Design of a framework for the automatic detection of context on the Android platform

Michiel Creve

Supervisor: Prof. dr. ir. Luc Martens
Counsellor: Dr. ir. Toon De Pessemier

Master's dissertation submitted in order to obtain the academic degree of Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Daniël De Zutter
Faculty of Engineering and Architecture
Academic year 2015-2016
Design of a framework for the automatic detection of context on the Android platform

Michiel Creve

Supervisor: Prof. dr. ir. Luc Martens
Counsellor: Dr. ir. Toon De Pessemier

Master's dissertation submitted in order to obtain the academic degree of Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Daniël De Zutter
Faculty of Engineering and Architecture
Academic year 2015-2016
Preface

In this way I would like to thank everybody that helped me to realise this dissertation. First, I would like to thank my counsellor dr. ir. Toon De Pesseiner for the useful guidance throughout the year. Next, I would like also express my gratitude to my promotor prof. dr. ir. Luc Martens for making it possible to work on this interesting subject.

Special thanks also go out to my eleven test users for willing to sacrifice some of their valuable time and smartphone battery life. Without them I could not have validated the correct functioning of the framework.

I also want to express gratitude to my fellow students and friends for helping to solve problems, giving useful input and feedback and their pleasant company throughout the years. Finally, a special word of thanks goes out to my girlfriend Ellen for all the support and to my parents, who gave me the opportunity to go for a second degree after getting a first one.

Michiel Creve, June 2016
Permission for use of content

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use.

In the case of any other use, the limitations of the copyright have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

Michiel Creve, June 2016
Design of a framework for the automatic detection of context on the Android platform

by

Michiel CREVE

Promotor: Prof. Dr. Ir. L. MARTENS
Counsellor: Dr. Ir. T. DE PESSEMIER

Master’s dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Daniël De Zutter
Faculty of Engineering and Architecture
Academic year 2015-2016

Summary

In today’s commerce, personal recommendations take an important role. Advertising products at a correct time to the correct customer has a positive impact on both the customer and the company. The company will generate more revenue and the customer will probably be less frustrated about annoying commercials. However, in order to provide the customers with the correct advertisements, the company should first get to know its customers. Collecting useful information, which can be a cumbersome task, should in fact not be the main activity while developing a commerce application. Therefore, this dissertation proposes a framework that collects information about the context of the users. This way the developers are released of the burden of collecting this information, such that they can solely focus on creating the commerce application itself. The framework first captures the user’s data, after which it is processed. The processing module consists of three submodules: a module for detecting and predicting the activity level of the user, for detecting and predicting the locations of the user and for labeling these locations in order to know to what kind of locations the user goes. The information generated by all modules is then made accessible through an API: the Context Façade. A user test has been conducted with participants of different categories. The results of this test show that the framework is capable of detecting and predicting the different user’s activity level and location pattern.

Keywords

Context recognition, Activity and location recognition and prediction, framework
Design of a framework for the automatic detection of context on the Android platform

Michiel Creve
Dr. Ir. Toon De Pessemier (Counsellor)
Prof. Dr. Ir. Luc Martens (Promotor)

Abstract—In today’s commerce, personal recommendations take an important role. Advertising products at a correct time to the correct customer has a positive impact on both the customer and the company. The company will generate more revenue and the customer will probably be less frustrated about annoying commercials. However, in order to provide the customers with the correct advertisements, the company should first get to know its customers. Collecting useful information, which can be a cumbersome task, should in fact not be the main activity while developing a commerce application. Therefore, this dissertation proposes a framework that collects information about the context of the users. This way the developers are released of the burden of collecting this information, such that they can solely focus on creating the commerce application itself. The framework first captures the user’s data, after which it is processed. The processing module consists of three submodules: a module for detecting and predicting the activity level of the user, for detecting and predicting the locations of the user and for labeling these locations in order to know what kind of locations the user goes. The information generated by all modules is then made accessible through an API: the Context Facade. A user test has been conducted with participants of different categories. The results of this test show that the framework is capable of detecting and predicting the different user’s activity level and location pattern.

Index Terms—Context recognition, Markov chains, Activity recognition and prediction, Location recognition and prediction, framework, extended abstract

1 INTRODUCTION

In order to provide accurate predictions, the developers of recommender system should first get to know their customers. To do this, information about the user’s context has to be gathered. In this work, the definition of context given by Abowd et al. in [1] is used: ‘Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.’ In this case, the entity is a person: the user to which the recommendations should be provided. The relevant information can be anything that is of use for recommendation systems. Thus, the solution to this problem should capture as much information about the user as possible. The current framework will detect and predict the user’s activity level and locations. It will also label the locations of the user so developers of recommender systems know what kind of locations the user visits.

First, some details of the framework will be explained in Section 2. Next, the results are presented in Section 3. Finally, a short conclusion will end this extended abstract in Section 4.

2 THE CONTEXT DETECTION FRAMEWORK

The context detection framework consists of two modules. The first module is responsible for the collection of sensor data on the user’s Android smartphone. It also captures the current activity (e.g. walking, cycling or being in a vehicle) using the Google Activity Recognition API. When this module finishes collecting a sample, it stores this information in a measurement database and alerts the second module that a new measurement is available. The second module will now process the sample. In short, the processing module will
detect and predict the user’s activity level and location. The processing module is explained in more detail in Section 2.1. The findings of the processing module are then made available through the Context Façade to enable developers of recommender system to easily query information about the user’s context.

2.1 Data Processing

As said earlier, the Data Processing module is mainly responsible for the detection and prediction of the user’s activity level and location. The detection of the activity level of the user boils down to checking the sensors of the smartphone to see if the phone is moving, has rotated, is in a noisy or illuminated environment, ... If one of the rules are triggered, the user is considered as active. The detection of the user’s location is based on the coordinates of the smartphone and the observed MAC addresses of nearby access points.

For the prediction of both the user’s activity level and locations a Markov based approach has been taken. In order to capture the pattern of an entire week, the Markov chains have a transition matrix for every 15 minutes of the week, resulting in 672 transition matrices per chain. Given the detected activity levels and locations, the transition matrices are gradually trained to reflect the user’s habits. Note that for the location predictions different sets of matrices are kept in parallel. This allows the framework to capture different types of periods, like regular working periods and holidays. Only the model that has made the best predictions at the end of week has its transition matrices updated, such that the matrices of the other models are not polluted with updates that belong to another pattern. A new set of matrices is created if no model succeeded to predict the locations sufficiently well. To make the predictions, one now has to multiply the current state with the transition matrices recursively. Two additional correction terms are added to the recursive computations in order counter some weaknesses of the models. The Data Processing module will predict the activity level and locations for the next 24 hours and updates these predictions every time a new sample has been taken. The most recent predictions are always available through the Context Façade.

Finally, the Data Processing module is also responsible for the labeling of the locations. It will automatically detect the home locations of the user, i.e. the locations where the user sleeps most often. Next to this it will also add information of Google Places and Foursquare to the locations. This makes the locations much more useful, since it is now known at what kind of locations the user is. Unfortunately, this labeling process with Google Places and Foursquare has to be manually controlled by the user. The reason for this is that not all locations exist in these services and that sometimes incorrect locations are proposed. Therefore the user has to approve whether or not the location proposals are correct.

3 Results

In this section the most important findings of the conducted tests will be presented. During the development of the framework the models were tested on one smartphone only. When the framework reached its current shape, a user test with 11 test subjects was conducted to validate the functionalities.

A first test of the activity submodule already showed that the framework is capable of both correctly detecting and predicting the activity levels of a user. Using the automatic detection functions in the Context Façade, the errors on the detection of waking up and going to sleep over a period of 28 days were 6 minutes and 11 minutes respectively. The errors on the predictions were, logically, somewhat higher: 40 minutes for the prediction of the time of waking up and 33 minutes for the time of going to sleep. This test was conducted when the framework was already running for over 20 weeks. Later, in the user test, the framework had only been running for about 5 to 6 weeks. The errors on the predictions during the user test were 69 minutes and 116 minutes for waking up and going to sleep respectively. This worse performance can be attributed to the fact that the framework is not sufficiently trained in the beginning. This shows that the
framework needs a period to warm up and that its predictions get better over the weeks.

Similar tests have been conducted for the location submodule. The error measure used to evaluate the location predictions is the logarithmic loss: 
\[ -\ln(p_{\text{correct}}) \]. \( p_{\text{correct}} \) is the predicted probability of the location that afterwards turned out to be the correct one. Thus, the logarithmic loss will be low for correct predictions with high confidence and high for incorrect predictions. In the tests during the development the results were as expected. A second model was automatically created at the start of the academic year, because the current model at that time was trained during the holiday and could not predict the new pattern. It is also observed that the loss decreased over the weeks and was lower during regular periods (e.g. school) than during more irregular periods (e.g. holiday). The user test also showed a decreasing logarithmic loss over the weeks, which indicates that the model actually learns the user’s habits. However, the creation of new models did not turn out as expected. It was expected that during the Easter Holiday some students would create a new model because their pattern would be different. However, no one reached the threshold to create a new model, except for one student that had a rather irregular week. It is clear that this calls for a new way of evaluating the models that should be implemented in the future, like for example ignoring the home locations of the users in the evaluation. During a holiday most people will still sleep at the same location, such that by ignoring the home locations, the real differences in the pattern are more focused on.

4 Conclusion
The results show that the framework is capable of detecting, learning and predicting the user’s activity level and location patterns using a Markov based mechanism. There still are some weaknesses (e.g. the creation of new location models) that should be fixed in the future, but possible solutions for most of these weaknesses are already proposed. The framework currently has limited functionality (mainly activity and location detection and prediction), but it should be fairly easy to add new modules.

REFERENCES
Contents

Preface i
Permission for use of content ii
Overview iii
Extended Abstract iv
Contents vii
List of Figures x
List of Tables xii
List of Acronyms xiii

1 Introduction 1
  1.1 Problem definition 1
  1.2 Solution 2
  1.3 Overview 2

2 Literature 4
  2.1 Existing services 4
     2.1.1 Foursquare 4
     2.1.2 Swarm by Foursquare 5
     2.1.3 Google Places API 5
     2.1.4 Google Location History 5
     2.1.5 Activity tracking apps 6
     2.1.6 Google Now 6
     2.1.7 Bento Homescreen 6
     2.1.8 Tasker 7
     2.1.9 CyanogenMod’s built in system profiles 8
  2.2 Ongoing research 9
     2.2.1 Platys: An active learning framework for place-aware application development 9
     2.2.2 Basic activity detection 11
     2.2.3 Activity detection and prediction 14
List of Figures

2.1 Bento homescreen vs. regular homescreen ............................................. 7
2.2 User Interface of the Tasker application ................................................... 8
2.3 CyanogenMod’s built in System Profiles feature ......................................... 9
2.4 Schematic overview of how ’Place’ is defined in [1] .................................. 10
2.5 Illustration of active learning in [1] ........................................................... 10
2.6 Standard orientation of axes on Android devices ...................................... 11
2.7 Raw accelerometer while cycling, as shown in [5] .................................... 12
2.8 Overall procedure of global-local co-training, as explained in [3] ............... 13
3.1 Overview of the proposed framework ......................................................... 18
4.1 Evolution of the activity level in the week from March 10 until March 16 .. 31
5.1 Screenshots of location module in data processing application .................. 45
5.2 Daily logarithmic loss of location predicting models before and during the first semester ................................................................. 48
5.3 Daily logarithmic loss of location predicting models during the holidays after the exams and the beginning of the second semester ....................... 49
5.4 Screenshot of incorrect most probable location during a lecture at 11.07 a.m. 49
5.5 Comparison of logarithmic loss of the a simple model and the best model during the first semester ................................................................. 50
5.6 Comparison of logarithmic loss of the a simple model and the best model during the first holidays ................................................................. 51
5.7 Evolution of the Anomaly Grade between the 22nd and 28th of February .... 52
6.1 Screenshots of location labeling process .................................................... 57
8.1 Usable activity profile that has clearly distinguishable active and inactive periods ...................................................................................................... 73
8.2 Unusable activity profile that does not have clearly distinguishable active and inactive periods ................................................................. 73
8.3 Overview of the prediction accuracy for all categories and the average prediction accuracy ................................................................. 78
8.4 Comparison of location predictions and basic model with the binary accuracy as error measure. 79
8.5 Comparison of location predictions and basic model with the logarithmic loss as error measure. 80

9.1 Screenshots of Personal Assistant giving relevant information for the next trip. 83
9.2 Screenshot of Personal Assistant giving a weather forecast for the expected time of waking up. 84
9.3 Screenshot of Personal Assistant suggesting to set the alarm. 85
9.4 Screenshot of Personal Assistant welcoming the user at the new location and suggesting to send a text message. 86
9.5 Screenshot of Personal Assistant when it is detected that the user is in a vehicle, giving the advice to drive safely. 87
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Features extracted from raw accelerometer data, as shown in [5]</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of activity level detections between ground truth, manually</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>filtered detection and Façade function detection</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of activity level prediction with ground truth</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Logarithmic loss of location models during 20 weeks</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Illustration of most probable locations during a lecture at 11.07 a.m.</td>
<td>48</td>
</tr>
<tr>
<td>8.1</td>
<td>Differences (in hh:mm) between detected and predicted times of waking up and going to sleep for all test users.</td>
<td>74</td>
</tr>
<tr>
<td>8.2</td>
<td>Average location prediction logarithmic loss over all categories</td>
<td>75</td>
</tr>
<tr>
<td>8.3</td>
<td>Location prediction logarithmic loss of category A users.</td>
<td>75</td>
</tr>
<tr>
<td>8.4</td>
<td>Location prediction logarithmic loss of category B users.</td>
<td>76</td>
</tr>
<tr>
<td>8.5</td>
<td>Location prediction logarithmic loss of category C users.</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Acronyms

AP     Access Point
API    Application Programming Interface
GLCT   Global-local co-training
GPS    Global Positioning System
HMM    Hidden Markov Model
MAC    Media Access Control
ME     Mixture of Experts
MM     Markov Model
NN     Neural Network
SSID   Service Set Identifier
SVM    Support Vector Machine
WiFi   Wireless Fidelity
Chapter 1

Introduction

Personalised recommendations are an important part of today’s commercial strategies. They can have a positive impact on both the consumers and the vendors of all kinds of products. Consumers, who in general do not want to be bothered with a lot of advertisements and recommendations, are likely to be less frustrated about it if the advertisements and recommendations are interesting. If consumers like the what they get to see, they might spend more money buying these recommended products. For this reason, a lot of research is done on how to create highly personalised recommendation systems.

1.1 Problem definition

In order to show the correct ads to his clients, the vendor should first get to know them. Unfortunately, this is not always as easy as it seems. Currently some accurate implementations of recommendation systems already exist. Some popular sectors in which recommendation systems are widely used are web stores (e.g. Amazon, Ebay, ..) or movie streaming services (e.g. Netflix, Youtube, ...). Web stores and movie streaming services build a profile that contains information of the customer. This profile is usually based on the previous purchases and preferences of the customer. According to these preferences and in combination with purchases made by other customers with a comparable taste, it is possible to suggest relevant products, which leads to higher sales.

However, in many cases it is a lot harder to gather relevant information. For example, if one wants to create a recommendation service for bars and restaurants, many factors have to be taken into account. If the weather is nice, people might prefer to sit outside in a bar with a lovely terrace. Has this person already had lunch and is he only looking for a drink or is he eager to have a nice meal? What is this person’s usual budget? These are only a few out of many questions that can help optimising the recommendation of the perfect place for a certain person. This example should make clear that gathering information is really necessary in order to provide accurate recommendations.
Many of these decision making factors from the previous example depend on the context of the user. Context is a very broad term that comprises different kinds of information. An accurate definition of context was given by Abowd et al. in [11]:

*Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

In this case, the entity is a person: the user to which the recommendations should be provided. The relevant information can be anything that is of use for recommendation systems. Thus, the solution to this problem (see Section 1.2) should capture as much information about the user as possible. Examples of context information that can be captured are: the user’s past, previous and future locations and the types of these location, the user’s activity level (awake or asleep), the current weather, the activity the user is performing (walking, running, driving,...), nearby events,...

### 1.2 Solution

In this master’s dissertation a framework to recognise the context of a person is proposed. With this framework it should be possible to query information about the user’s context, in order to create more accurate recommendation systems. Next to being more accurate, it will also facilitate the development of these recommendation systems, since the information is readily available in the framework and the developer does not have to worry about how this information has been collected.

An Android smartphone will be used to gather this context information. A smartphone is ideally suited for the purpose of context detection since the user usually always carries it with him. Furthermore, today’s smartphones usually are packed with diverse sensors. Many of these can be useful to detect the user’s context: the Global Positioning System (GPS) sensor can help deriving the user’s location, the accelerometer can be used to determine the user’s current activity,... The Android operating system allows to gather the raw data of all sensors in the device. Which sensors are used in the framework and for which purpose will be explained later in this dissertation.

### 1.3 Overview

In Chapter 2 some relevant literature and currently available related applications are discussed. Chapter 3 will give a general overview of the framework and all its modules. It also explains how all data is collected. Next, Chapter 4, Chapter 5 and Chapter 6 cover the different submodules of the data processing module of the framework: the Activity Submodule, Location Submodule and Location Labeling Submodule. After that,
it is explained in Chapter 7 how the data is made accessible for developers through the Context Façade. In Chapter 8 it is checked whether or not the framework works well when deployed on other devices in a user test. Chapter 9 describes a demo application that uses the Context Façade to query information of the framework. Finally, Chapter 10 discusses the strengths and weakness of the framework and concludes this master’s dissertation.
Chapter 2

Literature

This chapter contains a study of applications and literature that is related to this master’s thesis. The first part of the chapter shows some existing and popular services in the field of context detection and activity tracking. In the second part of this chapter, some useful and comparable state of the art research in this field will be summarised. Some of the principles mentioned in this research will also be adopted in this framework.

2.1 Existing services

In this section, some popular services and applications for context and activity detection will be given as an example. Since many of these applications and services exist in diverse domains, the list of available applications is not limited to the few examples mentioned below.

2.1.1 Foursquare

A first popular location service is Foursquare. Foursquare is a platform on which all kinds of interesting places have their own page on which people can post pictures and reviews. When you open the app on your smartphone, you can chose to actively search for a venue or place in a specific category, or you can discover nearby places. Foursquare is available on all popular platforms (Android, iOS, Windows Phone and through a regular web browser). It also has an extensive Application Programming Interface (API), with which it is possible to query nearby places with their corresponding ratings, reviews and popularity. It is also possible to refine the search and look for places in specific categories or within a certain budget range. The category tree in Foursquare is hierarchically structured and contains many sorts of venues [14]. Because of this, Foursquare is very suited to serve as location database for recommendation systems.
2.1.2 Swarm by Foursquare

Swarm is a platform created by the same developers as Foursquare. It is a social network on which people can check in at (Foursquare) places to let their friends know where they are. For developers, Swarm is available through the same API as Foursquare. Next to the possibility of looking up nearby locations, the Swarm API also contains a real-time service which can push an application everytime a user checks in at a location. This could possibly be very useful for the proposed framework, since this makes it possible to have a very accurate knowledge about the user’s physical location without having to use the (battery draining) GPS sensor.

2.1.3 Google Places API

Just like the Foursquare API, Google also provides an API through which locations and their categories can be retrieved. Where the Foursquare API contains the venues created by the users, Google Places API contains the locations that are available via Google Maps. This results in the fact that the Google Places API might contain less up-to-date information about popular (or temporary) places. Additionally, the Foursquare API has a more structured hierarchy of places in comparison with the Google Places API. An advantage of the Google Places API is that it is less complicated to implement, because it is already integrated in the Android SDK. Testing and user experience will have to decide whether the Google Places API or the Foursquare API (or a combination of both) will be most suitable for the proposed framework.

2.1.4 Google Location History

Google automatically tracks your location history if this function is enabled on your smartphone. This location history can be consulted (and edited) via a web browser, where you can browse through your past locations on a map. Since many people have this function enabled, it could be beneficial for the proposed framework to track and use this location information.

Unfortunately Google Latitude, the Google API with which location information could be retrieved programatically, retired as of August 9th, 2013 [12]. To download your location history Google now refers to Google Takeout, on which it is only possible to download location information in a file (JSON or KML formatted). The lack of an API makes it hard to collect and use this information in a program or application. There have been some people trying to create a workaround, but unfortunately these solutions are too cumbersome to use in an application. For example, Terence Eden explains on his blog how you can gather your location information programatically. Because the authentication cookies get invalidated after some time, re-authentication is needed quite often [13]. Since probably not all users will do this, this solution is not suited for the proposed framework.
2.1.5 Activity tracking apps

Another category of popular applications are activity tracking applications. This kind of applications are often used to track sporting activities. Some examples are Runkeeper, Endomondo and Strava. These applications use the GPS sensor of the smartphone to show the user’s trajectory and calculate metrics like the average pace, burned calories and so on. Among other functionalities, the user can set goals, generate statistics of his workouts or compare his activities with those of his friends.

2.1.6 Google Now

Android’s built in Google Now might be one of the biggest competitors of the proposed framework. Google Now acts as a personal companion on the user’s smartphone. It has an extensive set of functionalities. For example, it tells you how long you have to drive home from where you currently are, when you have to leave to be on time for a certain event in your calendar or where you have parked your car. Google Now also gives you a personalised feed of news articles which should be interesting for you. Just like Siri on iOS, it is also possible to give voice commands to set reminders, launch applications or perform web searches. Since Google Now is readily available on each Android device and automatically synced with the user’s Google account it already is a very popular recommender with increasing functionality.

2.1.7 Bento Homescreen

On its page in the Google Play Store, Bento is described as a contextual launcher (a launcher can be compared with a computer’s desktop). Instead of just showing app icons and some widgets, like the regular homescreen depicted in Figure 2.1b, Bento learns about you and tries to show relevant content on your homescreen. Bento is thus somewhat comparable with the standard built in Google Now. Among other things, Bento learns about the user’s musical tastes to recommend songs, interesting articles or even nearby concerts. Just like Google Now, the user also has a personalised news feed with articles that should be interesting to that particular user. A screenshot of the Bento Homescreen is shown in Figure 2.1a. Bento is currently still in a beta testing phase and only available in English.
2.1.8 Tasker

Tasker is an interesting application with which users can program their own tasks on their smartphone. The user can easily set certain triggers to perform some actions. This makes it possible to let the user program which settings the phone should have in which context (at home, at work, at school, on the road, ...). Possible settings are to disable Wireless Fidelity (WiFi) and enable mobile internet when leaving home (and vice versa), to mute all sounds when you are at a certain place or to automatically connect with the handsfree kit in your car using Bluetooth. Programming these tasks can be done through a graphical interface, as shown in Figure 2.2. The figure illustrates the categories of the actions that can be performed. More experienced users can use the Tasker specific programming language to set up larger and more complicated tasks.
2.1.9 CyanogenMod’s built in system profiles

CyanogenMod is an alternative Android operating system. It is free to install on a large set of devices and recently it also became the standard operating system on some devices (e.g. OnePlus One). CyanogenMod contains a whole collection of extra features compared to the stock Android versions. One of these features is called ‘System Profiles’, which is comparable to Tasker. With system profiles it is possible to set triggers (specific WiFi networks, Bluetooth devices or NFC tags) to change system settings like GPS, synchronisation, media volume, data connection, ... It is thus possible to automatically switch settings according to your current context. Screenshots with some example profiles and possible settings are depicted in Figure 2.3. In this example, the home WiFi network is set as trigger to use the ‘At Home’ profile.
2.2 Ongoing research

Most of the examples that were shown in the previous section are still rather basic. In this section, some ongoing research in the field of context and activity detection that reaches further than what is available on the market today will be summarised.

2.2.1 Platys: An active learning framework for place-aware application development

In [11] the authors present Platys, an active learning framework for place-aware application development. It is clear that this framework can be a source of inspiration for the framework proposed in this dissertation, since it applies similar concepts and has comparable goals. The authors give a very broad and high-level definition to the concept of 'place': "a place does not necessarily have to be a fixed physical location". For example, if a person is in the place 'School', it does not matter in which classroom or at which university campus the lecture is taking place. The physical location can differ, but this does not make any difference for the current activity taking place. To define a place, the authors combine both 'Space', 'Activity' and 'Social Circles'. A schematic overview of how places are defined is given in Figure 2.4.

Another important aspect in this paper is the active learning component. The framework will attempt to learn more about the user’s places in a proactive way. The authors illustrate this with an example to recognise WiFi networks and physical GPS locations.
In Figure 2.4 a possible timeline of a person is given, with annotations of which WiFi networks and location information were available at what time during the day.

If, for example, it is known that WiFi network $w_5$, $w_6$ and $w_7$ belong to the place 'Lab' the framework will proactively learn that WiFi network $w_4$ and location $g_2$ also belong to this place, since these are detected in combination with $w_5$ (of which it is known it is present in the lab). By acquiring this extra knowledge, the framework will also know that the user was in the lab at 14:00. This would not have been possible if only the information from 09:30 was used. A similar example can be seen when combining the information at 08:30 and 12:30 to know that GPS location $g_3$ is also at home.

This learning principle is very useful to adopt in the framework proposed in this dissertation. One should only periodically scan for available WiFi networks to have an accurate knowledge about at which place the user currently is. The learning can also be twofold: it can be applied to both the Service Set Identifier (SSID, the "name" of the network) and the medium access control (MAC) address. To get an accurate knowledge about the exact physical location, the MAC addresses of the wireless access points should be used, since
the MAC address is unique for each access point (AP). However, to recognise conceptual places like 'School', it might be a better idea to learn which SSIDs are near. For example, "eduroam" is present in a myriad of university buildings throughout the world [2]. If this SSID is detected, the framework could immediately know you are at the place called 'School'.

2.2.2 Basic activity detection

Another important goal of the proposed framework is to detect and predict the activities of the user. There are several aspects to a user's activity. Firstly there are basic activities, which determine the physical activity of the user (e.g. walking, running, cycling, standing still, ...). Secondly, activity also comprises a large set of other, higher level activities, which describe more complex activities (e.g. reading a book, drinking a beer on a terrace, studying, going to work, ...). In this section, basic activity detection will be considered. In the next few sections, the detection and prediction of some more complicated activities will be considered.

2.2.2.1 Basic activity detection using accelerometer data

A popular way to detect the physical activity of a person is by using the accelerometer data of the user’s smartphone. The data an accelerometer generates is represented by the acceleration values in \( m/s^2 \) for both the x, y and z axis. The position of the axes is the same for every Android smartphone and is fixed, unregarded the orientation of the smartphone. The standard orientation of the axes is depicted in Figure 2.6.

![Figure 2.6: Standard orientation of axes on Android devices](image)

Different users will generate data that is similar to the data generated by other users who carry their smartphone the same way (e.g. in their pockets or in a purse). Also, each kind of movement generates data that usually differs enough from other movements, such that they can be distinguished from each other.
During his master’s thesis at the University of Ghent in 2011, Ewout Meyns successfully tried to recognise these activities based on accelerometer data [5]. He managed to distinguish standing still, walking, running and cycling with an accuracy of (in some cases) 100%. The five features he extracted from the accelerometer data are listed in Table 2.1. An example graph of the raw accelerometer data is depicted in Figure 2.7.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature 1</strong></td>
</tr>
<tr>
<td><strong>Feature 2</strong></td>
</tr>
<tr>
<td><strong>Feature 3</strong></td>
</tr>
<tr>
<td><strong>Feature 4</strong></td>
</tr>
<tr>
<td><strong>Feature 5</strong></td>
</tr>
</tbody>
</table>

After the features were generated, a Support Vector Machine (SVM) was trained to classify a new set of features as the correct activity. Even though this has proven to yield satisfying results, a static approach has been applied in this work. Data has been generated and labeled to train the SVM, after which no new samples can be used to further improve the accuracy of the classification.

It could be beneficial for the classification accuracy if the system could continue its learning process with unlabeled data of which the system has a good confidence of being correct. Such an algorithm, global-local co-training (GLCT), is proposed in [4] and is applied to accelerometer data for low level activity detection in [3]. In GLCT, a set of labeled data...
(the initial training data) is used to train two models to classify new input data: a global and a local model. Where the global model takes all input data to make a decision, the local model consists of a set of classifiers, each trained with a specific subset of the labeled input data. The subsets of input data are often constructed by clustering the data, such that similar data are part of the same subset. The resulting set of classifiers of the local model is called a mixture of experts (ME). The decision of each of the experts is then fed into a gating network, resulting in one decision, based on the classification of all separate experts.

When after the initial training unlabeled data arrives at the system, this data is classified by both the global and the local model. If one of the models is confident enough about its decision, it uses the input data to train the other model. For example, if the local model is confident enough, it uses these samples to train the global model and vice versa. A schematic overview of the GLCT training process is depicted in figure 2.8. The results in [4] and [3] show that this approach indeed increases the classification accuracy. More details about this algorithm can be found in the papers mentioned above.

![Figure 2.8: Overall procedure of global-local co-training, as explained in [4]](image)

In [3] the ME is implemented by clustering all labeled input data to create different subsets which correspond to the different physical activities. These subsets are then separately used to train classifiers (the local experts), which can only classify one specific activity.

### 2.2.2.2 Basic activity detection using built-in Activity Recognition API

Even though the above mentioned methods could be successfully implemented, there might be an option using an easier implementation. Google offers an Activity Recognition API, which automatically recognises basic activities [12]. Just like in [5] and [3], the API has built-in functionality to detect walking, running, riding a bike, driving in a car and standing still. It also indicates rapid movement, in order to detect movements like picking
up the phone. If the Activity Recognition API yields comparable or better results as the above mentioned implementation, a custom implementation is not needed and more time will be available to focus on extra functionality.

2.2.3 Activity detection and prediction

The proposed framework will also contain a fairly large machine learning part, since a set of habitual activities can and should be predicted. Some possible predictable things are: sleeping pattern, location and whether or not the user has to go to work. If the framework is able to accurately predict the next activities of the user, it will also be possible to offer more accurate recommendations (e.g. someone who has to go to work will probably not be interested in some kind of event during the afternoon). Unfortunately, it is hard to predict this kind of activities deterministically, since some activities are user specific (e.g. most people do not work during the weekends, but some people do) and tend to vary over time.

2.2.3.1 Activity prediction using neural networks

One of the possible methods to overcome the complexity of some activities might be artificial neural networks (NN). The reason why neural networks can be a good solution is because they can be trained by providing a large set of examples. Also, the trained model can be adapted dynamically if some habits seem to change over time. By using the back propagation learning algorithm, the system will correct the neural network up to a certain degree to adapt to the user’s changing habits [7].

For this framework, the fact that neural networks need a large set of training data is not a problem. It is easy to log sensor data during a large period, even before the actual start of the development process. Also, when the framework is deployed, sensor data will automatically be monitored, yielding sufficient data to keep training and adapting the neural network.

In [7], it is shown that it is indeed possible to effectively predict habits when a lot of training data is available (an accuracy of up to 92% was reached). The authors use the recent room history (which room did a person visit before going to another room) to predict the next room that a person will visit. An extensive simulation of different parameters and setups is performed. For example, the authors test if it is best to use a global model for all users, or to use a local model and predict the next room for each person separately. As could be expected, the local model turned out to be the best option. However, this means that a lot more computations should be done, since the model has to be created for each user separately.

The authors also looked for the optimal number of previous rooms that should be used to
make the predictions. This number appeared to be six rooms. If neural networks are used during the development of the framework in this master’s thesis, the appropriate number of history data will also have to be defined. For example, in the case of the prediction of the sleeping pattern or location, this will probably be the activity information from the past couple of weeks, since weekends are most likely significantly different than regular weekdays. In [7] the computation time of the neural networks was not an issue because a history of only six rooms was used. However, if weeks of history data have to be used, computations might become an issue and a tradeoff will have to be made.

The same authors summarise a set of methods to make the above mentioned room predictions in [8]. A comparison is made between Bayesian networks, neural networks, Elman nets (a special kind of neural networks), Markov predictors and state predictors. The accuracy of all methods turned out to be comparable and each of the methods seems to have its own advantages. However, neural networks turned out to be one of the most computationally intensive methods. Since the framework needs a set of real-time detection and prediction methods that need continuous adaptation, other less computationally intensive methods should be explored.

### 2.2.3.2 Activity prediction using a (Hidden) Markov Model

A Markov Model (MM) describes a sequence of states and the probability of going from one state to another. The least complicated type of MMs are first order Markov Chains. In a first order Markov Chain, the transition probability between two states only depends on the current state (this is called the ‘memoryless property’ of a Markov Chain). In a Markov Chain of order R, the transition probability between states depends on R previous states instead of one.

A Markov Chain requires to have an exact knowledge of the current state. However, it will not always be possible to exactly define the current state. When the current state is only partially observable, a Hidden Markov Model (HMM) will have to be used. In case of a HMM, several observations can be made which yield a set of current state probabilities. Each of the states in a HMM have transition probabilities to other states, just like regular Markov Chains. Thus, the difference between Markov Chains and HMMs is that with HMMs, one can never be a 100% confident of the current state, since the states are only partially observable.

In [9], it turned out that a HMM can be at least as accurate as Neural Networks for next room prediction. Another study where location is predicted with the help of HMMs is elaborated in [10]. The major advantage of HMMs is that they are less computationally intensive than e.g. neural networks, since the prediction of the next state boils down to a simple table lookup. Since lookup tables have to be stored, HMMs tend to need more memory then e.g. the neural networks mentioned before. However, given the fact that
recent smartphones have less memory constraints than battery constraints, this should not be a great issue. Even with the memory and battery constraints, it still remains better to process the data locally because processing the data on a server requires a frequent internet connection. Compared with some other techniques, it is also fairly easy to adapt the transition probabilities of the HMM when changing habits are observed. If unlikely events are occurring more frequently, it is possible to gradually increase the probability of this transition, such that after a while this transition will become the usual habit.

If HMMs are applied to, for instance, location prediction of a user, the states will have to be defined with care. Firstly, transition probabilities will depend on what time it is. For example, the probability of making the trip from home to work will most likely be higher at 8 AM than at 2 AM or 5 PM. Additionally, the probability of making this trip will most likely also be higher on Wednesday than on Saturday. Thus, it will be necessary to keep several transition probabilities, depending on the current time and day.

The previous example illustrated that a Markov based approach can be a suitable method to make certain predictions. Additionally, Markov chains can also be used to detect things that occur infrequently. If someone is still at home at 10 AM on Friday, while in 95% of the cases this person is at work at that time on that day, this could indicate that this person has a day off. Rare events will not have a noticeable impact on the transition probabilities (which indicate the user’s habits), but this information can be exploited to make more accurate recommendations.
Chapter 3

The context detection framework

In this chapter, the proposed framework is decomposed into its different modules. It will be explained what the responsibilities of each module are and how the modules interact. In the first section a global overview is given, after which all separate modules are covered in the next sections.

3.1 Overview

An overview of the proposed framework is depicted in Figure 3.1. The framework mainly consists of five parts: the Data Collection module, the Data Processing module, the Measurements Database, the Prediction Database and the Context Façade. External applications can access the data and predictions by interacting with the Context Façade.

The different modules do not directly interact with each other, but store data in the shared databases. The data in the databases can then be accessed by the other modules. The only exception is the connection between the Data Collection and the Data Processing module. Whenever the Data Collection module has collected and stored new data, it sends an intent to the Data Processing module to indicate that new data has been stored in the Measurement Database. The Data Processing module will then read this new data and process it.

In the next sections, a short overview is given for each module.
3.2 Data Collection

The Data Collection module is responsible for the collection of all kinds of useful data. This module is, in contrast to the other modules, fairly simple. For this reason, no specific chapter will be devoted to this module any more. All information about this module is given in this section.

Every 7.5 minutes, the Data Collection module reads the sensors of the device and registers to available services during 10 seconds. After these 10 seconds, all measured data are stored as a new entry in the Measurement Database.

3.2.1 Simple measurements

The Data Collection module listens and registers to the following simple sensors and services:

- Linear acceleration sensor. This is a motion sensor that measures the acceleration of the device. Compared to a regular acceleration sensor, this sensor excludes the effect of earth’s gravity.
• Rotation sensor. This is a position sensor that measures the physical position of
the device. This sensor is used to measure the difference compared to the previous
measurement, and not to measure if the device is currently moving (in contrast with
the acceleration sensor).

• Magnetic sensor. This sensor measures the strength of the electric field in the
neighbourhood of the device. These data are not used since they have not proven
to be useful, but are stored for completion and possible future use.

• Light sensor. Every time a sample is taken, the data collection module registers the
current light intensity (unit of measure: lux).

• Proximity sensor: The proximity sensor registers whether or not an object is near
(i.e. a few centimetres) the device. For example, this near object can be the user’s
head during a phone call or the user’s pocket when he is not using his device.

• Battery monitor. This sensor measures all kinds of information about the battery.
The Data Collection module registers the charging state and temperature of the
battery.

• Sound monitor. During the 10 seconds that the module listens to the sensors and
services, it also registers the loudness of the environment. The highest sound am-
plitude is stored in the measurement.

• Location service. While measuring, the device also registers to the built in loca-
tion service. If any location information is available, the coordinates (latitude and
longitude) are stored, together with the measurement accuracy.

• WiFi monitor. The Data Collection module also stores information about the net-
works that can be seen during the measurement. The SSID (i.e. the name of the
network) and MAC address (the physical address of the access point) is stored for
each observed network.

### 3.2.2 Activity recognition

Another service that the Data Collection module listens to is a somewhat more complex
one: the built in Google Activity Recognition API. This API has already been mentioned
in Section 2.2.2.2 as a low effort alternative for a handmade machine learning activity
recognition solution. The API distinguishes between the following activities with corre-
ponding labels:

- The device is in a vehicle, such as a car (`vehicle`).
- The device is on a bicycle (`bicycle`).
Every time a sample of the sensor data is taken, the activity recognition is also activated. The activity recognition API then returns the list of most likely detected activities, together with the confidence it has in each of the detected activities. Since this API is readily available and has proven to be accurate in other (commercial) applications, it is also used in this framework. The effort that would be needed to make a custom activity recognition module that will most likely be inferior to Google’s solution can now be invested in the implementation of other, additional modules.

The response of the API (a list of activities and their confidence) is stored together with the measured sensor data every time a sample is taken. Recommender systems that use the framework will be able to consult the detected activities through the Context Façade (see Chapter 7).

### 3.2.3 Weather information

The last thing being measured is the current weather at the user’s location. When a location is received while taking the measurement, a request is sent to OpenWeatherMap [19] to ask for the current weather at the user’s current coordinates. If there is a sufficiently fast internet connection available such that the return arrives in time (before the 10 second deadline), a json string describing the weather is added to the measurement.

The returned string comprises a lot of information about the weather, such as a weather code describing the current weather condition [20], the current temperature, today’s minimum and maximum temperature, the wind speed and the air pressure. As will be explained in Section 7.2.4, the current version of the Context Façade (see Section 3.4) only returns the weather condition code, the current temperature and the current wind speed. However, all information is stored for possible later use.

### 3.3 Data Processing

The Data Processing module is more complex than the Data Collection module. It consists of two large and one smaller submodule: the Activity submodule, the Location submodule
and the Location Labeling submodule. Whenever the Data Collection module notifies this module that a new sample has been inserted in the Measurement Database, it reads the sample and processes it.

As will be explained in Chapter 4, the Activity submodule will detect and predict the activity level of the user. In short, it will detect and predict whether or not the user is active or will be active in the future. The Location submodule is decomposed in Chapter 5. This submodule detects the user’s current location and predicts his future locations. Both modules make predictions for the next 24 hours and store these predictions in the Prediction Database, to be accessed by the Context Façade.

The smaller Location Labeling submodule is related to the Location submodule. This submodule is responsible for the addition of meta-information to the locations that are detected and created in the Location submodule. Adding meta-information to a location consists of detecting the home location(s) of the user, adding the name of public venues and sorting the locations into different categories. Except for the detection of the home location, user input is needed. How this labeling process takes places is explained in Chapter 6.

3.4 Context Façade

The Context Façade is created to enable quick and easy access to the data that is collected and processed by the framework. Developers of e.g. recommender systems can use the Context Façade as an API to request the information they need for their system. This way they do not have to bother about collecting and processing the information themselves, which makes it possible to focus solely on their application.

How the Context Façade is used and which functions it currently supports is described in Chapter 7.
Chapter 4

Detection and prediction of user activity level

In this chapter it is explained how the activity level of the user is detected and predicted. There are two possible states for the user’s activity level: active (awake) or inactive (asleep). The framework will predict when the user will be active or inactive. This can be done for an arbitrarily far moment in the future, but it is constrained to 24 hours in the future for now. The results at the end of this chapter show that the framework is indeed capable of giving accurate predictions of the user’s activity level.

Section 4.1 summarises how Markov models are used to make predictions. The following sections use this as a basis to perform the detection and prediction of the user’s activity level.

4.1 Markov based prediction

As mentioned in Section 2.2.3.2 Markov Models (MMs) describe a sequence of states and the transition probabilities between these states. The probabilities of all possible states of the Markov Model are depicted in a row matrix X:

\[ X = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix} \text{ with } \sum_{i=1}^{n} x_i = 1 \]  

(4.1)

If for example one is sure that at the beginning of the observations the system is in state \( x_1 \), the matrix X equals:

\[ X = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \text{ with } \sum_{i=1}^{n} x_i = 1 \]  

(4.2)
To be able to make predictions for the next states, one has to know the probabilities to go from one state to another. The transition probabilities to go from certain states $x_m$ to $x_n$ are kept in an $m \times n$ matrix:

$$T_{m,n} = \begin{bmatrix} t_{1\rightarrow 1} & t_{1\rightarrow 2} & t_{1\rightarrow 3} & \cdots & t_{1\rightarrow n} \\
 t_{2\rightarrow 1} & t_{2\rightarrow 2} & t_{2\rightarrow 3} & \cdots & t_{2\rightarrow n} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 t_{m\rightarrow 1} & t_{m\rightarrow 2} & t_{m\rightarrow 3} & \cdots & t_{m\rightarrow n} \end{bmatrix} \tag{4.3}$$

The sum of each row should always equal 1, because a transition will always occur (the transition from a state to itself is also a transition):

$$\sum_{j=1}^{n} t_{i\rightarrow j} = 1 \tag{4.4}$$

If one knows the current state of the chain and the probabilities of the transitions between the states, it is easy to make a prediction of the next states based on the transition probabilities. As an example, the probability of the system being in state $x_1$ in the next state equals

$$p_{t+1}(x_1) = p_t(x_1) \times t_{1\rightarrow 1} + p_t(x_2) \times t_{2\rightarrow 1} + \ldots + p_t(x_m) \times t_{m\rightarrow 1} \tag{4.5}$$

One can see that this calculation equals multiplying the matrix $X$ with the first column of the transition matrix $T_{m,n}$. In order to calculate the probabilities of all states after a transition, one only has to multiply the matrix $X$ with the entire transition matrix $T_{m,n}$. It can easily be seen that to predict state probabilities further into the future, the outcome of the matrix multiplication has to be multiplied with the transition matrix again recursively.

### 4.2 Activity level prediction using Markov Models

#### 4.2.1 Determining the current level of activity

One part of behaviour prediction consists of determining the user’s activity level. This can be done by determining whether or not the user is active (i.e. when he is not asleep). The MM will contain only two states: an active state and a passive state. With this model it should be possible to predict if the user will most likely be active or passive at a given time in the future.
Several sensors of the user’s smartphone are combined to observe the user’s current level of activity. The user will be considered active if any of the following (empirically defined) decision rules are triggered:

- Light intensity ≥ 50 lux
- Sound intensity ≥ 225 on a scale of 32767
- Accelerometer indicates movement
- Proximity sensor value changed compared to the previous measurement (e.g. the phone has been taken out of the user’s pocket)
- Location of the user changed considerably
- The smartphone’s rotation changed (e.g. the phone has been picked up and put down on the table differently afterwards)
- The smartphone is now charging when it was not during the previous measurement or vice versa

The combination of these criteria seems to be accurate in determining whether or not the user is active, even though some exceptional cases can cause the system make a wrong conclusion. For example, the following cases can lead to incorrect conclusions:

- The user left his smartphone in a dark and silent room. Since it is not possible to force the smartphone users to always carry their smartphones with them, it will be impossible to correctly detect the activity level in these cases.
- The user left his smartphone in e.g. a rucksack in silent environments like classrooms during an exam or a library. In these cases, none of the decision rules will be triggered.
- The user is sleeping long during the weekend and left the smartphone in the kitchen, where it gets light when the sun rises.
- The user is sleeping on a night train. Since he is significantly moving, the system will conclude that the user is currently active.

Since most of the time the system will be able to correctly predict the user’s activity level, observations are always considered as 100% sure instead of assigning a belief to the observations. By neglecting the cases of incorrect conclusions, the complexity of the implementation boils down to a few decision rules that have to be checked. Additionally, there is a mechanism that will help filtering out many cases of incorrect conclusions when the system does not expect the user to suddenly turn active or inactive. This mechanism will be explained in Section 4.2.4.
4.2.2 State probability and transition matrices

Since there are two possible states, the probability matrix is a $1 \times 2$ matrix. It can be assumed that the user is active at the moment the framework is installed. Therefore the initial probability matrix looks as follows:

$$X = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad (4.6)$$

The transitions of the MM are somewhat more complex. The activity pattern of users will most likely not be the same every day of the week. Many people go out at night during the weekends and sleep longer the day after. Also, people that work part time might sleep longer on days where they do not start working early. These examples illustrate that there should be different transition matrices, depending on the time of the day and the day of the week.

The previous examples illustrate that there should be a mechanism to capture these different patterns. Thus, it will not suffice to maintain only one transition matrix. In this implementation, a $2 \times 2$ transition matrix (there are 4 possible transitions) is kept for every 15 minute interval for every day of the week. This results in 96 matrices per day or 672 matrices for 1 week. The transition matrices look as follows:

$$T_{t,m,n} = \begin{bmatrix} t_{t,a\rightarrow a} & t_{t,a\rightarrow p} \\ t_{t,p\rightarrow a} & t_{t,p\rightarrow p} \end{bmatrix} \quad (4.7)$$

In the notation above, the subscript $t$ is an index referring to a specific matrix that corresponds with a moment in the week (e.g. Monday at 4 p.m.). Initially, the probability to stay in the same state is set to 0.95. Every time a new measurement comes in, the measurement is used to detect the user’s activity level and update the transition matrix that corresponds with that time and day of the week. How this update mechanism works will be explained in Section 4.2.3. To speed up initialisation, the transition matrix updates triggered by the measurements that are made during the first day that the framework is running are copied to all other transition matrices of the entire week that correspond with that time of the day. As a consequence, after 24 hours the framework will most likely make better predictions than when it would have used a uniform distribution for the entire first week.

In order to converge faster to stable probabilities, it is also possible to assign higher weights to new updates during the first weeks. These weights should however be lowered after a few weeks, because otherwise the system will react too heavily to one-time deviations or incorrect detections.
4.2.3 Processing new measurements

Every time a new measurement comes in, the transition matrix that corresponds with the measurement should be updated. The transition matrix that is updated is the matrix that represents the moment of the week closest to the timestamp in the measurement. The transition matrix update mechanism will now be illustrated with an example. Suppose that the following matrices are the current state probabilities and transition matrix:

\[
X = \begin{bmatrix} 0.67 & 0.33 \end{bmatrix} \quad (4.8)
\]
\[
T = \begin{bmatrix} 0.92 & 0.08 \\ 0.61 & 0.39 \end{bmatrix} \quad (4.9)
\]

This situation can be an example of the matrices in the morning. If the user is already awake, he will very likely (92% certainty) also stay awake. However, if he is not yet awake, there is still a fair chance (39%) that the user will still be inactive when the next measurement is taken. Now suppose the user is sleeping a little longer today: the measurement indicates a Passive → Passive transition. Because this measurement increases the belief that if the user is still sleeping at this hour he will stay asleep, the probability of this transition should be increased. If new measurements are processed with a weight \(w\) (between 0 and 1), the update of the transition matrix will be performed as follows:

\[
T = \begin{bmatrix} 0.92 & 0.08 \\ 0.61 \times (1 - w) & 0.39 \times (1 - w) + 1 \times w \end{bmatrix} = \begin{bmatrix} 0.92 & 0.08 \\ 0.43 & 0.57 \end{bmatrix} \text{ for } w = 0.3 \quad (4.10)
\]

This update increased the probability of staying inactive if you were previously inactive. Note that when the previous measurement was passive, only the transitions from passive to passive or active are updated. If also the first row was updated with the same update mechanism, the probability that the user goes to sleep again directly after waking up increases. This however is a completely different situation and thus it is incorrect to update both rows of the transition matrix at once.

4.2.4 Determining the current state probabilities

As could already be noted, the state probability matrix in the previous example did not contain a 1 and a 0, but had a certain belief in both being active and inactive. This is the result of the mechanism that filters out unlikely transitions because of incorrect detections. For example, if the system wrongly detects that the user is inactive while he has just woken up 30 minutes ago, the system should know that it is rather unlikely that the user will already go to sleep again.
To be able to filter out these unlikely detections, the system should know how long the user is active and inactive on average. The values of these averages are initialised as 8 hours for being inactive and 16 hours for being active. These values will then be fine tuned depending on the habits of the user. A start of an active (or inactive) period will be taken as the moment on which the active (or inactive) state becomes the most probable. The period will end when the other state becomes the most probable again. In order to adapt to busy periods with less sleep or periods with more sleep like vacations, in fact not really an average is calculated. Instead, new periods are processed with a certain weight (different from the weight in the previous section). For example, if the previous average inactive period lasted 7.5 hours and a new measurement indicated 7 hours of inactive period, the average is updated as follows (for e.g. \( w = 0.2 \)):

\[
\text{Average inactive period} = 7.5 \times (1 - w) + 7 \times w \\
= 7.5 \times 0.8 + 7 \times 0.2 \\
= 7.4 \text{ hours}
\] (4.11)

This is done every time an active or inactive period is considered finished. This way, the averages should gradually converge to the user’s habits. Note that in order to keep the average realistic some values will not be accepted. If for example the periods take too long or too short (this can happen when incorrect measurements occurred), the values will not be used to update the average. Inactive periods shorter than 3 hours and longer than 24 hours and active periods shorter than 12 hours and longer than 24 hours will not be considered.

The calculation of the current state probabilities takes this average time of being active or inactive into account. Every time a new measurement arrives and a new probability has to be calculated, yet another weight \( w \) is calculated. Then this weight is used to update the probabilities as follows (illustrated for the case where the measurement is considered as active):

\[
X = \begin{bmatrix} p_{a,new} & p_{p,new} \end{bmatrix} = \begin{bmatrix} p_{a,old} + p_{p,old} \times (1 - w) & p_{p,old} \times w \end{bmatrix}
\] (4.12)

Thus, a fraction \( w \) of the probability of the other state is taken away and added to the detected state. This weight \( w \) is the sum of two terms:

\[
w = w_1 + w_2
\] (4.13)

\( w_1 \) is a fixed, basic weight which is set to 0.15 (for now). This is the lowest possible value of the weight \( w \), since \( w_2 \) is variable, depending on how long the current activity is already ongoing. The weight \( w_2 \) is calculated as shown below:
\[ w_2 = \frac{\text{Passed activity time}}{\text{Average activity time}} \times s \text{ with } s = \text{scaling factor} = 0.5 \quad (4.14) \]

Since the weight \( w \) equals \( w_1 + w_2 \), one can easily see that the weight increases linearly with time, starting from 0.15. When the average time of an activity has passed, the weight has reached 0.65. The maximum value the weight can reach is 1, since larger are forced back to 1.

The major consequence of this approach is that one needs more than a few samples to convince the system of an unlikely activity change. For example, the system will need to see a few inactive samples to believe that the user is actually sleeping during the afternoon if this is not his usual habit. If in this approach the basic weight \( w_1 \) was not added to the weight, it would have been nearly impossible for the user to 'return' to e.g. the active state after an incorrect measurement when the user has passed his average active time. The system will think that the user went to sleep and new measurements have very low weights since the user has been sleeping for just a little while. By adding a fixed value (\( w_1 \)) to the weight, the system will always be able to return to the active state after a few active samples.

### 4.2.5 Predicting future state probabilities

The prediction of the future state probabilities is similar to the approach used in determining the current state. The weight values are based on when the state with the highest (predicted) probability changed for the last time. Thus, when somewhere in the future the most likely state changes, this moment is used as the start of the new period and the weights will be calculated accordingly.

At the end of Section 4.1 it is said that predictions can be calculated by recursively multiplying the state probability matrix with the correct transition matrix. However, by introducing the weight, the calculations are slightly more complicated. As an example: in the case where the user is most likely active, the term that represents the probability of going from the active to the passive state \( (p_{a,old} \times t_{t,a\rightarrow p}) \) is reduced by multiplying it with the calculated weight. The probability that has been 'taken away' from this term is added to the probability of staying active. This makes sense because it is more likely to stay in the same state when the average time of being active has not passed completely. Hence, the predicted state probabilities can be calculated as follows:

\[
X_{predicted} = \begin{bmatrix}
p_{a,old} \times t_{t,a\rightarrow a} & p_{a,old} \times t_{t,a\rightarrow p} \times w \\
p_{p,old} \times t_{t,p\rightarrow a} & + p_{p,old} \times t_{t,p\rightarrow p} \\
p_{a,old} \times t_{t,a\rightarrow p} \times (1 - w)
\end{bmatrix} \quad (4.15)
\]
By using this approach for calculating the predictions, the average duration of the activities is taken into account. Unfortunately, this approach requires a 'custom' matrix multiplication. In order to be able to use regular matrix multiplications, a general formula that is valid for both an active or inactive most likely state can be written down equivalently as:

$$X_{predicted} = \begin{bmatrix} p_{a,old} & p_{p,old} \end{bmatrix} \times \begin{bmatrix} t_{t,a\rightarrow a} & t_{t,a\rightarrow p} \\ t_{t,p\rightarrow a} & t_{t,p\rightarrow p} \end{bmatrix}$$

$$+ \begin{bmatrix} p_{a,old} & p_{p,old} \end{bmatrix} \times \begin{bmatrix} a/p & -a/p \\ -1 + a/p & 1 - a/p \end{bmatrix}$$

$$\times (t_{a,p} \cdot a/p + t_{p,a} \cdot (1 - a/p)) \cdot (1 - w)$$  \hspace{1cm} (4.16)

As mentioned before, the weight $w$ depends on how long the active or passive state is already expected to be the most likely state. The parameter $a/p$ is 1 when the user will most likely be active and 0 when the user will most likely be passive. When expanding the above formula with $a/p = 1$ (active state is most probable), one will find the same probabilities as in equation 4.15.

### 4.2.5.1 Additional correction term: average activity level probability

To calculate the final new state probabilities, it has been empirically proven to be beneficial to add an additional correction term, based on the average activity level of the user at a certain moment of the week. A set of row matrices (one for every 15 minutes of the week), further called frequency matrices, keep track of the activity level of the user. Every time a measurement is processed, the corresponding column of the corresponding matrix is incremented. To allow gradually changing sleeping habits, the non-zero values are decreased by 0.1 before the increment takes place. As an example, suppose it is Wednesday evening around 6 p.m. In this case, the user will most likely have been active in the past. Considering a new measurement that also indicates that the user is active, the old and updated frequency matrix $F$ will be as follows:

$$F_{old} = \begin{bmatrix} 12.4 & 0.6 \end{bmatrix}$$  \hspace{1cm} (4.17)

$$F_{updated} = \begin{bmatrix} 13.3 & 0.5 \end{bmatrix}$$  \hspace{1cm} (4.18)

In order to have state probabilities, $F_{updated}$ is normalised:

$$F_{normalised} = \begin{bmatrix} 0.96 & 0.04 \end{bmatrix}$$  \hspace{1cm} (4.19)
The final predicted state probabilities can now be calculated by a simple weighted addition of 4.16 and 4.19. If the weights are set to 0.8 and 0.2, the predicted state probabilities equal:

\[
\text{Prediction} = 0.8 \times X_{\text{predicted}} + 0.2 \times F_{\text{normalised}}
\]  

(4.20)

These calculations can be performed recursively in order to predict the user’s activity further into the future.

### 4.2.6 Summary of calculation flow

In this section, a short summary of the calculation flow is given. The flow starts with the initial probabilities, a $1 \times 2$ matrix, as in formula 4.8. Then, according to whether or not the new measurement is considered as active, the weight $w$ is calculated according to equation 4.13. With this weight value, the new state probabilities can be calculated with equation 4.12. Next, the transition matrices are updated using equation 4.10.

To calculate the predicted probabilities of the future states, equation 4.16 and 4.19 should be calculated first. The predicted probabilities can then recursively be calculated using 4.20. At each iteration of the prediction, all terms and the weight $w$ have to be calculated again using the matrices and time periods that correspond to the moment in the future for which the calculations are being performed.

### 4.3 Results

#### 4.3.1 Activity level detection

##### 4.3.1.1 Evolution of activity level

In this section it shown how the activity level varies throughout the day. Figure 4.1 shows the evolution of the activity level during the week from March 10 until March 16. A clear periodic pattern can be seen, in which the activity level is close to 1 during the day and close to 0 at night.

However, the activity pattern is not as smooth as one would expect. Many false positives or negatives appear, mostly during the evening, at night or in the morning. For example, it could be that, when sleeping with opened windows, there is some noise outside that triggers the activity detection. Another example could be that during the evening the user leaves his smartphone in a silent and dark place. In the user test that is conducted (see Section 8.2) it appeared that these false positives really are an issue for the activity level detection.
The examples above should make clear that a developer that uses the framework has to be careful when interpreting the activity level. The framework will always return the activity level value, and not a boolean determining whether or not the user is most likely active (see Section 7.2.1). This gives the developer the freedom to choose his own thresholds and interpret the activity level the way he or she prefers.

### 4.3.1.2 Accuracy of activity level detection

To check the accuracy of the activity level detection, the time of going to sleep and waking up was logged during a couple of weeks. Comparing this ground truth with the automatic detections should give a clear idea about the accuracy of the activity level detection.

Table 4.1 shows the results of the experiment during one week. The table shows three values. The first two are the ground truth and the moment where the activity level drops below 0.50 or rises above 0.50. Note that this moment is manually filtered afterwards to be as closest to the ground truth as possible, thereby ignoring false positives or negatives before or after going to sleep or waking up. Because this is not really feasible in practice, functions have been implemented in the Context Façade that perform this filtering automatically. The functions have a variable threshold and check if the activity level stays below (or above) the threshold for a certain amount of time, hereby filtering out short spikes caused by false positives or negatives. More information about these functions can be found in Section 7.2.1. The last column of Table 4.1 is the time of detection by these functions, using the suggested thresholds.
Table 4.1: Comparison of activity level detections between ground truth, manually filtered detection and Façade function detection.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Ground Truth</th>
<th>Detection</th>
<th>Façade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>29/02/2016</td>
<td>Waking up</td>
<td>08:30</td>
<td>08:35</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:50</td>
<td>01:18</td>
<td>01:18</td>
</tr>
<tr>
<td>Tuesday</td>
<td>01/03/2016</td>
<td>Waking up</td>
<td>07:10</td>
<td>06:55</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:30</td>
<td>00:38</td>
<td>00:38</td>
</tr>
<tr>
<td>Wednesday</td>
<td>02/03/2016</td>
<td>Waking up</td>
<td>06:55</td>
<td>06:59</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:15</td>
<td>00:22</td>
<td>00:22</td>
</tr>
<tr>
<td>Thursday</td>
<td>03/03/2016</td>
<td>Waking up</td>
<td>05:45</td>
<td>06:02</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:10</td>
<td>00:13</td>
<td>00:13</td>
</tr>
<tr>
<td>Friday</td>
<td>04/03/2016</td>
<td>Waking up</td>
<td>06:50</td>
<td>06:51</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>02:00</td>
<td>02:07</td>
<td>02:07</td>
</tr>
<tr>
<td>Saturday</td>
<td>05/03/2016</td>
<td>Waking up</td>
<td>09:30</td>
<td>09:44</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>01:15</td>
<td>01:43</td>
<td>01:43</td>
</tr>
<tr>
<td>Sunday</td>
<td>06/03/2016</td>
<td>Waking up</td>
<td>09:40</td>
<td>09:43</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>01:00</td>
<td>01:16</td>
<td>01:16</td>
</tr>
</tbody>
</table>

The values in the table show that both the manual detection and the detection with the functions in the Context Façade are relatively correct. Both methods make some mistakes, but they all stay within an acceptable range. Over a period of 28 days, the average error for manual detection of waking up is 6 minutes and 11 minutes for the detection with the Context Façade function, for which 1 completely incorrect detection was discarded. The average errors on the detection of going to sleep are 12 minutes for both methods. Because the manual filtering when the activity level incorrectly drops below (or rises above) 0.50 is not feasible in practice, it is advised to use the automatic Context Façade functions, which do not seem to harm the correctness of detection a lot.

4.3.2 Activity level prediction

The activity module does not only detect the current activity level, it also predicts the future activity levels. After every sample, new activity levels are predicted for every quarter up to 24 hours into the future. To check the accuracy of the predictions, the same experiment as in the previous section is conducted. Whenever the predicted activity level drops below 0.50, this is considered as going to sleep. Whenever the predicted activity level rises above 0.50, this is considered as waking up. In Table 4.2, the results of the same week as in Table 4.1 are depicted.

The results in the table show that in ‘steady state’ (the experiment was conducted when over 20 weeks of data had been collected), the predictions converge to typical values. This
Table 4.2: Comparison of activity level prediction with ground truth.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Ground Truth</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>Waking up</td>
<td>08:30</td>
<td>07:45</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:50</td>
<td>00:45</td>
</tr>
<tr>
<td>Tuesday</td>
<td>Waking up</td>
<td>07:10</td>
<td>07:45</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:30</td>
<td>00:45</td>
</tr>
<tr>
<td>Wednesday</td>
<td>Waking up</td>
<td>06:55</td>
<td>07:45</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:15</td>
<td>00:45</td>
</tr>
<tr>
<td>Thursday</td>
<td>Waking up</td>
<td>05:45</td>
<td>07:30</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:10</td>
<td>00:45</td>
</tr>
<tr>
<td>Friday</td>
<td>Waking up</td>
<td>06:50</td>
<td>07:30</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>02:00</td>
<td>01:30</td>
</tr>
<tr>
<td>Saturday</td>
<td>Waking up</td>
<td>09:30</td>
<td>09:15</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>01:15</td>
<td>01:45</td>
</tr>
<tr>
<td>Sunday</td>
<td>Waking up</td>
<td>09:40</td>
<td>09:15</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>01:00</td>
<td>01:15</td>
</tr>
</tbody>
</table>

can be seen by looking at the predictions on weekdays, which are the same for almost every day. The predicted times are often somewhat later than the actual ground truth (e.g. 06:50 against a predicted time of 07:45). This can be attributed to the falses positive and negatives that appear often before going to sleep and after waking up (see Section 4.3.1.1).

The predictions are obviously not as accurate as the detections from the previous section, but they do give a correct indication of whether or not the user will wake up early. Some events, like the exceptional early wake up on the 3rd of March, simply can not be predicted which make the predictions look worse than they actually are. During the entire 27 day experiment, the average error on the wake up prediction was 40 minutes. The average error on predicting when the user goes to sleep was 33 minutes. It can be concluded that the predictions will not be very accurate, but are useful to get an idea about around what time the user wakes up or goes to sleep on average.
Chapter 5

Detection and prediction of user location

This chapter describes how the location of the user is predicted by the framework. The methodology used is mostly an extension of the previous chapter. The prediction is performed in the same way as for the user’s activity level, with just (many) more possible states. Even though the uncertainty increases with the number of states, the results at the end of the chapter prove that with some necessary additions, it is possible to predict the user’s location fairly accurate.

5.1 Locations

In order to be able to predict the user’s location, one first has to define what a location is exactly. The two basic building blocks of a location in this framework are:

- Coordinates: latitude, longitude and coordinate accuracy.
- A set of MAC-addresses that have been observed at this location.

A location can also be enriched with other data, like information from Google Places and Foursquare. Examples of this data are the category of the location, opening hours, telephone numbers,... How this information is gathered and added will be explained in Chapter 6.

Determining the location of a user is a lot easier when compared to determining the activity level of a user, as explained in the previous chapter. Firstly, MAC-addresses are unique and can thus be observed at one location only. Secondly, coordinates describe the location of a user very accurately. If the user’s coordinates lie within a certain radius of a location, the framework will consider the user at this location.
If a user is at a certain location for 2 or more subsequent measurements and 2 or more known MAC-addresses are observed, the information contained in the measurements will be added to the location if this information was not known before. This comes down to adding unseen MAC-addresses or more accurate coordinates to a certain location. The reason for doing this only if the user has been here for a while and after seeing already known MAC-addresses is to avoid that locations get mixed up because the received coordinates are not recent enough. For example, if a user arrives at a new location while his most recent coordinates are still the coordinates of the old location, the system might detect the old location by its coordinates. It would then add the MAC-addresses of the new location to the old one. By waiting a while and checking if MAC-addresses have been seen before, the framework ensures that locations do not get mixed up.

5.2 Location prediction

5.2.1 Matrices

The matrices used in a location model are very similar to those used for the activity prediction. Note that compared to activity prediction, location prediction will have many more possible states. The only implication of the higher number of states is that it will complicate computations, making them more expensive. The methodology however can be adopted completely from the activity prediction. In order to keep the number of states under control, a location will only be added to the matrices if the user has been at this certain location for two or more subsequent samples. Later in this chapter (in Section 5.4) it is discussed that the computations will always stay within certain limits, making it feasible to compute all predictions on a smartphone.

A first building block of the prediction process is the state probability matrix. This is a row matrix in which every column depicts a possible location known to the location module:

\[ X = \begin{bmatrix} p_{l_1} & p_{l_2} & \cdots & p_{l_n} \end{bmatrix} \] (5.1)

This state probability matrix will always contain just one ‘1’, while all other elements will be ‘0’. There will be no uncertainty concerning the current state, since the observed MAC-addresses and coordinates will yield a very accurate knowledge of the user’s position.

Just as in the activity prediction, the second type of matrices are the transition matrices. As already discussed in Section 4.2.2, multiple transition matrices are needed. Therefore, the location prediction will also use 672 transition matrices: 1 matrix per 15 minutes for every day of the week. The transition matrices have the following structure:
In the notation above, the subscript \( t \) is an index referring to a specific matrix that corresponds with a moment in the week (e.g. Monday at 4 p.m.). When a new location is added to the matrices, the new row contains all zeros for every transition probability except for the transition from the new location to itself. The extra column in the already existing rows will contain zero for every row. This is illustrated below, where a new location is added to a transition matrix that originally contained 3 locations:

\[
T_{t,m,n} = \begin{bmatrix}
  t_{t,l_1 \rightarrow l_1} & t_{t,l_1 \rightarrow l_2} & \cdots & t_{t,l_1 \rightarrow l_n} \\
  t_{t,l_2 \rightarrow l_1} & t_{t,l_2 \rightarrow l_2} & \cdots & t_{t,l_2 \rightarrow l_n} \\
  \vdots & \vdots & \ddots & \vdots \\
  t_{t,l_m \rightarrow l_1} & t_{t,l_m \rightarrow l_2} & \cdots & t_{t,l_m \rightarrow l_n}
\end{bmatrix}
\] (5.2)

After the new location is added to the matrix, the new measurement will be processed just like every other measurement. How new measurements are processed will be explained in Section 5.2.2.

Note that, just as in the activity prediction, the initialisation will be boosted by copying updates on the first day that the framework is running to the corresponding moments of the day in the entire week.

### 5.2.2 Processing new measurements

When a new measurement is taken, it is used to update the current state probabilities and the transition matrices. As already argued earlier, there is no uncertainty concerning the current location. If at least two MAC-addresses that correspond with a certain location are seen, or the user is within a certain range of a location (currently this radius is set to 75 meters), the user will be considered at this location with 100% certainty.

For example, suppose there are 4 locations and the user was previously at location 2. The old state probability matrix will be the following:

\[
X_{old} = \begin{bmatrix}
  0 & 1 & 0 & 0
\end{bmatrix}
\] (5.4)
The new measurement indicates that the user is now at location 4. The new state probability matrix will then be:

\[
X_{\text{new}} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}
\]  \hspace{1cm} (5.5)

The update mechanism for the transition matrices is the same as for the activity prediction. The row of the detected transition is updated with a certain weight \( w \). If for the previous example, the corresponding transition matrix was

\[
T_{t,4,4,\text{old}} = \begin{bmatrix} 0.8 & 0.1 & 0.05 & 0.05 \\ 0.2 & 0.4 & 0.1 & 0.3 \\ 0.1 & 0.6 & 0.15 & 0.15 \\ 0.2 & 0.2 & 0.4 & 0.2 \end{bmatrix}
\]  \hspace{1cm} (5.6)

and a transition from location 2 to location 4 is detected, the corresponding transition matrix will be updated as follows:

\[
T_{t,4,4,\text{old}} = \begin{bmatrix} 0.8 & 0.1 & 0.05 & 0.05 \\ 0.2 \times (1-w) & 0.4 \times (1-w) & 0.1 \times (1-w) & 0.3 \times (1-w) + 1 \times w \\ 0.1 & 0.6 & 0.15 & 0.15 \\ 0.2 & 0.2 & 0.4 & 0.2 \end{bmatrix}
\]  \hspace{1cm} (5.7)

With \( w = 0.3 \) the new transition matrix will be:

\[
T_{t,4,4,\text{old}} = \begin{bmatrix} 0.8 & 0.1 & 0.05 & 0.05 \\ 0.14 & 0.28 & 0.07 & 0.51 \\ 0.1 & 0.6 & 0.15 & 0.15 \\ 0.2 & 0.2 & 0.4 & 0.2 \end{bmatrix}
\]  \hspace{1cm} (5.8)

It is easy to see that the transition probability from location 2 to location 4 increased at this moment of the week and the transition probabilities from other locations remained the same. This is the desired effect, which can easily be illustrated with an example. Suppose a student usually is in the city in which he goes to school from Monday till Friday, but exceptionally he will only be there from Tuesday till Friday because there are no classes on Monday. While the student is still at home on Monday, the system should know he will not be going to school that day. This will be the case if the probability of the transition probability from the student’s home location to his school is not boosted when he goes to school on normal Mondays. By only updating the row of the previous state, the row of the transition from the user’s home location to school on the exceptional Monday off remains untouched.
5.2.3 Predicting future state probabilities

The prediction of future states is also very similar to the prediction of future activity levels. While the activity prediction takes the average time the user is active or inactive into account, the location prediction keeps track of the average time at the location. When the user has left a location, the time that he has been at the location is processed with a weight \( w \), which is set to 0.1. For example, if the previous average was 3 hours and the new measurement indicated a visit of 4 hours, the new average is calculated as follows:

\[
\text{Average time at location} = 3 \times (1 - w) + 4 \times w
\]
\[
= 3 \times 0.9 + 4 \times 0.1
\]
\[
= 3.1 \text{ hours}
\]  

This average value will be used to calculate a new weight \( w \), that is used in the prediction of future locations. The weight is the sum of 2 values:

\[
w = w_1 + w_2
\]  

\( w_1 \) is a fixed value that is set to 0.15. \( w_2 \) is calculated as follows:

\[
w_2 = \frac{\text{Passed time at location}}{\text{Average time at location}} \times s \text{ with } s = \text{scaling factor} = 0.6
\]  

One can easily see that the lowest possible value of \( w \) is 0.15 and that it increases linearly with the time at the location. If \( w \) grows larger than 1, it is forced back to 1.

Because of the weight that has to be taken into account, it is not possible to calculate the future location probabilities by simply recursively multiplying the current location probability matrices with the corresponding transition matrices. A more complicated set of matrix multiplications is needed to perform the desired probability calculations. After the regular matrix multiplication of the probability matrix with the transition matrix, part of the probability that ‘left’ the most probable location has to be moved back to the most probable location. Only a fraction \( w \) of the moved probability can stay at the new location. This means that \((1 - w)\) times the moved probability should be added back to the original, most probable location.

To illustrate this, consider 4 possible locations of which the second is the most probable. After the regular matrix multiplication (old probabilities multiplied with the transition matrix), the new probability matrix looks as follows:
\[ X_{\text{temp}} = \begin{bmatrix} p_{1,\text{temp}} & p_{2,\text{temp}} & p_{3,\text{temp}} & p_{4,\text{temp}} \end{bmatrix} = \begin{bmatrix} t_{t,l_1 \rightarrow l_1} & t_{t,l_1 \rightarrow l_2} & t_{t,l_1 \rightarrow l_3} & t_{t,l_1 \rightarrow l_4} \\ t_{t,l_2 \rightarrow l_1} & t_{t,l_2 \rightarrow l_2} & t_{t,l_2 \rightarrow l_3} & t_{t,l_2 \rightarrow l_4} \\ t_{t,l_3 \rightarrow l_1} & t_{t,l_3 \rightarrow l_2} & t_{t,l_3 \rightarrow l_3} & t_{t,l_3 \rightarrow l_4} \\ t_{t,l_4 \rightarrow l_1} & t_{t,l_4 \rightarrow l_2} & t_{t,l_4 \rightarrow l_3} & t_{t,l_4 \rightarrow l_4} \end{bmatrix} \] (5.12)

The temporary state probabilities should thus equal:

\[
\begin{align*}
p_{1,\text{temp}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_1} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_2} + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_3} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_4} \\
p_{2,\text{temp}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_2} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_2} + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_3} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_2} \\
p_{3,\text{temp}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_3} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_3} + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_3} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_3} \\
p_{4,\text{temp}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_4} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_4} + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_4} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_4}
\end{align*}
\] (5.13)

However, if the second location was the most probable, the desired new probabilities of \(X_{\text{new}}\) are:

\[
\begin{align*}
p_{1,\text{new}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_1} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_1} \times w + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_1} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_1} \\
p_{2,\text{new}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_2} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_2} + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_2} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_2} \\
&\quad + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_1} \times (1 - w) \\
&\quad + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_3} \times (1 - w) \\
&\quad + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_4} \times (1 - w) \\
p_{3,\text{new}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_3} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_3} \times w + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_3} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_3} \\
p_{4,\text{new}} &= p_{1,\text{old}} \cdot t_{t,l_1 \rightarrow l_4} + p_{2,\text{old}} \cdot t_{t,l_2 \rightarrow l_4} \times w + p_{3,\text{old}} \cdot t_{t,l_3 \rightarrow l_4} + p_{4,\text{old}} \cdot t_{t,l_4 \rightarrow l_4}
\end{align*}
\] (5.14)

This can be realised by adding a second term to the recursive calculation. Two extra matrices are needed to perform these calculations. The first matrix is called the correction matrix \(C\), a square matrix with the dimensions of the transition matrix. The off-diagonal elements are the same is those in the transition matrix. The diagonal elements equal 1 minus the diagonal elements of the transition matrix. In case of the above example where there are 4 possible locations, the matrix \(C\) looks as follows:

\[
C = \begin{bmatrix} 1 - t_{t,l_1 \rightarrow l_1} & t_{t,l_1 \rightarrow l_2} & t_{t,l_1 \rightarrow l_3} & t_{t,l_1 \rightarrow l_4} \\ t_{t,l_2 \rightarrow l_1} & 1 - t_{t,l_2 \rightarrow l_2} & t_{t,l_2 \rightarrow l_3} & t_{t,l_2 \rightarrow l_4} \\ t_{t,l_3 \rightarrow l_1} & t_{t,l_3 \rightarrow l_2} & 1 - t_{t,l_3 \rightarrow l_3} & t_{t,l_3 \rightarrow l_4} \\ t_{t,l_4 \rightarrow l_1} & t_{t,l_4 \rightarrow l_2} & t_{t,l_4 \rightarrow l_3} & 1 - t_{t,l_4 \rightarrow l_4} \end{bmatrix}
\] (5.15)

The second new matrix is the selection matrix \(S\), it is also a square matrix with the same dimensions as the transition matrix. It contains all zeros, except for one row: the row
of the most likely state. On this row, the diagonal element equals 1, the other elements are -1. It is called the selection matrix because it selects which column of the correction matrix C is used, depending on the most likely location. For the example above, where the second location is the most probable, the matrix S is the following:

\[
S = \begin{bmatrix}
0 & 0 & 0 & 0 \\
-1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]  

(5.16)

To reach the equations of (5.14) that lead to \(X_{\text{new}}\), the following calculation should be executed:

\[
X_{\text{new}} = X_{\text{temp}} + C \times S \times (1 - w)
\]  

(5.17)

The second term increases the probabilities of the most likely location by reducing the probability of the other locations (the bold part of the equations in (5.14)).

While predicting, the correction matrix \(C\) will always contain the transition probabilities that correspond with those of the moment in the future for which you are currently predicting. The selection matrix \(S\) will also have its non-zero elements on the row of the predicted most likely location at that time in the future. For the calculation of the weight \(w\), the average time at the most likely location in the future is taken. The time at that location starts at the moment in the future that the location is predicted to become the most likely.

### 5.2.4 Additional correction term: average location probability

Although the formula of equation (5.17) is capable of making accurate predictions, there is one weak point that calls for an additional correction term. The weak point is that when the user is at a new location or at a location at which the user has not been before at this time of the week, the system will not have transitions in its transition matrices from this location to another location in the near future.

For example, consider that a user usually goes to a bar on Friday night. One day, he goes to this bar on Thursday night for the first time. The transition matrices will not contain transitions from this bar to the user’s home (or another location) before Friday night. This will cause the system to predict that the user stays at this bar for more than a day, until the time that the user usually goes home on Friday night. This is obviously a bad prediction.

To solve this, the location model will keep track of the ‘average location’ of the user. A set of row matrices, called frequency matrices, will store at which location the user was at
a certain moment of the week. Every time a sample is processed at a certain location, the corresponding value in the corresponding frequency matrix will be incremented. In order to allow gradual changes in location patterns, all non-zero values are decreased by 0.1 before incrementing the correct value. As an example, consider the user is at the second out of 4 locations. The old and updated frequency matrices will be as follows:

\[
F_{\text{old}} = \begin{bmatrix} 4.8 & 8.2 & 0.2 & 0 \end{bmatrix}
\]

\[
F_{\text{updated}} = \begin{bmatrix} 4.7 & 9.1 & 0.1 & 0 \end{bmatrix}
\]

To put this matrix into the predicted state probabilities, the matrix is first normalised:

\[
F_{\text{normalised}} = \begin{bmatrix} 0.34 & 0.65 & 0.01 & 0 \end{bmatrix}
\]

Since this matrix has the same dimensions as \(X_{\text{new}}\) from equation \(5.17\), the corrected predicted state probabilities can be calculated as simple as a weighted addition of \(X_{\text{new}}\) (from equation \(5.17\)) and \(F_{\text{normalised}}\) (see equation \(5.20\)). The weight depends on the average time the user is expected to be at this location. If the current location is a new location, the average time calculated over all locations is used since no specific average time is known for this location yet. The weight is calculated as follows:

\[
w = 0.1 \times \left( \frac{\text{Time at location}}{\text{Average time at location}} \right)^2
\]

If \(w\) grows larger than 0.1, it is forced back to 0.1 in order to avoid excessive weight values. Thus, the final predicted state probabilities equal:

\[
\text{Prediction} = (1 - w) \times X_{\text{new}} + w \times F_{\text{normalised}}
\]

### 5.2.5 Summary of calculation flow

The initial state probabilities with \(n\) possible locations will be a \(1 \times n\) matrix with all zeros and one 1, as shown in equation \(5.4\). After a measurement has been processed, the update of the new state probabilities is rather simple, the detected location has 100% probability, as shown in equation \(5.5\). Next, the transition matrix can be updated using equation \(4.10\).

To recursively predict the future location probabilities, the \(X_{\text{new}}\) has to be calculated as described in equation \(5.17\). Finally, the corrected predictions can be calculated using equation \(5.22\).
5.3 Error Measure: Logarithmic Loss

Once the predictions are made, it is also desirable to be able to express how well the module performs at making these predictions. Only checking all predictions of the model to see if the location with the highest probability indeed was the correct location would give a rather oversimplified measure. In situations where multiple locations have a non-negligible probability there is fair chance that the most probable location turns out to be incorrect.

Because of the above mentioned reason, there is a need for an error measure that takes into account the confidence in the predicted location. An error measure that satisfies this need is the logarithmic loss. The logarithmic loss can be calculated as follows:

\[
\text{logarithmic loss} = -\ln p_{\text{correct}}
\]  \hspace{1cm} (5.23)

In the above equation, \( p_{\text{correct}} \) is the predicted probability of the correct location. One can derive from the formula that if the probability of the correct location is high, the loss will be close to zero. If the probability of the correct location is low, the loss will be high. For practical reasons (to avoid an infinite or very high loss), the probability of the correct location will always be lower bounded by 0.001. By doing this, a loss higher than (approximately) 6.91 will never be encountered.

The logarithmic loss serves as a good measure to compare different models. Whenever a sample is processed, a model will perform 24 predictions: one for every hour after the measurement took place. The logarithmic loss is calculated for every prediction by comparing it with the collected ground truth afterwards. By averaging over all calculated logarithmic losses, the performance of a model can be expressed.

5.4 Parallel prediction

It will not be possible to capture every scenario in just one model. For example, people visit different locations during holidays than during regular weeks. For this reason, different models will be maintained in parallel. During the entire week, all models will be updated and saved temporarily. The predictions of the model will continuously be checked by comparing them with the ground truth that is collected after performing the predictions.

The framework will use the predictions of the model that has proven to be the best model up to that moment in the week. The model that has made the best predictions at the end of the week will be saved. The other temporary models will be rolled back to the
state of the previous week. If none of the models performed better than a maximum logarithmic loss (empirically defined to be 3), all models are rolled back to their state at the end of the previous week. Afterwards a new model will be created, initialised with the measurements of the past week. With this approach, the model will always automatically choose the model that fits best to the current daily pattern of the user, which will increase the overall prediction accuracy.

Another benefit of this approach can be illustrated with the following example. A student might leave to the city in which he studies on Sunday night during the academic year, but he will not do this during the holidays. If there would be only one model, the transition probability of going to another city on Sunday night will heavily decrease during holidays. This will have as an effect that when school starts again, the model will not know that the student is leaving. If there were two models, the transition probabilities in the model that captures the academic year remain untouched during holidays.

A last benefit of the parallel model approach lies in the area of computation cost. Each model will not have all existing locations in its matrices (e.g. people mostly do not go to work when they are on holiday). Thus, the matrices are smaller and the computations will be less expensive. This means that the computations are more likely to stay within reasonable boundaries, justifying that detections and predictions are performed on the smartphone itself instead of moving them to a server. Performing the computations locally decreases the overall system complexity and ensures that there is no need to have an always running internet connection.

5.5 Quantifying the degree of abnormality

In some situations it is impossible to predict the user’s location, for example when the user occasionally goes to a party or takes a day off from work. In these cases the predictions will obviously be completely wrong. However, useful information can still be derived from this situation. In some cases it can be beneficial to know when a user left his usual behavioural pattern. Therefore, a number that quantifies the degree of the abnormality of the user’s behaviour is created. This number is called the Anomaly Grade and can be calculated using the previous Anomaly Grade and the accuracy of the predictions of the current sample:

\[
Anomaly\ Grade_{\text{new}} = 0.9 \times Anomaly\ Grade_{\text{old}} + 0.1 \times \frac{\text{Avg. log. loss}_{\text{sample}}}{-\ln 0.001} \quad (5.24)
\]

When looking at the formula, one can see that if a series of unusual samples is encountered, the Anomaly Grade will go up. The initial value of the Anomaly Grade is 0. The maximal
value is 1, because the second term normalises the average log loss with the maximal possible value (see Section 5.3). When the user went back to his usual behaviour, the Anomaly Grade will gradually go down again. The results in Section 5.6.4 show that this number can indeed clearly indicate unusual behaviour, which can be very useful for developers of recommender systems. The Anomaly Grade is available for developers through the Context Façade, as illustrated in Section 7.2.2.

5.6 Results

5.6.1 Location prediction examples

In the data processing application of the framework, the most likely locations for the next 12 hours of the currently best performing model are always displayed. It also shows the average time at the location, the coordinates of the location, the number of the subsequent samples at this location and how long you are already at this location.

A few screenshots are given in Figure 5.1. Figure 5.1a is a screenshot taken at the beginning of a weekly lab session on Thursday morning at location 181. The system correctly predicts when the lab session will end and when I will go back to my studio (location 26) again. Figure 5.1b is taken on a Friday morning, before a lecture that starts at 11:30 a.m. The framework correctly indicates that I will go to this lecture (location 176), stay at my studio (location 26) for a short while after the lecture and then go home (location 6) for the weekend, because it is Friday evening.

Figure 5.1c and 5.1d are, just as Figure 5.1a, made on Thursday morning, when the lab session should start at 8:30 a.m. However, this week the lab session was cancelled. The system cannot know this beforehand, and thus predicts that the lab session will start at 8:30 a.m. at location 181. However, when the system noticed that I was still at location 26 at 8:40 a.m., it concluded that I would not be going to this lab session anymore, because it already would have started. The reason why the system does not predict this any more is because it has not seen transitions from location 26 to location 181 later than 8:30 a.m. on Thursday morning before.
Figure 5.1: Screenshots of location module in data processing application
5.6.2 Prediction accuracy

The average logarithmic loss of the models at the end of every week between the start of the measurements and the end of the year is written in Table 5.1. Note that week 5 and 6 were merged because of a lack of data due to bugs and app crashes. As could be expected, starting from the beginning of the academic year (week 7), a second model was needed, because the daily location pattern and the visited locations differed a lot from what was known to the current model up to then. Because the weeks during the academic year are a lot more regular (due to fixed time slots for classes and lab sessions), the logarithmic loss is significantly lower during this period. It is also clear that the longer this model is trained, the higher the accuracy becomes. In the first weeks of the academic year, the second model reaches loss values of about 0.80 on average. In later weeks, loss values of 0.50 are reached almost every week.

It can also be noticed that when the semester has ended, in week 20, Model 1 becomes the best model again. This is because the daily location pattern now became the same of the first weeks again, and Model 1 is the model that has been trained for this situation. This indicates that using multiple models is really useful to adapt to different situations.

The same conclusion can be drawn from Figure 5.2. In this figure, the average logarithmic loss of the models since the beginning of the week is plotted once per day before and during the first semester. It can easily be seen that Model 2 starts in the second week of the academic year (28/09/2015), when Model 1 has proven to be insufficient to model the scholar pattern (it has a logarithmic loss greater than 3). Because Model 2 is trained with only scholar pattern data, it is always better than the model that was trained during the holidays (except for November 2nd, which was a holiday on Monday, on which I stayed at home). At the end of the semester, during the Christmas Holidays, Model 1 seems to be the best model again. This is so because during this period other locations are visited again: the same locations as in August and September. This again clearly proves that both models have their own expertise: Model 1 is trained to make accurate predictions when at home, Model 2 is trained to make accurate predictions when at school in Ghent.

One may see two other strange things in the graph. The first is the zero logarithmic loss on the 22nd of September. The cause for this is that due to a bug, only 4 samples were taken that day which seems to have resulted in no mistakes at all. However, this zero logarithmic loss is not representative for the performance of the model. The other remarkable thing that can be noticed is the periodic behaviour of, especially, the first model. This is because during the week, when in Ghent, this model performs very bad. But during the weekend, when at home, it knows the visited locations, and decreases its average loss.

Another prediction accuracy plot is given in Figure 5.3. This figure shows the daily
Table 5.1: Logarithmic loss of location models during 20 weeks

<table>
<thead>
<tr>
<th>Period</th>
<th>Logarithmic Loss</th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>10/08/2015-16/08/2015</td>
<td>2.02</td>
<td>/</td>
</tr>
<tr>
<td>Week 2</td>
<td>17/08/2015-23/08/2015</td>
<td>1.28</td>
<td>/</td>
</tr>
<tr>
<td>Week 3</td>
<td>24/08/2015-30/08/2015</td>
<td>0.93</td>
<td>/</td>
</tr>
<tr>
<td>Week 4</td>
<td>31/08/2015-06/09/2015</td>
<td>1.04</td>
<td>/</td>
</tr>
<tr>
<td>Week 5/6</td>
<td>07/09/2015-20/09/2015</td>
<td>2.31</td>
<td>/</td>
</tr>
<tr>
<td>Week 7</td>
<td>21/09/2015-27/09/2015</td>
<td>4.14</td>
<td>/</td>
</tr>
<tr>
<td>Week 8</td>
<td>28/09/2015-04/10/2015</td>
<td>2.48</td>
<td>0.70</td>
</tr>
<tr>
<td>Week 9</td>
<td>05/10/2015-11/10/2015</td>
<td>4.84</td>
<td>0.68</td>
</tr>
<tr>
<td>Week 10</td>
<td>12/10/2015-18/10/2015</td>
<td>3.22</td>
<td>1.01</td>
</tr>
<tr>
<td>Week 11</td>
<td>19/10/2015-25/10/2015</td>
<td>3.65</td>
<td>0.33</td>
</tr>
<tr>
<td>Week 12</td>
<td>26/10/2015-01/11/2015</td>
<td>2.26</td>
<td>0.88</td>
</tr>
<tr>
<td>Week 13</td>
<td>02/11/2015-08/11/2015</td>
<td>2.63</td>
<td>0.89</td>
</tr>
<tr>
<td>Week 14</td>
<td>09/11/2015-15/11/2015</td>
<td>2.71</td>
<td>0.47</td>
</tr>
<tr>
<td>Week 15</td>
<td>16/11/2015-22/11/2015</td>
<td>2.91</td>
<td>0.50</td>
</tr>
<tr>
<td>Week 16</td>
<td>23/11/2015-29/11/2015</td>
<td>3.74</td>
<td>0.45</td>
</tr>
<tr>
<td>Week 17</td>
<td>30/11/2015-06/12/2015</td>
<td>3.05</td>
<td>0.49</td>
</tr>
<tr>
<td>Week 18</td>
<td>07/12/2015-13/12/2015</td>
<td>2.62</td>
<td>0.41</td>
</tr>
<tr>
<td>Week 19</td>
<td>14/12/2015-20/12/2015</td>
<td>2.87</td>
<td>0.66</td>
</tr>
<tr>
<td>Week 20</td>
<td>21/12/2015-27/12/2015</td>
<td>1.68</td>
<td>3.72</td>
</tr>
</tbody>
</table>

average logarithmic loss of the same models during the holidays after the first semester exams and the first part of the second semester. It again shows that from the moment the second semester starts, Model 2 takes over again. However, it must be noted that the loss is now slightly higher than during the first semester, especially in the beginning of the semester. This can be explained by the fact that this model has learned the schedule of the first semester and will thus expect the user to have the same schedule during the second semester. This is obviously not the case and the model will gradually have to learn the new schedule. It is thus logical that the loss decreases after a few weeks.

It is important to note that these results are not only based on the most probable location. It is, as mentioned before, possible that in case of an incorrect most probable location, the correct location is also quite probable. In this case the most probable location is incorrect, but the prediction is not as bad as it seems. For example, Figure 5.4 shows a screenshot of the most probable locations during a lecture at the beginning of the second semester. Because the course schedule changed in this semester, the predictions of the most probable location are wrong: the correct prediction would be location 43 in the next
4 hours instead of location 42. This looks as a bad mistake, but in Table 5.2 one can see that the probability of location 43 is also non-negligible. The other relatively probable location, location 35, is a local sandwich place. This example clearly illustrates that it is dangerous to only consider the most probable location, which proves the benefit of using the logarithmic loss. Developers of recommender systems using the framework are thus advised to consider more locations than only the most probable location (e.g. all locations that sum up to at least 90% of all probability). This will also be supported by the Context Façade, as shown in Section 7.2.2.

Table 5.2: Illustration of most probable locations during a lecture at 11.07 a.m.

<table>
<thead>
<tr>
<th>Location</th>
<th>Probability</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>0.5049</td>
<td>School</td>
</tr>
<tr>
<td>42</td>
<td>0.2929</td>
<td>Home</td>
</tr>
<tr>
<td>35</td>
<td>0.1732</td>
<td>Sandwich Place</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 5.2: Daily logarithmic loss of location predicting models before and during the first semester.
Figure 5.3: Daily logarithmic loss of location predicting models during the holidays after the exams and the beginning of the second semester.

Figure 5.4: Screenshot of incorrect most probable location during a lecture at 11.07 a.m.
5.6.3 Comparison with simple model

In this section, the location models are compared with a simple model that always predicts the most visited location since the beginning of the week with 100% confidence. It can be expected that, for example, during a week of the academic year a student spends most of the time at his studio. The simple model in Figure 5.5 will thus predict this location. In this figure, the basic model is compared to the best model during the first semester of the academic year.

Figure 5.5: Comparison of logarithmic loss of the a simple model and the best model during the first semester.

It is clear that throughout the semester, the best model is better than the simple model. This is the case because when not at school the simple model is correct with confidence 1, but when at school or somewhere else, the simple model is completely incorrect, resulting in a loss of 6.91 for that prediction. Since it makes many of those mistakes, the average logarithmic loss is rather high. However, when comparing the logarithmic loss of the simple model with the logarithmic loss of Model 1 in Figure 5.2, one can see that the simple model does perform better than Model 1. This can be explained by the fact that the simple model always predicts the most probable location, while Model 1 predicts the locations it has seen during the holidays. It should thus not come as a surprise that the simple model performs worse than Model 1 during the holidays, as illustrated in Figure 5.6.

As a conclusion, it can be said that using the location models does have an added value compared to some basic predictor. It is clear that the models specialise in a pattern
they know. This can be concluded by the fact that both models perform better than the simple model in the periods they are trained for, but perform worse otherwise, because they simply do not consider the locations that are visited during these periods.

5.6.4 Illustration of Anomaly Grade

The Anomaly Grade describes, as mentioned in Section 5.5, the degree of abnormality in the location behaviour of the user. This is quantified by a number between 0 (normal) and 1 (abnormal). Figure 5.7 shows the Anomaly Grade of the best model between the 22nd and the 28th of February.

The 2 highest peaks correctly indicate unlikely events. On the 25th, I went to a local restaurant in the evening and on the 27th there was a family gathering. Even though the predictions were wrong, a developer can now be alerted about the fact that unusual things are going on. The less significant peaks indicate events like a lesson that took place later because of a job fair and going shopping.
Figure 5.7: Evolution of the Anomaly Grade between the 22nd and 28th of February.
Chapter 6

Adding meta-information to
locations

6.1 Overview

Now the locations of the user can be detected and predicted, it is also useful to enrich these locations with meta-information. Some examples of interesting meta-information are the following:

- Name of the location
- Category of the location
- Rating of the location (e.g. restaurants, bars, ...)
- Popularity of the location (e.g. number of check-ins on foursquare)
- Whether or not this is the home location of the user

Seen from the perspective of recommender systems, the category of the location is probably one of the most interesting meta-information types. The category of the location will tell much about what the user currently is doing or what he has been doing recently. Recommender systems will act completely different when the user is at a bar or a certain event, compared to a normal day at the office.

In order to gather all this meta-information, the Foursquare and Google Places API will be used. Both APIs contain many of locations and are easily accessible. In order to automatically detect the home location of the user, a simple algorithm is developed.

6.2 Automatic detection of home location

If there is one assumption that can be made about the home location of a user, it is that the user usually sleeps at this location. For this reason, the home detection algorithm
is ran every time the user wakes up (after being inactive for at least 4 hours). The algorithm should also support the detection of multiple home locations, since it should also be detected that students, for example, live at two locations: the city in which they study during the week and at their parents during the weekend.

For every location it is kept how frequent the user has woken up at this location. This information is stored in a row matrix with as many columns as there are locations:

\[ F = \begin{bmatrix} f_{l_1} & f_{l_2} & f_{l_3} & \ldots & f_{l_n} \end{bmatrix} \]  

(6.1)

Every time the user wakes up, the value of the corresponding location in the frequency matrix \( F \) is incremented by 1. However, in order to support that users change home, all other frequencies are decreased with the fixed value of 0.01 before incrementing the current location frequency. By gradually decreasing the frequencies, the system will sooner detect that the user now lives at another place. The actual home detection is done by normalising the matrix (divide every element by the sum of all elements). All locations that have a normalised frequency larger than 0.10 are considered as a home location.

To illustrate this algorithm, consider a student that arrives in the city in which he studies on Sunday evening and goes back home on Friday evening. Suppose the framework is initialised on Sunday and it knows 5 locations, of which his parents’ house is location 1 and the student’s studio is location 4. Initially, after the initialisation on Sunday the frequency matrix \( F_{\text{Sunday}} \) will contain all zero values:

\[ F_{\text{Sunday}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]  

(6.2)

The student will wake up at location 4 (his studio) from Monday till Friday, and at location 1, his parents’ house, on Saturday and Sunday. All frequency matrices and normalised frequency matrices for every day of the week are depicted below.

On Monday, location 4 is incremented:

\[ F_{\text{Monday}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]  

(6.3)

\[ F_{\text{Monday,normalised}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]  

(6.4)

Before incrementing location 4 again on Tuesday, all current values are decreased by 0.01:
The same is done from Wednesday till Friday:

\[
F_{\text{Wednesday}} = \begin{bmatrix} 0 & 0 & 0 & 2.98 & 0 \end{bmatrix} \quad (6.7)
\]

\[
F_{\text{Wednesday, normalised}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (6.8)
\]

\[
F_{\text{Thursday}} = \begin{bmatrix} 0 & 0 & 0 & 3.97 & 0 \end{bmatrix} \quad (6.9)
\]

\[
F_{\text{Thursday, normalised}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (6.10)
\]

\[
F_{\text{Friday}} = \begin{bmatrix} 0 & 0 & 0 & 4.96 & 0 \end{bmatrix} \quad (6.11)
\]

\[
F_{\text{Friday, normalised}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (6.12)
\]

During the weekend, the student stays at location 1:

\[
F_{\text{Saturday}} = \begin{bmatrix} 1 & 0 & 0 & 4.95 & 0 \end{bmatrix} \quad (6.13)
\]

\[
F_{\text{Saturday, normalised}} = \begin{bmatrix} 0.168 & 0 & 0 & 0.832 & 0 \end{bmatrix} \quad (6.14)
\]

\[
F_{\text{Sunday}} = \begin{bmatrix} 1.99 & 0 & 0 & 4.94 & 0 \end{bmatrix} \quad (6.15)
\]

\[
F_{\text{Sunday, normalised}} = \begin{bmatrix} 0.287 & 0 & 0 & 0.713 & 0 \end{bmatrix} \quad (6.16)
\]

One can see that by Sunday, the algorithm considers both the student’s studio and his parents’ house as home locations. This would also have been the case if the student only slept at his parents’ house on Saturday night instead of both Friday and Saturday night.

Now suppose the student occasionally sleeps at another location, e.g. location 3. The frequency matrices will look as follows:
\[ F'_{\text{Monday}} = \begin{bmatrix} 1.98 & 0 & 1 & 4.93 & 0 \end{bmatrix} \]  \quad (6.17)

\[ F'_{\text{Monday,normalised}} = \begin{bmatrix} 0.250 & 0 & 0.127 & 0.623 & 0 \end{bmatrix} \]  \quad (6.18)

With a home location threshold of 0.1, the system will now also consider location 3 as a home location, which is incorrect. Fortunately, the longer the algorithm will be active, the more robust it will be against occasional sleeping at different places. More samples at other locations will be needed to remove existing home locations or to let new locations be considered as home location. For example, suppose this occasional sleepover took place after 2 complete weeks in the regular pattern, in which a week consists of 5 nights at the studio and 2 nights at the house of the parents. The frequency matrices now show that this location is not considered as a home location any more:

\[ F_{2 \text{ weeks} + 1 \text{ day}} = \begin{bmatrix} 3.91 & 0 & 1 & 9.86 & 0 \end{bmatrix} \]  \quad (6.19)

\[ F_{2 \text{ weeks} + 1 \text{ day,normalised}} = \begin{bmatrix} 0.265 & 0 & 0.067 & 0.668 & 0 \end{bmatrix} \]  \quad (6.20)

This example shows that after a short time period the algorithm is already able to detect less frequent home locations, while still being robust to occasional sleepovers at locations that are not the home location of the user.

### 6.3 Location labeling with Google Places and Foursquare

To enrich the locations with meta-information like their name, category, popularity or rating, the Google Places API and Foursquare API are used. Both APIs contain many locations and are easily accessible. The major difference between both APIs is that Google supervises its places, while in the case of Foursquare every user is able to create venues with arbitrary information. So called Foursquare superusers can control these venues and edit or remove them if necessary, but it is still not guaranteed that every location contains the correct information \[16\]. One can thus say that if for a certain location information is present in both APIs, the information of the Google Places API is more trustworthy than the Foursquare information.

When a user is at a certain location for at least two samples in a row, a request is posted to both APIs asking for the most likely near locations. This returned information will then be used to label the current location. How this process takes place will be explained in the next sections.
6.3.1 Supervised location labeling

In contrast to the detection of home locations, location labeling can unfortunately not be done automatically. This is because the user’s coordinates will not always be accurate enough to precisely determine where the user is. Further, not all locations will be locations that are known to Google Places, Foursquare or any other database.

Thus, it is unavoidable that the user will have to indicate whether or not he is at a certain location. To make this process as simple as possible, the user is prompted with a notification after the second sample at a certain unlabeled location (see Figure 6.1a). The user then has the choice to choose one of these proposed labels if it corresponds to his current location (see Figure 6.1b). If the user denies all label proposals, new labels (if there are any left) will be sent to the user after the next measurement took place. If the user is confident that both Google or Foursquare will not contain the user’s current location, he can choose to deny any further proposals. In this case, the system will not send new label proposals for this location to the user.

Once the user has chosen a label for a certain location, the useful information about this location is stored. However, there are some legal restrictions to the storage of this information. These legal restrictions will be handled in Section 6.3.3.

![Figure 6.1: Screenshots of location labeling process](image)

(a) Notifications shown to the user  
(b) An example label proposal
6.3.2 Joint location categories

Both Google Places and Foursquare have different ways of representing the location categories. Google Places has a flat structure of 97 categories [15], while Foursquare has an entirely different way of representing location categories: it uses a (very!) extensive tree of categories with 10 categories at the root of the tree [14].

In order to allow the usage of both APIs to determine the location category, a mapping between both category representations has been made. The framework will use a flat representation of 13 joint categories, which contains the following categories with corresponding category codes:

- Arts & Entertainment (Code 1)
- Education (Code 2)
- Event (Code 3)
- Food (Code 4)
- Nightlife (Code 5)
- Outdoors & Recreation (Code 6)
- Residence (Code 8)
- Shop & Service (Code 9)
- Travel & Transport (Code 10)
- Medical (Code 11)
- Religion (Code 12)
- Government (Code 13)
- Other (Code 0)

This representation is mostly derived from the Foursquare category tree. It consists of the 10 root categories (of which some names have slightly been changed) with 3 additional categories. The additional categories (‘Medical’, ‘Religion’ and ‘Government’) were mostly derived from places in the ‘Other’ category because it seemed useful to put some of these places into separate categories. Also, some related places where not in the same category. For example, the places ‘Middle School’ and ‘High School’ were not put in the ‘Education’ category, but in ‘Others’. This made it necessary to go through the ‘Others’ category and make a custom mapping of the categories.
When a location is labeled with Google Places or Foursquare, the joint category is updated using the above mentioned joint category mapping. If category information of both Google Places and Foursquare is available, the Google Places category will have priority over the Foursquare category, because the Google Places information is considered to be more trustworthy (as argued at the beginning of this section).

### 6.3.3 Legal restrictions

Both Google and Foursquare have some legal restrictions concerning the storage of information of their APIs. Foursquare says a developer may ‘not cache or store any Foursquare places information (including tips and venue photos) for more than 30 days without refreshing’[17]. When it comes to the Google Places API, a developer may not pre-fetch, cache, index, or store any Content to be used outside the Service, except that you may store limited amounts of Content solely for the purpose of improving the performance of your Maps API Implementation due to network latency (and not for the purpose of preventing Google from accurately tracking usage), and only if such storage is temporary (and in no event more than 30 calendar days), secure, does not manipulate or aggregate any part of the Content or Service and does not modify attribution in any way[18].

Thus, both APIs severely restrict the storage of their information. Foursquare seems to be the most tolerant, since they only require that their information has to be refreshed every 30 days. For this reason, an automatic, monthly refresh mechanism should be implemented in order to comply to the legal restrictions of this API. The storage and usage of Google Places is a lot more restricted. Next to the same 30 day interval as Foursquare, there also is a set of rules the application should comply with in order to be allowed to store the places.

Since this is a research project that is currently not commercially available, the framework does not comply with the restrictions of the APIs yet. Of course there has to be looked after these legal restrictions if the framework would become available for other developers.
Chapter 7

Context Façade

The Context Façade is created for developers of recommender systems to enable easy and quick access to the data that is collected and processed by the framework. Developers can add the façade to their project (as explained in Section 7.1) and ask for the information they need. As a whole, the supported functions make available all information that has been gathered and processed by the framework.

The documentation for all these functions is given in Section 7.2. The list of possible functions is not limited to the functions provided here. The functions that are currently supported are there to show what is possible using the collected and processed data. It is still possible to add more functionality using the data that is available. Of course, developers that use the library can also create more intelligent functions using the currently available functions as a basis.

### 7.1 How to use the Context Façade

The Context Façade is created as an Android Archive (.aar) library. To add it to a project, one should just copy the facade.aar file to the application libs folder. Now, to compile the library with the project using Android Studio and Gradle, the following lines have to be added to the application’s build.gradle file:

```groovy
repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    compile (name: 'facade', ext: 'aar')
}
```
If there are other dependencies in the project (like e.g. the `appcompat` library), the dependencies have to be merged.

Once the Context Façade has been added to the project, the developer can create a `ContextFacade` object using the application context as a parameter for the constructor. The `ContextFacade` object supports all functions mentioned in Section 7.2. Obviously, using these functions requires having installed the `Context Collection` and `Context Processing` application (i.e. having a running framework).

### 7.2 Supported functions

The supported functions can be divided into four categories: Activity Level functions, Location functions, Activity Recognition functions and Weather functions.

#### 7.2.1 Activity level functions

This category of functions uses the activity level (a value between 0 and 1) of the user as defined in Chapter 4. A higher activity level indicates a higher confidence in being active.

The following functions are supported:

- `getCurrentActivityLevel()`
  - **Input parameters:** /
  - **Returns:**
    * **Type:** double
    * **What:** The user’s last known activity level. Returns -1 if the activity level is not available.

- `getActivityLevelAt(String date)`
  - **Input parameters:**
    * **String date:** String defining the moment for which the activity level is requested. The String should be ISO8601 formatted as follows: `yyyy-MM-dd'T'HH:mm:ss.S'Z'` (e.g. `2016-03-12T18:23:14.5Z`).
  - **Returns:**
    * **Type:** double
    * **What:** The user’s activity level closest to the defined moment. Returns -1 if no activity level could be found.

- `getActivityLevelIn(int quarters)`

61
- **Input parameters:**
  
  * **int quarters:** The number of quarters in the future of which the activity level is requested. The predictions go 24 hours in the future, so 96 is the highest available amount of quarters.

- **Returns:**
  
  * **Type:** double
  * **What:** The predicted activity level in the defined amount of quarters. Returns -1 if there are no predictions available or an incorrect amount of quarters is given.

- **wokeUpAt(String date, double threshold)**

  - **Input parameters:**
    
    * **String date:** String defining the day for which the time of waking up is requested. The String should be ISO8601 formatted as follows: `yyyy-MM-dd'T'HH:mm:ss.S'Z'` (e.g. `2016-03-12T18:23:14.5Z`), but can be truncated up to `yyyy-MM-dd` (e.g. `2016-03-12`).
    * **double threshold:** Double value between 0 and 1, defining the threshold to consider an activity level as active. It is advised to choose the threshold around 0.25, since this yielded the best results after some experimenting.

  - **Returns:**
    
    * **Type:** String
    * **What:** An ISO8601 formatted String (see input parameters) defining the moment of waking up. If no moment of waking up is found, null is returned.

As described in Section 4.3.1.1, activity level detection is very sensitive to false positives and false negatives. To be as robust as possible, the algorithm works as follows:

1. Start by pointing at the very first measured activity level of the given day.
2. If the first activity level is not yet below the threshold, find the first inactive activity level starting from the current pointer by scanning forward.
3. Go through all subsequent samples until an active sample is encountered again. This is a potential wake up event.
4. Go through the subsequent samples. If an active period of more than 1.5 hours is encountered, return the start of this active period. Otherwise, when an inactive sample is encountered earlier, go back to step 3 if no more than 24 hours of samples have been scanned through. If the entire day has been analysed without finding an active period of more than 1.5 hours, null is returned.
• `wentToSleepAt(String date, double threshold)`

  – Input parameters:
    * **String date**: String defining the day for which the the time of going to sleep is requested. The String should be ISO8601 formatted as follows: `yyyy-MM-dd'T'HH:mm:ss.S'Z'` (e.g. `2016-03-12T18:23:14.5Z`), but can be truncated up to `yyyy-MM-dd` (e.g. `2016-03-12`).
    * **double threshold**: Double value between 0 and 1, defining the threshold to consider an activity level as active. It is advised to choose the threshold around 0.50, since this yielded the best results after some experimenting.

  – Returns:
    * **Type**: String  
    * **What**: An ISO8601 formatted String (see input parameters) defining the moment of going to sleep. If no moment of going to sleep is found, null is returned.

As described in Section 4.3.1.1, activity level detection is very sensitive to false positives and false negatives. To be as robust as possible, the algorithm works as follows:

1. Start by pointing at the very last measured activity level of the given day.
2. Find the beginning of a series of inactive samples. If the current activity level is already below the threshold, look back to find the first inactive sample. If the current activity level is not yet below the threshold, go further to find the first inactive sample. Note that these samples are samples taken on the day after the defined day! The first inactive sample is a potential going to sleep event.
3. Go through the subsequent samples. If an inactive period of more than 1.5 hours is encountered, return the start of this inactive period. Otherwise, when an active sample is encountered earlier, go back to step 2 if no more than 24 hours of samples have been scanned through. If the entire day has been analysed without finding an inactive period of more than 1.5 hours, null is returned.

### 7.2.2 Location functions

This category of functions comprises everything that has to do with the locations the user visits. As will be clear from the function definitions, a developer that asks for a location will always receive the unique ID of the location. He can then ask for more information about this location using the received unique ID.
The first series of functions are functions that return the IDs of the locations:

- **getCurrentLocation()**
  
  - **Input parameters:** /
  
  - **Returns:**
    * **Type:** int
    * **What:** The ID of the current location. If the current location is unknown, -1 is returned.

- **getLocationsBetween(String date1, String date2)**
  
  - **Input parameters:**
    * **String date1:** String defining the lower boundary of the interval. The String should be ISO8601 formatted as follows: yyyy-MM-dd'T'HH:mm:ss.S'Z' (e.g. 2016-03-12T18:23:14.5Z), but can be shortened in order to meet the desired accuracy (e.g. 2016-03, 2016-03-12 and 2016-03-12T18 are also valid, where the left out numbers are considered 0). If an empty string is given, all locations since the first measurement are returned.
    * **String date1:** String defining the upper boundary of the interval. The String should be ISO8601 formatted as follows: yyyy-MM-dd'T'HH:mm:ss.S'Z' (e.g. 2016-03-12T18:23:14.5Z), but can be shortened as illustrated above. If an empty string is given, all locations up to the last measurement are returned.
  
  - **Returns:**
    * **Type:** Set<Integer>
    * **What:** A Set containing the IDs of all locations visited between the two defined dates. Returns an empty Set if no locations have been found.

- **getLocationAt(String date)**
  
  - **Input parameters:**
    * **String date:** String defining the lower boundary of the interval. The String should be ISO8601 formatted as follows: yyyy-MM-dd'T'HH:mm:ss.S'Z' (e.g. 2016-03-12T18:23:14.5Z), but can be shortened as illustrated in the previous function. The left out numbers are considered 0.
  
  - **Returns:**
    * **Type:** int
    * **What:** The ID of the location at the defined moment. If no location was known at that time, -1 is returned.
• getLocationsIn(int hours, double threshold)
  
  - **Input parameters:**
    * int hours: The number of hours in the future for which the location predictions are requested.
    * double threshold: A value between 0 and 1 that defines how many locations are returned. Locations are added to the returned List until the sum of the confidence in all returned locations is greater than or equal to the threshold value.
  
  - **Returns:**
    * Type: List<Entry<Double, Integer>>
    * What: The returned List contains Entry objects containing the most probable locations. The Entry class is provided in the library and is nothing more than a key-value pair. The keys are the probabilities of the locations, the values are the IDs of the locations. The probabilities of the locations sum up to at least the threshold value. They are added in order, such that when one iterates over the List, the most probable location is encountered first. An empty List is returned when an incorrect amount of hours or an incorrect threshold is given or when no predictions are available.

A second series of functions are functions to request more information about a specific location:

• lastTimeAtLocation(int id)
  
  - **Input parameters:**
    * int id: The ID of the location.
  
  - **Returns:**
    * Type: String
    * What: An ISO8601 formatted string (e.g. 2016-03-12T18:23:14.5Z) indicating the last sample that was taken at the provided location. If the user is currently at this location, the last time during the previous visit is returned (and not the current sample). If no previous visit is found, null is returned.

• getHoursAtCurrentLocation()
  
  - **Input parameters:** /
  
  - **Returns:**
* Type: double
* What: The amount of hours that the user is at the current location. If the current location is unknown, -1 is returned.

- **getAnomalyGrade()**
  - Input parameters: /
  - Returns:
    * Type: double
    * What: The current anomaly grade, as defined in Section 5.3

- **isHomeLocation(int id, boolean reload)**
  - Input parameters:
    * int id: The ID of the location.
    * boolean reload: Whether or not the locations have to be refreshed before executing the function. It is advised to refresh the locations frequently if the same ContextFacade object is used for a long time.
  - Returns:
    * Type: boolean
    * What: Whether or not the location is considered to be a home location of the user.

- **getCategory(int id, boolean reload)**
  - Input parameters:
    * int id: The ID of the location.
    * boolean reload: Whether or not the locations have to be refreshed before executing the function. It is advised to refresh the locations frequently if the same ContextFacade object is used for a long time.
  - Returns:
    * Type: int
    * What: The category code of the mapping between Google Places and Foursquare categories, as defined in Section 6.3.2

- **getName(int id, boolean reload)**
  - Input parameters:
    * int id: The ID of the location.
* boolean reload: Whether or not the locations have to be refreshed before executing the function. It is advised to refresh the locations frequently if the same ContextFacade object is used for a long time.

- Returns:
  * Type: String
  * What: The name of the location after it has been labeled by the user. If no label is present, null is returned. If both a Google Places and Foursquare label are available, the Google Places label is returned.

- getAverageHoursAtLocation(int id, boolean reload)

  - Input parameters:
    * int id: The ID of the location.
    * boolean reload: Whether or not the locations have to be refreshed before executing the function. It is advised to refresh the locations frequently if the same ContextFacade object is used for a long time.

  - Returns:
    * Type: double
    * What: The average amount of hours that the user is expected to be at the location. If no average amount of hours is available (e.g. on the first visit), -1 is returned.

- getCoordinates(int id, boolean reload)

  - Input parameters:
    * int id: The ID of the location.
    * boolean reload: Whether or not the locations have to be refreshed before executing the function. It is advised to refresh the locations frequently if the same ContextFacade object is used for a long time.

  - Returns:
    * Type: CoordinateInfo
    * What: The coordinate information of the location is returned as a CoordinateInfo object. The CoordinateInfo class is provided in the library and contains three elements: the latitude, the longitude and the coordinate accuracy. The coordinate information can be requested with the following functions: getLatitude(), getLongitude() and getAccuracy(). All functions return a double value. If no coordinates are available, null is returned.
7.2.3 Activity recognition functions

This function category is about the recognised activities using the Google Activity Recognition API (see Section 3.2.2). It is possible to get the current activities of the user or to ask for the activities at a certain moment in history using the functions mentioned below.

Activities are returned as a `List<Entry<Double, String>>`. The Entry class is provided in the library and is nothing more than a key-value pair. The keys are the confidence in the detected activities, the values are Strings describing the activities (see Section 3.2.2). The label ‘unknown’ is excluded. The activities are added in order, such that when one iterates over the List, the most probable activities are encountered first. An empty List is returned when no activities are recognised or the Activity Recognition API is unavailable.

- **getCurrentActivities()**
  - **Input parameters:** /
  - **Returns:**
    * **Type:** List<Entry<Double, String>>
    * **What:** A List containing the recognised activity Entry objects, as defined above.

- **getActivitiesAt(String date)**
  - **Input parameters:**
    * **String date:** String defining the moment for which the activities are requested. The String should be ISO8601 formatted as follows: `yyyy-MM-dd’T’HH:mm:ss.S’Z’` (e.g. `2016-03-12T18:23:14.5Z`), but can be shortened as much as wanted. All left out numbers are considered 0.
  - **Returns:**
    * **Type:** List<Entry<Double, String>>
    * **What:** A List containing the recognised activity Entry objects at the specified date, as defined above.

7.2.4 Weather functions

The last function category is about the weather. The following functions support requesting the current weather, searching the weather history and asking for weather forecasts. Note that the weather information that is stored contains more information than the functions return. However, for simplicity it is chosen for now to constrain the returned values to a weather condition code, the temperature and the wind speed.

The weather information is returned in a `WeatherInfo` object. The `WeatherInfo` class is provided in the library and contains three values: the weather condition code (as defined
by OpenWeatherMap \cite{20}, the temperature in degrees Celsius and the wind speed in meter/second. The weather information can be requested from the *WeatherInfo* objects using the following functions:

- `int getConditionCode()`
- `double getTemperature()`
- `double getWindSpeed()`

The following functions are supported by the Context Façade:

- `getMostRecentWeather()`
  - **Input parameters:** /
  - **Returns:**
    * **Type:** `WeatherInfo`
    * **What:** A `WeatherInfo` object, as defined above. If no weather information was received recently, old weather information will be returned. If no weather information has ever been received up to now, null is returned.

- `getWeatherAt(String date)`
  - **Input parameters:**
    * **String date:** String defining the moment for which the weather information is requested. The String should be ISO8601 formatted as follows: `yyyy-MM-dd'T'HH:mm:ss.S'Z'` (e.g. `2016-03-12T18:23:14.5Z`), but can be shortened as much as wanted. All left out numbers are considered to be zero.
  - **Returns:**
    * **Type:** `WeatherInfo`
    * **What:** A `WeatherInfo` object that contains the weather information at the specified date. If no weather information was received at that time, older weather information will be returned. If no weather information had ever been received up to the specified date, null is returned.

- `getWeatherForecast(int location)`
  - **Input parameters:**
    * **int id:** The ID of the location.
  - **Returns:**
    * **Type:** `List<WeatherInfo>`
* **What:** A List containing WeatherInfo objects with the forecast for the next 5 days at the specified location. For every 3 hours there is one element in the list containing weather information. This sums up to a total of 40 elements for the next 5 days (8 per day). However, due to a slower or no internet connection, the List may contain less elements or may even be empty.

- **getWeatherForecastIn(int location, int hours)**
  - **Input parameters:**
    * int id: The ID of the location.
    * int hours: The number of hours in the future for which the weather forecast is requested.
  - **Returns:**
    * Type: WeatherInfo
    * **What:** A weather forecast retrieved from the *getWeatherForecast()* function. The functions picks the most accurate available element and returns it. If, for example, the weather in 2 hours is requested, the weather in 3 hours is returned. If a moment more than 5 days ahead in time is requested, the last available forecast is returned. If no weather forecast is available, null is returned.
Chapter 8

User Test

To validate whether or not the framework works well in practice for people with different daily patterns, a user test has been conducted. During a period of approximately 5 to 6 weeks, 11 people collected and processed data with their smartphones. As will become clear in Section 8.3, not all users started and stopped the test at the same time. After the experiment, it is checked if the framework did a good job at detecting and predicting the locations and activity level, creating new models if necessary, ...

In the next section the distribution of the test subjects is given. Afterwards, Section 8.2 and Section 8.3 present the results of the Activity Submodule and Location Submodule. Next, a short note about how the users perceived the test is given. Finally, the chapter will end with a discussion and conclusion of the user test.

8.1 Test subjects

To be able to test the framework in various situations, both students and working people were chosen as test subjects. There were eight students and three working people involved. Three of the students commute to school on a daily basis, the others stay in the city in which they study during the week. Unfortunately, all working test users had a job with a rather fixed time schedule, so it could not be tested how the framework reacts to people that work in shifts. What could be interesting is that one of the three working test users works in a different city, while the other two work at about ten minutes from their home location.

Although there are not that much test users, it will be interesting to see the differences between the different categories of users. The usage of user categories will prove to be most useful during the evaluation of the location prediction in Section 8.3. The user categories are assigned the following labels:

- Category A: The three working test users.
- Category B: The three commuting students.
• Category C: The five students that (mostly) live in the city in which they study throughout the week, but at home during the weekend.

8.2 Activity level prediction results

In this section the results of the activity level prediction will be summarised. It will be analysed how accurate the predictions for waking up and going to sleep are, and if and how well the framework is able to learn the activity pattern of the users.

Unfortunately not all user data could be used to validate the activity level prediction. Due to some issues with the database, the data of four users could not be used. There was also one user that always turned off his phone when he went to sleep, which caused the framework to assume the user was permanently active at night. Next to these five users, there was yet another issue causing some data to be unusable. As mentioned in Section 4.3.1.1 the activity level evolution is not always as smooth as expected due to false positives or negatives, especially at night. Most often the triggers for the false positives are sounds from the outside or noise (or too sensitive trigger settings) on the acceleration or rotation sensor. Two users severely suffered from this, which caused the predictions of the time of waking up to be unusable for both users. For one of these users, the predictions of the time of going to sleep also turned out to be unusable. Figure 8.1 and Figure 8.2 respectively show a usable and unusable activity profile. In this case, the profile turned out to be unusable because the user slept with his smartphone in his bed, causing the accelerometer to trigger the activity detection. This shows that for some users, there is a need for a better activity level algorithm. At the end of Section 10.1.2 some possible improvements to solve this issue are suggested.

Given the issues mentioned in the previous paragraph, the data of five users remained useful to analyse the quality of the predictions of the time of going to sleep. For the wake up time predictions, the data of four users could still be used. Table 8.1 shows the average deviations between the detected and predicted times for all weeks and the total average deviation. It also shows the same deviation values averaged over all users.

When looking at the results one can see that, on average, the deviations decrease when weeks pass by. For most users the deviations in the last week are the lowest (or one of the lowest) of all weeks. Averaged over all users and all weeks, the errors are 1 hour and 39 minutes and 1 hour and 56 minutes for the time of waking up and going to sleep respectively. Note that the used detected time of waking up and going to sleep is the time that is detected by the Context Façade functions of Section 7.2.1. In Section 4.3.1.2 it is shown that the average errors of the Context Façade functions are 6 minutes and 11 minutes for waking up and going to sleep respectively, when completely incorrect detections are discarded. Because the occasionally incorrect detections are not discarded
Figure 8.1: Usable activity profile that has clearly distinguishable active and inactive periods.

Figure 8.2: Unusable activity profile that does not have clearly distinguishable active and inactive periods.

Here, one should keep in mind that for some samples a non-negligible error can also be caused by the detection functions instead of the predictions.

Compared to Section 4.3.2, where the average deviation was 40 minutes and 33 minutes for respectively waking up and going to sleep, the deviations in the user test are significantly higher. However, it should be noted that the framework had been trained for about 25 weeks in Section 4.3.2. Therefore it seems logical that the errors in the user test are higher and that they are expected to keep decreasing in the future.
Table 8.1: Differences (in hh:mm) between detected and predicted times of waking up and going to sleep for all test users.

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Waking up</td>
<td>02:35</td>
<td>00:27</td>
<td>01:32</td>
<td>00:59</td>
<td>01:06</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>03:25</td>
<td>02:58</td>
<td>01:48</td>
<td>00:47</td>
<td>01:07</td>
</tr>
<tr>
<td>B1</td>
<td>Waking up</td>
<td>02:18</td>
<td>02:11</td>
<td>00:45</td>
<td>00:34</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>02:45</td>
<td>01:37</td>
<td>00:36</td>
<td>01:10</td>
<td>/</td>
</tr>
<tr>
<td>B2</td>
<td>Waking up</td>
<td>02:10</td>
<td>00:28</td>
<td>00:43</td>
<td>01:11</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>00:36</td>
<td>00:56</td>
<td>01:07</td>
<td>01:02</td>
<td>/</td>
</tr>
<tr>
<td>B3</td>
<td>Waking up</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>04:12</td>
<td>02:37</td>
<td>01:10</td>
<td>01:37</td>
<td>01:28</td>
</tr>
<tr>
<td>C3</td>
<td>Waking up</td>
<td>03:36</td>
<td>03:17</td>
<td>02:27</td>
<td>03:05</td>
<td>00:49</td>
</tr>
<tr>
<td></td>
<td>Going to sleep</td>
<td>03:42</td>
<td>01:02</td>
<td>01:20</td>
<td>00:25</td>
<td>01:11</td>
</tr>
</tbody>
</table>

| Mean  | Waking up | 02:39  | 01:36  | 01:22  | 01:27  | 00:57  | 01:39  |
|       | Going to sleep | 02:56  | 01:50  | 01:12  | 01:00  | 01:15  | 01:56  |

8.3 Location prediction results

In this section the location prediction results of the user test are evaluated. First, the results are evaluated using the (as argued in Section 5.3) most suitable error measure to compare models: the logarithmic loss. However, the logarithmic loss may be somewhat hard to grasp because it just produces a dimensionless number. Therefore, the models are also compared by their binary accuracy in Section 8.3.2 to give a better understanding. The accuracy is the percentage of the predictions for which the most probable location was correct.

8.3.1 Evaluation using logarithmic loss

The logarithmic loss for all weeks, averaged over all participating users is depicted in Table 8.2. Averaged over all weeks, the logarithmic loss equals 1,329. As could be expected, one can see that in general the predictions get better over the weeks (because the logarithmic loss decreases). Note that not only the average loss decreases: the standard deviation also decreases, which shows that the predictions get better for all users and not only a few of them. This already indicates that the framework is capable of learning the different location patterns for various categories of users.

It is also interesting to investigate whether or not the prediction accuracy differs for different user categories. Therefore, Table 8.3, Table 8.4 and Table 8.5 show the logarithmic losses for all users and all categories. In the tables it is marked how many measurements
Table 8.2: Average location prediction logarithmic loss over all categories.

<table>
<thead>
<tr>
<th>Date Range</th>
<th>Mean</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-03-2016 / 20-03-2016</td>
<td>1.511</td>
<td>0.766</td>
</tr>
<tr>
<td>21-03-2016 / 27-03-2016</td>
<td>1.906</td>
<td>0.698</td>
</tr>
<tr>
<td>28-03-2016 / 03-04-2016</td>
<td>1.590</td>
<td>0.834</td>
</tr>
<tr>
<td>04-04-2016 / 10-04-2016</td>
<td>1.148</td>
<td>0.665</td>
</tr>
<tr>
<td>11-04-2016 / 17-04-2016</td>
<td>1.084</td>
<td>0.554</td>
</tr>
<tr>
<td>18-04-2016 / 24-04-2016</td>
<td>0.886</td>
<td>0.427</td>
</tr>
<tr>
<td>25-04-2016 / 01-05-2016</td>
<td>0.883</td>
<td>0.401</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1.329</strong></td>
<td><strong>0.647</strong></td>
</tr>
</tbody>
</table>

were taken by the users in every week by giving each cell a color (red = 0 to 500 samples, yellow = 500 to 1000 samples and green = more than 1000 samples). When a cell is marked in red, one should take care when drawing conclusions based solely on this cell. For example, the 0.0 loss of user C4 in the first week is based on 26 samples. It is also important to note that all means and standard deviations are also calculated weighted by the amount of samples that represent every loss value, such that representative values weigh more than less representative values.

Table 8.3: Location prediction logarithmic loss of category A users.

<table>
<thead>
<tr>
<th>Date Range</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Mean</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-03-2016 / 20-03-2016</td>
<td>1.17</td>
<td>/</td>
<td>/</td>
<td>1.17</td>
<td>/</td>
</tr>
<tr>
<td>21-03-2016 / 27-03-2016</td>
<td>2.00</td>
<td>1.80</td>
<td>1.86</td>
<td>1.91</td>
<td>0.10</td>
</tr>
<tr>
<td>28-03-2016 / 03-04-2016</td>
<td>0.50</td>
<td>1.00</td>
<td>1.49</td>
<td>0.96</td>
<td>0.53</td>
</tr>
<tr>
<td>04-04-2016 / 10-04-2016</td>
<td>0.40</td>
<td>0.56</td>
<td>/</td>
<td>0.45</td>
<td>0.10</td>
</tr>
<tr>
<td>11-04-2016 / 17-04-2016</td>
<td>0.94</td>
<td>0.79</td>
<td>1.64</td>
<td>1.15</td>
<td>0.44</td>
</tr>
<tr>
<td>18-04-2016 / 24-04-2016</td>
<td>0.50</td>
<td>0.38</td>
<td>1.07</td>
<td>0.73</td>
<td>0.38</td>
</tr>
<tr>
<td>25-04-2016 / 01-05-2016</td>
<td>/</td>
<td>/</td>
<td>0.81</td>
<td>0.81</td>
<td>/</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>1.09</strong></td>
<td><strong>0.32</strong></td>
</tr>
</tbody>
</table>

A first interesting finding is that category A, the working people category, has the lowest average logarithmic loss. This can probably be explained by the fact that these users have a more regular life compared to students, because most of their social (and often unpredictable) activities take place during the weekend. Even though a standard deviation of three users is not very representative, the lower standard deviation compared to other classes might also indicate more regular behaviour of the users of class A. Another thing that can be seen is that in Table 8.3 for user A1 there is a large increase of loss, from 0.40 to 0.94, after the Easter Holiday (28-03-2016 / 10-04-2016). This can be explained by the fact that user A1 is a teacher who, just like other students, had an entirely different
pattern during the Easter Holiday. Because the models are still very sensitive to changes after a few weeks, the model has been trained to learn the Easter Holiday pattern. This explains why after the Easter Holiday the model did not perform well. During this week the school locations were trained, which caused the loss to decrease to 0.50 in the next week.

Table 8.4: Location prediction logarithmic loss of category B users.

<table>
<thead>
<tr>
<th>Dates</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>Mean</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-03-2016 / 20-03-16</td>
<td>2.67</td>
<td>/</td>
<td>/</td>
<td>2.67</td>
<td>/</td>
</tr>
<tr>
<td>21-03-2016 / 27-03-16</td>
<td>1.58</td>
<td>1.24</td>
<td>2.20</td>
<td>1.57</td>
<td>0.39</td>
</tr>
<tr>
<td>28-03-2016 / 03-04-16</td>
<td>2.21</td>
<td>1.24</td>
<td>0.80</td>
<td>1.44</td>
<td>0.75</td>
</tr>
<tr>
<td>04-04-2016 / 10-04-16</td>
<td>1.22</td>
<td>1.07</td>
<td>0.25</td>
<td>0.83</td>
<td>0.54</td>
</tr>
<tr>
<td>11-04-2016 / 17-04-16</td>
<td>1.81</td>
<td>0.64</td>
<td>1.61</td>
<td>1.49</td>
<td>0.54</td>
</tr>
<tr>
<td>18-04-2016 / 24-04-16</td>
<td>0.84</td>
<td>0.43</td>
<td>1.55</td>
<td>1.14</td>
<td>0.53</td>
</tr>
<tr>
<td>25-04-2016 / 01-05-16</td>
<td>/</td>
<td>/</td>
<td>1.22</td>
<td>1.22</td>
<td>/</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1.32</td>
<td>0.56</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When looking at the student categories, one can see that the mean loss of category B is lower than the loss of category C and that both losses are higher than the loss of category A. When looking at the standard deviation it is clear that the prediction quality varies more in class C. These things might be attributed to the fact that students that commute every day are, just like working people, less likely to go out during the week. However, due to things like varying course schedules, the students of category B are still harder to predict than the working people of category A. Some students of class C probably do more unplanned (and unpredictable) things during the week, which might explain the higher loss and standard deviation. Again, note that three or five test subjects is not enough to have a representative standard deviation and thus the conclusions could vary in a larger test.

Table 8.5: Location prediction logarithmic loss of category C users.

<table>
<thead>
<tr>
<th>Dates</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>Mean</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-03-2016 / 20-03-16</td>
<td>1.54</td>
<td>0.62</td>
<td>1.10</td>
<td>0.00</td>
<td>2.25</td>
<td>1.32</td>
<td>0.64</td>
</tr>
<tr>
<td>21-03-2016 / 27-03-16</td>
<td>1.84</td>
<td>3.65</td>
<td>1.17</td>
<td>/</td>
<td>1.99</td>
<td>2.09</td>
<td>1.04</td>
</tr>
<tr>
<td>28-03-2016 / 03-04-16</td>
<td>1.34</td>
<td>0.80</td>
<td>2.02</td>
<td>2.72</td>
<td>2.94</td>
<td>1.97</td>
<td>0.89</td>
</tr>
<tr>
<td>04-04-2016 / 10-04-16</td>
<td>2.37</td>
<td>1.82</td>
<td>0.83</td>
<td>1.37</td>
<td>1.30</td>
<td>1.53</td>
<td>0.59</td>
</tr>
<tr>
<td>11-04-2016 / 17-04-16</td>
<td>0.21</td>
<td>1.58</td>
<td>0.46</td>
<td>1.16</td>
<td>0.84</td>
<td>0.84</td>
<td>0.55</td>
</tr>
<tr>
<td>18-04-2016 / 24-04-16</td>
<td>1.64</td>
<td>0.82</td>
<td>0.57</td>
<td>1.21</td>
<td>0.52</td>
<td>0.85</td>
<td>0.42</td>
</tr>
<tr>
<td>25-04-2016 / 01-05-16</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>0.20</td>
<td>0.20</td>
<td>/</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1.44</td>
<td>0.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An unexpected result is that only for one user (C2) a second model was created (and this due to an irregular and thus unpredictable school week). Initially it was expected that most students of category B and C would create a second model during the Easter Holiday (from 28-03-2016 to 10-04-2016). It quickly became clear that the users of category B would certainly not get a second model. The reason for this is probably that even though these user’s pattern may vary during holidays, they still sleep at the same location, which is enough to keep the logarithmic loss below the threshold value of 3. To solve this, the error measure should be changed such that it ignores the obvious locations that are the same in every pattern. This fix, in which the home location of the user is ignored for the loss calculations, is proposed as future work in Section 10.1.3.

It still remains to be explained why the users of category C did not create a second model during the Easter Holiday. In a questioning afterwards it appeared that most of the users continued working at school or in their school’s city since all of them were graduate students. Because graduate students spend much time at home to work on their master’s dissertation, it is partially explained why many of them did not have bad predictions during the Easter Holiday. However, some users did go home for at least one week. For example, user C1 was at home during the second week of the Easter Holiday (from 04-03-2016 to 10-04-2016). As expected, the predictions during this week are worse than other weeks, but not bad enough to exceed the 3.0 threshold. Again, this confirms the need for the improved error measure proposed in Section 10.1.3 of Chapter 10 in which the home location of the user is discarded in the loss calculations.

8.3.2 Evaluation using binary accuracy

The binary accuracy is the percentage of the predicted most likely locations that were correct. This measure is probably easier to understand than the logarithmic loss because the result is a clear percentage instead of some number that represents some error. Therefore, it is also useful to evaluate the location prediction results with this measure in order to get a clearer idea about how good the predictions turn out to be.

In Figure 8.3 an overview of the binary prediction accuracy is given for all user categories, together with the accuracy averaged over all categories. The average accuracy at the end of the experiment is situated around 70%. In general, it can be seen that the accuracy increases over the weeks, except for some points that break the monotonicity of the curves. This can often be attributed to deviations in the user pattern like the Easter Holiday, which is unpredictable in the current system. However, for almost all categories the first and the last point of the curve does not show the expected behaviour. This can be explained by looking at Table 8.3, Table 8.4 and Table 8.5. The first and last weeks often have not so many samples (the cells marked in red and yellow in the tables), which often causes the accuracy to be rather unreliable. So, when ignoring these first and last
values the average accuracy is monotonically increasing, which shows that the accuracy improves over the weeks. Thus, it can be concluded that the system indeed learns the user’s location pattern.

Figure 8.3: Overview of the prediction accuracy for all categories and the average prediction accuracy.

8.3.3 Comparing the predictions with a simple model

To evaluate whether or not the model effectively gives an added value, the predictions are compared with a simple model. The simple model is the same model as in Section 5.6.3 where it always predicts the most visited location since the beginning of the week with 100% certainty.

First, the binary accuracy is used as an error measure. A comparison of both models using this measure is given in Figure 8.4. At first sight, the performance does not appear to be as well as expected. In the beginning of the experiment the performance of the location predictions is mostly worse than the simple model and it is only in the last few weeks that the model has learned enough to have a better accuracy than the simple model. It can be expected that the simple model performs reasonably well because most people spend the largest of part of the day at home. Therefore it is not a surprise that for both models the accuracy is comparable. However, the predictions of the real model will probably be more useful, since they will capture the actual location pattern of the user and not only predict the most obvious location. Finally, it is also interesting to note that the performance of the simple model is, as expected, more or less constant throughout the experiment, while
the accuracy of the real model increases. This again shows that the real model is actually learning while the simple model is just always predicting the same thing.

![Figure 8.4: Comparison of location predictions and basic model with the binary accuracy as error measure.](image)

Next, the same comparison is made with the logarithmic loss as error measure. This comparison is depicted in Figure 8.5. It can immediately be seen that here the real model always (and increasingly) performs better than the simple model. The reason for this is that the simple model is punished harder for its mistakes because it always predicts a location with 100% confidence. The real model is a lot more careful with its predictions, causing multiple locations to have a significant probability. If one of these locations appears to be correct, the loss will not be as high as the simple model. Again, note that the simple model always has more or less the same loss, while the loss of the real model decreases. This again indicates that the real model learns the habits of the user and increasingly performs better than the simple model.

So, to conclude one can state that the differences between the simple and the real model are not very visible when looking at the binary accuracy. This is because the simple model will always have a relatively high accuracy, since most people spend a large part of their day at home. However, the differences become clear when comparing the models using the logarithmic loss. Here it is obvious that the real model makes less severe mistakes: it considers multiple locations and assigns significant probabilities to all possible locations. If one of these locations turns out to be the correct one, the loss will not be as high as when an incorrect location had been predicted with 100% confidence.
8.4 User perception

During the experiment, the test users were asked to check the predictions every once in a while. By the end of the testing period the framework seemed to perform generally well. Most users said that the framework more or less captured their activity level pattern (except for the users for which the Activity Submodule turned out to be unusable). Some users also indicated that the framework was capable of predicting when the user would go home, leave for work, or some other location (i.e. fixed habits that are the same every week). Some users also said that they found it remarkable that the system knew more or less when they would go back home when they were at a previously unknown location. Because in this case the predictions depend on the average time at a location and the location where the user mostly is at this moment, the framework can usually make a fairly good prediction. This effect is caused by the correction term of Section 5.2.4, which appears to be beneficial for the perception of the predictions.

As could be expected, some users said in the first weeks that the predictions of both the Activity Submodule and the Location Submodule were not that accurate, which can be attributed to the lack data. Again, one can conclude that the framework needs some weeks to warm up the predictions in order have a general idea about the user’s location and activity pattern.

Another important aspect of the user perception is the usage of battery. Most test users
indicated that there was a noticeable impact on their smartphone’s battery life. However, nobody said that the impact was so dramatic they could not use their smartphone in a proper way anymore. However, measures should be taken in order to further reduce the battery usage. An example of such a measure could be the remote computations mentioned in Section 10.2.1.

Finally, the framework also has an impact on the smartphone’s memory. Since only users with more recent smartphones were asked to participate, nobody ran into problems due to a full memory. However, most of the users were surprised that the samples took up a few hundred megabytes in only a few weeks time. It is clear that in the future something should be done to fix this. Likely measures are remote storage of measurements and partially erasing the measurements after processing, as explained in Section 10.2.2 and Section 10.1.1.

8.5 Conclusion

It can be concluded that the framework works reasonable well for different user categories. For the activity level predictions, the errors were still non-negligible after a few weeks. However, a decreasing trend is observed and it is expected that the prediction accuracy would further increase in the future. Unfortunately, the issue of false positives turned out to be greater than expected. Therefore, measures should be taken in order to filter out or decrease these false positives.

The location prediction accuracy also turned out to be reasonably good. Next to some unexpected observations, the results were as expected (e.g. working people are more predictable than students). Given the increasing accuracy over the weeks, it is also expected that the location prediction correctness would increase if the experiment would have lasted longer.

Given the fact that the errors on both the activity and location predictions decrease over the weeks, it can be concluded that the framework indeed learns the user’s habits.
Chapter 9

Demo Application

To demonstrate the use of the Context Facade (see Chapter 7), a small demo application has been developed. The concept of a personal assistant application looked like a good way to demonstrate as much functionality as possible. A typical personal assistant application should notify the user about things that are likely to happen, assist in anticipating on these events and give some fun facts about the past. The small demonstration application contains at least one function for each of these things. Which functionality the application has is explained in the next section.

9.1 The Personal Assistant application

The Personal Assistant consists of five functions that use mostly all functionality the Context Façade supports and uses all types of information collected and processed by the framework. All functions are explained in detail below.

9.1.1 Relevant information about the next trip

A first function the Personal Assistant supports is giving relevant information for the expected next trip. Whenever the user is expected to be at a different location in two hours (while he is still expected to be at the same location in one hour), the Personal Assistant creates a new notification.

The notification contains a message that tells which location the user is expected to go to (e.g. home, the school building, ...). It also provides the user with relevant information about the trip to the new location. The personal assistant looks back four weeks in time to check which activities were detected at that time in the past. If the activities indicate that the user was more running, walking or cycling than he was in a vehicle, a weather forecast is shown. In the other case traffic information should be shown. However, retrieving traffic information is not yet supported by the framework. Therefore the notification only shows a message saying that the user is expected to use a vehicle for his trip, without
traffic information. In Figure 9.1a and Figure 9.1b, two example notifications are shown.

(a) Screenshot of Personal Assistant that should give traffic information for the next trip.

(b) Screenshots of Personal Assistant giving relevant information for the next trip.

Figure 9.1: Screenshots of Personal Assistant giving relevant information for the next trip.

9.1.2 Weather forecast for the time of waking up

A weather forecast is not only given when the user is expected to go to another location. Based on the activity level prediction, a notification that shows a weather forecast is shown 12 hours before the user is expected to wake up. A screenshot of such a notification is shown in Figure 9.2.
Another notification that is based on the prediction of the activity level assists the user in setting his alarm. If the user is expected to be awake in 12 hours, a notification pops up to ask if the user wants to set his alarm in order to wake up in time. If he chooses to do so, the alarm is set 30 minutes earlier. Figure 9.3 shows such a notification.
9.1.4 **Telling the user about the last time at the current location**

Whenever the user arrives at a new location, the Personal Assistant welcomes the user at this location (thereby mentioning the name of the location if it is available) by posting a new notification. The application also uses the location history: it tells the user how long it has been since he was at that location for the last time. Additionally, the user can choose to send an automated text message to his ICE (In Case of Emergency) contact to let the contact know he arrived safely at this location. An example of this notification is shown in Figure 9.4.
Figure 9.4: Screenshot of Personal Assistant welcoming the user at the new location and suggesting to send a text message.

9.1.5 Detecting when the user is in a vehicle

The last notification is shown when the framework detects that the user is in a vehicle. In that case, the Personal Assistant asks the user to not use his phone while driving (as shown in Figure 9.5). A later version of the Personal Assistant might automatically block all phone functions except navigation and hands-free calling. Some other applications already do this. An example of such an application is the ’Drive Safe App’ of the Dutch insurance company a.s.r. [21].
Figure 9.5: Screenshot of Personal Assistant when it is detected that the user is in a vehicle, giving the advice to drive safely.
Chapter 10

Discussion and Conclusion

This last chapter wraps up this master’s dissertation about the context collecting framework. First, weaknesses and possible improvements for the currently existing modules are identified. Next, possible future extensions are discussed. Finally, the conclusion summarises this work.

10.1 Weaknesses and improvements

10.1.1 Context Collection Module

Most of the test subjects indicated that the framework’s battery usage was not dramatically large. However, they all said they did experience a larger battery drain than usual. Therefore, some measure could be taken to reduce the battery consumption. An effective measure could be to have an adaptive sampling rate instead of a fixed sampling rate (now every 7.5 minutes). During periods that the user looks active (or is expected to be active), the sampling rate could be increased to capture changes rapidly. On the contrary, when the user is inactive (or at the same location for a long time) the sampling rate can be reduced. Care has to be taken about the fact that the sampling rate should not be too low when the user turns active again. For this reason, the sampling rate should depend not only on what the user is currently doing, but also on what he is expected to do.

Another point that can be improved is the memory usage of the stored measurements. Currently too many information is stored, e.g. the complete 10 seconds of accelerometer data while only a few samples are effectively used. There has to be looked after which data is really necessary and useful to capture. Another interesting thing can be to remove part of the measurements after they have been processed. Fields like the current location, detected activities and activity level have to be stored to have a useful history (which is used for some Context Façade functions). However, accelerometer, rotation and other sensor data are not likely to be used once they have been processed. Therefore, it might
be interesting to nullify some fields after processing, which will also drastically reduce memory usage.

10.1.2 Activity Submodule

It has to be investigated whether or not the usage of multiple models, like it is implemented in the Location Module, can be useful for the Activity Module. People tend to have (slightly) different sleeping patterns during vacation periods, which the current solution does not capture. Additionally, mixing up both patterns degrades the performance of the predictions.

The measure to express how well the time of waking up and going to sleep is predicted should be different than for the location module, because the differences between the models will be a lot more subtle. The difference in time of waking up and going to sleep will probably be at most a few hours, while the differences in location patterns can vary during the entire day. It might be a good idea to use the functions to detect the time of waking up and going to sleep that is used in the Context Façade. The functions are explained in Section 7.2.1. In Section 4.3.1.2 it is shown that the functions are reasonably accurate at detecting these moments. A possible error measure could then be the average deviation of the predicted time compared to the detected time. If the average deviation is above a certain value, a new activity model could be created.

Another issue with the Activity Submodule is the amount of false positives, especially at night. In the user test (see Section 8.2 of Chapter 8) it appeared that some users generated useless results because of the amount of false active detections. Therefore, measures should be taken in order filter out as much of these false positives as possible. It might be a good idea to set variable thresholds for sensors like the light and sound sensor. If these thresholds depend on the average values for that user, it could be possible to filter out some false positives (when e.g. there is a constant noise on the acceleration sensor). Another measure that will reduce the number of false positives could be to require multiple detection rules to be triggered before considering a sample as active. Finally, yet another fix could be to reduce the impact of every sample by reducing the weight of new measurements in Equation 4.12.

10.1.3 Location Submodule

The user test of Chapter 8 taught some interesting things about the Location Submodule. In Section 8.2 one of the conclusions was that for people that always sleep at the same location throughout the week the mechanism of parallel models did not work out as expected. It was expected that a second model would be created during a holiday, but it appeared that sleeping at the same location was enough to stay below the logarithmic loss threshold of 3.
To improve this, the logarithmic loss threshold could be made variable, depending on the location pattern of the user. However, this might turn out to be rather complicated, since the threshold will differ for every user. It might be easier to ignore some locations when calculating the logarithmic loss. For example, all predictions for the home locations (as mentioned in Section 6.2, a user can have multiple home locations) could be discarded when calculating the loss. This would mean that only the (non-home) locations throughout the day have an impact on the accuracy of the model. In this case, if a user visits other locations during a holiday, the regular model will not have predicted this and the loss will increase. Sleeping at the same locations as usual does not decrease the loss and a new model is expected to be created at the end of the week.

10.1.4 Context Façade

The current Context Façade still contains a limited set of functions (see Chapter 7) that enable querying the information produced by the framework. However, it is still possible to implement more complex functions that combine the information of different modules. An example of such a function could be a function to see if a user already visited a restaurant in the past day, combining location history and the location labeling module. Obviously, more functions can also be instantiated when new modules are added to the framework (see Section 10.2.3).

10.2 Future extensions

10.2.1 Remote computations

A significant amount of the battery usage of the framework can be attributed to the computations performed during processing, like predicting the future locations and activity level. It would be beneficial for the battery usage and processing speed (e.g. location models that contain many locations need a long time for performing the calculations because of the large matrices) if the computations could be done on a remote server. The framework would just have to send its measurements to the server, after which it receives the predictions without having to process them. Note that this would require an always on internet connection (or some kind of measurement caching mechanism when the smartphone is not always connected to the internet). However, given the fact that most smartphones are at least sometimes connected to the internet, this will probably not be a major issue.

10.2.2 Remote storage

As perceived by the test users (see Section 8.4), the proposed framework uses quite some memory space. Not only did the measurements take up a few hundreds of megabytes
in a couple of weeks time, the models (and model back-ups) and locations also need about 50 megabytes of storage (depending on how long the framework has been running and how many locations are present in the models). Therefore, it might be a good idea to move the storage of the measurements and models to a server. If the computations already take place on a server, this might even be the most logical thing to do. To avoid unwanted costs, the measurement transactions should only take place when having a WiFi connection. Therefore, some sort of measurement caching system should be implemented to avoid having to transfer the measurements over the mobile internet connection of the user and being able to process measurements that have been taken when there was no internet connection.

10.2.3 New modules

As stated in Section 1.1, context is defined by Abowd et al. in [11] as follows:

*Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

From this definition, it is clear that context can be so much more than just the current activities, the activity level and the locations of the user. Next to extending and improving the currently implemented modules, it is easy to implement new modules as well. By doing that, more context of the user is captured which makes it possible to make more accurate recommendations. Numerous extension can be thought of, like e.g. modules that read the phone’s calendar and reminders, the user’s surfing habits, Swarm check-ins or social media activity.

10.3 Conclusion

In this dissertation a framework is presented to capture the context of the user of the framework. The presented framework mainly consists of two modules. The first module, the Data Collection Module (Section 3.2), collects sensor and other data via the user’s smartphone. Next, the Data Processing module processes the captured data. This module consists of several submodules. First, the Activity Level Submodule (Chapter 4) detects and predicts the activity level of the user (i.e. whether or not the user is active). The Location Submodule (Chapter 5) detects and predicts the locations of the user. These locations are labeled using the Location Labeling Submodule (Chapter 6). This module also detects the home location(s) of the user. Subsequently, all this information is made accessible through and API: the Context Façade (Chapter 7).

The results show that the framework is capable of detecting, learning and predicting the user’s activity and location patterns using a Markov based mechanism. This is illustrated...
in the chapters of the modules itself and the chapter about the user test (Chapter 8). It is observed that the predictions get better with time. However, the results at the end of the user test that lasted about five weeks were already rather satisfactory. Unfortunately also some issues and weaknesses turned up. The weaknesses of the framework are summed up at the beginning of this chapter, together with some possible solutions.

In short, it can be said that this dissertation proves that it is possible to capture the context of a user using a Markov based approach. The framework currently has limited functionality (mainly activity and location detection and prediction) but it should be fairly easy to add new modules.
Bibliography


