement, test and evaluate a software application which provides a centralised, vendor-neutral hub for fitness wearables using modern web technologies.

The following objectives have been identified to achieve this:

- Carry out a comparative review of existing fitness wearables
- Investigate the Bluetooth Low Energy specification
- Research existing projects in the area
- Identify the specification of a fitness tracker
- Create ontology of a fitness tracker
- Design software application model

- Implement application using modern web technologies
- Test the application with different fitness trackers
- Evaluate the results
- Complete the final report

2 Literature and Technology Review

2.1 Introduction

The literature and technology review explores the question whether it is possible to implement a vendor-neutral software application that would unify currently available fitness wearables. It also aims to analyse whether there are any available standards to interact with those devices. To answer those questions, the review will analyse the main aims of fitness trackers, the technology used both from hardware and software perspective and solutions available to third party developers to integrate the devices into their system. Based on the output, it will also provide additional information about the legality of reverse-engineering, existing open-source solutions and descriptions of technologies used in the practical section of the project.

2.2 Fitness trackers

2.2.1 History

According to research done by the Wearable, the predecessor of fitness trackers was a ‘lie detector’ also known as the Polygraph back in 1921. Surprisingly, the same technology that was used to measure body statistics such as the galvanic skin response, pulse rate and blood pressure, is now used in modern fitness trackers to reveal whether the user is putting enough effort in the workout (Axworthy, 2016).

An invention closer to an actual fitness tracker known nowadays was a 1965 Manpo-kei pedometer (Fig. 1) introduced by a Japanese professor Dr Yoshiro Hatano, who tried to combat obesity in Japan by influencing people to walk more. His research stated that a daily rate of ten thousand steps is a key in achieving a balance between optimum weight and fitness that would decrease the obesity in the population. This amount of steps is set as a default to most fitness trackers today (Axworthy, 2016).

Figure 1: Manpo-kei pedometer (Axworthy, 2016)

Technological improvements in the late 20th and early 21st century changed the way how progress in fitness activities was monitored. Automated systems became more popular and easier to wear as more often they were integrated into watch-size devices (Wikipedia, 2017). The first wearable heart rate monitor designed for sports professionals and athletes, the Sports Tester PE2000 (Fig. 2), was founded by a company called POLAR in 1982 (POLAR, 2013).

Figure 2: Sports Tester PE2000 (POLAR, 2013)

Consumer-grade activity trackers started to become more popular in the early 2000s with the introduction of the original Fitbit tracker (Fig. 3) in 2009 – a clip worn at waist or wrist with the included band, which acted as a digital pedometer able to detect the kind of activity performed (sitting, walking or running) and track movements when sleeping, thanks to the included 3-dimensional accelerometer. Fitbit came with a base station enabling the user to synchronise the tracker with a computer to view the activity data and upload it to the Fitbit servers (Ewalt, 2010).

Figure 3: Fitbit tracker (Savov, 2009)

2.2.2 Overview

According to the Cambridge Dictionary, a fitness tracker can be defined as an "electronic device, usually worn on the wrist, that is designed to record information about person's physical fitness and activity" (Cambridge University Press, n.d.). A typical tracker will measure the number of steps made; distance travelled and calorie consumption. A more advanced device may include a heart rate monitor, an ability to track the quality of sleep or a GPS module for more accurate monitoring. Apart from being worn on the wrist, fitness trackers are often attached to the waist or worn on the arm. The main aim of modern activity trackers is to be easily integrated into peoples' life, due to a small size and easy to use interface (Gallaga, 2015).

Fitness trackers are used in multiple different areas from athletes and fitness professionals, casual users, to academic researchers investigating the impact of

technology on different aspects of life. As software applications provided with trackers often provide a social way of networking with others by informing of a start of a run or progress in achieving a fitness goal, the new generation of fitness devices start a movement where people self-monitor their fitness activity to compete with others (Miller, 2013). This opens a new possibility for academic researchers to explore the connection between maintaining a healthy weight and social competition. One of the studies attempted to fight childhood obesity by introducing fitness trackers to a group of children and engaging them in a system called StepStream, which pulled student's individual fitness progress into a social network site in a game format. It resulted in children being so competitive that they attempted to take the pedometers everywhere to earn extra points in the game (Miller, 2013). Other academic studies focus on the use of fitness trackers in the medical and health department, such as the ability to monitor and assess sleep for adults (de Zambotti, Claudatos, Inkelis, Colrain, & Baker, 2015), monitor glucose level for diabetics (Anonymous, 2016) or use tracker as a digital fitness coach (Schmidt, Benchea, Eichin, & Meurisch, 2015). Surprisingly, similar technologies are used to track the activity of wildlife animals and their energy consumption (Curry, 2014).

According to a report made by Research and Markets, the business market of fitness wearables is likely to grow in 2016-2020 at a very high pace, due to an increase of chronic diseases such as diabetes and obesity, and an increase in some fitness clubs and people interested in healthy lifestyle. The biggest companies mentioned in the report include Fitbit Inc., Garmin Ltd., Jawbone Inc. and Xiaomi Inc. ('Global Fitness Wearable Device Market 2016-2020 - Dominated by Fitbit, Garmin, Xiaomi & Jawbone', 2016).

2.2.3 Specification

2.2.3.1 Introduction

As the 'activity tracker' term is widely used across different aspect of the industry, there is no standardised specification for fitness trackers. However, as similarities between consumer-grade devices trackers can be seen, the minimal hardware specification can also be identified. The following section will analyse the specification of three activity trackers from well-known brands in different price range.

2.2.3.2 Xiaomi Mi Band

Xiaomi Mi Band (Fig. 4) is the most affordable device in comparison with an asking price of $14.99 (about £12). It features a wristband in six different colours with the main unit attached in the middle of the band. The device is splash and dust resistant with an IP67 rating and a very light in construction. The sensors and components include an accelerometer and a vibration motor. The tracker uses a Bluetooth 4.0 radio module and is equipped with a 41 mAh lithium-polymer battery. It is equipped with a three-led display that lights up according to fitness achievements in the day or other activities. It is designed to track walking, running and sleeping activities (Xiaomi, 2016). Is it unknown what is the size of the internal memory and how many days of activity tracking data can be stored. The only way to interact with the device is through the software application on a smartphone.

Figure 4: Xiaomi Mi Band (Xiaomi, 2016)

2.2.3.3 Fitbit One

Fitbit One (Fig. 5) is a clip-style fitness tracker, a successor to the original Fitbit from 2009 and later Fitbit Ultra, priced at £79.99 and available in two different colours. It is sweat, rain and splash proof with an IP67 rating. The sensors and components include a 3-axis accelerometer, an altimeter and a vibration motor. The radio transceiver located in the device is a Bluetooth 4.0 module, with a lithium-polymer. The tracker is equipped with an OLED display and a button to interact with, displaying various daily statistics and a clock. The built-in memory allows saving seven days of motion data in

detail and 23 days of daily total statistics. Apart from walking and running, the tracker is also able to track sleeping activity, and the number of floors climbed (Fitbit, 2017a).

Figure 5: Fitbit One (Fitbit, 2017a)

2.2.3.4 Garmin Vívosmart 3

The fitness tracker from Garmin, the Vívosmart 3 (Fig. 6), is the most expensive device in the comparison priced at £129.99, however, targeting fitness hobbyists more than casual users. It is a wrist-worn tracker, available in three different colours and two sizes. It is fully water resistant with an IP68 rating. The sensors include an accelerometer, barometric altimeter, heart-rate monitor and an ambient light sensor. The tracker is equipped with a Bluetooth 4.0 and ANT+ modules for connectivity and a lithium-ion battery. It features an OLED screen displaying daily activity statistics, weather forecast and clock. The memory allows storing seven timed activities and fourteen days of detailed tracking data. It can monitor different types of activities from running, floor climbing, weight lifting to swimming. It also features an automatic repetition counting for fitness and strength training (Garmin, 2017c).

Figure 6: Garmin vívosmart 3 (Garmin, 2017c)

2.2.3.5 Side-by-side comparison

	Xiaomi Mi Band	**Fitbit One**	**Garmin vívosmart 3**
Sensors	Accelerometer, vibration motor	Accelerometer, altimeter, vibration motor	Accelerometer, altimeter, heart-rate monitor, ambient light sensor, vibration motor
Connectivity	Bluetooth 4.0	Bluetooth 4.0	Bluetooth 4.0, ANT+
Display	LED indicator lights	OLED	OLED
Memory	Data not available	7 days of detailed motion data, 23 days of statistic	7 timed activities, 14 days of detailed motion data

IP rating	IP67 (dust and splash proof)	IP67 (dust and splash proof)	IP68 (fully water resistant)
Activities tracked	Walking, running, sleeping	Walking, running, sleeping, floor climbing	Walking, running, sleeping, floor climbing, swimming, fitness and weight training
Price	$14.99 (about £12)	£79.99	£129.99

Table 1: Side-by-side comparison of fitness trackers from Xiaomi, Fitbit and Garmin

2.2.3.6 Summary

Based on the side-by-side comparison, it is clear that certain similarities can be found between devices. A typical activity tracker consists of an accelerometer and a vibration motor, a Bluetooth 4.0 module, memory, battery and some form of display, from basic LED indicator lights to full OLED screen. It is also dust, splash and sweat-proof with a minimal rating of IP67. The minimal set of tracked activities includes walking, running and sleeping.

2.2.4 Official Software Development Kits

Most manufacturers provide a form of extending the functionality of the device, either by a web API or a direct SDK. It is very uncommon to release specification for individual Bluetooth endpoints or general information about the interface implemented in the trackers. In fact, the use of web API is mostly the only way of accessing and manipulating the data gathered by the device. This approach makes it more difficult for developers to work with fitness trackers directly, as data stored on the proprietary servers may not always be in sync with real-time use.

Fitbit offers a web API only solution, where the developer has to register an application to use the web services. The web API offers access to synchronised data such as fitness activities, sleep logs or heart rate data. Fitbit developed a ‘Subscriptions API’, which allows developers to set up an endpoint to be notified when new data is synchronised with the Fitbit cloud (Fitbit, 2017b).

Similarly, Garmin offers a set of multiple web APIs ranging from paid-for professional ‘Connect API’ to a less detailed ‘Health API’, which offers access to fitness metrics such as the number of steps made, the intensity of the workout, heart rate data or sleep logs. The API is only available to approved developers (Garmin, 2017b). Garmin also offers an SDK called the ‘Connect IQ’, which enables to extend the functionality of a device and direct communication over Bluetooth or ANT+, however, it is only available to a selection of smartwatches and GPS trackers (Garmin, 2017a).

On the contrary, Xiaomi offers a web API with clients in written in the majority of different languages including PHP, Python, Java (Android) and Objective-C (iOS). However, it is only available for registered developers, and the documentation is written in Chinese, as seen in Appendix 5. As no translation is available, it is not possible to investigate whether the API provides fitness related data.

It is unclear why manufacturers do not offer a direct way of communicating with the devices over Bluetooth, however, the most obvious reason is being in full control of the data stored on the device and synced with the cloud. Lack of direct access also prevents developers from accessing the devices in an unintended way.

2.3 Bluetooth Low Energy

2.3.1 Overview

Bluetooth Low Energy (BLE) also known as “Bluetooth Smart”, is a light-weight part of the classic Bluetooth, mostly targeting devices requiring lower power consumption. It was introduced as a subset of Bluetooth 4.0 core specification. It is supported by a majority of operating systems nowadays, starting with (Townsend, 2014):

- Microsoft Windows 8
- Apple OS X 10.6
- GNU/Linux Vanilla BlueZ 4.93
- Android 4.3
- iOS5

Similarly to the classic Bluetooth, BLE protocol stack is created out of a set of a Controller and a Host (Fig. 7a). The Controller usually includes the Physical Layer and the Link layer, as a part of a small System-on-Chip with an integrated radio interface.

The Host uses the application processor to offer “upper layer functionality” such as the Attribute Protocol (ATT), Generic Attribute Profile (GATT) or Generic Access Profile (GAP) (Gomez, Oller, & Paradells, 2012).

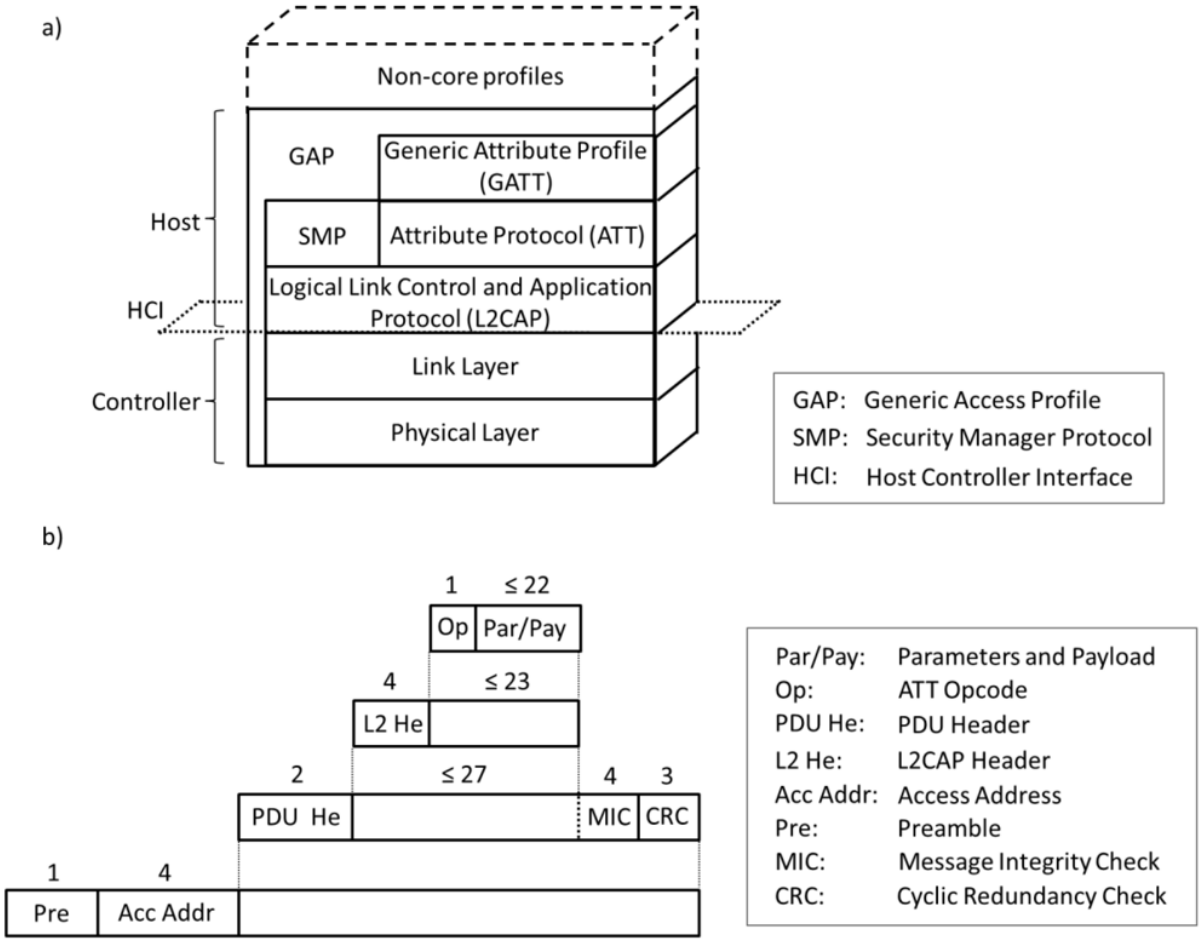


Figure 7: (a) BLE protocol stack; (b) structure of a BLE data unit (Gomez et al., 2012)

2.3.2 Comparison of Bluetooth Classic and Bluetooth Low Energy

Bluetooth Low Energy aims to reduce the energy cost and delay between messages and provide an increase performance of discovery, by decreasing latency and application throughput to slower speeds (Harris, Khanna, Tuncay, Want, & Kravets, 2016). A full comparison between BLE and Bluetooth Classic can be seen below (Table 2).

Technical specification	Classic *Bluetooth* technology	*Bluetooth* low energy technology
Radio frequency	2.4GHz	2.4GHz
Distance/Range	~10-100 meters	~10-100 meters
Symbol rate	1-3Mbps	1Mbps
Application throughput	0.7 – 2.1Mbps	305kbps
Nodes/Active slaves	7	Unlimited
Security	56 to 128 bit	128-bit AES
Robustness	FHSS	FHSS
Latency (from not connected state to send data)	100+ ms	<6ms
Government regulation	Worldwide	Worldwide
Certification body	Bluetooth SIG	Bluetooth SIG
Voice capable	Yes	No
Network topology	Point-to-point, scatternet	Point-to-point, star
Power consumption	1 (reference value)	0.01 to 0.5 (use case dependent)
Service discover	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, headsets, stereo audio, automotive, PCs etc.	Mobile phones, gaming, PCs, sport & fitness, medical, automotive, industrial, automation, home electronics etc.

Table 2: Comparison between BLE and Bluetooth Classic (Bluegiga Technologies, 2011)

2.3.3 GAP

Generic Access Profile (GAP) is responsible for controlling connection between devices, discovery and advertising of services (Townsend, 2014) and security (Gomez et al., 2012). GAP defines four roles on the controller: Broadcaster, Observer, Peripheral and Central (Gomez et al., 2012), with the most important concepts being the last two (Townsend, 2014). A device in the Broadcaster role can only broadcast data via advertising channels and does not support connections from other devices. On the contrary, the observer can only receive data sent by the Broadcaster. The Peripheral role is designed for small, low power and resource constrained devices, which require connecting to a device in the Central role, which is responsible for initiating and managing connections between devices (Gomez et al., 2012). For example, a fitness tracker or a heart rate monitor is a Peripheral device, as it requires a mobile phone or a tablet, a Central device, to connect to and process the data (Townsend, 2014).

Advertising of services is done in two different ways, one by sending the Advertising Data payload and the other by scanning the Response payload. Both data structures are exact, with a maximum size of 31 bytes of data. The advertising device transmits data packets with a specific interval, while the scanning device listens to the response payloads and if it is interested in the service, it can optionally request it. The entire process can be seen in Figure 8 and 9. Longer delays between advertising and scanning processes save power but make the communication feel less stable and responsive (Townsend, 2014)

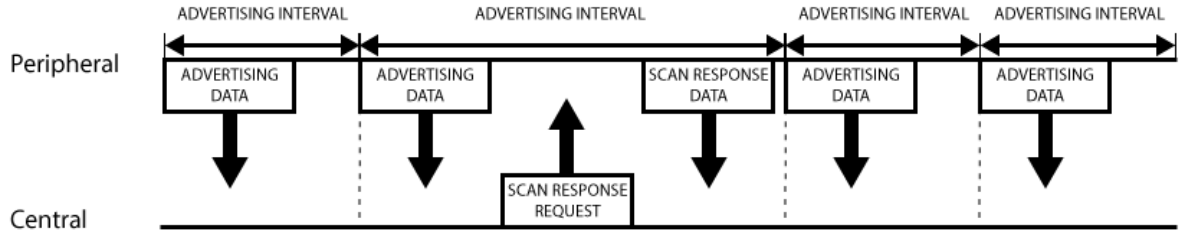


Figure 8: GAP request exchanges between the Advertising and Scanning Processes (Townsend, 2014)

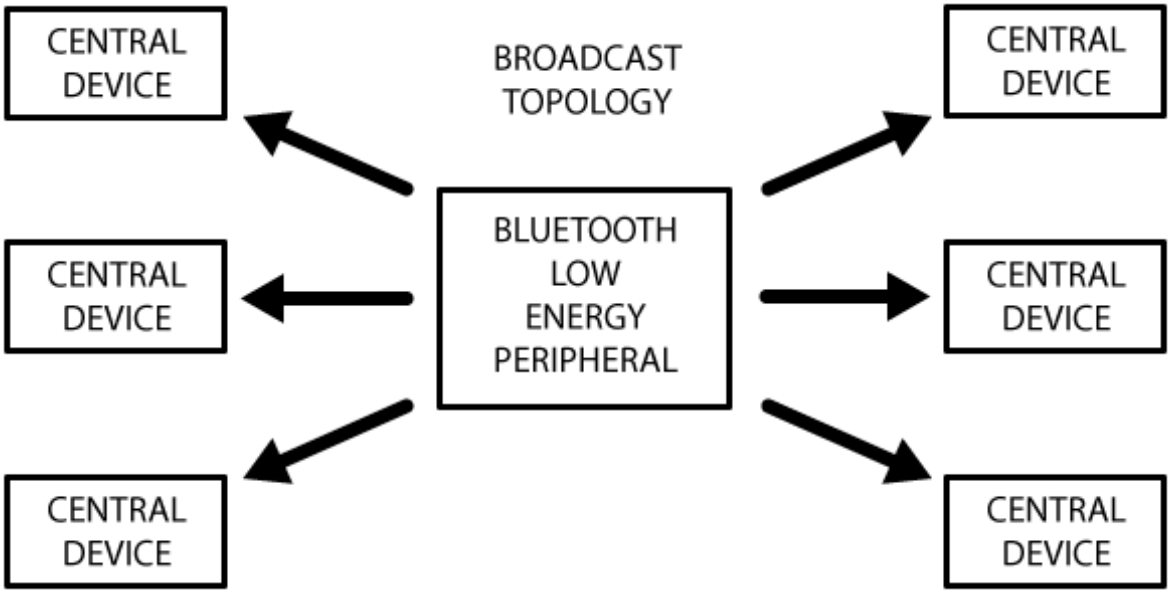


Figure 9: Broadcast topology (Townsend, 2014)

When a connection is made between the Peripheral and Central device, the advertising process will stop propagating data, and further communication is made through GATT services and characteristics in unidirectional exchange. Is it worth noticing that the connection then becomes exclusive, as a BLE Peripheral device can only be connected to a single Central device at a time (Townsend, 2014).

2.3.4 GATT

Generic Attribute Profile (GATT) defines the framework used between two BLE devices to exchange data using concepts called Services and Characteristics (Fig. 10). It uses a generic data protocol in Bluetooth called the Attribute Protocol (ATT) to store Services and Characteristics and data related to the service in a lookup table using 16-bit ID for each entry (Townsend, 2014). As an example, a Peripheral device that includes a temperature sensor may offer a service with a 'temperature' characteristic that offers separate attributes for describing the sensor, storing the temperature measurement values and units (Gomez et al., 2012). GATT uses a client-server relationship when communicating between devices (Townsend, 2014).

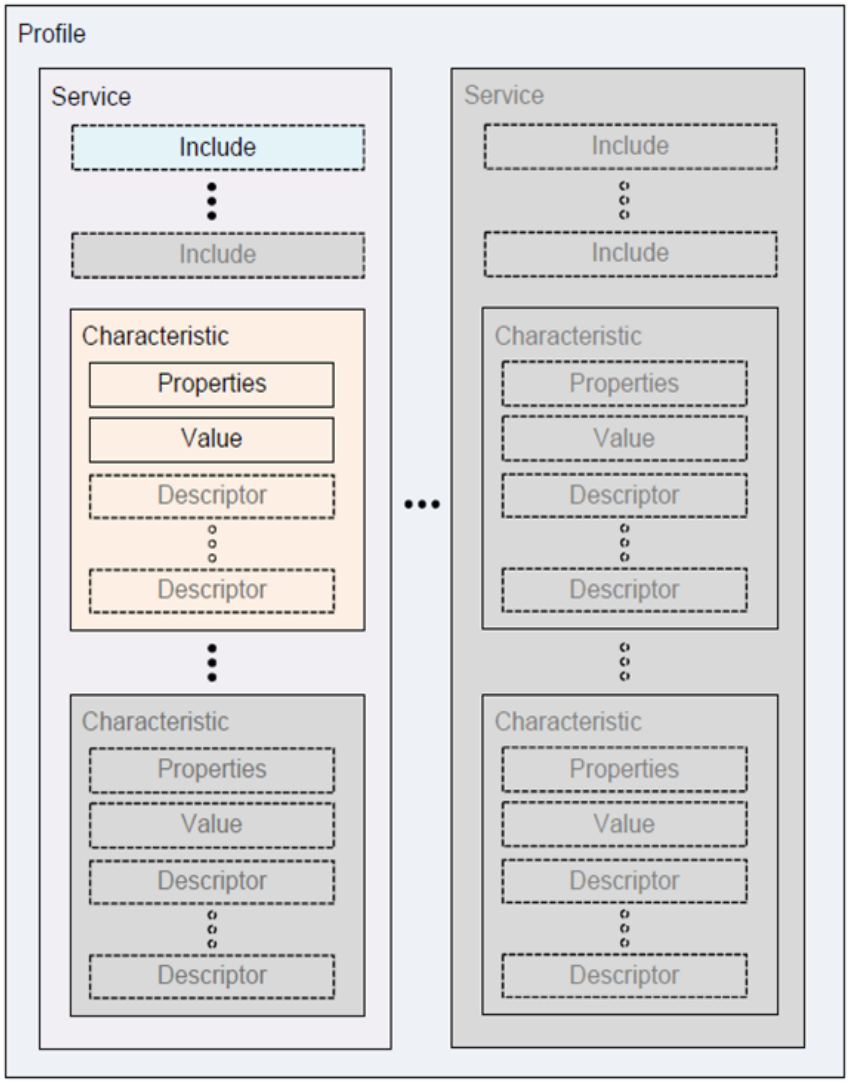


Figure 10: GATT Profile Hierarchy (Bluetooth SIG Inc, 2017a)

A profile is not stored on the Peripheral device itself, as it is only a simple collection of services, but is available in the Central device. Profiles are used to group services that serve selected purposes, such as the Heart Rate Profile, which combines the Hearth Rate and Device Information services (Townsend, 2014). A full list of GATT profiles is available in Appendix 6.

Many profiles were adapted from Bluetooth Classic specification, implementing some of the services in Health Device Profile (HDP) specification (Bluetooth SIG Inc, 2017b). Most notable GATT profiles related to fitness include the Blood Pressure, Body Composition, Cycling Power, Fitness Machine, Heart Rate and Weight Scale. However, there is no profile responsible for collecting step count data which could be used in fitness trackers, which leads to a lack of standards in the specification.

2.4 Reverse engineering

2.4.1 Introduction

As the Bluetooth Core specification lacks a dedicated profile for GATT step count tracking, manufacturers implement their custom services that do not follow any standard and differ between devices. Unfortunately, is very uncommon to publicly release the specification of the implemented interfaces. As the main goal of this project is to investigate whether the creation of vendor-neutral application is possible, an attempt to reverse-engineer a fitness tracker will be made.

2.4.2 Legal aspect

European Union Directive 2009/24/EC specifies details on the intellectual property law binding in the European Union countries. It states that “unauthorised reproduction, translate, adoption or transformation” of the software or hardware by a third-party is illegal and breach the exclusive rights of the author. The directive specifies, however, situations where such an action is legal and does not require permission from the legal owner (European Parliament, 2009).

One of such scenarios is described in the Article 6 of the directive, specifying that a person who is in legal possession of a program or hardware device, is permitted to perform necessary actions to achieve “interoperability” between the independently created program and other software, including reverse-engineering the protocol, if the

necessary information has not been previously shared with public (European Parliament, 2009). In the context of the academic project, this legislation enables to legally undertake reverse engineering actions against software and hardware of a selected fitness tracker.

2.4.3 Process

Reverse engineering of a software application is achieved by a process called decompiling or disassembling the machine-readable code back to a human-readable form. It is a very complicated process, as the source code for a program is usually written in a higher-level programming language, which contains comments, labels and variables names, which are all stripped out when compiled to assembly instructions. As a result of such disassembly, the source code lacks human explanation and is very difficult to interpret. Reading decompiled code is often comparable to reading a novel without all adjectives, adverbs, articles, paragraphs, chapters, which makes the entire process very chaotic. Even the most experienced computer programmer will require a reasonable amount of time to understand how the piece of software works, before being able to identify the specification of an interface or general process of a program (Daughtrey, 1994).

The process of reverse engineering combined software and hardware is more complex, especially when there is limited knowledge about the device's operations. In such a situation, there is a need for a combined software and hardware co-comprehension process (LaRoche & Cox, 2004). In the case of reverse engineering a fitness band, the process will require a deep understanding of the purpose of individual GATT service and its effect on the device.

2.4.4 Conclusion

While the process of reverse engineering of copyrighted software or hardware is an illegal practice in the European Union, the Article 6 of European Union Directive 2009/24/EC allows for reverse-engineering in a situation where the person is in legal possession of a program or device and wants to achieve "interoperability" between other computer systems or software. In other words, extending the functionality of a device or software is permitted in the legal aspect.

2.5 Existing open-source projects

2.5.1 Introduction

There is some existing open source project attempting to reverse engineer interfaces used in fitness trackers to build libraries for extracting and manipulating data. Some of those projects focus on selected device only, while others attempt to create a unified solution for a range of devices from selected manufacturer. The most interesting project, however, is attempting to build a free, cloudless and vendor-neutral application for Android system that aims to support multiple fitness trackers, smart watches and other fitness monitors, and unify their functionality, while giving the user full control of data gathered. The following section will discuss most notable projects from each category.

2.5.2 Mibanda + Mibui

Mibanda is a Python library made by Oscar Acena, which aims to provide connectivity with Xiaomi Mi Band from any operating system with Python 3 installed, however, mostly targeting Linux subsystems. It provides a precompiled package for Debian systems, however, can also be compiled from source from any system. Functionality-wise it enables to control and read every GATT characteristic offered by the fitness tracker, for instance, it enables to read the number of steps made for each day, stream real-time step counts, set a daily goal or display custom colour on the LED display on the band (Acena, 2016a). It is implemented using pygattlib - a Python library that enables to communicate with BLE devices using the GATT protocol (Acena, 2016b).

The same author has also built a graphical interface for his library, named Mibui (Fig. 11), that allows a user using the Debian-based system to interact with the fitness tracker without implementing the library in a custom application. The application is available as a precompiled Debian package or can be built from source code publicly available (Acena, 2015).

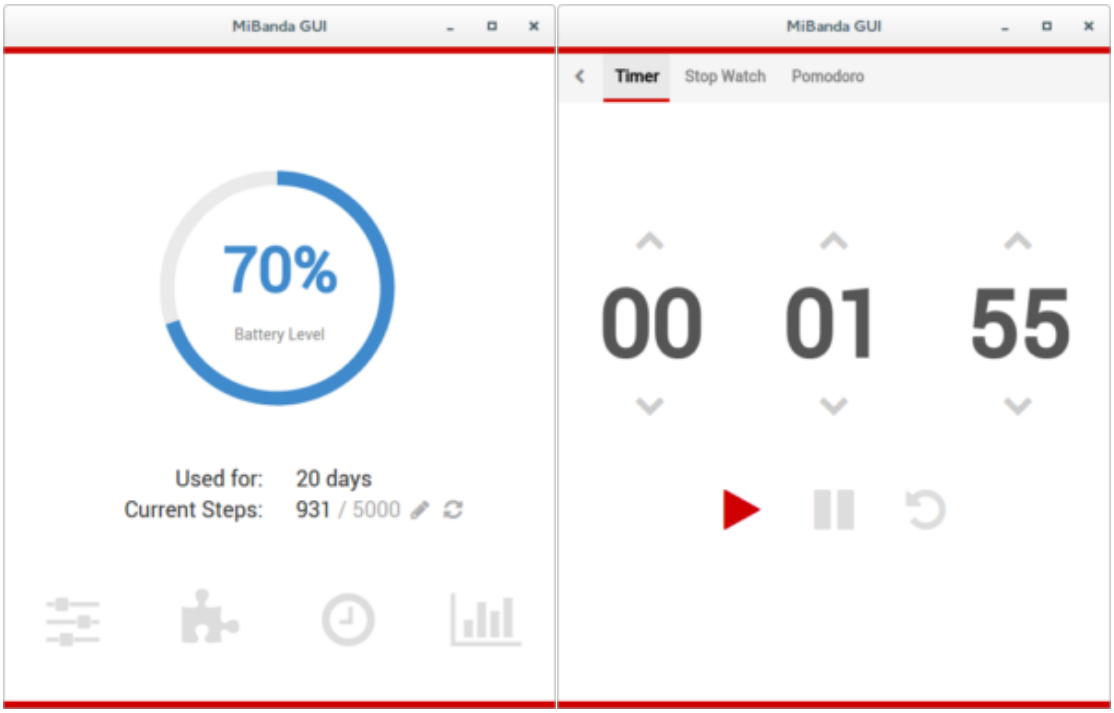


Figure 11: Mibui - Mibanda graphical user interface (Acena, 2015)

2.5.3 OpenYou Project

OpenYou project, also known as the Open Source Health Hardware Access project, is an interesting attempt started by the OpenYou organisation (most notably Kyle Machulis, the founder), which aims to reverse-engineer multiple fitness and health-related devices and provide libraries to interact with them. Instead of focusing on BLE connectivity, it mostly targets reverse-engineering drivers to USB-based dongles used to synchronise the wearable with a host machine to understand the interface of a device (OpenYou Organisation, 2013).

One of the most notable projects is libfitbit – a Python 2.7 based library, which aims to provide connectivity with Fitbit Ultra tracker. The tracker itself was a successor to the original Fitbit from 2009 and used ANT wireless network technology to communicate with a USB dongle provided. The library implemented data retrieval protocol and enabled to synchronise the fitness tracker to download stored data, process it and optionally upload to the Fitbit cloud services (Machulis, 2011).

OpenYou has also created some other libraries for accessing fitness and health wearables, such as the libfuelband – a Python 2.7 library for communicating with the

Nike Fuelband and libsensewear – a Python 2.7 library for Bodybugg (OpenYou Organisation, 2013).

2.5.4 Galileo

Galileo is a Python command line tool for synchronising newer Fitbit trackers utilising BLE communication protocol. The project supports the full range of Fitbit products including:

- Fitbit One
- Fitbit Zip
- Fitbit Force
- Fitbit Charge
- Fitbit Charge HR
- Fitbit Alta (experimental support)
- Fitbit Surge (experimental support)
- Fitbit Blaze (experimental support)

The utility targets multiple Linux distributions, providing prebuilt binaries for each, besides being able to compile from source code. Galileo features the ability to synchronise a Fitbit tracker locally, store data dumps for analysis and securely upload to the Fitbit server over HTTPS. The main idea behind the project was to enable compatibility on Linux, as Fitbit does not offer any first-party solution for the operating system (Allard, 2017). It is worth noticing that the project is kept up-to-date with the latest hardware from Fitbit, with new code commits being sent each day.

2.5.5 Gadgetbridge

Gadgetbridge (Fig. 12) is the most advanced open source project from the above mentioned, as it attempts to create a first-class, vendor-neutral and cloudless software application for Android based devices unifying multiple fitness trackers and smartwatches from different vendors. Its main aim is to allow to use a Pebble or Mi Band wearable without the vendor's closed source application installed on the host device and the need to create an account and transmit data to the manufacturer's servers. The application designed its abstract interface of a wearable, which is then implemented in individual adapters for each device. The list of supported devices includes (Freeyourgadget, 2017):

- Pebble smartwatches (original Pebble, Pebble Time, Pebble Time Steel, Pebble Time Round, Pebble 2)
- Mi Band fitness trackers (Mi Band, Mi Band 1A, Mi Band 1S, Mi Band 2)
- Sony Ericsson LiveView smartwatch
- HPlus Devices (ZeBand and others)

Supported functionality differ based on the connected device but enable to interact with the device, store and retrieve data and update firmware (Freeyourgadget, 2017). It is in active development, being constantly updated by the open source community on a daily basis.

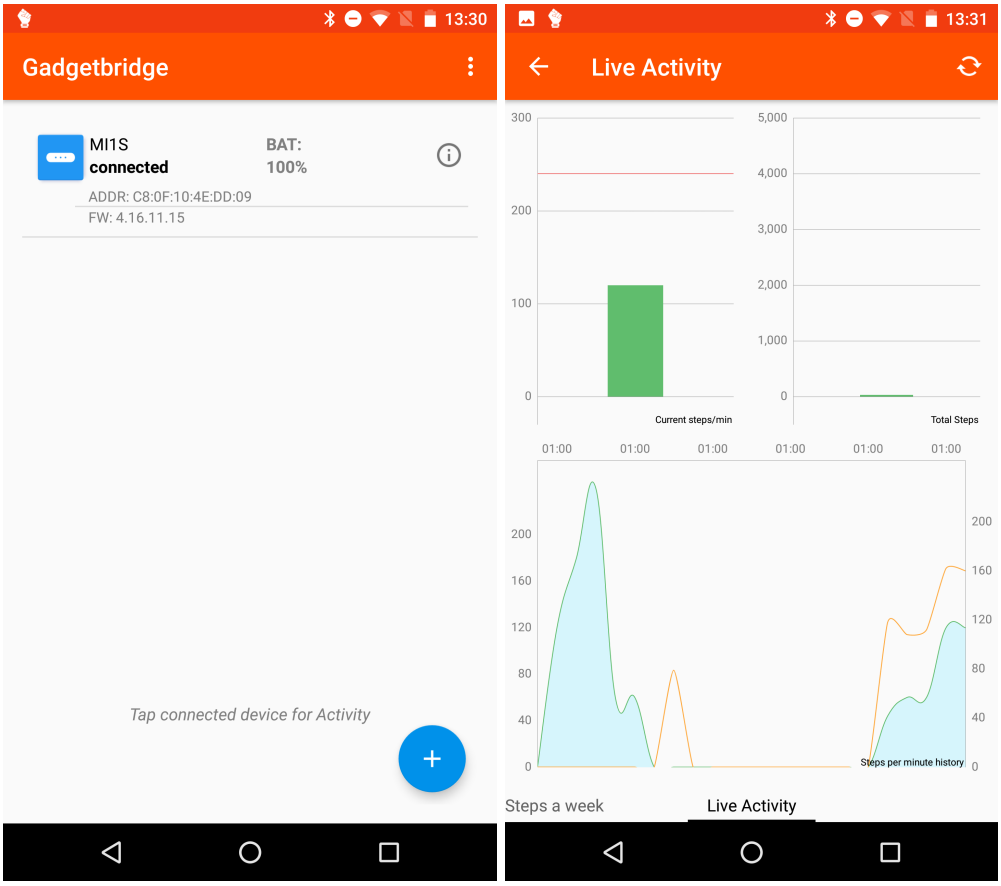


Figure 12: Gadgetbridge Android application

2.5.6 Conclusion

Multiple open source projects exist that attempt to reverse engineer fitness wearables to extend the functionality of a device or provide support for other operating systems. Even though Bluetooth Low Energy has no standard for fitness trackers, one of the

projects attempts to build a generic and vendor-free solution that unifies the functionality of the wearables.

2.6 Ontology and OWL

In the context of computer sciences, an ontology is a semantic data model that defines a set of primitives used to model a domain of knowledge. Those primitives are typically classes, attributes and relationships among other class members. Primitives are described with information on their meaning and their position in a system (Gruber, 2009). In the context of fitness wearables, ontology can be used to describe the level of abstraction of data models, their attributes and relationship to other individuals and interfaces used.

Historically, the term “ontology” comes from philosophy field, where it was used as a theory of the nature of being and existence. In computer and information science, it is a technical term defining that an object is designed for the certain purpose that helps to model the knowledge about given domain, real or imagined. During the early 1990s, ontology layer was used as a standard component of knowledge system due to the effort to create interoperability standards in the overall technology stack (Gruber, 2009).

Ontologies are part of the W3C standard stack, defined as the Web Ontology Language (OWL), for the Semantic Web that specifies a standardised way to exchange data among systems to allow interoperability across multiple services (Bechhofer, 2009). In other words, OWL is designed to represent rich and complex information about web resources, their groups and relations between other objects in the Web format.

2.7 Web Bluetooth API

Web Bluetooth API is a specification that allows websites to communicate with local Bluetooth devices in a secure and privacy-preserving way. The specification is currently in a draft form with ongoing active development. It was published by the Web Bluetooth Community Group and is not a part of the W3C standard (Web Bluetooth Community Group, 2017a).

Web Bluetooth API attempts to provide a secure way of connecting to the local Bluetooth devices, by removing access to certain Bluetooth features that are difficult to be implemented in a secure way. Instead of being based on certificate authentication, it allows the user to manually select and approve access to a specific device (Fig. 13). The first version of the specification allows website to run only in the Bluetooth Central role to connect to GATT servers using Bluetooth Low Energy connection (Web Bluetooth Community Group, 2017a).

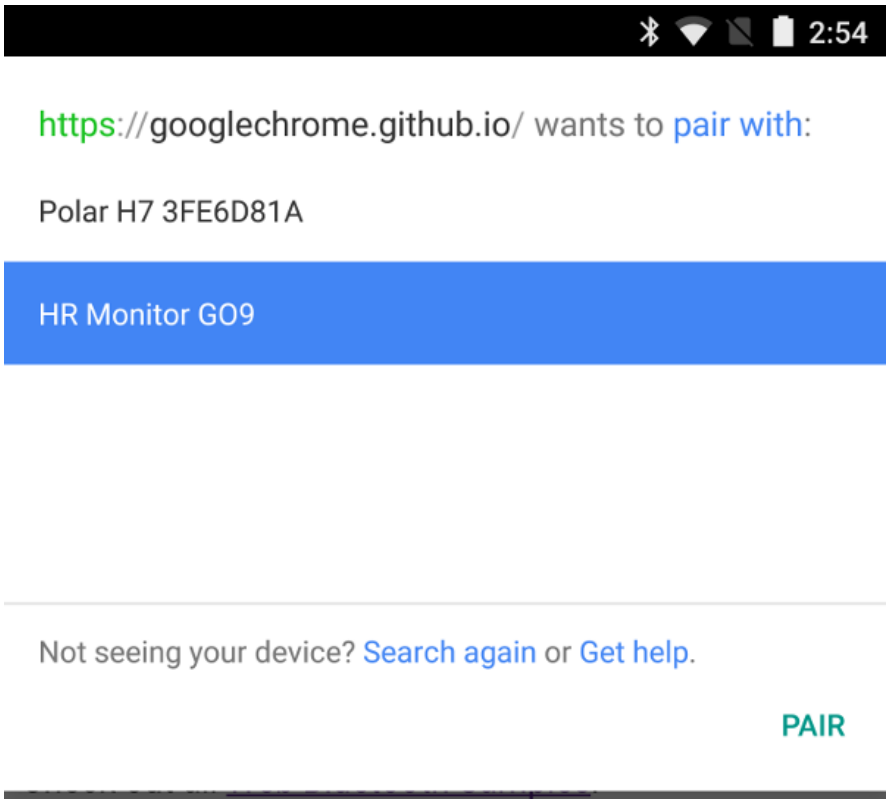


Figure 13: Web Bluetooth API device selection prompt (Yasskin, 2016)

Because the ability to gain physical access to the Bluetooth interface of a device introduces several security issues, the community group responsible for maintaining the specification is constantly assessing the risk involved. As it turns out, the advantage of reducing the number of native applications installed on the device is a major aspect, which with additional security measures implemented, outweighs the potential risks involved (Yasskin, 2016).

Web Bluetooth API is currently limited to Chrome and Opera web browser. It also introduces several operating system requirements (Web Bluetooth Community Group, 2017b):

- Android: Requires Android 6.0 Marshmallow or later.
- Mac: Requires OS X Yosemite or later.
- Linux: Requires Kernel 3.19+ and BlueZ 5.41+ installed.
- Windows: Requires Windows 8.1 or later.

It does not currently support Apple's iOS operating system (Web Bluetooth Community Group, 2017b).

2.8 Conclusion

With the wealth of the researched knowledge, especially with the lack of standards for fitness wearables in Bluetooth Low Energy specification and the number of existing open source projects targeting compatibility with selected devices, the foundation of the project has been partially assured. It is possible to create a model for a vendor-neutral software application for fitness wearables. However, the abstract interface would require individual adapters being built for each supported device. As manufacturers very rarely publish a detailed technical specification for the released devices, each fitness tracker would require a considerable amount of time spent on reverse-engineering the protocol before being integrated into the system. This situation could be easily avoided by the introduction of a standard for the data collection of fitness-related devices in the Bluetooth Low Energy specification, however, with the release of Bluetooth 5.0, this does not seem to be planned in the Bluetooth SIG organisation.

3 Requirements Analysis

3.1 Introduction

To be able to fully understand each requirement of the Vendor-Neutral Software Application for Fitness Wearables, a technique called the MoSCoW analysis will be used. It is a prioritisation tool that is used to compare the importance of various requirements in the system (Miranda, 2011). While it is mostly used in the business communication with external clients to understand the project grounds best, it was very useful in defining the requirements of this project, due to several unknown challenges at the start of the project.

The MoSCoW ratings are as follows (Miranda, 2011):

- Must have: this is the Minimum Usable Subset of requirements which the project guarantees to deliver.
- Should Have: this rating specifies important but not vital requirements of the project. Failing to meet these criteria is tolerable but may have an impact on the results of the project
- Could have: this rating describes the desirable requirements in the project, but unnecessary to complete the project.
- Would have: this is a rating for requirements already known to be not worth to be implemented at this stage of the project

The requirements in this project and their MoSCoW analysis ratings are as follows:

Requirement	**MoSCoW rating**
The Software Architecture design allows using fitness trackers from multiple vendors	Must have
The Software Architecture design allows adding new fitness trackers over time	Must have
The web application allows connecting to a single fitness tracker	Should have

The web application allows reading basic information from a single fitness tracker (name, Bluetooth mac address, etc.)	Should have
The web application allows retrieving a step count data from a single fitness tracker	Should have
The web application works on different operating systems supporting Web Bluetooth API	Could have
The web application allows reading advanced information from a single fitness tracker (battery level, firmware version, etc.)	Could have
The web application allows using a heart rate monitor in the selected fitness tracker (if present)	Could have
The web application allows setting user details for fitness calculations such as age, gender, weight and height	Could have
The web application can calculate fitness statistics such as the number of calories burn; the distance travelled, etc.	Could have
The web application can store historical data	Could have
The web application allows connecting to multiple fitness trackers from different vendors	Would have

3.2 Programming environment

3.2.1 JavaScript

Over the last few years, JavaScript programming language has gained a lot of momentum, both on the client and server side. The popularity of cloud-based applications has attracted much attention in the industry, which resulted in many improvements in the language as well as new functionality being added to the core specification.

As the aim of this project is to create a web-based solution utilising modern technologies, JavaScript will be the language of choice for this project.

3.2.2 Web Bluetooth API

With the addition of Web Bluetooth API specification, the ability to connect to a local Bluetooth device is possible. This interface will be used as a method of communication between the fitness tracker and a host device.

3.2.3 Text Editor

The text editor of choice in this project will be the WebStorm, as it provides great support for JavaScript and is kept up-to-date with latest additions to the language. It also offers a built-in web server which will make the development process easier.

3.2.4 Source Control

Git will be used as a revision control system for the implementation of the web application. To make sure the implemented code is stored safely, a private remote repository is kept on the Bitbucket hosting service and updated using the SourceTree GUI.

3.3 Hardware

This software application will rely heavily on the choice of hardware, as every fitness tracker offer a different set of Bluetooth services. As the “should haves” described in section 3.1 of the project specify support of a single fitness tracker, while the “would have” define support for multiple devices, single activity tracker will be selected in the process. Additionally, as the list of supported operating systems in the Web Bluetooth API is limited to UNIX-based systems, a very specific host device will be chosen for the development of the project.

The project will use the following hardware

- Xiaomi Mi Band Pulse fitness tracker
- Retina MacBook Pro (mid-2014 model)

3.3.1 Xiaomi Mi Band Pulse

Xiaomi Mi Band Pulse, also known as the 1S, is a very similar device to the original Mi Band discussed in the section 2.2.3.2 of this project, with the exemption of having an additional heart rate monitor added. Full device specification can be found below:

	Xiaomi Mi Band Pulse
Sensors	Accelerometer, heart rate monitor, vibration motor
Connectivity	Bluetooth 4.0
Display	White LED indicator lights
Memory	Data not available
IP rating	IP67 (dust and splash proof)
Activities tracked	Walking, running, cycling, sleeping
Price	$15 (about £12)

Table 3: Xiaomi Mi Band Pulse specification (Xiaomi, 2017)

Figure 14: Xiaomi Mi Band Pulse fitness tracker (Xiaomi, 2017)

4 System Design

4.1 Introduction

The previous chapter on the Requirements Analysis specified the basic requirements which the system must complete to be successful. The diagrams in the following section of the report will help to illustrate the software architectural design of the system.

4.2 Ontology of a Fitness Tracker

To be able to represent a fitness tracker in an abstract way, an open source tool called Protégé was used to design the ontology in the OWL standard.

Below is the final visualised version of the ontology of a fitness tracker.

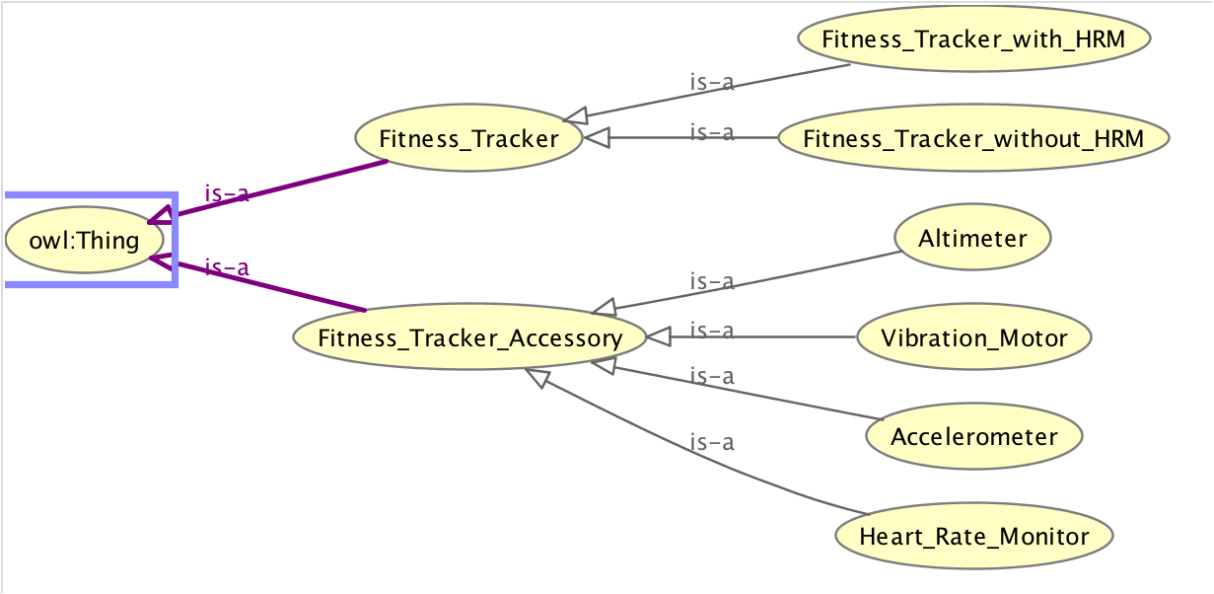


Figure 15: Ontology of a Fitness Tracker

Based on the research section of the Literature and Technology review, a consumer-level fitness tracker was defined as a device that utilises the accelerometer and the vibration motor as its base. Additionally, it can have a heart rate monitor and an altimeter to measure extra fitness statistics. Those parts were defined as an accessory of a fitness tracker as seen in Figure 15

.

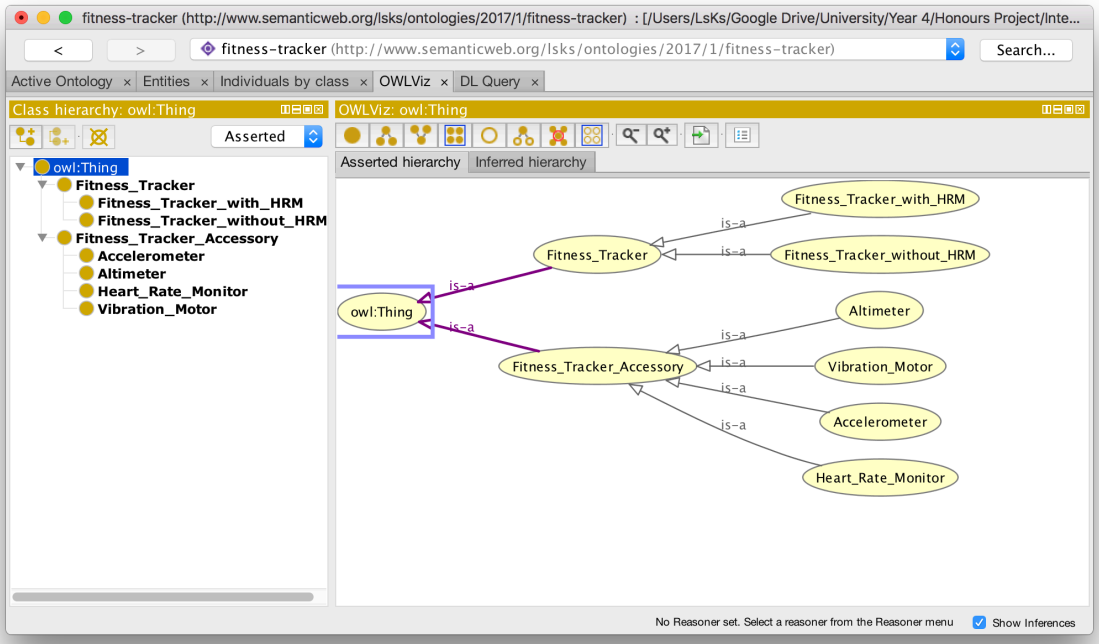


Figure 16: Visualised ontology of a Fitness Tracker using the Protégé program

The base *Fitness_Tracker* entity was defined as a superclass of the *Fitness_Tracker_wih_HRM* and *Fitness_Tracker_without_HRM* to identify whether a fitness tracker is equipped with a heart rate monitor.

Fitness_Tracker_without_HRM was defined as an object that must have an accelerometer and vibration motor with an optional altimeter as seen in Figure 17. Many existing fitness tracker devices were identified to fall into this category.

Similarly, *Fitness_Tracker_with_HRM* was defined as an object that requires an accelerometer, vibration motor and heart rate monitor, with the addition of altimeter. The design of this model can be seen in Figure 18.

What is more, existing fitness trackers were identified in the list of individuals and categorised based on the offered functionality (Fig. 19). Basic data properties were also described as potential data sets offered by each fitness tracker (Fig. 20).

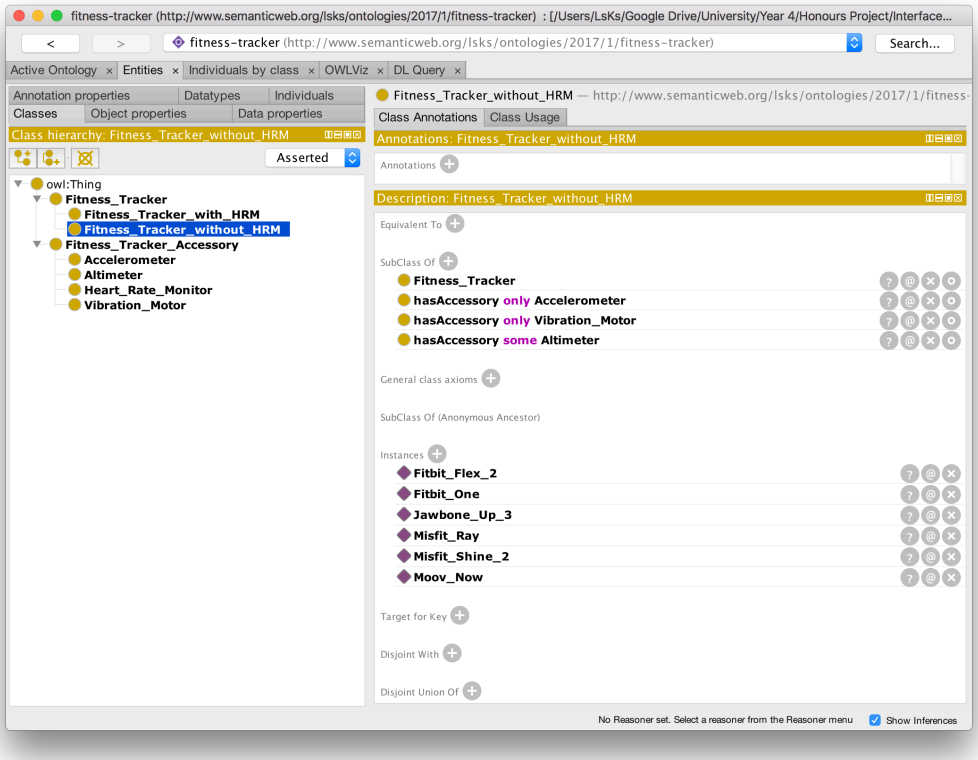


Figure 17: Fitness Tracker class without heart rate monitor

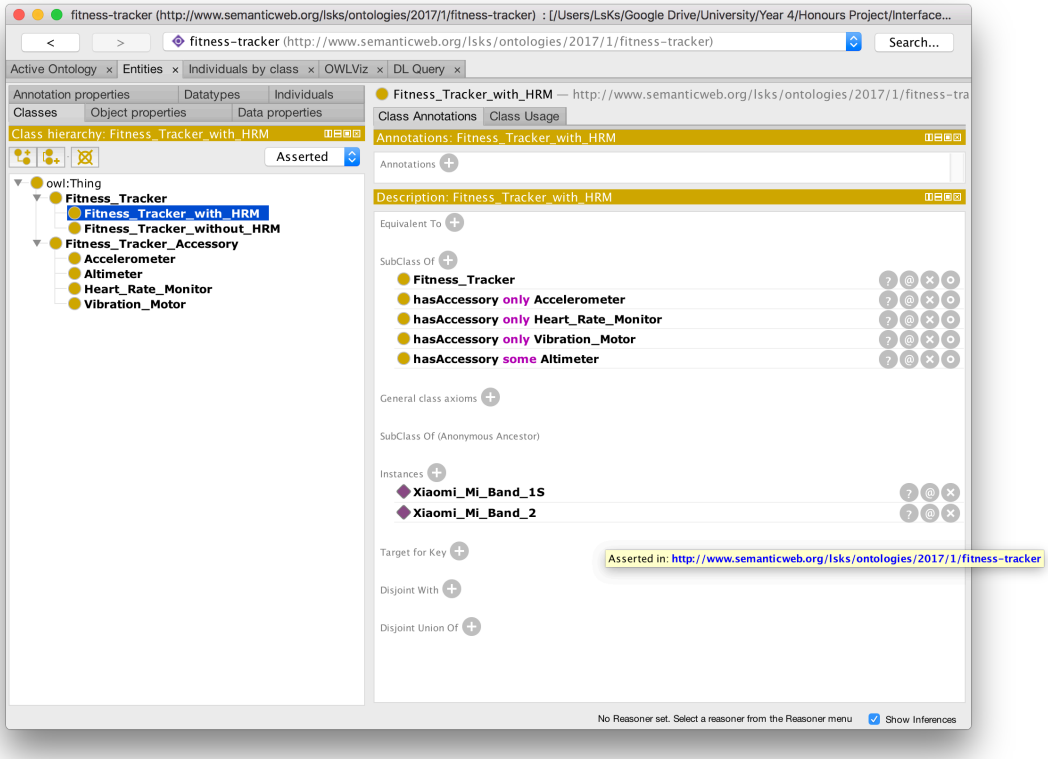


Figure 18: Fitness Tracker class with heart rate monitor

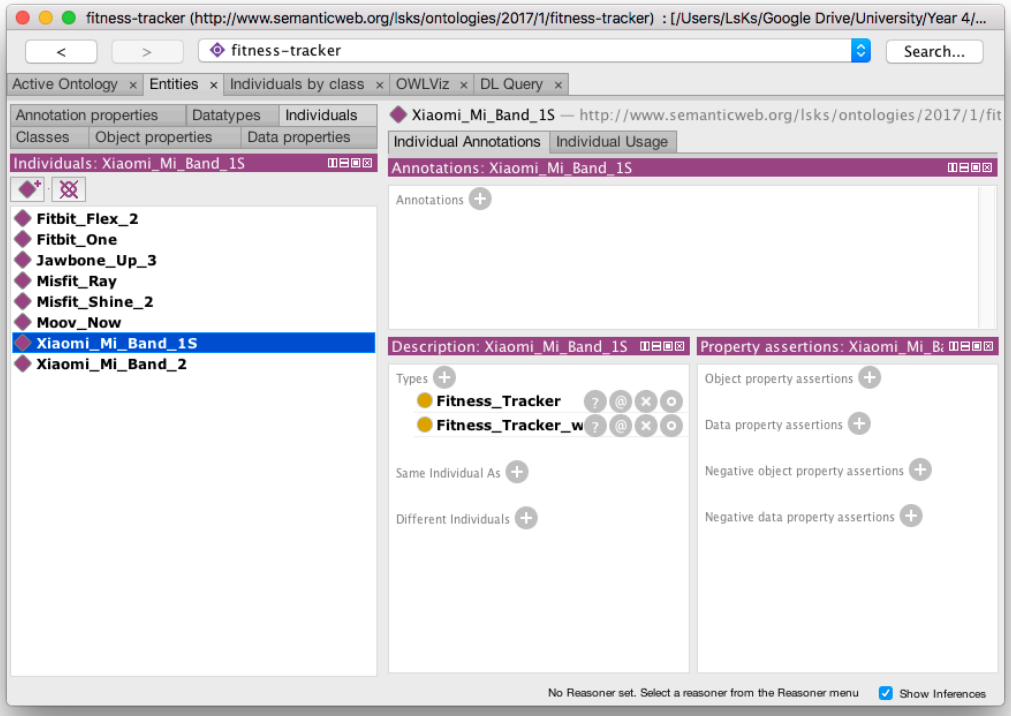


Figure 19: List of individual fitness trackers categorised in the ontology

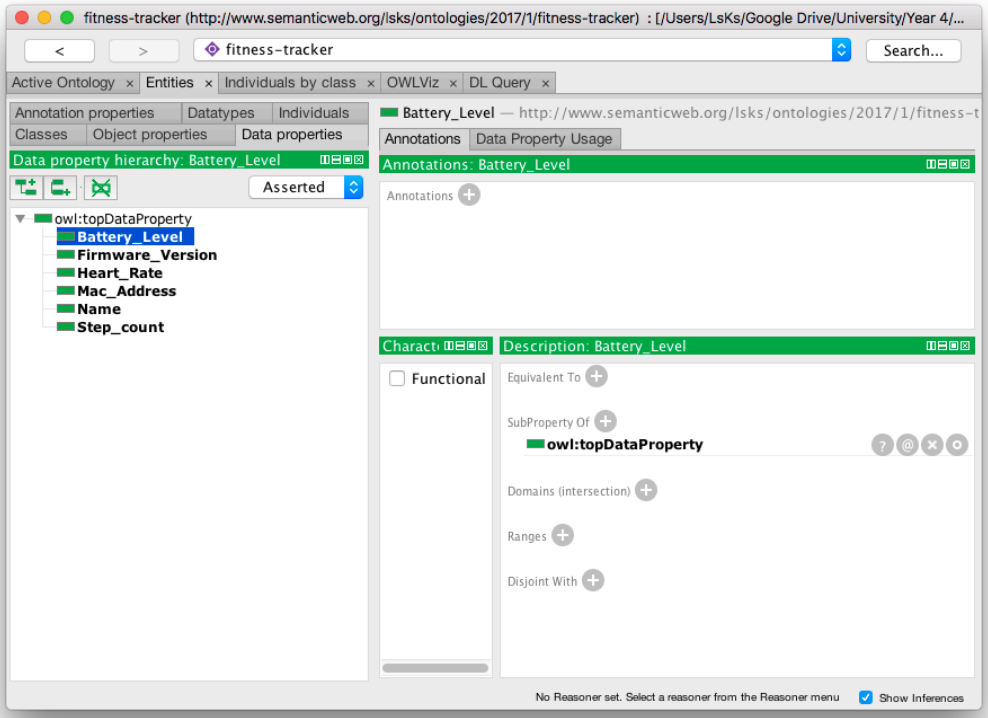


Figure 20: Theoretical data properties offered by a fitness tracker

4.3 Software Application Model

As the aim of the application is to create a vendor-neutral web application for fitness wearables that provides support to multiple different fitness trackers, the software application model has to be designed with a certain degree of flexibility. Because JavaScript is a hybrid between object-oriented and functional programming language, it does not provide standard ways of building relations between object. As a result, there are no specific ways of implementing an interface inside an object, apart from declaring it as a function and providing sufficient documentation for it. Unfortunately, these are the drawbacks of dynamically typed languages. To test if an object contains the function of given name, a process called “duck typing” is used, which checks whether a function exists in an object before calling it. The project assumes that each supported fitness tracker in the application will implement a subset of the described interface (Fig. 22), like functions with specified name.

The software application model divides every part of the functionality into separate interfaces, to make sure that each fitness tracker can be supported. This way the activity tracker object will export only the implemented functionality. After connecting to a device, the application will then test the availability of described functions (Fig. 21) and dynamically adjust the offered functionality in the user interface. It will also attempt to synchronise with the fitness tracker providing basic information about the device.

4.3.1 Activity Diagram

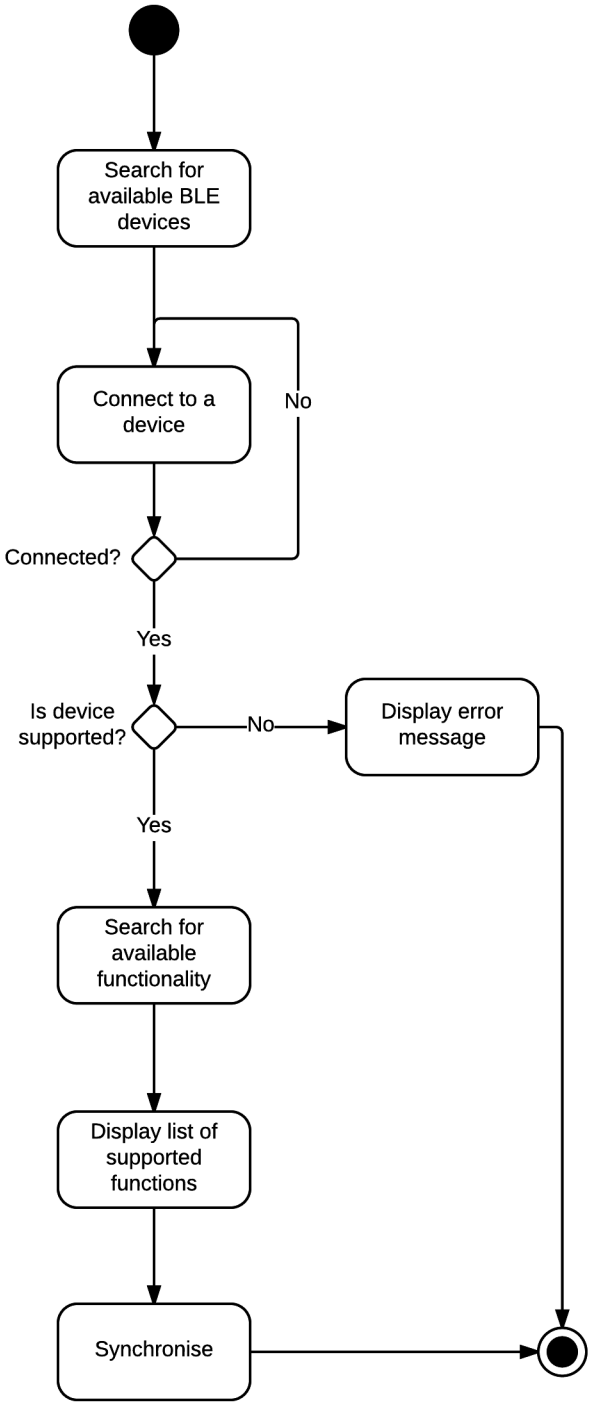


Figure 21: Activity diagram

4.3.2 Class Diagram

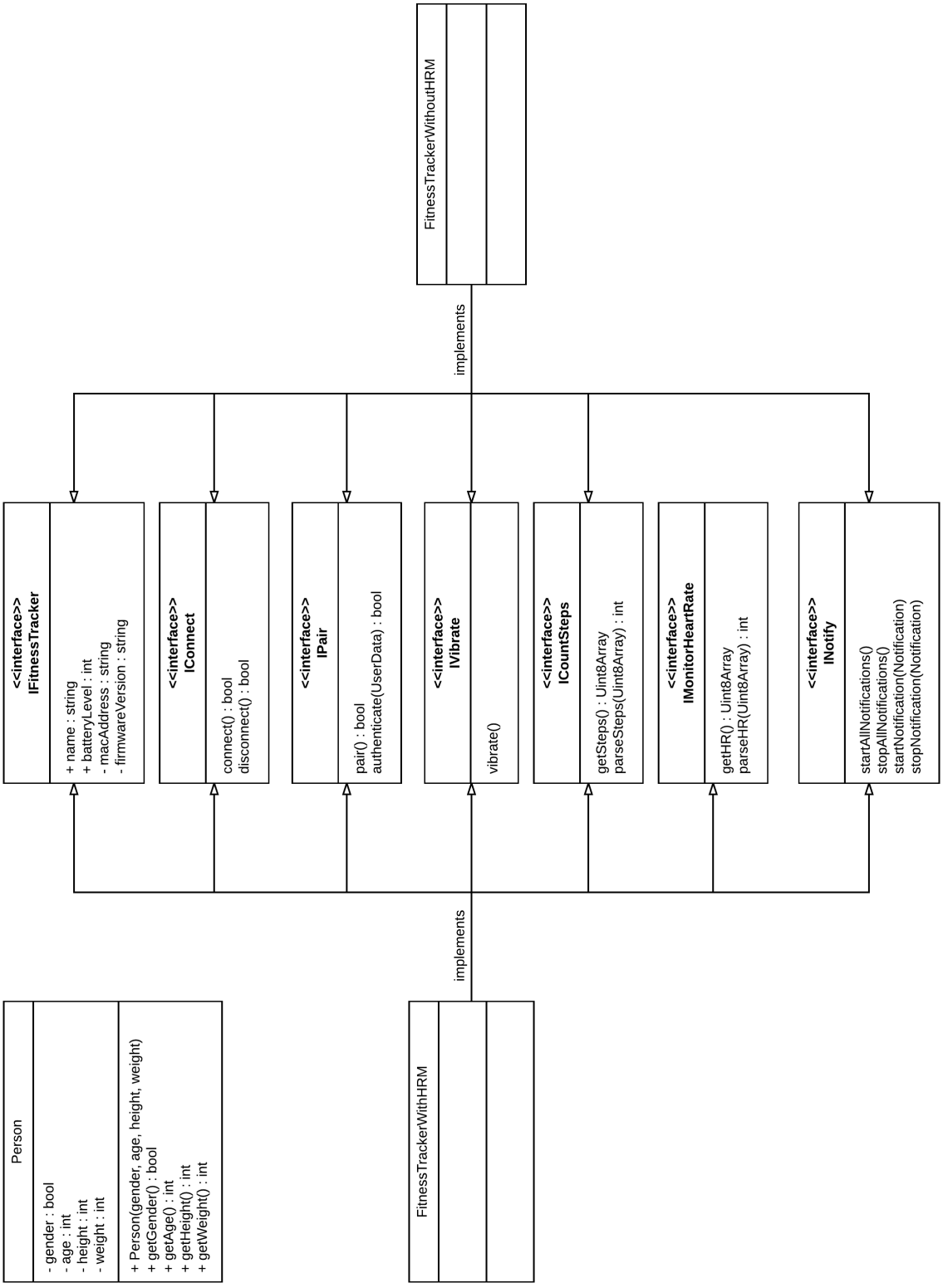


Figure 22: Class diagram

5 Implementation

5.1 Bluetooth Service Discovery

Implementation started from attempting to reverse-engineer the set of services offered by the Mi Band 1S fitness tracker. To do so, a tool called nRF Connect (Nordic Semiconductor ASA, 2017) was used to discover the advertised Bluetooth services.

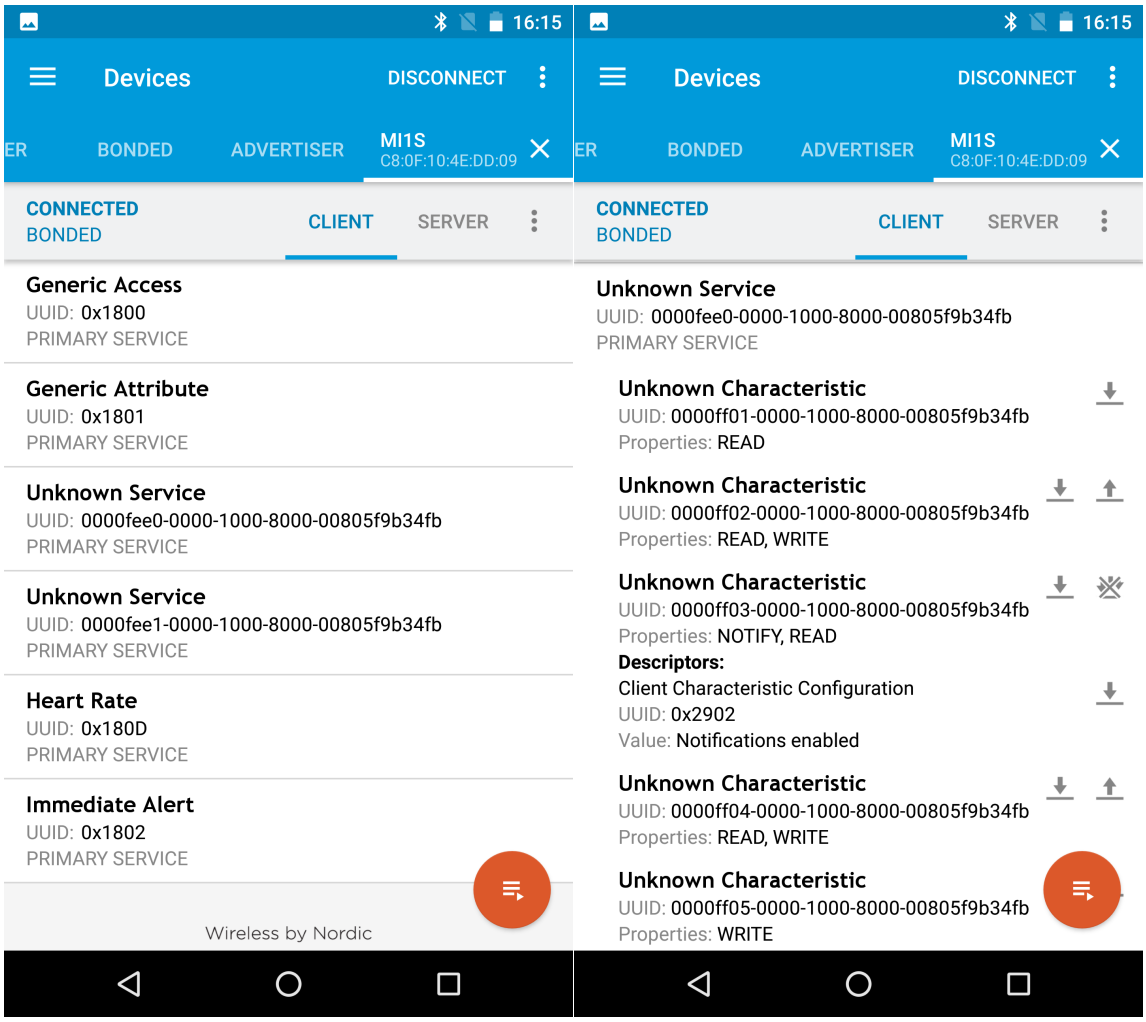


Figure 23: nRF Connect used to discover Bluetooth Services of Mi Band 1S

As it turned out, Xiaomi has implemented standardised Bluetooth services for general information about the device (UUID: 0x1800 and 0x1801), vibration (UUID: 0x1802) and partially heart rate (UUID: 0x180D), however, it was not responding to the requests. All fitness-related functionality was implemented in the two unknown services with long UUIDs.

With the help of one of the open source project discussed in the literature review, Gadgetbridge, there was no need to attempt the lengthy process of reverse engineering the protocol. Gadgetbridge provides a clear description of each service and characteristic offered by the Mi Band tracker (Fig. 24).

```
00001800-0000-1000-8000-00805f9b34fb:Generic Access Service
- 00002a00-0000-1000-8000-00805f9b34fb:Device Name
- 00002a01-0000-1000-8000-00805f9b34fb:Appearance
- 00002a02-0000-1000-8000-00805f9b34fb:Peripheral Privacy Flag
- 00002a04-0000-1000-8000-00805f9b34fb:Peripheral Preferred Connection Parameters

00001801-0000-1000-8000-00805f9b34fb:Generic Attribute Service
- 00002a05-0000-1000-8000-00805f9b34fb:Service Changed

0000fee0-0000-1000-8000-00805f9b34fb:MiBand Service
- 0000ff01-0000-1000-8000-00805f9b34fb:Device Info
- 0000ff02-0000-1000-8000-00805f9b34fb:Device Name
- 0000ff03-0000-1000-8000-00805f9b34fb:Notification
- 0000ff04-0000-1000-8000-00805f9b34fb:User Info
- 0000ff05-0000-1000-8000-00805f9b34fb:Control Point
- 0000ff06-0000-1000-8000-00805f9b34fb:Realtime Steps
- 0000ff07-0000-1000-8000-00805f9b34fb:Activity Data
- 0000ff08-0000-1000-8000-00805f9b34fb:Firmware Data
- 0000ff09-0000-1000-8000-00805f9b34fb:LE Params
- 0000ff0a-0000-1000-8000-00805f9b34fb:Date/Time
- 0000ff0b-0000-1000-8000-00805f9b34fb:Statistics
- 0000ff0c-0000-1000-8000-00805f9b34fb:Battery
- 0000ff0d-0000-1000-8000-00805f9b34fb:Test
- 0000ff0e-0000-1000-8000-00805f9b34fb:Sensor Data

0000fee1-0000-1000-8000-00805f9b34fb:Unknown service
- 0000fedd-0000-1000-8000-00805f9b34fb:Unknown characteristic
- 0000fede-0000-1000-8000-00805f9b34fb:Unknown characteristic
- 0000fedf-0000-1000-8000-00805f9b34fb:Unknown characteristic

0000fee7-0000-1000-8000-00805f9b34fb:Unknown service
- 0000fec7-0000-1000-8000-00805f9b34fb:Unknown characteristic
- 0000fec8-0000-1000-8000-00805f9b34fb:Unknown characteristic
- 0000fec9-0000-1000-8000-00805f9b34fb:Unknown characteristic
```

Figure 24: Reverse-engineered interface of a Mi Band 1S (Freeyourgadget, 2017)

This information was used and rewritten in JavaScript using ES6 *const* keywords to define the constants of the Bluetooth interface (Fig. 25).

```
/* Custom Bluetooth Service UUIDs */

  const MIBAND_SERVICE_UUID = 0xFEE0;
  const IOS_SERVICE_UUID = 0xFEE7;

  /* Custom Bluetooth Characteristic UUIDs */

  const BATTERY_INFO_UUID = 0xFF0C;
  const BLE_CONNECTION_PARAMETERS_UUID = 0xFF09;
  const CONTROL_POINT_UUID = 0xFF05;
  const DATETIME_UUID = 0xFF0A;
  const DEVICE_INFO_UUID = 0xFF01;
  const DEVICE_NAME_UUID = 0xFF02;
  const NOTIFICATIONS_UUID = 0xFF03;
  const STEPS_UUID = 0xFF06;
  const HRM_UUID = 0x2A39;
  const ACTIVITY_DATA_UUID = 0xFF07;
  const USER_INFO_UUID = 0xFF04;
  const MAC_ADDRESS_UUID = 0xFEC9;
```

Figure 25: Mi Band 1S Bluetooth interface in JavaScript

This has enabled to implement basic functionality such as connecting and disconnecting from the device (Fig. 27). After successfully connecting to the device, the application will scan available Bluetooth characteristics and cache them in internal store (Fig. 26).

As a starter project for Web Bluetooth API, an open source boilerplate project was used created by one of the developers of Google Chrome (Beaufort, 2017).

```
  /* Utils */
  _cacheCharacteristic(service, characteristicUuid) {
    return service.getCharacteristic(characteristicUuid)
      .then(characteristic => {
        this._characteristics.set(characteristicUuid, characteristic);
      });
  }
```

Figure 26: JavaScript function for storing Bluetooth characteristics

```
class MiBand {
    constructor() {
      this.device = null;
      this.server = null;
      this._characteristics = new Map();
      this._debug = true;
    }
    connect() {
      let options = {
        filters: [{
          name: 'MI1S'
        }],
        optionalServices: [MIBAND_SERVICE_UUID]
      };
      return navigator.bluetooth.requestDevice(options)
        .then(device => {
          this.device = device;
          return device.gatt.connect();
        })
        .then(server => {
          this.server = server;
          return Promise.all([
            server.getPrimaryService(MIBAND_SERVICE_UUID).then(service => {
              return Promise.all([
                this._cacheCharacteristic(service, BATTERY_INFO_UUID),
                this._cacheCharacteristic(service, BLE_CONNECTION_PARAMETERS_UUID),
                this._cacheCharacteristic(service, CONTROL_POINT_UUID),
                this._cacheCharacteristic(service, DATETIME_UUID),
                this._cacheCharacteristic(service, DEVICE_INFO_UUID),
                this._cacheCharacteristic(service, DEVICE_NAME_UUID),
                this._cacheCharacteristic(service, NOTIFICATIONS_UUID),
                this._cacheCharacteristic(service, STEPS_UUID),
                this._cacheCharacteristic(service, ACTIVITY_DATA_UUID),
                this._cacheCharacteristic(service, USER_INFO_UUID),
                this._cacheCharacteristic(service, MAC_ADDRESS_UUID),
              ])
            })
          ]);
        })
    }
    disconnect() {
      if (this.device) {
        this.device.gatt.disconnect();
        console.log(`Disconnected device: ${device.name}`);
      }
    }
```

Figure 27: JavaScript code used to establish connection with the fitness band

5.2 Pairing problems

Unfortunately, the fitness tracker kept disconnecting after several seconds for no reason. It was also not responding to any other request. As it turned out after a couple of weeks of development, straight after establishing a successful connection with the central device, the fitness tracker waits until it receives an authentication packet that consists of an array of bytes in a specific format. Some indexes contain device specific bytecode, while others require user profile details such as the gender, age, weight and height to be specified. Again, with the help of Gadgetbridge application, the following Java code (Fig. 28) could be rewritten into JavaScript (Fig. 29).

```
public byte[] getData(DeviceInfo mDeviceInfo) {
    byte[] sequence = new byte[20];
    int uid = calculateUidFrom(alias);

    sequence[0] = (byte) uid;
    sequence[1] = (byte) (uid >>> 8);
    sequence[2] = (byte) (uid >>> 16);
    sequence[3] = (byte) (uid >>> 24);

    sequence[4] = (byte) (gender & 0xff);
    sequence[5] = (byte) (age & 0xff);
    sequence[6] = (byte) (height & 0xff);
    sequence[7] = (byte) (weight & 0xff);
    sequence[8] = (byte) (type & 0xff);

    int aliasFrom = 9;
    if (!mDeviceInfo.isMili1()) {
        sequence[9] = (byte) (mDeviceInfo.feature & 255);
        sequence[10] = (byte) (mDeviceInfo.appearance & 255);
        aliasFrom = 11;
    }

    byte[] aliasBytes = alias.substring(0, Math.min(alias.length(), 19 -
aliasFrom)).getBytes();
    System.arraycopy(aliasBytes, 0, sequence, aliasFrom, aliasBytes.length);

    byte[] crcSequence = Arrays.copyOf(sequence, 19);
    sequence[19] = (byte) ((CheckSums.getCRC8(crcSequence) ^
Integer.parseInt(this.btAddress.substring(this.btAddress.length() - 2), 16)) &
0xff);

    return sequence;
}
```

Figure 28: Java code used to authenticate with the Mi Band 1S (Freeyourgadget, 2017)

```
    setUserInfo(reset) {
      return this.getMacAddress().then(macAddress => {
        let uuid = 1550050550; // UUID must have 10 digits.
        let gender = 1; // Gender (Female 0, Male 1)
        let age = 22; // Age in years.
        let height = 188; // Height in cm.
        let weight = 85; //2 Weight in kg.
        let type = reset ? 1 : 0; // If 1, all saved data will be lost.

        let userInfo = [];
        for (let i = 0; i < 4; i++) {
          userInfo.push(uuid & 0xff);
          uuid >>= 8;
        }
        userInfo.push(gender);
        userInfo.push(age);
        userInfo.push(height);
        userInfo.push(weight);
        userInfo.push(type);
        for (let i = 0; i < 10; i++) { /* Alias */
          userInfo.push(0);
        }

        // taken from Gadgetbridge
        userInfo[9] = 0x4; // mDeviceInfo.feature TODO: code it
        userInfo[10] = (0x0 && 0xff); // mDeviceInfo.appearance TODO: code it

        let crc = ((this._computeCRC(userInfo) ^ parseInt(macAddress.slice(-2),
16)) & 0xff);
        userInfo.push(crc);

        return this._writeCharacteristicValue(USER_INFO_UUID, new
Uint8Array(userInfo));
      });
    }
```

Figure 29: JavaScript code used to authenticate with Mi Band 1S

5.3 Step Count

After overcoming the initial difficulties of getting the fitness tracker to maintain a connection, reading the step count of the device was as easy as requesting the correct Bluetooth characteristic and parsing the output to a readable format (Fig. 30).

```
    getSteps() {
      return this._readCharacteristicValue(STEPS_UUID)
        .then(data => {
          return this.parseSteps(data);
        });
    }
    parseSteps(data) {
      return data.getUint8(0) | (data.getUint8(1) << 8) |
        (data.getUint8(2) << 16) | (data.getUint8(3) << 24);
    }
```

Figure 30: Reading the step count data from the fitness tracker using JavaScript

Several other Bluetooth services were also implemented, such as reading the device battery level, a number of charges, firmware version and status code (Fig. 31).

```
    getDeviceInfo() {
      return this._readCharacteristicValue(DEVICE_INFO_UUID)
        .then(data => {
          let deviceInfo = new Map();
          deviceInfo.set('firmwareVersion', data.getUint8(15) + '.' +
data.getUint8(14) + '.' + data.getUint8(13) + '.' + data.getUint8(12));
          deviceInfo.set('profileVersion', data.getUint8(11) + '.' +
data.getUint8(10) + '.' + data.getUint8(9) + '.' + data.getUint8(8));
          return deviceInfo;
        });
    }
    getBatteryInfo() {
      return this._readCharacteristicValue(BATTERY_INFO_UUID)
        .then(data => {
          let lastChargeDate = new Date(2000 + data.getUint8(1),
            data.getUint8(2),
            data.getUint8(3),
            data.getUint8(4),
            data.getUint8(5),
            data.getUint8(6));
          let batteryInfo = new Map();
          batteryInfo.set('batteryLevel', data.getUint8(0));
          batteryInfo.set('batteryStatusCode', data.getUint8(9));
          batteryInfo.set('batteryStatusText',
batteryStatusCodes.get(data.getUint8(9)));
          batteryInfo.set('batteryCharges', 0xffff & (0xff & data.getUint8(7) |
(0xff & data.getUint8(8) << 8)));
          batteryInfo.set('batteryLastCharge', lastChargeDate);
          return batteryInfo;
        });
    }
```

Figure 31: Reading additional data from the fitness tracker

The very last implemented functionality utilised the JavaScript's event based system to allow streaming the step count data from the device in real time (Fig. 32).

```
    startNotifications() {
      return this._startNotifications(NOTIFICATIONS_UUID);
    }
    stopNotifications() {
      return this._stopNotifications(NOTIFICATIONS_UUID);
    }
    startNotificationsSteps() {
      return this._startNotifications(NOTIFICATIONS_UUID)
        .then(() => {
          return this._startNotifications(ACTIVITY_DATA_UUID);
        })
        .then(() => {
          return this._startNotifications(STEPS_UUID);
        });
    }
    stopNotificationsSteps() {
      return this._stopNotifications(NOTIFICATIONS_UUID)
        .then(this._stopNotifications(ACTIVITY_DATA_UUID))
        .then(this._stopNotifications(STEPS_UUID));
    }

function handleSteps(stepsCharacteristic) {
  stepsCharacteristic.addEventListener('characteristicvaluechanged', event => {
    // […]
    // Some code was omitted here
  });
}
```

Figure 32: Live streaming data using JavaScript's notification system

5.4 Final design

The final design of the application was very influenced by choice of supporting a single fitness tracker only. The user interface allows to connect to the fitness tracker using the "GET" button in Figure 33; then the user has to select a device from a drop-down list (Fig. 34). After establishing a successful connection, the screen will transition to the dashboard showing important information about the device, including current step count data and battery level (Fig. 35) and will start live streaming the step tracking using notification system (Fig. 36). All Bluetooth transactions are logged to the browser's console to enable easier debugging (Fig. 37).

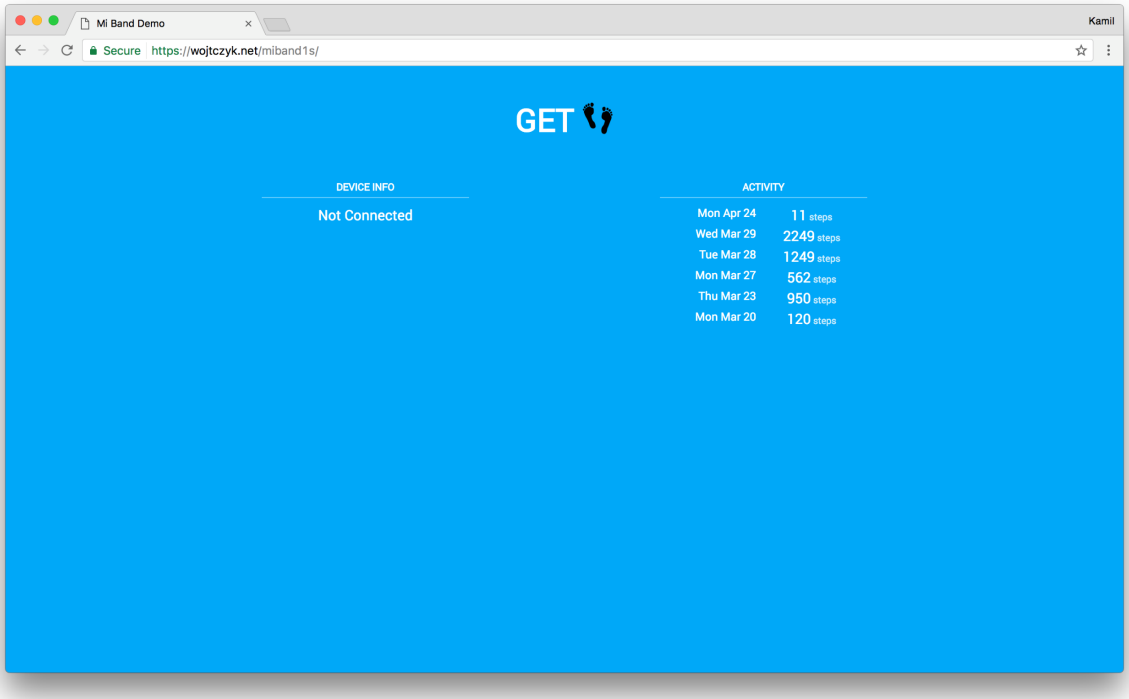


Figure 33: Interface of the implemented JavaScript application

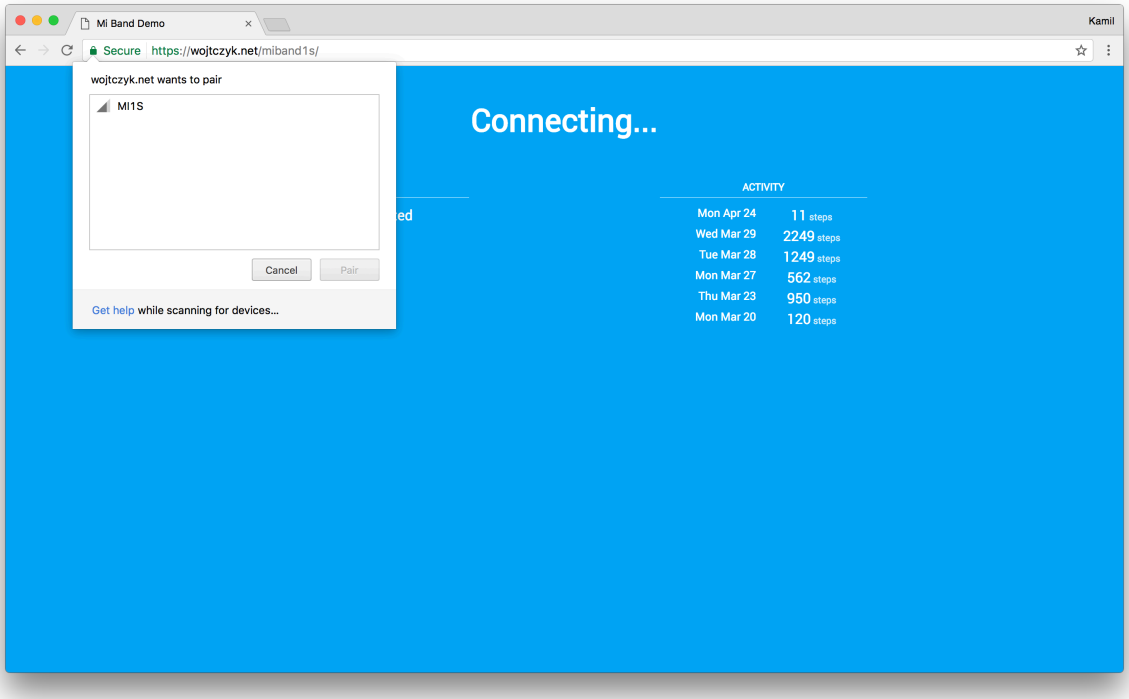


Figure 34: Connecting to the fitness tracker

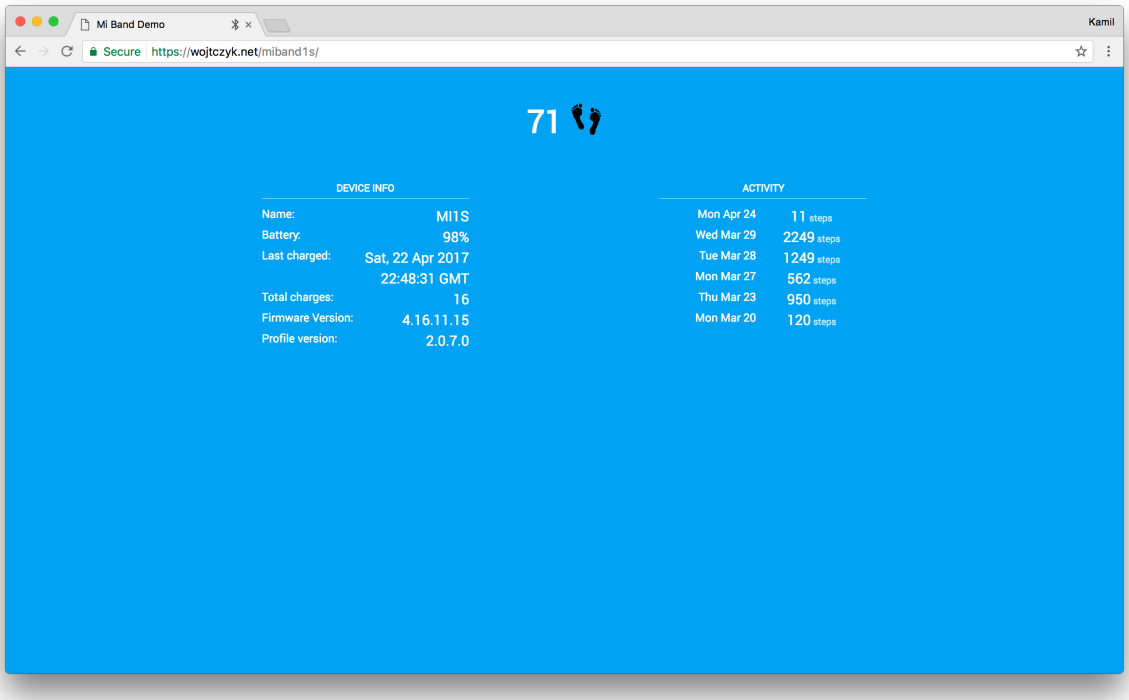


Figure 35: Main dashboard of the application

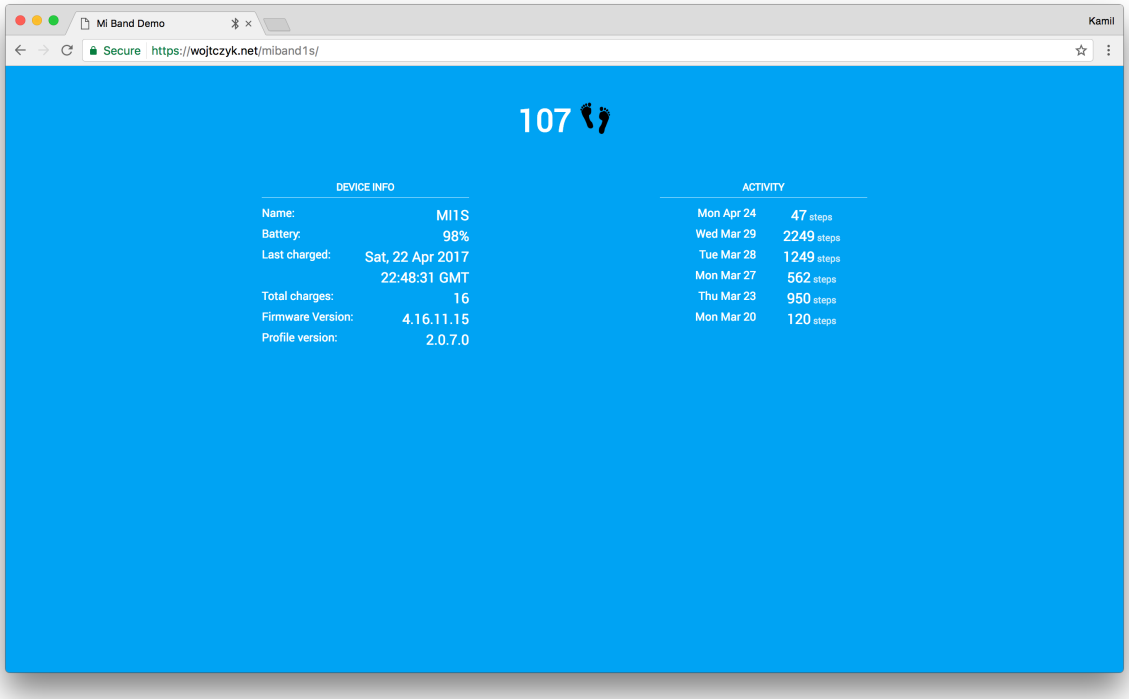


Figure 36: Increasing step count when live streaming the data

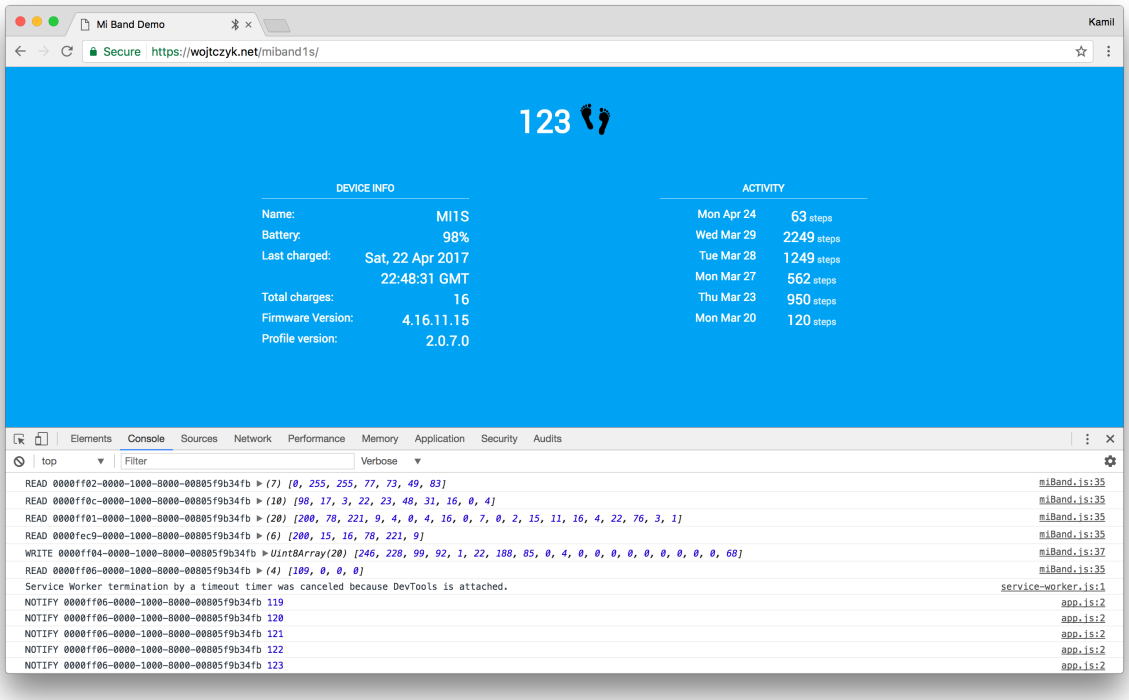


Figure 37: Debugging console of the JavaScript application

6 Evaluation

6.1 Fulfilment of requirements

One way to evaluate the outcomes of the project is to compare it with the technical requirements specified in section 3.1.

Requirement	**MoSCoW rating**	**Completion status**
The Software Architecture design allows using fitness trackers from multiple vendors	Must have	Completed
The Software Architecture design allows adding new fitness trackers over time	Must have	Completed
The web application allows connecting to a single fitness tracker	Should have	Completed
The web application allows reading basic information. from a single fitness tracker (name, Bluetooth mac address, etc.)	Should have	Completed
The web application allows retrieving a step count data from a single fitness tracker	Should have	Completed
The web application works on different operating systems supporting Web Bluetooth API	Could have	Completed
The web application allows reading advanced information from a single fitness tracker (battery level, firmware version, etc.)	Could have	Completed
The web application allows using a heart rate monitor in the selected fitness tracker (if present)	Could have	Not completed

The web application allows setting user details for fitness calculations such as age, gender, weight and height	Could have	Not completed
The web application can calculate fitness statistics such as the number of calories burn; the distance travelled, etc.	Could have	Not completed
The web application can store historical data	Could have	Partially completed
The web application allows connecting to multiple fitness trackers from different vendors	Would have	Not completed

Table 4: Requirements evaluation

All of the most important requirements rated as “Must have” and “Should have” have been implemented in the project, as well as three less critical “Could have” out of six. The “Would have” requirement has been left out of the scope of the project as the complexity of the task went way beyond the timespan of the project.

6.2 Vendor-neutral approach

The system was designed to allow the use of multiple fitness trackers from different manufacturers. To demonstrate that the system could be extended without modifying the underlying structure, a device from a new manufacturer will be introduced to the project. The entry-level fitness tracker from Jawbone will be used, namely the Jawbone Up Move (Fig. 38). Based on the software application design it will be then determined, whether the tracker could be supported or not.

Jawbone Move Up is an affordable, clip-style activity tracker that costs only £18.95. It is available in five different colours of the clip. The device is splash and dust resistant with an IP67 rating and a very light in construction. The sensors and components include an accelerometer and a vibration motor, as well as a Bluetooth 4.0 radio module. The device has a LED display and a button that enables to interact with it directly. It is designed to track activity, the distance through step count and sleeps quality (Jawbone, 2017).

Figure 38: Jawbone Up Move fitness tracker (Jawbone, 2017)

	Jawbone Up Move
Sensors	Accelerometer
Connectivity	Bluetooth 4.0
Display	LED display, with a button
Memory	Data not available
IP rating	IP67 (dust and splash proof)
Activities tracked	Walking, running, cycling, sleeping
Price	£18.95

Table 5: Jawbone Up Move specification (Jawbone, 2017)

As functionality-wise the fitness tracker is very similar to the Xiaomi Mi Band Pulse used in the project, and it falls into the “Fitness_Tracker_without_HRM” category of the ontology, it is believed that the device would fit in the system without any modification.

The only required aspect would be to develop an object model that implements described functions in the interface.

6.3 Cross-platform support

As the last evaluation method to determine whether the web application supports different host devices and indeed provides a web-based cross-platform support, a different test device will be introduced. All previous tests were run on MacOS operating system, while the new device will be running underneath the Android operating system. The selected test device is the Google Nexus 6p.

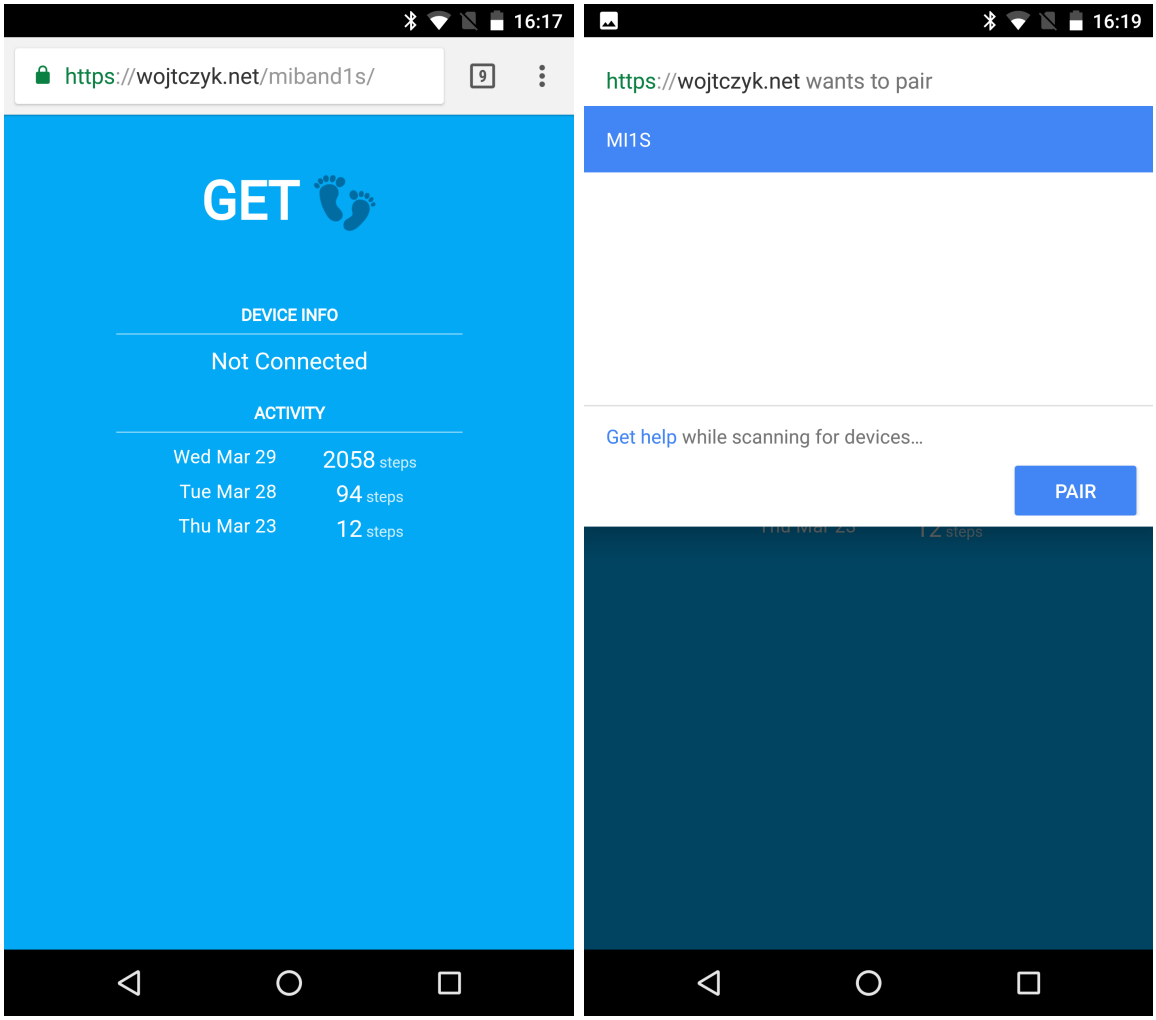


Figure 39: Testing web application on the Google Nexus 6p

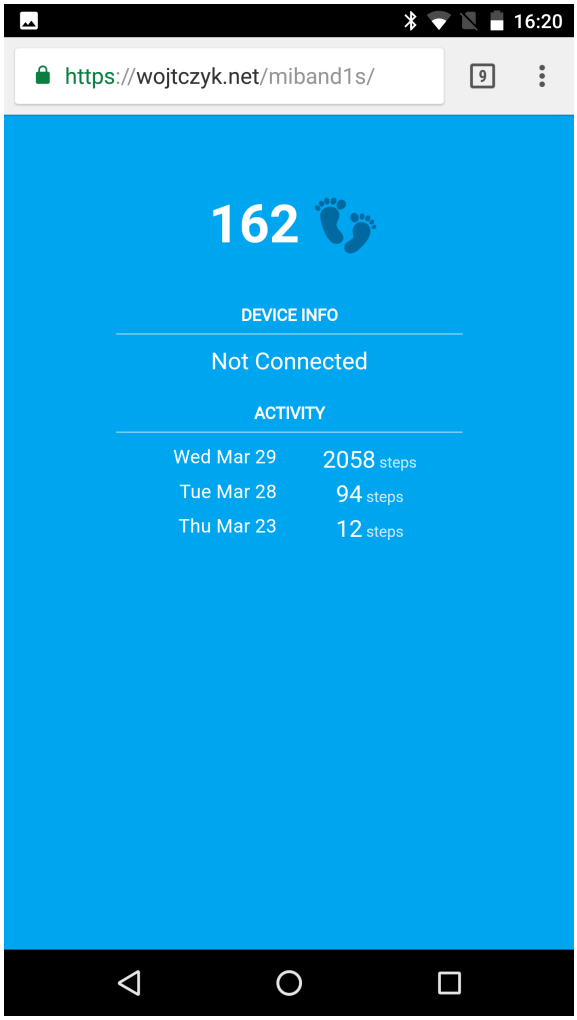


Figure 40: Fitness tracker successfully connected to the Google Nexus 6p

As seen in the figures 39 and 40, the application provides a fully cross-platform support, due to utilising the latest web technologies. There was no need to download additional software to able to support the Xiaomi Mi Band Pulse fitness tracker, as all required code was delivered in the JavaScript format.

7 Conclusions

7.1 Achievement of aims

As discussed in the previous chapters, the implementation of the Vendor-Neutral Software Application for Fitness Wearables reaches the technical goals and initial aims, as it developed a working abstract model of a fitness tracker and the required interface to implement new devices, despite the differences between them. It has provided a working prototype of a web application that utilised modern web technologies, including Web Bluetooth API, and enabled to connect to a selected fitness tracker without the need for manufacturer's software.

In its current version, however, it does not provide support to multiple fitness trackers, as based on the findings from the literature and technology review, there is no standard for such devices in the Bluetooth 4.0 Low Energy specification. As a result, each manufacturer implements their own Bluetooth Services and Characteristics, forcing the implementation of individual adapter for each supported device. Additionally, in the process of building the web application, the graphical user interface was not considered, which would play a viable role in the full implementation of the system.

7.2 Retrospective

The project was extremely challenging and risky, as each aspect of it had to be fully understood before being able to specify the further direction. In the beginning, there was no clear answer whether such a system is possible and specifying clear aims and objectives seemed impossible. As the project went on and certain discoveries were made through research, it started to take shape. The discoveries in the literature review section were shocking, as there is a gap in the current specification of the Bluetooth 4.0 Low Energy. Further research showed that there are many open source project with very similar aims, either focusing on reverse engineering individual interfaces of BLE devices to extend its functionality or building a fully vendor-neutral and cloud-free solutions. It was a really interesting experience to find out that other developers are also interested in the similar field.

From today’s perspective, there are many things that should be done differently in the project. Starting from focusing on a single part of the application – either by providing

a comprehensive design with no implementation attempted, or by extending an existing project in the area, by adding support for a selected fitness tracker. The choice of the new emerging technology, Web Bluetooth API, was an interesting experience from a developer's point of view, but a bit too challenging to begin with and to provide in the honours project. Aside from those points, the ability to work with modern Bluetooth technology was a delight and provided an interesting aspect of the project.

7.3 Further Work

Each component of the project could be individually improved in the future, as the structure of the application is complex and requires much attention in every degree.

The ontology could be improved with more abstraction about the relations between the objects, to provide more realistic semantic description of a fitness tracker. It could also be extended by describing more types of a fitness tracker and the internal accessories used, with suitable examples provided. The ontology could then be validated against W3C specification and published to the public domain.

Most importantly, the software application should be redesigned to match against JavaScript design patterns to fully utilise the language potential. The model should implement a fitness tracker in an abstracted way, allowing more flexibility to support additional devices. The web application should also have a suitable graphical user interface designed, which would standardise the output of different fitness trackers. Moreover, the application should provide support to additional devices, which should be updated over time. Finally, the system should allow to use the gather data in greater extend, by being able to visually it and export it to different formats and external providers.

8 References

Acena, O. (2015). Mibui. Retrieved 22 April 2015, from https://bitbucket.org/OscarAcena/mibui

Acena, O. (2016a). Mibanda. Retrieved 22 April 2017, from https://bitbucket.org/OscarAcena/mibanda

Acena, O. (2016b). Pygattlib. Retrieved 22 April 2017, from https://bitbucket.org/OscarAcena/pygattlib

Allard, B. (2017). Galileo. Retrieved 22 April 2017, from https://bitbucket.org/benallard/galileo/overview

Anonymous. (2016). An Activity Tracker for Monitoring Glucose. *Chemical Engineering Progress*. New York.

Axworthy, J. (2016). The origins of the fitness tracker. Retrieved 16 April 2017, from https://www.wareable.com/fitness-trackers/the-origins-of-the-fitness-tracker-1234

Beaufort, F. (2017). Web Bluetooth sandbox. Retrieved 20 April 2017, from https://github.com/beaufortfrancois/sandbox

Bechhofer, S. (2009). OWL: Web ontology language. In *Encyclopedia of Database Systems* (pp. 2008–2009). Springer.

Bluegiga Technologies. (2011). Classic Bluetooth vs. Bluetooth low energy. Retrieved 20 April 2017, from http://www.mt-system.ru/sites/default/files/docs/documents/bluetooth_le_comparison.pdf

Bluetooth SIG Inc. (2017a). Generic Attributes (GATT) and the Generic Attribute Profile.

Bluetooth SIG Inc. (2017b). Health Device Profile. Retrieved 21 April 2017, from https://www.bluetooth.com/specifications/assigned-numbers/health-device-profile

Cambridge University Press. (n.d.). Meaning of 'fitness tracker' in the English Dictionary. Retrieved 19 April 2017, from http://dictionary.cambridge.org/dictionary/english/fitness-tracker

Curry, A. (2014). Survival of the fittest: using a wildlife version of fitness trackers, biologists can finally measure how much energy animals need to stay alive.(FEATURE: NEWS)(Report). *Nature*, *513*(7517), 157.

Daughtrey, S. C. (1994). Reverse engineering of software for interoperability and analysis. *Vanderbilt Law Review*, *47*(1), 145–187.

de Zambotti, M., Claudatos, S., Inkelis, S., Colrain, I. M., & Baker, F. C. (2015). Evaluation of a consumer fitness-tracking device to assess sleep in adults. *Chronobiology International*, *32*(7), 1024–1028.

European Parliament. (2009). Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs. *Official Journal of the European Union*, *L111*(April), L111/16-L111/22.

Ewalt, D. M. (2010). Getting Fitbit. Retrieved 19 April 2017, from https://www.forbes.com/2010/06/11/fitbit-tracker-pedometer-lifestyle-heatlh-lifetracking.html

Fitbit. (2017a). Fitbit One. Retrieved 20 April 2017, from https://www.fitbit.com/uk/shop/one

Fitbit. (2017b). Subscriptions API. Retrieved 20 April 2017, from https://dev.fitbit.com/docs/subscriptions/

Freeyourgadget. (2017). Gadgetbridge. Retrieved 22 April 2017, from https://github.com/Freeyourgadget/Gadgetbridge

Gallaga, O. L. (2015). Wearables may be hot, but they've got a ways to go. *TCA Regional News*.

Garmin. (2017a). Connect IQ Devices. Retrieved 20 April 2017, from https://developer.garmin.com/connect-iq/compatible-devices/

Garmin. (2017b). Health API. Retrieved 20 April 2017, from https://developer.garmin.com/health-api/overview/

Garmin. (2017c). vívosmart® 3. Retrieved from https://buy.garmin.com/en-GB/GB/p/567813/pn/010-01755-03#specs

Global Fitness Wearable Device Market 2016-2020 - Dominated by Fitbit, Garmin, Xiaomi & Jawbone. (2016). *M2 Communications*.

Gomez, C., Oller, J., & Paradells, J. (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, *12*(9), 11734–11753.

Gruber, T. (2009). Ontology. In *Encyclopedia of Database Systems* (pp. 1963–1965). Springer.

Harris, A., Khanna, V., Tuncay, G., Want, R., & Kravets, R. (2016). Bluetooth Low Energy in Dense IoT Environments. *Ieee Communications Magazine*, *54*(12), 30–36.

Jawbone. (2017). Jawbone Up Move.

LaRoche, P., & Cox, A. (2004). Combined software and hardware comprehension in reverse engineering. *11th Working Conference on Reverse Engineering*. USA. https://doi.org/10.1109/WCRE.2004.16

Machulis, K. (2011). Libfitbit. Retrieved 22 April 2017, from https://github.com/openyou/libfitbit

Miller, A. (2013). Fitness trackers. *XRDS: Crossroads, The ACM Magazine for Students*, *20*(2), 24–26.

Miranda, E. (2011). Time boxing planning: buffered moscow rules. *ACM SIGSOFT Software Engineering Notes*, *36*(6), 1–5. https://doi.org/10.1145/2047414.2047428

Nordic Semiconductor ASA. (2017). nRF Connect for Android.

OpenYou Organisation. (2013). OpenYou. Retrieved 22 April 2017, from http://www.openyou.org/

POLAR. (2013). Innovations. Retrieved 19 April 2017, from https://www.polar.com/us-en/about_polar/who_we_are/innovations

Savov, V. (2009). Fitbit tracker starts shipping, ready to monitor your fitness, sleep, piety. Retrieved 19 April 2017, from https://www.engadget.com/2009/09/29/fitbit-tracker-starts-shipping-ready-to-monitor-your-fitness-s/

Schmidt, B., Benchea, S., Eichin, R., & Meurisch, C. (2015). Fitness tracker or digital personal coach: how to personalize training. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers* (pp. 1063–1067). ACM.

Townsend, K. (2014). Introduction to Bluetooth Low Energy. Retrieved 20 April 2017, from https://learn.adafruit.com/introduction-to-bluetooth-low-energy?view=all

Web Bluetooth Community Group. (2017a). Web Bluetooth. Retrieved 20 April 2017, from https://webbluetoothcg.github.io/web-bluetooth/

Web Bluetooth Community Group. (2017b). Web Bluetooth Implementation Status. Retrieved 20 April 2017, from https://github.com/WebBluetoothCG/web-bluetooth/blob/master/implementation-status.md

Wikipedia. (2017). Activity tracker. Retrieved 19 April 2017, from https://en.wikipedia.org/w/index.php?title=Activity_tracker&oldid=773450801

Xiaomi. (2016). Mi Band. Retrieved 20 April 2017, from http://www.mi.com/en/miband/

Xiaomi. (2017). Mi Band Pulse. Retrieved 21 April 2017, from https://xiaomi-mi.co.uk/xiaomi-mi-band/xiaomi-mi-band-pulse-black/

Yasskin, J. (2016). The Web Bluetooth Security Model. Retrieved 20 April 2017, from https://medium.com/@jyasskin/the-web-bluetooth-security-model-666b4e7eed2

Appendix 1 Project Overview

Kamil Wojtczyk 40128893

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Vendor-neutral software application for fitness wearables

Overview of Project Content and Milestones

The main idea of the project is to create a web-based application that enables users to synchronise their fitness tracking devices and explore the collected data, using single generic solution which would not require downloading any separate programs or using vendor-specific tools, but only visiting a web page.

To achieve this functionality, multiple modern technologies will be used, including: Web Bluetooth API - a client-side interface to interact with Bluetooth Low Energy devices, use of local storage to provide ability to store data in browser, service workers to provide support for offline use of the application and utilisation of progressive web app technology to enhance mobile experience.

Milestones:

- Week 4: investigate publicly available APIs for fitness devices (open-source or vendor-specific)
- Week 5: research of the devices used in the fitness market
- Week 6: research of the interfaces used in fitness tracking Bluetooth devices from different vendors - whether they use a standardised layer of communication or custom structure
- Week 8: research of the Web Bluetooth API and an attempt to initialise a connection between the host and the device
- Week 10: design of the application structure and API based on the results of previous researches (generic API or vendor-specific modules)
- Week 13: application development; enabling data synchronisation between device and host
- Week 6 (2nd semester): implementation of a user interface that would utilise the collected data and visualise in a useful way
- Week 8 (2nd semester): offline application development (service workers)

The Main Deliverable(s):

Provide an ability to connect to a generic fitness tracking device using Web Bluetooth API and synchronise collected data.

The Target Audience for the Deliverable(s):

- Fitness enthusiasts
- Bluetooth LE and Web Bluetooth API research community

The Work to be Undertaken:

- Extensive research on the Bluetooth Low Energy technology and interfaces used in fitness tracking devices.

- Analysis of multiple devices used in the market.Application system design
- Analysis of the collected data.
- Implementation using web-based solutions.
- Testing with multiple trackers from different manufacturers

Additional Information / Knowledge Required:

- Extending current skills in building client-side software applications
- Gaining knowledge in smart wearable devices and BLE technology
- Use of new technologies including:
    - Web Bluetooth API
    - Statically-typed JavaScript (TypeScript) utilising modern standards (ES2015 and ES2016)
    - Web and Service Workers
    - Local storage
    - Progressive Web App technology for mobile devices

Information Sources that Provide a Context for the Project:

W3C: Draft Community Group Report
https://webbluetoothcg.github.io/web-bluetooth/

Google Developers: Web Bluetooth API
https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web?hl=en

Mozilla: Web Bluetooth API
https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API

Web Bluetooth Samples
https://googlechrome.github.io/samples/web-bluetooth/index.html

Google+ Web Bluetooth community
https://plus.google.com/communities/108953318610326025178

The Importance of the Project:

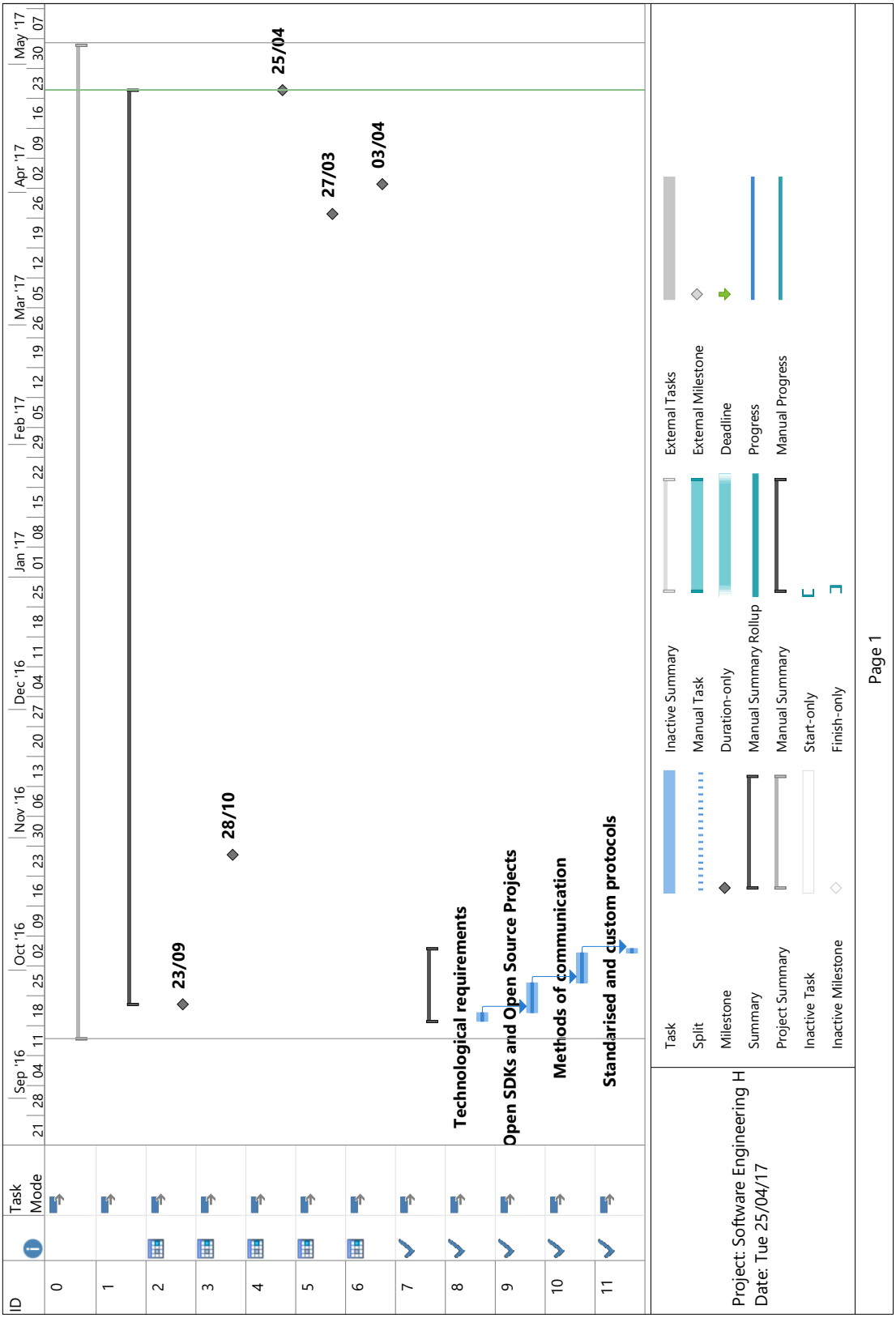
A generic solution for fitness tracking devices has not been implemented yet, at the moment users have to rely on a specific application provided by the manufacturer. This project is going to research and implement a new way of interacting with those gadgets.

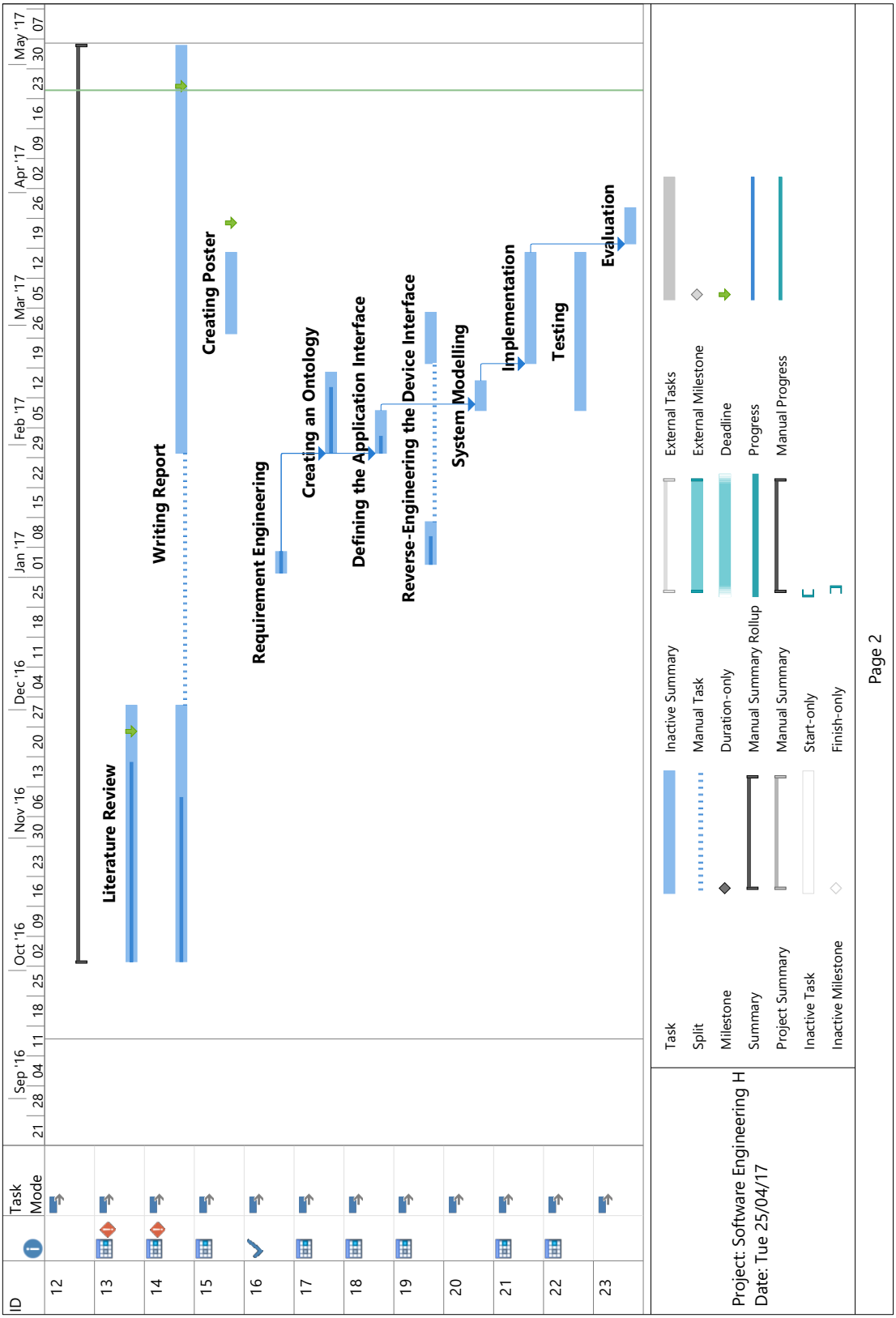
The Key Challenge(s) to be Overcome:

The main challenge would be researching whether Bluetooth-enabled fitness trackers use generic way of structuring the data and communicating with end device, as it there may be not enough academic researches available, due to the topic being fairly new. There is a possibility of a test-driven approach.

There may be several debugging related issues during the development as Web Bluetooth API is an experimental technology implemented only on selected operating systems and browsers. The final interface may also be changed in the future.

Appendix 3 Gantt Chart & Diary Sheets




ID
Task Mode
Sep '16
Oct '16
Nov '16
Dec '16
Jan '17
Feb '17
Mar '17
Apr '17
May '17
Literature Review
Writing Report
Creating Poster
Requirement Engineering
Creating an Ontology
Defining the Application Interface
Reverse-Engineering the Device Interface
System Modelling
Implementation
Testing
Evaluation
Project: Software Engineering H
Date: Tue 25/04/17
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress
Page 2

Appendix 5 Xiaomi Developers Site

Appendix 6 Bluetooth GATT Specification

specifications

Adopted Specifications

Bluetooth 5 Core Specification is an important update to the Bluetooth Core Specification, significantly increasing the range, speed and broadcast messaging capacity of Bluetooth applications.

Bluetooth 5 Core Specification

With up to 4x the range, 2x the speed and 8x the broadcasting message capacity, the enhancements of *Bluetooth*® 5 focus on increasing functionality for the Internet of Things (IoT). Bluetooth 5 delivers a “connectionless” IoT, advancing beacon and location-based capabilities in home, enterprise and industrial applications. This will create significant advantages for developers and manufacturers, while providing a better user experience for their customers.

Links to tools and information:

- Frequently Asked Questions
- Bluetooth Brand Guide
- Download the Core Specification

Bluetooth Qualification Test Requirements

If you are looking for more information about the Bluetooth Qualification Testing Requirements, please visit the Testing Requirements page.

Adopted Bluetooth Core Specification

Specification	Date Adopted	Status
Core Version 5.0	06 December 2016	Active
Core Specification Supplement (CSS) v7	06 December 2016	Active
Core Specification Addendum (CSA) 5	15 December 2015	Active

Core Version 4.2	02 December 2014	Active

Adopted Bluetooth Profiles, Services, Protocols and Transports

To view the structure of definitions that are used in the adopted GATT profiles, services, and characteristics use the XML Definition Browser. You can also download the definitions in an XML format.

GATT-Based Specifications

Profile	Specification	Version	Status	Date Adopted
ANP	Alert Notification Profile	1.0	Active	13 September 2011
ANS	Alert Notification Service	1.0	Active	13 September 2011
AIOP	Automation IO Profile	1.0	Active	14 July 2015
AIOS	Automation IO Service	1.0	Active	14 July 2015
BAS	Battery Service	1.0	Active	27 December 2011
BCS	Body Composition Service	1.0	Active	21 October 2014
BLP	Blood Pressure Profile	1.0	Active	25 October 2011
BLS	Blood Pressure Service	1.0	Active	25 October 2011
BMS	Bond Management Service	1.0	Active	21 October 2014
CGMP	Continuous Glucose Monitoring Profile	1.0.1	Active	15 December 2015

CGMS	Continuous Glucose Monitoring Service	1.0.1	Active	15 December 2015
CPP	Cycling Power Profile	1.1	Active	03 May 2016
CPS	Cycling Power Service	1.1	Active	03 May 2016
CSCP	Cycling Speed and Cadence Profile	1.0	Active	21 August 2012
CSCS	Cycling Speed and Cadence Service	1.0	Active	21 August 2012
CTS	Current Time Service	1.1	Active	07 October 2014
DIS	Device Information Service	1.1	Active	29 November 2011
ESP	Environmental Sensing Profile	1.0	Active	18 November 2014
ESS	Environmental Sensing Service	1.0	Active	18 November 2014
FMP	Find Me Profile	1.0	Active	21 June 2011
FTMP	Fitness Machine Profile	1.0	Active	14 February 2017
FTMS	Fitness Machine Service	1.0	Active	14 February 2017
GLP	Glucose Profile	1.0	Active	10 April 2012
GLS	Glucose Service	1.0	Active	10 April 2012
HIDS	HID Service	1.0	Active	27 December 2011
HOGP	HID over GATT Profile	1.0	Active	27 December 2011
HPS	HTTP Proxy Service	1.0	Active	

				06 October 2015
HRP	Heart Rate Profile	1.0	Active	12 July 2011
HRS	Heart Rate Service	1.0	Active	12 July 2011
HTP	Health Thermometer Profile	1.0	Active	24 May 2011
HTS	Health Thermometer Service	1.0	Active	24 May 2011
IAS	Immediate Alert Service	1.0	Active	21 June 2011
IPS	Indoor Positioning Service	1.0	Active	19 May 2015
IPSP	Internet Protocol Support Profile	1.0	Active	16 December 2014
LLS	Link Loss Service	1.0.1	Active	14 July 2015
LNP	Location and Navigation Profile	1.0	Active	30 April 2013
LNS	Location and Navigation Service	1.0	Active	30 April 2013
NDCS	Next DST Change Service	1.0	Active	13 September 2011
OTP	Object Transfer Profile	1.0	Active	17 November 2015
OTS	Object Transfer Service	1.0	Active	17 November 2015
PASP	Phone Alert Status Profile	1.0	Active	13 September 2011
PASS	Phone Alert Status Service	1.0	Active	13 September 2011
PXP	Proximity Profile	1.0.1	Active	14 July 2015
PLXP	Pulse Oximeter Profile	1.0	Active	

				14 July 2015
PLXS	Pulse Oximeter Service	1.0	Active	14 July 2015
RSCP	Running Speed and Cadence Profile	1.0	Active	07 August 2012
RSCS	Running Speed and Cadence Service	1.0	Active	07 August 2012
RTUS	Reference Time Update Service	1.0	Active	13 September 2011
ScPP	Scan Parameters Profile	1.0	Active	27 December 2011
ScPS	Scan Parameters Service	1.0	Active	27 December 2011
TDS	Transport Discovery Service	1.0	Active	17 November 2015
TIP	Time Profile	1.0	Active	13 September 2011
TPS	Tx Power Service	1.0	Active	21 June 2011
UDS	User Data Service	1.0	Active	27 May 2014
WSP	Weight Scale Profile	1.0	Active	21 October 2014
WSS	Weight Scale Service	1.0	Active	21 October 2014

Traditional Profiles

Profile Specification	Version	Status	Date Adopted
3DSP	1.0.3	Active	15 December 2015
A2DP	1.3.1	Active	14 July 2015

AVRCP	1.6.1	Active	15 December 2015
BIP	1.2	Active	24 July 2012
BPP	1.2	Active	27 April 2006
CTN	1.0	Active	18 September 2014
DI	1.3	Active	26 March 2007
DUN	1.2	Active	06 November 2012
FTP	1.3.1	Active	15 December 2015
GAVDP	1.3	Active	24 July 2012
GNSS	1.0	Active	13 March 2012
GOEP	2.1.1	Active	15 December 2015
GPP	1.0.1	Active	15 December 2015
HCRP	1.2	Active	27 April 2006
HDP	1.1	Active	24 July 2012
HFP	1.7.1	Active	15 December 2015
HID	1.1.1	Active	15 December 2015
HSP	1.2	Active	18 December 2008
MAP	1.3	Active	17 May 2016
MPS	1.0	Active	02 July 2013
OPP	1.2.1	Active	15 December 2015
PAN	1.0	Active	20 February 2003

PBAP	1.2.1	Active	15 December 2015
SAP	1.1.1	Active	15 December 2015
SPP	1.2	Active	24 July 2012
SYNCH	1.2.1	Active	15 December 2015
VDP	1.1	Active	24 July 2012

Protocols

Protocol Specification		Version	Status	Date Adopted
AVCTP	A/V Control Transport	1.4	Active	24 July 2012
AVDTP	A/V Distribution Transport	1.3	Active	24 July 2012
BNEP	Bluetooth Network Encapsulation Protocol	1.0	Active	20 February 2003
IrDA	IrDA Interoperability	2.0	Active	26 August 2010
MCAP	Multi-Channel Adaptation Protocol	1.0	Active	26 June 2008
RFCOMM	RFCOMM	1.2	Active	06 November 2012

Adopted Bluetooth Errata Service Releases

Release	Adopted Date	Notes
ESR10	06 December 2016	
ESR09	15 December 2015	

ESR08	02 December 2014	
ESR07	03 December 2013	
Erratum 5256	22 October 2013	Mandatory when claiming compliance to 3DS Synchronization Profile v1.0
ESR06	04 December 2012	
Erratum 4656	07 February 2012	Mandatory when claiming compliance to v2.0 + EDR and later
ESR05	23 August 2011	
ESR04	18 December 2008	
ESR03	06 February 2008	
Erratum 747	25 November 2005	Enabling Boot Mode Only (HID Lite) Hosts
ESR02	21 June 2005	
ESR01	25 August 2003	

If you are looking for other specifications including withdrawn or deprecated, go here.