VRIJE UNIVERSITEIT AMSTERDAM

MASTERS THESIS

SWAN expression in multi-device multi-sensor environment

Author:
Veaceslav MUNTEANU

Supervisor:
Prof. Dr. Ir. Henri E. BAL
Roshan Bharath Das

A thesis submitted in fulfilment of the requirements for the degree of MSc.

in the

Faculty of Sciences

August 2016

Abstract

A typical mobile phone application will most likely want to take better advantage of the sensors embedded in the phone. It is easy to access data about sensors on an Android device but things start to be difficult when we want good battery savings. The best case scenario is to delegate all the troubles of having efficient implementation to a library, or a middleware project, such as SWAN. SWAN offers a simple and intuitive way to get data from available sensors, just by registering a simple expression. To make things even simpler, there is a configuration activity for the given sensor, so that developers can learn faster what options are available to them and how to get the best expressions for the needs. The sensors in the phone offer great flexibility but with proximity sensors it becomes increasingly difficult to preserve the original design. For example, applications want to take advantage from sensors located on Bluetooth connected smartwatch or nearby beacons . The functionality offered by SWAN extends beyond the simple sensor data retrieval. Advanced scenarios include waking up the application when the sensor value exceed a certain threshold. This allow applications to sleep and only wake up when a certain event occurs.

Acknowledgements

We thank the municipality of Alkmaar for sponsoring us with Bluetooth Low Energy Beacons, Android Smartwatch and other Bluetooth connected devices used for this project. This work was done in the context of ACBA(Amsterdam Center of Business Analytics) led by Frans Feldberg. I would like to thank Prof. Henri Bal, PhD Roshan Bharath Das, PhD Nicolae Vladimir Bozdog and Drs. Kees Verstoep for their valuable help.

probably add Marc Makkes (second reader) and Data Science Alkmaar.

Contents

Abstract i

Acknowledgements ii

Contents iii

List of Figures v

List of Tables vi

1 Introduction 1

2 Related Work 3
- 2.1 General . . . 3
- 2.2 Swan Plus . . . 3

3 Background 5
- 3.1 SWAN . . . 5
- 3.2 Beacon Frame Standards . . . 6

4 Beacon Implementation 7
- 4.1 SWAN Data Specifications . . . 7
  - 4.1.1 Proposed Solutions . . . 7
    - 4.1.1.1 Data Mapping approach . . . 8
    - 4.1.1.2 Expression Location Approach . . . 9
  - 4.1.2 Implemented Approach . . . 10
- 4.2 Beacon based Bluetooth Sensors . . . 11
  - 4.2.1 AltBeacon Library . . . 12
  - 4.2.2 Beacon Frame Formats . . . 13
  - 4.2.3 Retrieving data from specific Beacons . . . 13

5 Smartwatch Implementation 15
- 5.1 Value Based Smartwatch Sensors . . . 15
- 5.2 Swan Application Layout . . . 16
  - 5.2.1 Splitting SWAN sources into libraries . . . 16
- 5.3 Expression Based Smartwatch Sensors . . . 17

6 Power Efficiency Analysis **19**
6.1 Introduction 19
6.2 Design 19
6.3 Experiment Planning 20
6.3.1 Context Selection 20
6.3.2 Variable Selection 20
6.3.3 Hypothesis Formulation 21
6.3.4 Subject Selection 22
6.3.5 Experiment Design 23
6.3.6 Threats to Validity 23
6.3.7 Instrumentation 24
6.3.8 Experiment Execution 26
6.3.9 Preliminary Tests 27
6.3.10 Test Application 27
6.4 Results 28
6.4.1 Preliminary Analysis 28
6.5 Hypothesis Testing 32
6.6 Final Results Discussion 33

7 Conclusion and Future Work **35**
7.1 Conclusions 35
7.2 Future Work 35

A Power Analysis Data **37**

B Merging SWAN Repositories **41**

Bibliography **42**

List of Figures

3.1 Swan Expression 6

4.1 Estimote Nearable Format 14

6.1 Phone vs Wear Based Expressions, delay 100 ms 29
6.2 Phone vs Wear Based Expressions, delay 1000 ms 29
6.3 Run time of different number of sensors, baseline 4800 seconds 30
6.4 Expression Power Consumption 31
6.5 Expression Power Consumption 31
6.6 Dropping values in Phone and Wear Based Expressions 34

List of Tables

A.1 Phone Based Expression with delay 100 ms 37
A.2 Phone Based Expression with delay 1000 ms 38
A.3 Wear Based Expression with delay: 100 ms 38
A.4 Wear Based Expression with delay: 1000 ms 38
A.5 Individual Phone Based Expression power consumption, delay 100 ms . . 39
A.6 Individual Wear Based Expression power consumption, delay 100 ms . . . 39
A.7 Phone and Wear based Test Sensor, delay 100 ms 39
A.8 Phone and Wear based Test Sensor, delay 1000 ms 40

Chapter 1

Introduction

With the smartphone user base reaching almost two billion users, new technologies arise to take advantage of our pocket device, which also packs a considerable amount of processing power. Smartwatches that pair with our smartphones aim to replace the regular watch, beacons aim to replace WI-FI location support. However, we must not forget that limited battery capacity is still a problem even after so many years of development. The available power is even more limited on the smartwatch, because of its small form factor. With Android ecosystem being mature enough, there are little room for new application ideas. Having reliable access to new features and taking advantage of the new Bluetooth Low Energy standard enable us to come with new, innovative usage scenarios. SWAN application aims to facilitate the access to sensors available of the phone. We try to take advantage of the new technologies by extending SWAN application.

The overall goal of the project was to integrate a variety of new Bluetooth Low Energy devices into SWAN, including beacons and smartwatches. We encountered several challenging problems that we study in more detail, including the following questions:

- What is the best method of acquiring data from a sensor located on the smartwatch? Can SWAN operate on the smartwatch, assuming that we can only keep its core functionality?
- What are the power consumption on the watch and on the phone, when SWAN is actively retrieving data from smartwatch?
- What changes should be applied to expression based SWAN, to add support for multiple sensors of the same type located on beacons?

- How can we take full advantage of Bluetooth Low Energy Beacons? How do we seamlessly integrate multiple beacon frame formats without increasing the program's complexity?

The master thesis is structured as follows. In Chapter 2 we will describe the work related to our project, Chapter 3 covers essential details about the SWAN project and Chapter 4 is dedicated to data format changes that we applied on SWAN expression to allow support of multiple sensors. Chapter 5 describes all implementation details related to scanning beacons and communication with a smartwatch and Chapter 6 offers a detailed analysis of the power consumption for two implemented methods of accessing smartwatch sensors. Chapter 7 concludes the master thesis and describes the future work.

Chapter 2

Related Work

→ Introductory sentence missing!

2.1 General

For our implementation of smartwatch sensors and Bluetooth beacon sensors we used the standard Android API to send, scan and receive information. There are other ways of communicating with the smartwatch, for example using Beetle[16] service. Using Beetle, it is also possible to retrieve data from heart rate monitors in real time, in a similar way of how our SWAN service on the watch works.

Describe why our work is better

The implementation on the SWAN service on the smartwatch does not have notable differences compared to a regular Android phone application. An extensive description of the operating system on the watch is available in ~~Understanding Characteristics of Android Wear~~[18]. In comparison with our power consumption experiment, the paper's experiment was performed with power monitors attached directly to hardware, which gives higher accuracy.

of

the

here you say our work is bad??

The ArmTrak[20] program, which aims to better understand user's hand gestures, is implemented using the same approach as our phone based smartwatch sensors. ArmTrak requires high accuracy and low latency for some scenarios, and with the new wear based SWAN expressions, some computation can be done locally, on the watch.

2.2 Swan Plus

The SWAN Plus[17] project, ~~which~~ aims to offer expression evaluation on nearby devices, is closely related to our work on adding support for smartwatches and Bluetooth devices. Our implementation of sending SWAN expressions on the watch was coded on top of an already existing mechanism of sending expressions to nearby devices and forwarding

It

other Bluetooth devices??

results back to the expression registrar. On the other hand, SWAN Plus focuses on multiple devices and uses a more close-to-metal approach of scanning and discovering the Bluetooth devices.

(Try to add more related work)

→ Rearrange related work

→ First describe others work

→ then compare our work with that

(Try to prove that our work is needed!!)

For example,

Beetle does a.....

But, we do .....

Chapter 3

Background

In this chapter

Before discussing implementation details and research questions, we provide a brief description of the SWAN program. The first subsection will focus only on key features of SWAN, relevant to our work. The second subsection will briefly describe few Beacon Frame Standards relevant for our research.

framework

3.1 SWAN

The core functionality of SWAN is to act as a middleware between the phone applications and the actual hardware or software sensors. We will further refer to the expression which is being passed to SWAN from application as SWAN Song expression, or simply SWAN Song. There are multiple types of Swan Song Expressions:

give reference

- Value Expression - Retrieve values from sensors
- Tristate Expression - Perform evaluation on the values before sending them to applications

The application's interface with SWAN is always the same, the only component that varies is the SWAN Song expression passed to SWAN. Swan Song Expression(Figure 3.1) encapsulates all the information required by SWAN. Besides passing data to the application, SWAN also takes care of evaluation and storage. Relevant parameters are also embedded into the SWAN Song expression.

As part of our research we will also apply changes on the SWAN Song Expression and the meaning of different parameters.

give brief description of what you see in the figure

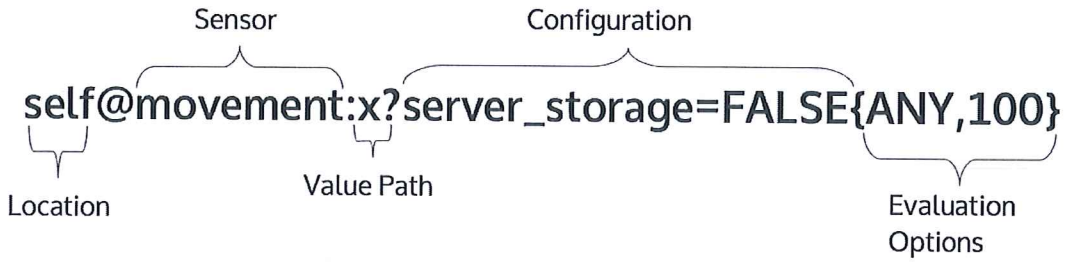

FIGURE 3.1: Detailed swan expression).

3.2 Beacon Frame Standards

With Bluetooth Low Energy devices market in its incipient stage, the Beacon emitters suffer from high fragmentation. To help improve the situation, Apple and Google stepped up to offer frame specifications and further enforce the standard.

Apple's format, called iBeacon is simple and only focused on proximity applications. On the other hand, Google offers and extensive standard, with 3 different frame specifications and more specifications coming soon.

Chapter 4

Beacon Implementation

Beacon implementation for SWAN aims to support a large array of beacon formats. But we discovered that current SWAN data format was not suitable for our multi-sensor environment. This chapter describes the changes applied to data formats and then gives more details about beacon implementation.

4.1 SWAN Data Specifications

Initially SWAN was designed to take a single value from the sensor data. The value should be a primitive type such as Integer, Float, Double etc. This constraint applies because SWAN evaluation engine allows the programmer to perform MIN, MAX and AVERAGE operations on given sensor data. Also, History reduction is important when using the single value.

By adding multiple sensors, it became clear that taking a single value is not scalable enough. The further changes were delayed by adding multiple value paths to the sensor implementation. A clear example was accelerometer data: You will probably need the values for all 3 axes: x, y, z. And you need to register 3 expressions for each value path. The limitation can no longer be avoided after implementing Beacon Sensors, because now we need to add a full array of beacon identifiers under the same timestamp, the key-value to provide individual temperature data or even key-array of values if we want the accelerometer data from all beacons.

4.1.1 Proposed Solutions

All the proposed solutions should meet the following requirements:

- Support multiple sensors of the same type (beacon sensor)
- Support multiple values for each sensor
- Provide scheme for single sensor single value data
- Provide implementation for Evaluation Engine Application

We will further discuss two options that meet the requirements from above.

4.1.1.1 Data Mapping approach

Data Mapping tries to solve the problem by proposing a new approach of how data is being passed form the sensor to the evaluation engine. The model is aiming to solve future problems when it comes to supporting multiple sensors with multiple values at the same time. The drawback of this solution is the number of changes that are required to be done in the Evaluation Engine and sensors, in order to implement the new approach. Also there is a high risk of breaking compatibility with older expressions.

The format of data for the following function calls(value parameter):

```
1 AbstractSwanSensor.putValueTrimSize(final String valuePath,
                    final String id,
3                   final long now,
                    final Object value):
```

In order to satisfy the new requirements, we propose the new data format to be encoded as:

```
Map<ValuePath, Map<sensorID,  ArrayList<Object>>
```

where:

- ValuePath = sensor valuepaths, ex Accelerometer x, y, z, total
- sensorID = self or unique Sensor Identifier(BeaconID)

We identify four sensor formats and recommend the following representation for each type of sensor:

- Single Sensor Single Value(sensor name = stepcounter):

  - Example: Map: { steps={self=[13434545] }}

- Single Sensor, Multiple Values(sensor name =accelerometer):
  - Key - name of the sensor
  - Array of values - multiple values in the same order
  - Example: Map: { x={self=[0.5]},
    y={self=[0.6]}, z={self=[-0.5]} }

- Multiple Sensor, Single Value(Beacon Distance):
  - Key - the sensor identifier(id)
  - Array of values - single value in the array
  - Example: Map: {distance={beaconID1=[1.5],
    beaconID2=[0.5], beaconID3=[0.2]}}

- Multiple Sensor, Multiple Values(sensor name = Beacon Accelerometer):
  - Key - the sensor identifier(id), should be unique
  - Array of values - multiple values in the same order
  - Example: Map:{x ={beaconID1=[-0.5], beaconID2=[-0.1],
    y = {beaconID1=[0.2], beaconID2=[-0.4]}}

4.1.1.2 Expression Location Approach

Compared to the Data Mapping, the Expression Location Approach tries to preserve the single return value and value paths for each sensor. It works in addition to the current implementation and maintains the backwards compatibility with the previous expression format. The drawback of this approach is very low flexibility when the number of sensors are really high. Also, it tries to solve an immediate problem rather than focusing on future challenges.

Instead of using a map to send the values to the evaluation engine, we add an extension to the current location of the SWAN expression. The evaluation engine recognizes keywords such as **self**, **wear** and **remote**. To support the multiple sensors, we add the sensor's unique identifier instead of the location. In case of Beacons, each beacon has beacon Id, which should be unique for all kind of applications.

The recommended SWAN expressions for the 4 categories of sensors:

- Single Sensor Single Value(sensor name = stepcounter):

  - Register an expression:

    ```
    self@stepcounter:steps{ANY, 1000}
    ```

  - Use putTrimValue() with a single value

- Single Sensor, Multiple Values(sensor name =accelerometer):

  - Register an expression with value path:

    ```
    self@accelerometer:x{ANY, 1000}
    ```

  - Use putTrimValue() with a single value

- Multiple Sensor, Single Value(Beacon Distance):

  - Use Beacon Discovery Sensor to get a list beaconIDs:

    ```
    self@beacon_discovery:ibeaconuuid{ANY, 10}
    ```

  - How is beacon distance received? (show expression)

- Multiple Sensor, Multiple Values(sensor name = Beacon Accelerometer):

  - Use Beacon Discovery Sensor to get a list beaconIDs:

    ```
    self@beacon_discovery:ibeaconuuid{ANY, 10}
    ```

  - Register individual expressions with individual valuepaths for each BeaconID:

    * `beaconID@beacon_movement:x{ANY,1000}`
    * `beaconID@beacon_movement:y{ANY,1000}`

Note these expressions can not be evaluated by Evaluation Engine, because it outputs an array instead of a single value.

4.1.2 Implemented Approach

The implemented solution was the Expression Location Approach. While the Data Mapping approach brings more novelty, it comes with severe disadvantages:

- Unable to keep backward compatibility with older applications which are using SWAN - before the changes, applications expect a single primitive value. Using an array or a map may result in application crashes.
- While the Value Based Expressions are easy to adapt, changing Tristate Expressions is more difficult - Evaluation engine needs to be redesigned to adapt the new approach

- Big number of changes may render SWAN unstable and it will take a long time to narrow down bugs

On the other hand, the Location Based approach does not interfere with the Evaluation Engine's algorithm, and location can be easily passed to the sensor implementation and handled by sensor implementations.

Implementing the location extension was easy, we just added an extra parameter of type **Map** which we pass to the sensor when we bind it. The existing sensors preserved their implementation without changes and the beacon based sensors were adapted to get the beacon ID from the location parameters. Also, for Bluetooth sensor discovery purpose, we left the Discovery sensor to output an array instead of a single value, assuming that no Tristate Expression will be registered with that sensor.

4.2 Beacon based Bluetooth Sensors

Bluetooth sensors allow us to get proximity data from the nearby beacons. Beacon sensor integration into SWAN expression based approach was not seamless. The new data specifications which include beacon extension are discussed in Chapter 3.

Before exploring the implementation details we must discuss the available Beacon technology and Beacon Frame format. The Bluetooth Low Energy Beacons can broadcast various data with different transmission power and broadcast interval. The broadcast power is measured in dBm[13]. The transmission power can vary from -30dBm to +4dBm. Higher transmit power increases the range, but also shortens the battery life. The broadcast interval can vary from 300ms to more than 5 seconds. Higher resolution improves the accuracy of data, especially for real-time applications, but also shortens the battery life.

To avoid the fragmentation of the Beacon Market, Google and Apple stepped in and proposed two standards for Beacon Frame Layout:

- Apple iBeacon - simple beacon format, mostly used for proximity(distance measuring) applications
- Google Eddystone - complex standard, with 3 available frame formats:
  - Eddystone UID - similar to iBeacon, broadcast unique ID
  - EddystoneTLM - telemetry frame format, stores information about the beacon, such as temperature, battery level, number of packets sent

  - Eddystone URL - frame layout which encodes a 17 bytes long URL in the frame

The standards from above are industry recognized and widely implemented by various Beacon Manufacturers. Unfortunately, even the Google Eddystone Format is not flexible enough and some companies need to come up with their own format to support extra functionality added to their beacon products. We will also discuss the following frame formats which are proprietary to companies selling the test beacons, but they enable us to explore the new way of using beacons:

- AltBeacon Beacon Format - default beacon format present in the library used by SWAN for beacon scanning
- Estimote Nearable - Beacon Frame Format developed by Estimote[3], with accelerometer and movement data embedded into the format

The beacon implementation is revolving around adding support for the four beacon formats listed above. Implementation details were further split into following steps:

- Implement Bluetooth Low Energy Scan
- Add support for existing Beacon Formats
- Implement Bluetooth Beacon Discovery
- Implement Bluetooth Based Sensors

4.2.1 AltBeacon Library

The AltBeacon Library is developed by Radius Networks[8] company and provide a free and open-source implementation for applications that want to scan for beacons. We choose to use this library over a custom implementation because of the easy way to add new beacon formats. The library does not support beacons with encrypted identifiers, but it allows us to parse almost any existing, non-encrypting beacon frame. A proof of flexibility is our beacon layout for Estimote Nearable that we were able to implement in SWAN. AltBeacon runs as a service, so SWAN needs to register to get the results of every scan. All the data from available beacons are available at once, so we decided to go for observer pattern, and register all sensors that want to access beacon data in our singleton class. This allows us to scan once for all sensors.

4.2.2 Beacon Frame Formats

The support for Apple iBeacon, Eddystone UID, Eddystone TLM, Eddystone URL and AltBeacon Format was already available in the library. The Estimote format was left for us to analyse and provide the proper parsing expression to the AltBeacon Library.

The process of parsing the non- standard frame starts with finding the specifications for each byte in the frame. Fortunately, we found a NodeJs implementation that parses the Estimote Nearable Frame. The data encoded is presented in Figure 5.1. Before decoding the data we must tell the AltBeacon Library where to look for the frame identifier. The debugging mode allows us to see how the Bluetooth Low Energy frames are being parsed.

4.2.3 Retrieving data from specific Beacons

Original SWAN only allowed to return a single value of the primitive type. To comply with the new SWAN Data Specifications, we test the location part of expression to see which information we should provide:

- If location is **self** we retrieve a random beacon from the list, which matches our frame type and give the result to the user
- if location contains a **beacon ID** we search for the beacon in the list and if it is present, we will output the result

Other types of operations can be implemented, but first, a new revision of data formats should be made, to set the guidelines for all sensor implementations.

Our implementation relies on a singleton class which gets the result of the scanning and then distributes it to all registered beacon sensors. The AltBeacon library will automatically stop scanning if no beacon sensors are registered.

Bytes	Data	Values
0	Identifier	0x5d
1		0x01
2		0x01
3	ID	
4		
5		
6		
7		
8		
9		
10		
11	Type	SB0
12	Firmware	-127,-126,-125
13	Temperature	
14		
15	Battery Voltage	
16	Accelerometer:x	
17	Accelerometer:y	
18	Acceleromenter:z	
19	Current Motion State Duration	
20	Previous Motion State Duration	
21	Power/Firmware State	

FIGURE 4.1: Estimote Nearable Format

Chapter 5

Smartwatch Implementation

Initial smartwatch support for SWAN was added using the value based approach. Later on, the opportunity to run a mini version of SWAN on the smartwatch arose. But before we were able to run SWAN on the smartwatch, we had to apply changes on application layout, especially, to split it into libraries. In this chapter, we will describe our value based approach of gathering data from smartwatch, then we briefly describe the new SWAN layout and we conclude with the mini SWAN implementation for the smartwatch.

5.1 Value Based Smartwatch Sensors

Implementing the value based smartwatch sensor was the first option and it derived from the Android's guidelines of performing the minimum amount of computation on the watch. The other requirement was to keep code for Android Wear as small as possible, since the performance of the integrated components is lower compared to computing power on the smartphone.

To keep the implementation simple, we have a singleton class responsible for communication between the smartphone and watch. We also issue start and stop commands to our service on the smartwatch, so when no smartwatch sensor is active, the SWAN will not run on the watch.

We used the data path [2] for reliable communication between devices. The use of data path guarantee that the value will always be delivered at the cost of increased delay and jitter[23] which affect our delay enforcing policy in SWAN.

The watch service is implemented as a foreground service[4]. Running in foreground requires us to display the notification on the watch, but on the other hand we have the

guarantee that our service will not be put to sleep by the operating system. Each SWAN sensor on the phone is able to send start and stop commands, and the watch service will always count how many sensors are active. When no sensors are active, the service will stop.

Despite the Google's guidelines of keeping the Android Wear applications small, value based smartwatch sensors have poor energy efficiency. More about the power consumption experiment, in Chapter 6.

5.2 Swan Application Layout

In the process of implementing the support for SWAN on smartwatch(also referred as SWAN WEAR) and Beacon Based Sensor, SWAN application layout has changed dramatically.

We will cover the following main changes applied to SWAN project structure during the last 6 months:

- Splitting the SWAN project sources in libraries
- Uploading the SWAN JARS in publicly available repository - JCenter

Before each change from above was made, a merge of all SWAN repositories was made. You can read more about it in Appendix B. Big changes on the project structure have a big impact on SWAN and could bring unexpected bugs.

5.2.1 Splitting SWAN sources into libraries

After the implementation of the smartwatch sensors and beacon sensors, as part of research topic, SWAN had to be ported on smartwatch. There were multiple options to handle the challenge:

- Clone all SWAN code and adapt it for the smartwatch
- Extract all common code and make it a library

I choose the second option, since it will avoid duplicate code and will allow easier maintenance in the future. As a starting point, I decided to see if the Evaluation Engine code can be extracted and called from a library. The dependencies were extensive. The Evaluation Engine was depending on:

- Swan Interface code - Classes that applications need to communicate with SWAN
- Swan Song Implementation - Parser and Data structure implementation for SWAN Expression
- Sensor Interfaces - used to implement new sensors in SWAN
- Beacon Implementation
- Proximity Implementation (SWAN Plus)

The first three components were separated from the SWAN Phone application and added to the SWAN core. Unfortunately, the dependency on the last two items was still present and it had to be removed from the code, since the implementation was relevant only for phone and not for the smartwatch. To preserve the functionality, all the methods depending on Proximity Manager and Beacon Discovery were replaced with empty bodies and the Evaluation Engine on the phone was sub-classing the Evaluation Engine in the library and implementing the missing functionality. The same design was applied to Abstract SWAN Sensor which was relying on Sense Android Library to offload values[21].

After **SWAN Core** was made independent, we decided to extract **SWAN Song** and **SWAN Interface** from it and make them separate libraries. The main reason for this library separation was the way applications interact with SWAN. SWAN Interface is often imported as a separate JAR in any application that wants to register a SWAN expression. To improve the distribution of the Swan Interface, we decided to make it available[1] on Android Main Repository - JCenter[5].

Because we are maintaining the packages on JCenter now, the old packages and package building scripts in the repository were discarded. The JCenter is not storing the packages in JAR format, but in the new format made to accommodate the Android Resources: AAR [1]. The new package format can be built and uploaded directly from Android Studio.

More information about how to manage and upload swan libraries is available on SWAN wiki on Github[11].

5.3 Expression Based Smartwatch Sensors

Expression based smartwatch sensor implementation aims to bring SWAN on your smartwatch. Before, we didn't consider to have SWAN on the watch a good idea. SWAN

[1] Uploading the JARS required us to follow the tutorial by The Cheese Factory[12].

is a big application and running it on the watch could result in sever performance issues on the watch. The opportunity of running SWAN on the watch arose after we managed to split SWAN source-code into multiple libraries. Some performance incurring features, such as database storage were left behind, and now the source-code was small enough to run on the watch.

Explain more about difference between SWAN and miniSWAN

Compared to normal SWAN expression, a wear expression has the location string set to **wear** keyword. Evaluation Engine is responsible for changing the location to **self** and forwarding the expression to SWAN on the smartwatch. The implementation of expression based smartwatch sensors added on top of the existing value based implementation. New commands were added to start and stop a SWAN expression on the watch. The values sent back use the same implementation as for value based sensors.

The main difference between value based and expression based sensors is that the values are not sent to SWAN sensor implementation for evaluation, but rather added to Evaluation Engine list of cached values. This approach offers very low power consumption on the phone, as we can see from the power analysis in Chapter 6.

change to
5.3 → 5.2
and put 5.2.1 → SWAN application layout
5.2.2 → Explain about miniSWAN

make 5.3 → difference b/w Value vs Expression based sensors.

Chapter 6

Power Efficiency Analysis

6.1 Introduction

The current implementation of the SWAN WEAR allows us to choose which way we want to process the data from the smartwatch application:

- Perform minimum amount of computation on the watch and just send values to be processed by the SWAN PHONE application
- Send the expression to SWAN WEAR application for evaluation

Typically, the recommended approach is to minimize the computation on the watch to save battery, but in certain cases, when the expression does not require frequent evaluation, we might get better power savings if we choose to perform the evaluation on the watch.

6.2 Design

Before proceeding to the data analysis, according to the GQM[14][15] paradigm, we should define the goal, the questions and the metrics. Our goal is to **analyse the power consumption** of phone and smartwatch, for the purpose of **comparing two different methods of acquiring sensor data** with respect to **differences in power consumption of phone and smartwatch**, in context of **SWAN android application**.

Question 1: What is the overhead of gaining data from smartwatch, compared to running evaluation on the watch?

- Metric 1: Battery level on Android Phone
- Metric 2: Battery level on Android Smartwatch
- Metric 3: Expression runtime in seconds

Question 2: What is the overhead of gaining data from test sensor[1] compared to real hardware sensor?

- Metric 1: Battery level on Android Phone
- Metric 2: Battery level on Android Smartwatch

6.3 Experiment Planning

6.3.1 Context Selection

The experiment will be run on the simulated(laboratory) environment, composed of Android Phone and Android Wear Smartwatch. We will consider our experiment a real life problem because:

- The tests are performed on real devices, which are used as reference for many Android developers
- The test suite is composed of various sensor data, which reflect the actual SWAN performance in real life applications.

6.3.2 Variable Selection

The main dependent variable is **power consumption** expressed as battery power consumed after the experiment. The independent variables are **data acquisition method** and **delay**. Data acquisition methods have two options:

- Phone based SWAN expressions
- Wear Based SWAN Expressions

Delay is set by us, but to avoid spending too much time on experiment, we will choose between two options: fast(100ms) and slow(1 second).

[1]Test Sensor is our reference implementation of SWAN sensor

6.3.3 Hypothesis Formulation

We consider μ the battery power consumed by the test suite. Since we have two devices to measure power consumption from, μ will be calculated as $\delta_{\text{phone}} + \delta_{\text{wear}}$, where δ is the difference in battery levels before and after the experiment.

Special case: If the power consumption is better on the phone but worse on the watch for the given approach, we cannot test the hypothesis:

- $(\delta_{\text{wear test type 1}} - \delta_{\text{wear test type 2}}) > 0$
- $(\delta_{\text{phone test type 1}} - \delta_{\text{phone test type 2}}) < 0$

The reason why we can't apply our hypotheses is because the smartwatch and the phone have different hardware, idle power consumption and battery installed. We cannot compare them directly, but we can claim that one approach is better for the phone but worse for the watch and let the developers make decision on what they want to value the most: Phone or watch battery.

Question 1: What is the overhead of gathering data from smartwatch, compared to running evaluation on the watch?
Conjecture(P): There is a difference between gaining data and evaluating expression on the smartwatch
Consequence(Q): The difference between only gaining data and evaluating expression on the smartwatch is significant.
Null hypothesis - No observable difference in power consumption between only gathering data and evaluating expression on the watch
H0: $(\mu_{\text{phone based}} - \mu_{\text{wear based}} = 0)$
Alternative hypothesis - There is a difference in power consumption between only gaining data on watch and running a SWAN expression on the watch
H1:$(\mu_{\text{phone based}} - \mu_{\text{wear based}} \neq 0)$

Question 2: What is the overhead of gaining data from test sensor compared to real hardware sensor?
Conjecture(P): There is a difference between using test sensor and real sensor on the smartwatch
Consequence(Q): The difference between only gaining data and evaluating expression on the smartwatch is significant.
Null hypothesis - No observable difference in power consumption between using test sensor and hardware sensor on the watch

H0: (μ_{test} - $\mu_{\text{real}} = 0$)
Alternative hypothesis - There is a difference in power consumption between using test sensor and hardware sensor on the watch
H1:(μ_{test} - $\mu_{\text{real}} \neq 0$)

6.3.4 Subject Selection

For this experiment we will use a batch of expressions targeting multiple sensors available on the smartwatch. The batch for phone based tests contains the following expressions:

- `self@wear_movement:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `self@wear_gamerotation:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `self@wear_linearacceleration:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `self@wear_gravity:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `self@wear_heartrate:heart_rate?delay={$delay}$server_storage=FALSE{ANY,0}`

The batch for wear based expressions contains the following expressions:

- `wear@movement:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `wear@gamerotation:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `wear@linearacceleration:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `wear@gravity:x?delay={$delay}$server_storage=FALSE{ANY,0}`
- `wear@heartrate:heart_rate?delay={$delay}$server_storage=FALSE{ANY,0}`

The **{$delay}** will be replaced by the appropriate value when registering an expression

To evaluate the performance of the real sensor compared to the test sensor, we will run the test sensor for the same amount of values and we will compare its power consumption with power consumption from hardware sensors. There will be 2 implementations of the test sensor, one for each data acquisition approach.

6.3.5 Experiment Design

For the first part of the experiment we will have two factors to vary: **data acquisition approach** and **delay**. This will give us four different scenarios to test:

Factor A: Data acquisition Method			
Phone Based Evaluation		Wear Based Evaluation	
Factor B: Delay		Factor B: Delay	
Delay: 100ms	Delay: 1000 ms	Delay: 100ms	Delay: 1000 ms
3x Experiments	3x Experiments	3x Experiments	3x Experiments

The second part of the experiment will focus on quantifying the discrepancy between power consumption when using a test sensor versus using a real, hardware sensor. In this test scenario we also apply delay of 100 ms and 1000ms, to be fully aware of the implications of all factors.

Delay 100 ms:

Factor A: Data acquisition Method			
Phone Based Evaluation		Wear Based Evaluation	
Factor B: Sensor Type		Factor B: Sensor Type	
Real Sensor	Test Sensor	Real Sensor	Test Sensor
3x Experiments	3x Experiments	3x Experiments	3x Experiments

Delay 1000 ms:

Factor A: Data acquisition Method			
Phone Based Evaluation		Wear Based Evaluation	
Factor B: Sensor Type		Factor B: Sensor Type	
Real Sensor	Test Sensor	Real Sensor	Test Sensor
3x Experiments	3x Experiments	3x Experiments	3x Experiments

6.3.6 Threats to Validity

External: The experiment is being performed on a specific setup. Even if the devices in our tests are chosen to represent the reference, the Android market is fragmented, and different Android versions, hardware and smartwatches can yield to a different result

Internal: Basing our power consumption results on battery levels increase the total error rate. Nonlinear discharge rate, battery wear level[22] can influence the final conclusion. To avoid these battery limitations, we will perform tests only when the battery is fully charged, so the discharge pattern will be the same. Also to reduce the battery wear impact, the tests will be performed with limited time delay, and without any phone or smartwatch usage between the tests.

Internal: The batch execution of the expression can induce extra error if different types of sensors have different power consumption. We will try to limit the impact of this error type by proving that the selection of the sensors has the same power draw by performing single sensor benchmark with initial full battery.

Internal: The communication using Bluetooth may vary energy consumption based on distance or obstacles between devices. We minimize the impact of Bluetooth, by placing phone and smartwatch in close proximity with no obstacles in between.

External: Heart rate sensor can be unpredictable. In some circumstances, the heart rate sensor can stop giving values. Since we were not wearing the smartwatch during our test, we placed it on the charging dock, with no charging cable connected and the phone in close proximity. By satisfying these conditions, the experiment setup and results can be verified.

6.3.7 Instrumentation

The following devices are available to perform our experiment:

- Phone - Nexus 6P - manufactured by Huawei, with latest Android 6.0 Marshmallow installed and with Android security patch level: 1 June 2016.
- Smartwatch - Motorola 360 gen 2, with latest Android 6.0 Marshmallow, android wear version 1.5

We could have a bigger selection of phones to run SWAN on them, since we only have one smartwatch available, the different processors and Bluetooth chips might induce more randomness in our observations. Besides we can argue that Nexus phones are used by default by lots of smartwatch manufacturers for tests. Before proceeding to experiment setup, the devices were factory reset and the only Android Wear app necessary for smartwatch connection was installed. We observed that by default a lot of Google applications are installed on the Nexus Phone. Some of them also have watch apps packaged. To avoid possible battery drain from other applications we decided to disable the following apps on the Nexus Phone:

- Google Drive
- Google Play Games
- Google Play Music
- Hangouts
- Google Maps
- Google Photos
- Youtube
- Google Docs

Measuring power consumption of both approaches requires us to have accurate tools. The power consumption should be measured on both phone and Android Wear device, so we could better understand what are the pros and cons of using each method. We have two methods of measuring the power consumption:

- Using the Android battery starts, which provide the current power draw if the device is equipped with a hardware fuel gauge(Our test devices Nexus 6P and Moto 360 gen 2)[6].
- The standard setup: Running each method for a prolonged amount of time and measure the battery level after the experiment is done.

Using the hardware power gauge is a very useful feature to measure the power consumption. Unfortunately, it suffers from two major disadvantages:

- We need to run another battery sensor, so the results may not reflect the constant power consumption of the SWAN application running.
- Hardware fuel gauge has its own limitations, and the computed power usage may not be accurate.

Using battery level measurement can also be affected by battery maximum capacity, charge-discharge cycle, but since we repeat the experiment for both options on the same hardware, and without delay we can argue that results can be compared. Additionally, measuring the battery levels allow us to have a better average value.

6.3.8 Experiment Execution

After initial research we concluded that the data from hardware fuel gauge is highly inaccurate on Android Wear. The reason is exceptionally power efficient hardware and a high error rate for the power consumption sensor. Also we are unable to test what is the power implication of running the SWAN WEAR foreground service. In order to obtain some power metrics, we will monitor battery levels on the phone and the smartwatch after the SWAN expression returns a given amount of sensor values, in both scenarios. The results that we get will be analysed based on power consumption on both phone and smartwatch. The experiment steps should be performed in the following order:

- Factory reset both phone and smartwatch.
- Pair smartwatch to the phone
- Install SWAN on the phone and smartwatch
- Run a preliminary test, to see if SWAN is ready to operate
- Charge both phone and smartwatch to 100%
- Install test application
- Disable Wi-Fi and radio, leave only Bluetooth
- Start test application and turn off the screen of the phone
- Test application should take the phone and wear battery levels and remaining Power values before starting test evaluations
- Wait for application to finish the test suite
- Test application will take the battery levels and remaining power values after the tests are finished and will write them on local storage for further analysis

Mention how many times you conducted each experiment

For the testing purpose, we will avoid sensors that are dependent of user motion to actually work:

- Step counter sensor - will send updates only if the user walked after the last query. It might also be related to hardware implementation of the sensor in our test device[9].
- Light sensor - the updates are triggered by changes in light level and since our test setup is configured to run for a long time, we cannot include a sensor triggered by external, undefined behaviour

Surprisingly, the Heart Rate Sensor will still give values even if it is not mounted on the wrist, so we can use it for our test purpose. The heart rate sensor is different compared to regular sensors found on the phone. The smallest granularity for heart rate results is one second, so specifying a smaller delay will not make any sense. Delays bigger than one second instead will be fulfilled by SWAN implementation. Unfortunately, due to different delay, we could not prove that the heart rate sensor consumes the same amount of power as the rest of the sensors in the test suite. We still provide our findings, but we will not include heart rate in power comparisons.

6.3.9 Preliminary Tests

Preliminary tests shows us that we cannot rely on the estimated power reported by the fuel gauge sensors. In case of our phone device, Nexus 6P, the reported values in Nano-watts were not accurate at all. Instead, the current discharge current was reported, but we can not use it in our tests, since it will interfere with our test setup. We encountered the similar problem on our smartwatch, Motorola 360 generation 2. The reported value in Nano-watts was not corresponding with the maximum theoretical energy that the watch battery is able to hold, according to the specifications.

To get comparable results we are supposed to guarantee equal conditions for both test scenarios. Since SWAN is quite power efficient and sensors that we are targeting are also very power efficient, the test should run for a long period of time. We estimated that a notable difference in battery levels should occur after running SWAN expressions for around two hours' period. Running tests for a longer period of time can help us identify even smaller differences, but the necessity of running the experiments with different types of expressions and also the number of repetitions for each scenario make the use of longer runtimes prohibitive. Also we should take into consideration the charging time necessary after each run of the experiment, with smartwatch normally losing the biggest battery percentage and having long recharge times.

6.3.10 Test Application

The test application was built in order to run tests in batch order and gather the battery level data. To avoid power consumption from external sources, such as phone screen, we implemented the testing code as foreground service[4] which is not hibernated by the Android system when the screen is off. The testing application has the following workflow: For each SWAN expression:

- Register a SWAN expression and get the battery level of the phone before the experiment
- Register a SWAN expression and get the battery level of the smartwatch before the experiment
- Register the start time
- Calculate the number of values necessary for the run, based on the given expected runtime in microseconds
- Register the experiment expressions and wait until the number of values are reached
- Register expression end time
- Get the battery level of the phone after the experiment
- Get the battery level of the smartwatch after the experiment

Once the tests for the expression are done, the application will write on local storage, in a text file the expressions name, battery levels before and after for each device the number of values received and the total runtime. The values are comma separated, so they can be imported in Excel for further processing.

Even if the program runs in a separate service, we are not allowed to block in the main thread. We are actually required to wait for the SWAN expression to terminate, so we sent the computation in another thread and we used Latch to wait for expression execution.

6.4 Results

We performed all experiments according to the Experiment Design and we obtained a lot of data. To improve readability, we decided not to include all the raw data in this chapter. If you are interested to see the raw data, check Appendix A, for tables containing the average values of 3 experiments.

6.4.1 Preliminary Analysis

We present the results of our data in graphs below. The Figures 6.1-6.5 contain the data relevant for our hypothesis testing.

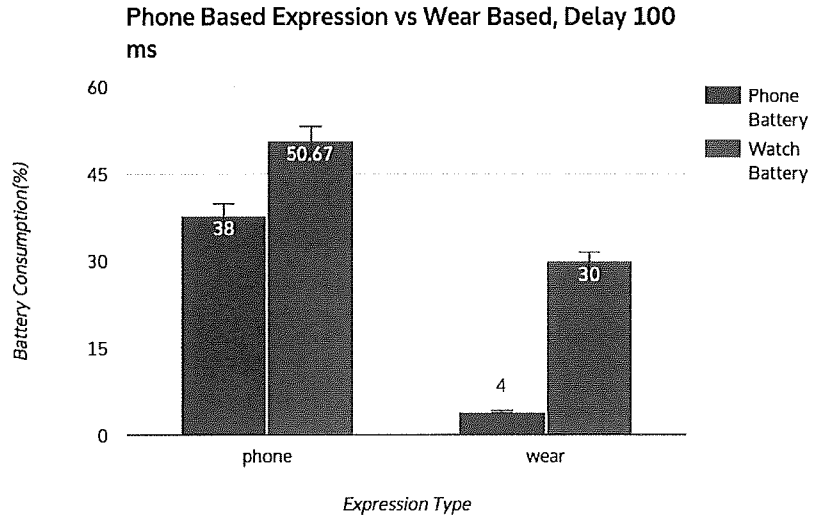

FIGURE 6.1: Phone vs Wear Based Expressions, delay 100 ms

Figure 6.1 represents the power consumption of phone based expressions and wear based expressions for very fast delay, and fixed number of received values. As we can see, the phone based SWAN expressions are highly inefficient, phone battery dropped **34%** more and watch battery dropped **20 %** more than watch based SWAN expression.

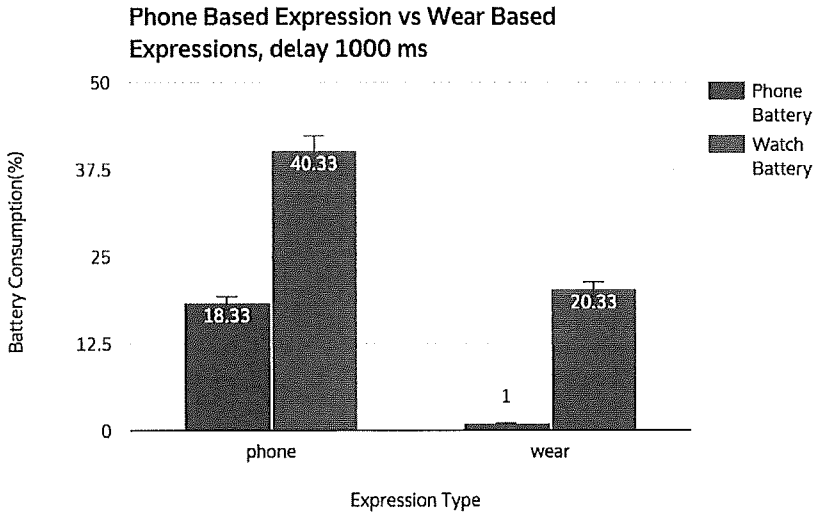

FIGURE 6.2: Phone vs Wear Based Expressions, delay 1000 ms

Figure 6.2 represents the power consumption of phone based expressions and wear based expression for normal delay, also with fixed number of received values. This test scenario

highlights the very good power savings for phone battery, with only **1%** drop in phone battery for watch based SWAN expressions. Phone based expressions perform better than in fast delay scenario, but still very power inefficient for both watch and phone.

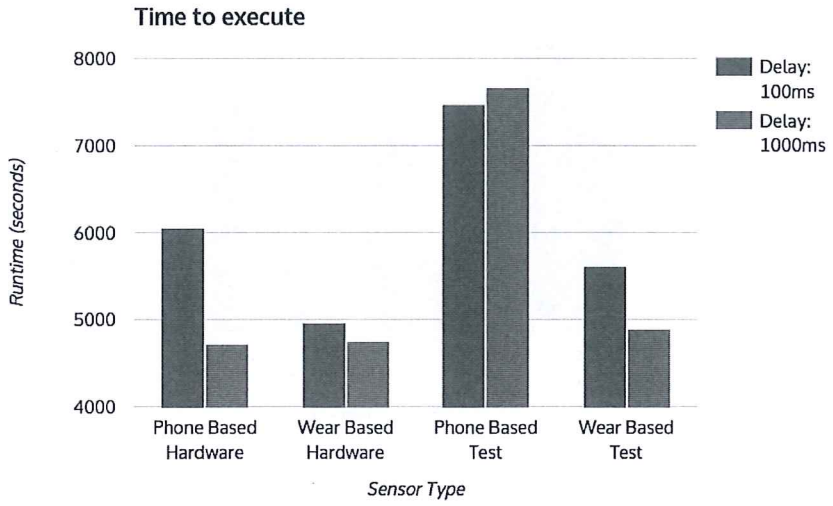

FIGURE 6.3: Run time of different number of sensors, baseline 4800 seconds

As part of the experiment, we noticed that different types of expression have different runtimes for the same number of recorded values. The expected run time for all expressions, given the delay, was 4800 seconds. Figure 6.3 shows the execution times of wear and phone based expressions, for both real sensors and our software test sensor. Longer runtimes translate into more battery consumption, but we will not take the run times into consideration, and we will focus on the fixed amount of values that our test application should receive.

The graphs show clearly that phone based expressions are not efficient. We should further validate our approach by excluding the variable update rate from a real hardware sensor. Test sensor will always send updates with the specified delay, and as we can see in Figure 6.4, the shorter delay from hardware sensor is responsible for more power used.

For watch based expressions, the phone battery improvements are non-existent(or, at least do not surpass our error threshold the be considered a difference). On the other hand, a stable delay still translates into **5 to 10%** less battery consumed on the watch as we can see in Figure 6.5

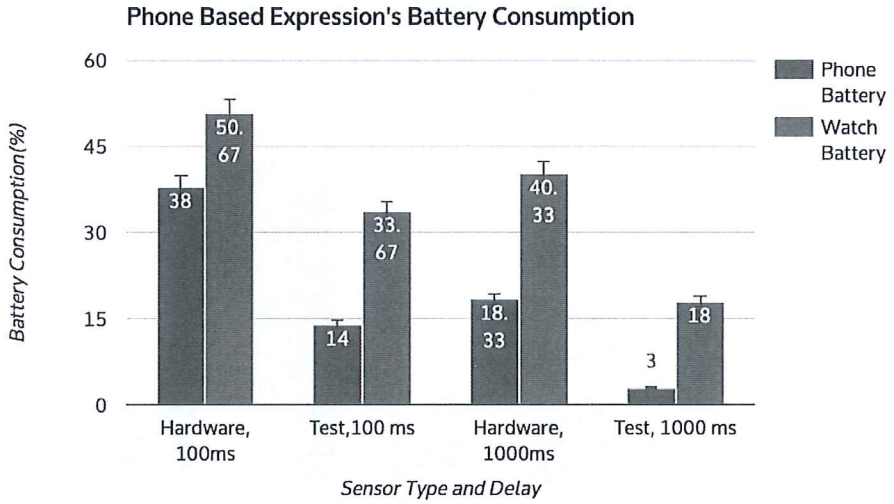

FIGURE 6.4: Phone Based Expression Power Consumption

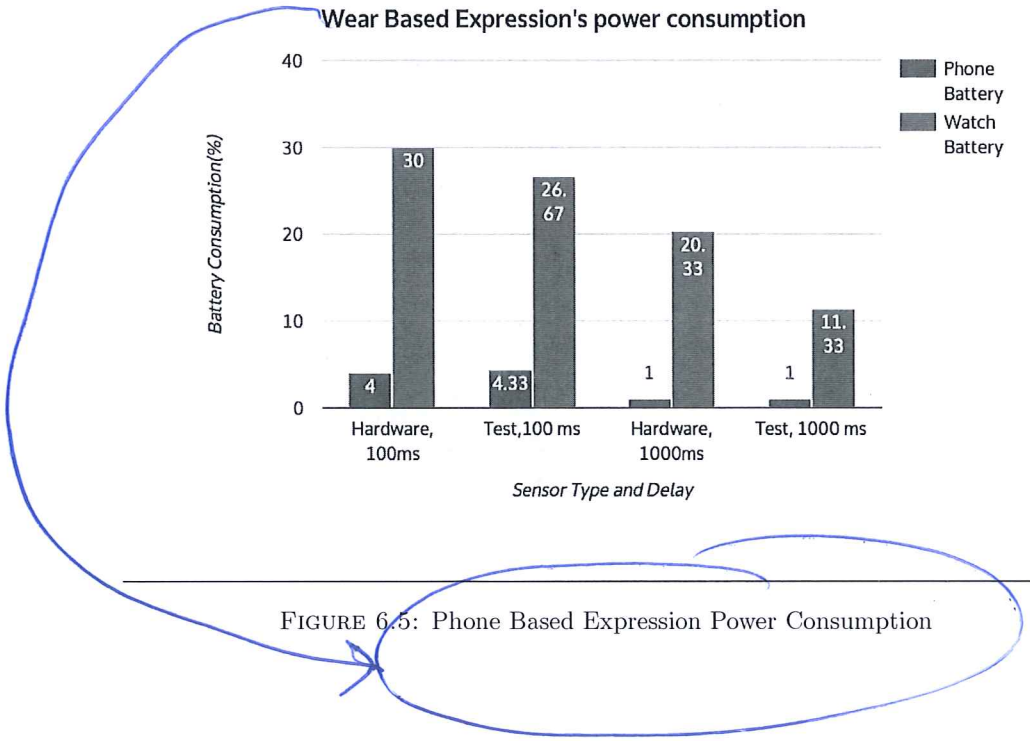

FIGURE 6.5: Phone Based Expression Power Consumption

6.5 Hypothesis Testing

After preliminary data analysis we notice that we don't have the special case, so we can proceed to test out hypothesis.

Research Question 1: What is the overhead of gathering data from the smartwatch, compared to running evaluation on the watch?
Following the formulas described in Hypothesis Formulation section, we observe that for delay of 100 ms, phone based expressions consume **34% more** phone battery and **20% more** smartwatch battery. Taking into account the 5% error by the imperfection of our instruments, we reject the Hypothesis H0, and accept the Hypothesis H1, for the given delay of 100 ms.

For given delay of 1000ms, we observe that phone based expressions consume **17% more** phone battery and **20% more** smartwatch battery. Taking into account the 5% error by the imperfection of our instruments, we reject the Hypothesis H0, and accept the Hypothesis H1.

Research Question 2: What is the overhead of gaining data from test sensor compared to real hardware sensor?
Following our further research of why the phone based expressions consume so much power, we wanted to test the sensor update delay, as a factor which increases the power consumption. Our test sensor implementation always honours the delay specified in the expression, compared to a very weak delay guarantee given by a real hardware sensor.

For given delay of 100ms, hardware sensor consumed **14% more** phone battery and **17% more** smartwatch battery than the test sensor, assuming that both test and hardware sensor is implemented as phone based expression. Taking into account the 5% error by the imperfection of our instruments, we reject the Hypothesis H0 and accept the Hypothesis H1.

For given delay of 1000ms, hardware sensor consumed **15% more** phone battery and **22% more** smartwatch battery than the test sensor, assuming that both test and hardware sensor is implemented as phone based expression. Taking into account the 5% error by the imperfection of our instruments, we reject the Hypothesis H0 and accept the Hypothesis H1.

Comparing the wear based expressions is more complicated. We could not record any notable differences on phone battery during our tests, so we will focus on smartwatch battery.

For given delay of 100ms, hardware based sensor consumed **3% more** watch battery than the test sensor. Our 5% error rate requires to have a difference of at least 1.5% in battery levels (5% of 30% battery drop is equal to 1.5%). We reject hypothesis H0 and accept Hypothesis H1.

For given delay of 1000ms, hardware based sensor consumed **9% more** watch battery than the test sensor. Our 5% error rate requires to have a difference of at least 1% in battery levels (5% of 20% battery drop is equal to 1%). We reject hypothesis H0 and accept Hypothesis H1.

6.6 Final Results Discussion

After performing all the tests from our suite, we revealed that using phone based expression in current SWAN implementation is not a good idea.

Specifically, the phone is getting hot on low delays and the phone battery drop is one order of magnitude higher than using SWAN expression evaluation on the watch.

After careful analysis we identified two possible reasons why phone based expression consumes so much power, but also what can further improve the smartwatch based expressions:

- The hardware sensor does not honour the requested delay:
  - A lot of values are being sent over Bluetooth and the phone has to drop them because they do not arrive in the given timestamp.
  - Wear based expression drop values locally, so no extra values are sent over Bluetooth.
  - Phone based test sensor show much better battery when the values arrive at given delay.
- The SWAN mechanism of accepting values drops too many values, that were supposed to be accepted[2]:
  - The Bluetooth and synchronization delay change the arrival time of the values, and the value-accept-function takes into consideration arrival time, not the time when values were recorded.
  - Test Sensor revealed poor performance of the accept value implementation, which translates two times longer runtimes than predicted in our test scenarios.

[2]See Figure 6.6 and the explanation below

– Having the expected runtime as close as possible to real runtime can save battery

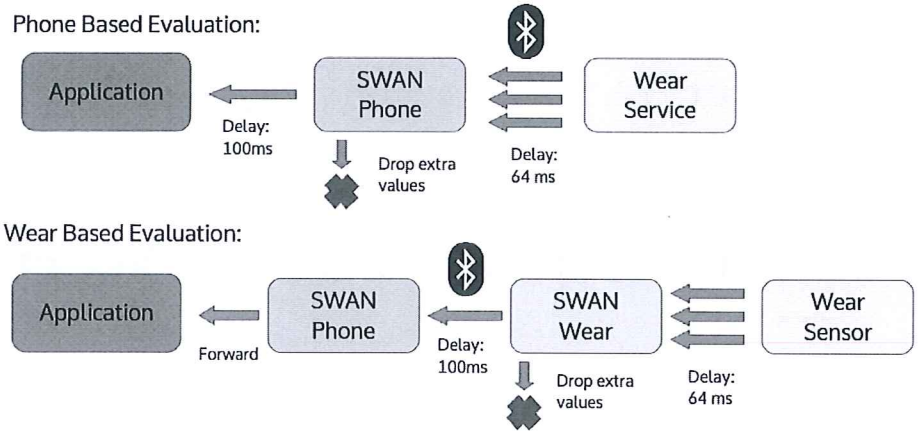

FIGURE 6.6: Dropping values in Phone and Wear Based Expressions

Furthermore, we noticed that hardware sensors will not enforce the delay and SWAN should take care about this. As we can see in Figure 6.6, phone based approach will drop the values after they were sent through Bluetooth, while in the wear based expressions, values are dropped locally, which translates into less power consumption. Even with out test sensor, some values are dropped because of the extra delay incurred by Bluetooth.

The accept values implementation performance needs to be further investigated. As the result of our experiments we can conclude that the test sensor, with ideal delay, has lower battery consumption. The excessive runtime as side effect of the test sensors, and log analysis(we record when the values are being dropped) give us some clues to the problem, but we didn't manage to fully isolate the problem so we could say for sure that the accept method is responsible for long runtimes.

Chapter 7

Conclusion and Future Work

7.1 Conclusions

The master thesis presents the use of new technologies, such as smart watches and beacons in the context of mobile phone applications. The added support for beacon based sensors allowed us to provide specifications for data formats which can be used as guidelines for adding support of other sensor types. We proposed two specification drafts and we opted for implementing the option which has the best compatibility with the existing implementation. We noticed that the beacon market suffers from very high fragmentation despite the effort from Google and Apple to offer a standard. However, we managed to overcome this obstacle by using a single, open-source library for parsing. Moreover, we proved that even a closed source format such as Estimote Nearable can be easily analysed and supported without the use of the proprietary development kit.

We proposed and implemented two methods of accessing smartwatch features and provided an extensive power analysis of using them in real life scenarios. As results of our research we demonstrated that SWAN is able to run on any Android Wear device. The SWAN running on the smartwatch was able to surpass simple data collection in any test scenario in power efficiency, providing superior power savings on both phone and watch.

7.2 Future Work

Our power saving tests revealed inconsistent run times for the same amount of requested values and constant delay. For some test cases we noticed a run time value twice as high as expected. Long run times can also translate in poor power efficiency. We were able

to identify few problems with ~~SWAN~~ delay enforcing ~~implementation~~, but the problem should be further investigated.

We were able to prove that multiple sensors of the same type can be integrated with SWAN, by integrating support for multiple beacons and implementing few test sensors to retrieve values. Unfortunately, we were only able to validate the sensors with Swan Monitor[10] application. Building a real proof-of-concept application that use beacons to present real use cases should be done in near future.

General remark:

→ Throughout the paper use either past or present tense, BUT NOT BOTH

→ Try to seperate the background section from your contribution section (Chapter 4 & 5). Because There should be a clear seperation between existing work and your Contribution

→ Remove all bold ~~letter~~ words except for headings.

Appendix A

Power Analysis Data

We recorded a lot of data as part of the power consumption experiment. We do not include it in the main thesis body, but we offer the reader the possibility of analyzing and reading raw data.

All the tables from below include the average of 3 experiments performed. The average function used it the AVERAGE macro from Google Docs.

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	12000	100	100	94.67	86.67	1521.67
gamerotation	12000	94.67	86.67	87.67	74	1517.67
linear acceleration	12000	87.67	74	80.67	62	1507
gravity	12000	80.67	62	73.67	49.33	1515
heartrate	1200	73.67	9.33	73.33	46.33	1288

TABLE A.1: Phone Based Expression with delay 100 ms

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	1200	100	100	96.67	89.33	1178.33
gamerotation	1200	96.67	89.33	92	79.33	1178
linear acceleration	1200	92	79.33	86.67	69.33	1178
gravity	1200	86.67	69.33	81.67	59.67	1178.33
heartrate	1200	81.67	59.67	80.67	55.33	1987.33

TABLE A.2: Phone Based Expression with delay 1000 ms

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	12000	100	100	100	92	1254
gamerotation	12000	100	92	98.33	85	1248.67
linear acceleration	12000	98.33	85	97.33	77	1208.67
gravity	12000	97.33	77	96	70	1253.67
heartrate	1200	96	70	95.33	67	1279

TABLE A.3: Wear Based Expression with delay: 100 ms

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	1200	100	100	100	94.33	1189
gamerotation	1200	100	94.33	100	89.67	1176.67
linear acceleration	1200	100	89.67	100	84.67	1203.67
gravity	1200	100	84.67	99	79.67	1190
heartrate	1200	99	79.67	98	75.33	2201.33

TABLE A.4: Wear Based Expression with delay: 1000 ms

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	12000	100	100	95	87	1506
gamerotation	12000	100	100	95	87	1515
linear acceleration	12000	100	100	95	87	1513
gravity	12000	100	100	95	87	1514
test sensor	12000	100	100	97	90	1862

TABLE A.5: Individual Phone Based Expression power consumption, delay 100 ms

Sensor Name	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
movement	12000	100	100	100	91	1281
gamerotation	12000	100	100	100	91	1265
linear acceleration	12000	100	100	100	91	1207
gravity	12000	100	100	100	91	1214
test sensor	12000	100	100	100	92	1325

TABLE A.6: Individual Wear Based Expression power consumption, delay 100 ms

Sensor Location	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
phone	48000	100	100	86	66.33	7480.33
wear	48000	100	100	95.67	73.33	5617

TABLE A.7: Phone and Wear based Test Sensor, delay 100 ms

Sensor Location	Number of values	Phone battery before	Smart watch battery before	Phone battery after	Smart watch battery after	Runtime (sec)
phone	4800	100	100	97	82	7663.67
wear	4800	100	100	99	88.67	4899

TABLE A.8: Phone and Wear based Test Sensor, delay 1000 ms

Appendix B

Merging SWAN Repositories

At the beginning of the master thesis project, the main repository of SWAN has holding the stable version of it, which was being used in production by a company. Our SWAN project has a big team, each member work on different part of the SWAN or SWAN related functionality. To avoid any breaking changes to the current SWAN version, I decided to make a different branch and put all my smartwatch related changes in it. When I was done with implementing the first prototype of SWAN WEAR sensors, other PhD students' projects: SWAN Plus[17] and IoT SWAN[19] were also ready. Unfortunately, the commits of the projects stated above were in different repositories so merging them into master branch was proven to be difficult. We followed this procedure when merging the SWAN:

- Create the release branch with the old, stable SWAN
- Create development branches for SWAN Plus and IoT SWAN
- Export commits from SWAN Plus and IoT SWAN as patches
- Apply the patches to each development branch
- Merge Smartwatch code into Master
- Merge SWAN Plus into master and resolve conflicts
- Merge IoT SWAN into master and resolve conflicts

The most challenging part was to apply patches from the two different repositories. Merging commits were the reason why some history was lost, and patches were hard to apply. This also proves why you should always use the rebase option instead of merging option when you push commits to the remote. Atlassian article[7] explains the main advantage of rebasing over using merging commits.

Bibliography

[1] Aar format specifications. http://tools.android.com/tech-docs/new-build-system/aar-format. Accessed: 2016-07-11.

[2] Android wear communication, data path sync. https://developer.android.com/training/wearables/data-layer/data-items.html. Accessed: 2016-07-11.

[3] Estimote. http://estimote.com/. Accessed: 2016-07-11.

[4] Google developers: Running service in foreground. https://developer.android.com/guide/components/services.html#Foreground. Accessed: 2016-07-11.

[5] Jcenter main page. https://bintray.com/bintray/jcenter. Accessed: 2016-07-11.

[6] Maxim fuel gauge hardware. https://www.maximintegrated.com/en/products/power/battery-management/battery-fuel-gauges.html. Accessed: 2016-07-11.

[7] Merging vs. rebasing. https://www.atlassian.com/git/tutorials/merging-vs-rebasing/workflow-walkthrough. Accessed: 2016-07-11.

[8] Radius networks website. http://www.radiusnetworks.com. Accessed: 2016-07-26.

[9] Step counter accuracy of motorolla 360. http://forums.androidcentral.com/moto-360/532428-step-counter-accuracy.html. Accessed: 2016-07-11.

[10] Swan monitor application. https://github.com/swandroid/SwanMonitor. Accessed: 2016-07-11.

[11] Swan wiki page: Managing libraries. https://github.com/swandroid/swan-sense-studio/wiki/Managing-swaninterface-and-swansong-jcenter()-libraries. Accessed: 2016-07-11.

[12] Tutorial to upload your android libraries to jcenter. `https://inthecheesefactory.com/blog/how-to-upload-library-to-jcenter-maven-central-as-dependency/en`. Accessed: 2016-07-11.

[13] Ieee standard definitions of terms for antennas. *IEEE Std 145-1993*, December 1993.

[14] V. R. Basili. *Software modeling and measurement - the Goal-Question-Metric paradigm.* 1992.

[15] H. Koziolek. *Dependability metrics: Goal-Question-Metric paradigm.* Springer-Verlag Berlin, Heidelberg, 2008.

[16] A. Levy, J. Hong, L. Riliskis, P. Levis, and K. Winstein. Beetle: Flexible Communication for Bluetooth Low Energy. In *Proceedings of the 14th International Conference on Mobile Systems, Applications and Services (MobiSys)*, June 2016.

[17] H. B. Nicolae Vladimir Bozdog, Roshan Bharath Das. Swan-Lake: Opportunistic Distributed Sensing for Android Smartphones. November 2016.

[18] F. X. L. Renju Liu. Understanding the Characteristics of Android Wear OS. In *Proceedings of the 14th International Conference on Mobile Systems, Applications and Services (MobiSys)*, June 2016.

[19] A. v. H. H. B. Roshan Bharath Das. Swan-fly : A flexible cloud-enabled framework for context-aware applications in smartphones. *Sensors to Cloud Architectures Workshop (SCAW-2016)*, (2), March 2016.

[20] R. R. C. Sheng Shen, He Wang. I am a Smartwatch and I can Track my Users Arm. In *Proceedings of the 14th International Conference on Mobile Systems, Applications and Services (MobiSys)*, June 2016.

[21] N. Sovaiala. A layered storage design for swan system. Master's thesis, Vrije Universiteit Amsterdam, 2016.

[22] J. Winter, M.; Brodd. What are batteries, fuel cells, and supercapacitors? *Chemical Reviews 104*, (12), September 2004.

[23] D. H. Wolaver. *Phase-Locked Loop Circuit Design.* 1991.