Markus Perndorfer

Detecting social interactions via mobile sensing

Master's Thesis

Graz University of Technology

Knowledge Technologies Institute
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Supervisor: Dipl.-Ing. Dr.techn. Viktoria Pammer-Schindler

Graz, April 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

With this thesis we try to determine the feasibility of detecting face-to-face social interactions based on standard smartphone sensors like Bluetooth, Global Positioning System (GPS) data, microphone or magnetic field sensor.

We try to detect the number of social interactions by leveraging Mobile Sensing on modern smartphones. Mobile Sensing is the use of smartphones as ubiquitous sensing devices to collect data. Our focus lies on the standard smartphone sensors provided by the Android Software Development Kit (SDK) as opposed to previous work which mostly leverages only audio signal processing or Bluetooth data.

To mine data and collect ground truth data, we write an Android[2] app that collects sensor data using the Funf Open Sensing Framework[1] and additionally allows the user to label their social interaction as they take place.

With the app we perform two user studies over the course of three days with three participants each. We collect the data and add additional meta-data for every user during an interview. This meta-data consists of semantic labels for location data and the distinction of social interactions into private and business social interactions. We collected a total of 16M data points for the first group and 35M data points for the second group.

Using the collected data and the ground truth labels collected by our participants, we then explore how time of day, audio data, calendar appointments, magnetic field values, Bluetooth data and location data interacts with the number of social interactions of a person. We perform this exploration by creating various visualization for the data points and use time correlation to determine if they influence the social interaction behavior.

[2] http://en.wikipedia.org/wiki/Android_(operating_system)

We find that only calendar appointments provide some correlation with the social interactions and could be used in a detection algorithm to boost the accuracy of the result. The other data points show no correlation during our exploratory evaluation of the collected data. We also find that visualizing the interactions in the form of a heatmap on a map is a visualization that most participants find very interesting. Our participants also made clear that labeling all social interactions over the course of a day is a very tedious task.

We recommend that further research has to include audio signal processing and a carefully designed study setup. This design has to include what data needs to be sampled at what frequency and accuracy and must provide further assistance to the user for labeling the data.

We release the data mining app and the code used to analyze the data as open source under the MIT License[3]

[3] http://opensource.org/licenses/MIT

Zusammenfassung

Mit dieser Arbeit wird untersucht, ob es möglich ist mithilfe von Standard Smartphone Sensoren wie Global Positioning System (GPS), Bluetooth, Mikrophone oder Magnetfeld-Sensor, soziale Interaktionen, die von Angesicht zu Angesicht stattfinden, zu erkennen.

Wir versuchen die Anzahl der sozialen Interaktionen zu einem gegebenen Zeitpunkt mithilfe von Mobile Sensing zu erkennen. Mit Mobile Sensing wird die Nutzung von Smartphones als Ubiqutious Sensing Device bezeichnet. Smartphones dienen hier als Data Mining Geräte. Wir konzentrieren uns hierbei im Gegensatz zu anderen Arbeiten auf die Standard Smartphone Sensoren. Andere Arbeiten konzentrieren sich meist auf Bluetooth oder Audio-Signalverarbeitung.

Um Daten und die Ground Truth zu sammeln, wurde eine Android[4] App erstellt, die es, neben Data Mining, unseren Probanden erlaubt ihre sozialen Interaktionen zu dem Zeitpunkt zu markieren an dem sie stattfinden.

Mit der App führten wir zwei Nutzerstudien durch. Beide Nutzerstudien umfassten jeweils drei Probanden und dauerten jeweils drei Tage. Zusätzlich wurden von allen Probanden nach der Studie noch weiter Meta-Daten erhoben. Diese Meta-Daten waren semantische Labels für die Positionsdaten und eine Unterscheidung der sozialen Interaktionen in Privat und Geschäftlich. Für die erste Gruppe wurden 16 Mio. Datenpunkte erhoben, für die zweite Gruppe 35 Mio.

Mit den erhobenen Daten wurde der Zusammenhang zwischen Tageszeit, Audio Signalen, Kalendereinträgen, Magnetfeld-Stärke, Bluetooth Daten, Positionsdaten und den sozialen Interaktionen untersucht. Für die Untersuchung wurde ein explorativer Ansatz gewählt, bei dem für jedes Datenset

[4] https://de.wikipedia.org/wiki/Android_(Betriebssystem)

und den sozialen Interaktionen geeignete Visualisierungen erstellt wurden, die die zeitliche Korrelation zwischen den Sensordaten und den Interaktionen zeigt.

Unsere Resultate zeigen, dass nur Kalendereinträge teilweise mit den sozialen Interaktionen korrelieren. Dieser Zusammenhang könnte helfen die Genauigkeit eines Erkennungsalgorithmus zu verbessern. Alle anderen Ansätze zeigten keine Korrelation während unserer Analyse. Es zeigte sich auch, dass die Visualisierung des Zusammenhangs zwischen Ort und Anzahl der sozialen Interaktionen in der Form einer Heatmap von den Probanden sehr interessant gefunden wurde. Unsere Probanden fanden außerdem das Labeln der sozialen Interaktionen sehr anstrengend und mühsam.

Wir empfehlen, dass sich weitere Arbeiten auf diesem Gebiet auf Audiosignal-Verarbeitung konzentrieren sollten und dass eine Nutzerstudie genau geplant werden sollte, im Bezug auf welche Daten in welcher Frequenz und wie lange gesammelt werden. Die Probanden sollten auch beim Markieren der sozialen Interaktionen durch die App unterstützt werden.

Die Data Mining App, sowie der Code der zur Analyse verwendet wurde, ist unter der MIT Lizenz[5] veröffentlicht.

[5] http://opensource.org/licenses/MIT

Contents

List of Figures

List of Tables

1. Introduction

This thesis explores the feasibility of detecting face-to-face social interactions based on standard smartphone sensors.

This is an interesting question because a reliable detection would provide some new and interesting possibilities to enhance interactions. We envision a system that detects the social interaction as it takes place and also identifies the interaction partner. Using this information, it could provide additional information concerning the interaction partner, for example recent electronic exchanges like e-mail or text messages.

Another use for the social interaction count would be in the health care sector, especially for stress related diseases. Social support and social groups are linked to reduced impact of stress [11, 12]. We think that social interactions correlate with social behavior and therefore the number of social interactions during a day could be a good indicator on how well protected a person is from stress.

To discover the conversations a person has over the day, we need sensor data. We analyze this data and try to find hypothesis that allow us to determine the number of interactions, based on sensor readings by the device. To get this data, we need a device that fulfills following requirements:

- It is always with the person
- It does not interfere with the persons daily business
- It provides a multitude of different sensors

Eagle and Pentland [19] first stated that a smartphone fulfills all the requirements mentioned above. People take them everywhere they go, they provide a multitude of sensors (see Chapter 2). This makes them ideal for this kind of study.

2. Related work

In this chapter we will describe how this work fits into the existing work in the field of Mobile Sensing.

First we will give a broad overview over Mobile Sensing, starting with a general definition of the term. Then we will show different classifications for Mobile Sensing and describe sample work for every category.

After the overview and classification, we go into further detail about the area of social interaction detection. We describe notable prior work that influences this thesis.

Mobile Sensing, in general terms, is the use of smartphones as ubiquitous sensing devices to collect data. This collected data is then used to provide insight about the behavior of the user, a defined group, or to gain further information about the environment the user inhabits.

Mobile Sensing uses smartphones because they[8]

- provide a multitude of built-in sensors (see Table 2.1)
- are widely available (estimates are that the worldwide total of smartphone users will be 1.75 billion by the end of 2014 [1]
- are a standardized platform in terms of Operating System (OS), hardware capabilities and software
- offer a low-cost and low-energy platform for sensoring
- are widely compatible with existing wireless infrastructure (for example Global System for Mobile Communications (GSM) or WiFi) out of the box
- are easy to use, portable and will continue to improve in terms of miniaturization, processing power and availability

[1] http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536, Last visited: Jan. 16, 2014

The above points show that smartphones are an interesting and viable choice to implement sensing applications

Smartphones are also used because they are truly personal and ubiquitous devices: one person - one device, and people carry them nearly everywhere

Sensor	Function	Technology	Sample application
Accelerometer	Rotating Screen	Acceleration	Vibration Sensors
Ambient Light	Light Sensor	Detection of light intensity	Screen brightness adjustment & energy saving
Camera	Taking picture & video	Recording of scene	Image analysis & colorimetry
Digital compass	Compass	Meassuring magnetic fields by hall effect	Magnetic field detection
Gyroscope	Postures of mobile device	Measuring angular rate by Coriolis effect	Detection of position & stance
Global Positioning System (GPS)	GPS receiver	Radio Frequency	Position & Clock
Microphone	Conducting voice	Sound	Spectrum analyzer for sound & noise measurement
Near field communication (NFC)	Radio-frequency identification (RFID)	NFC	Payment, ticketing & security
Proximity	Photoelectric sensor	Detecting distance	turn off screen

Table 2.1.: Typical sensors in smartphones and their usage.
Table recreated from [8]

2.1. Classification of Mobile Sensing

Mobile Sensing is used as very broad term and implies many fields of application and usage scenarios. In this section we describe two different classification systems for Mobile Sensing and also classify this thesis in the context of the two system.

2.1.1. Sensing Scale and Sensing Paradigm

Lane et al. [25] divide Mobile Sensing along two dimensions, "Sensing Scale" and "Sensing Paradigm". Every Mobile Sensing application falls into one category of these two dimensions.

Sensing scale

The sensing scale defines the scope of the sensing application, with regard to the size of the affected user group. Lane et al. [25] define three scales:

Personal Personal sensing applications provide information for a single user. They mostly collect data about one user and refine the data to be utilized by one person only

Group Group sensing application collect data from individuals with similar interests and provide feedback and information for the whole group. An example would be an exercise tracking app where users can compete which each other. For groups privacy can be a concern with regard to sharing information with others

Community Community sensing applications use large-scale data collection, data sharing and data analysis to provide helpful information to a community. These applications need many data points and therefore many participants that provide data to become helpful. An example would be the generation of a "noise map", a visual representation of the average loudness (usually

as a heat map[2]) of a given geographical area, for example a city district or even a whole city

Sensing paradigm

The sensing paradigm specifies the user involvement. Or in other words, how much or how often the user has to interact with the sensing application.

Lane et al. [25] define two sensing paradigms:

Participatory The user actively takes part in the sensing and inputs additional data to the information gathered by the sensors. The user may also be involved in the sensing. For example, the sensing application may inquire to take pictures of certain places or events. [23]

Opportunistic Sensing works completely autonomous. No further interaction with the user is required.

Our application has a personal sensing scale. We try to sense the social interactions of an individual user, and while we may employ data from other participants in the analysis, the main benefit is for the individual user.

As a sensing paradigm, we use participatory sensing. The user has to label the social interactions as they take place, so we are able to extract useful correlations between the collected data and the social interactions.

2.1.2. Areas of Mobile Sensing

Khan et al. [23] analyzed various published papers and applications regarding Mobile Sensing. They divide the fields of application in seven groups, see Fig. 2.1

In this section, we explain the different fields in detail and describe some interesting examples per field.

[2]For some examples of heat maps see http://en.wikipedia.org/wiki/Heat_map

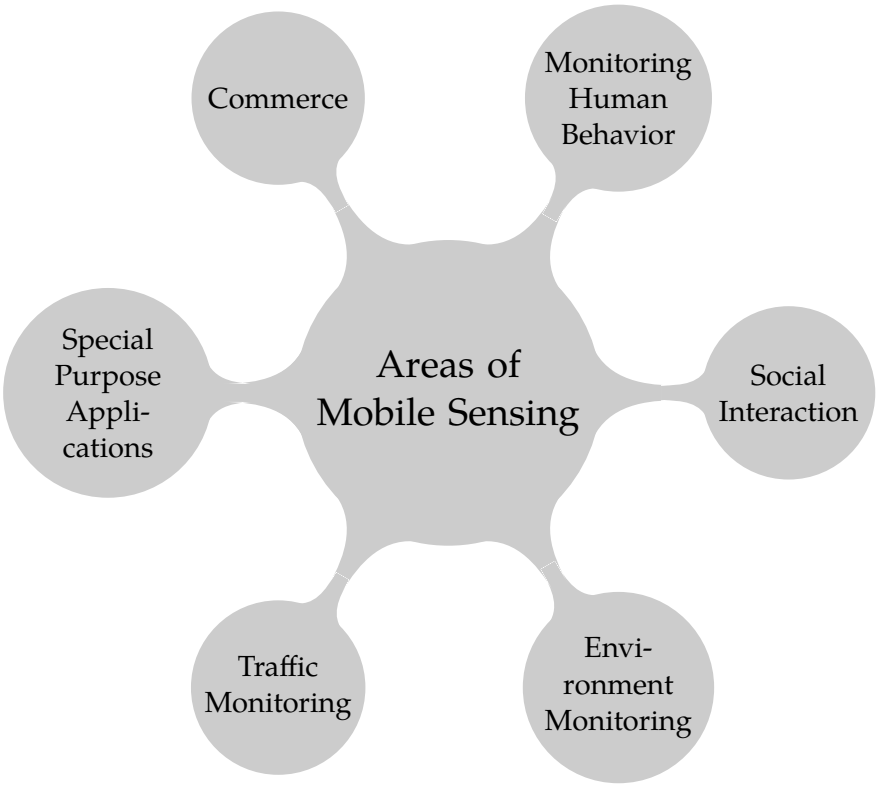

Figure 2.1.: Fields of application of Mobile Sensing. Figure recreated from [23]

Health Monitoring

There are several ways to use Mobile Sensing for health monitoring. In this portion of the thesis we will provide a brief overview of the different interesting approaches discussed in research.

Lu et al. [29] propose StressSense, a system that monitors the stress level in the voice of the user. The use of smartphones provides the opportunity to continuously and unobtrusively monitor the stress level during various events and conditions over the course of the day.

Another interesting application of Mobile Sensing for health monitoring is "SPA: A Smart Phone Assisted Chronic Illness Self-management System with Participatory Sensing". Sha et al. [40] propose a system where the users biometric data is constantly measured via various sensors (for example a blood oxymeter, or the build-in accelerometer to determine the user's activity level) which in turn use a smartphone to send the data to a central processing and evaluation system. The mined data is used to find out what specific behaviors trigger symptoms and therefore aid in finding a custom fit treatment plan for the user. Furthermore, based on the sensed data, the system could also trigger alerts or notifications without additional human interference and thus reduces the necessary involvement of health care personnel.

Oliver and Flores-Mangas [37] propose a system that uses the processing capabilities of the smartphone to analyze sensor readings and present the results understandably to the user. More specifically, they use a blood oxymeter to detect sleep apnea events.

Chen et al. [9] use the smartphone to analyze Electro-cardio gramm (ECG) data to find abnormalities and alert the user or a health care professional about it, if needed. Additionally, if an abnormality is detected, the ECG data is sent to a server, where a doctor can review and monitor the data. Jin et al. [22] also monitor ECG, but focus on the analysis of the data directly on the smartphone.

There are also various systems to measure the user's activity level or energy expenditure versus the caloric intake. [20, 14, 15]

These are only some examples for the use of Mobile Sensing for health monitoring. There are many possibilities to use the smartphone as sensing platform to monitor the user's health and well being.

Traffic Monitoring

Another field of application of Mobile Sensing is traffic monitoring. Here the focus shifts away from the individual user towards the community.

Mohan, Padmanabhan, and Ramjee [35] use the various sensors on a smartphone to detect honking, potholes, bumps and sudden slowdowns. The aim for this work is to detect various traffic conditions for roads in developing regions, where traffic tends to be more chaotic and stressful than in the developed world.

A different use of Mobile Sensing for traffic monitoring is proposed by Thiagarajan et al. [41]. They developed a system that uses the data mined by smartphones to detect traffic delays and to estimate the travel time in real time. The basic idea is that the users wants to avoid congested streets (for example a main street during rush hour). The system would reroute the user around detected traffic hotspots to avoid delays.

Environment Monitoring

Another use of Mobile Sensing is environment monitoring. Environment monitoring systems use the smartphone (and sometimes additional sensors) to gather data about the environment of the users.

Maisonneuve et al. [30] propose a system that uses the microphone built into smartphones to measure the noise pollution in the user's environment. The system allows citizens to measure there individual exposure to noise. Additionally, every citizen can contribute his data to generate a noise map for the whole community. Bilandzic et al. [4] describe a different approach for the same idea.

Rana et al. [39] also describe a system to generate a noise map for an urban environment. Additionally, they try to solve the problem of sparse and incomplete spatio-temporal data for the noise map. They propose using Compressive Sensing (see [17] for the mathematical theory behind Compressed Sensing) to extrapolate missing data. This enables the system to calculate data for specific times of day or spatial coordinates, even if no sample was yet recorded that matches these parameters.

While the first two examples try to determine the noise level of an urban environment, Mun et al. [36] describe a system that measures the environmental impact of the user. Their system measures the carbon footprint of the user when traveling, how much smog the user experiences and, interestingly, how many fast food restaurants the user passes on a journey.

Social Interaction

Mobile Sensing applications for social interactions focus on the user and his social peers and how he or she interacts with her social environment

The work of Miluzzo et al. [32] focuses on providing new ways to interact with people. Their system senses different aspects about the user and his current environment (user is in conversation, walking, at the gym …) and publishes theses aspects to various social networking sites or instant messengers, if the user wishes to do so.

Another interesting approach to enhance social interactions is proposed by Beach et al. [3]. Their systems enables smartphones to exchange so called "social networks ids" with other smartphones int their vicinity via Bluetooth. With these "social network IDs" further information about the user of the smartphone may be obtained, for example from his facebook profile. As a protoype for their framework they describe a context-aware music player, that uses the system to obtain the music tastes of all users in its proximity to adjust the play list according to the users tastes.

The work of Bao and Roy Choudhury [2] as a different perspective of the purpose of Mobile Sensing in the context of social interactions. They use the data mined by different smartphones to detect and capture interesting moments in a social setting. The interesting moments are then combined into a

highlight reel of the event. They also describe several triggers to detect interesting events, for example when the most users change their facing to the same direction (they argue this can happen when a group turns towards a public speech or a stage). Additionally, they describe different ways to determine groups, based on audio signal processing and image processing of regular snapshots provided by the smartphone camera.

Monitoring Human Behavior

Mobile Sensing applications that monitor human behavior are very focused on the individual user, and try to determine the physical and even emotional behavior or state of the user.

For example, Rachuri et al. [38] use audio signal processing on Nokia phones to classify the users emotional state. They use the emotion classification and various other classifiers to conduct psychological experiments.

While the previous work focuses on the emotional component, Kwapisz, Weiss, and Moore [24] focus on the physical activity of the user and try to determine the activity the user is currently performing. This includes walking, jogging, ascending stairs, sitting or standing.

The work of Dong, Lepri, and Pentland [16] focuses on the relations between members of a social group. They use the smartphone to mine data about the co-location of group members and how this influences the development of relationships and behavior in this group.

Commerce

While the previous areas of application focus on very different aspects of users or user groups, the applications do not concern themselves with monetization or applying Mobile Sensing to business processes.

Deng and Cox [13] and Bulusu et al. [7] both use the camera built in smartphones and Optical character recognition (OCR) to detect prices for groceries. The difference is that Deng and Cox [13] use the receipt of a store transaction extract grocery prices, while Bulusu et al. [7] use the price tags

of the products. Deng and Cox [13] additionally describe a system to detect fuel prices by photographing the fuel price boards of fuel stations.

Special Purpose Applications

Lastly, there are some applications that can not be classified in one of the previous categories. These are called "Special Purpose Applications" and cover a wide range of topics.

For example, Zhang et al. [43] describe a system that accurately (2 cm median error) measures the distance between two phones in real time. Their measurement system cross-correlates audio signal measurements and timestamps exchanged by the smartphones over WiFi to calculate the distance between two phones while being robust against noise and Doppler effect issues. They use the system to implement two prototype games, SwordFight and ChaseCat, that demonstrate the use of distance measurement in real time on smartphones.

Another, completely different usage of Mobile Sensing is the work of Liu and Liao [27]. The propose PaperUI, where the smartphone is used to link the printed and digital versions of a document together. With the help of Augmented Reality (AR) the user can annotate the digital version of the document by interacting with the paper print out on the smartphone.

There are also several works that try to infer the semantics of a place a user visits. [26, 10]

Lastly, the work of Lu et al. [28] uses the smartphone to classify ambient noise into sound events. They use the term "sound events" for distinct sounds that the user encounters in his everyday life, for example car horns, vacuum cleaners and so on. They describe an unsupervised learning algorithm that detects such sound events and allows the user to label them. Additionally, they classify the ambient noise in three coarse categories: music, human voice and general ambient noise and propose to further classify within these categories. But they only describe a male/female classifier for human voice in their prototype.

We classify this thesis into the "social interaction" category. One could also argue that this work falls into the "special purpose application" category, because we only try to detect face to face social interactions, without trying to further enhance the users social interactions.

2.2. Social Interaction Detection

As described in the previous section, there are lots of different applications for mobile sensing. In this section we will describe various other works that are closely related to detecting social interactions as we define them (see Chapter 1).

First we want to discuss the work of Miluzzo et al. [32]. Their paper "CenceMe – Injecting Sensing Presence into Social Networking Applications" is one of the first that tries to detect if a person is actually talking to another person. While the paper primarily describes a system that enhances a person's social interactions by sharing sensed data on various social platforms, they also describe a "Conversation detection" classifier. The classifier is a binary classifier and detects if the user is in a conversation or not.

More specifically, it samples the ongoing audio stream provided by the smartphones built in microphone and performs a discrete Fourier transform (DFT) on the samples [34]. The DFT works on a range of 250 Hz to 600 Hz, because they determined that most of the signal power of a human voice is located at this frequency spectrum. From the DFT result the mean and the standard deviation are extracted as feature vectors for the classifier. The classifier itself applies a threshold to mean and standard deviation to determine if "talking" is present in the sample or not.

The result of the audio classifier is further evaluated at the backend server of CenceMe. Because a conversation also includes pauses or moments of silence, the backend uses a rolling window of five audio samples to determine if the person is in a conversation or not. If two or more samples in the current window indicate a conversation, the classifier returns "in conversation". Otherwise (four samples have state "silence"), it returns "not in conversation".

Another relevant paper is the work of Miluzzo et al. [33]. They describe a system to perform speaker recognition on smartphones.

The paper details a novel approach for machine learning on smartphones. In their system smartphones:

- automatically update their learned model with usable new samples for a given model (*Classifier evolution*)
- share their learned models with other smartphones in the system to increase the sensing capabilities and scalability of the system (*Model pooling*)
- combine their sensing result with results from other adjacent phones to boost the classification performance (*Collaboration inference*)

Their speaker recognition use case details the use of these three techniques to perform speaker recognition.

They model the users voice via Gaussian Mixture Model (GMM) based on the Mel Frequency Cepstral Coefficients (MFCCs) of the audio signal provided by the built in microphone of the smartphone.

Initially, every user only trains his speaker model with a 15 s voice sample of his own voice. With the use of *model pooling* the different speaker models are distributed to all participating phones in the system. *Classifier evolution* increases the performance of the classifier for any given user over time, as more data for the model is collected in different environments. The ever increasing diversity of environments and the additional data gradually enhances the model of a speaker. They also describe how *collaboration inference* boosts the system's detection rate in various adverse environments, for example detecting the speaker in a loud restaurant or during a walk on a busy street, because different participating phones may be in a better position (on the table / out of the trouser pocket, closer to the speaker or farther way from noise and other variants) to correctly infer the speaker, and therefore the combined result provides a better classification.

Rachuri et al. [38] also describe a system that performs speaker recognition on smarpthones. They also use GMMs as model for the speaker recognition, but use Perceptual Linear Predictive (PLP) coefficients. The main difference between their work and the work of Miluzzo et al. [33] (described previously)

is that they perform all their model training offline and a priori for every user using their system.

Probably the most directly related work for this thesis is the work of Xu et al. [42]. They describe a system to determine the number of speakers currently talking. Their system requires no user interaction, no a priori training and, additionally, needs to be deployed only on smartphones, which means that there are no additional servers required.

To count the number of speakers, they split the incoming audio stream in audio segments of 3 s length. After filtering out audio segments that do not contain human voice using the pitch of the audio signal, they calculate the MFCC vector of the segment using a frame length of 32 ms. The pitch and MFCC vector a compared to the set of existing speakers. If pitch and vector match an existing speaker, the model for the speaker is updated. If not, a new speaker is added to the set of detected speakers. The match between an existing speaker and the given sample is determined via the Cosine Similarity (CS) of the MFCC vector and the frequency range of the pitch. They use the pitch frequency to determine the gender of the speaker. If the CS is below an empirically determined threshold θ_s and the gender matches, the sample belongs to the same speaker. If the sample is above another empirically determined threshold θ_d or the gender does not match, the sample belongs to a new speaker. When none of the previous conditions apply, the system abandons the sample.

They report an average error distance of 1.5 speakers to the real number of speakers in a conversation. They also provide various performance evaluations for different parameters like:

- Environment (indoor, outdoor)
- Noise level (noisy, quiet)
- Number of speakers
- Multiple distinct groups that are in conversation
- Utterance length
- Audio segment length

Most of the work described in this section tries to perform speaker recognition without any regard for the actual duration of the interaction. Furthermore, it also some a priori training required to be able to perform speaker

recognition, except for Crowd++ [42], that is able to count speakers without any a priori training and human interaction. But they also describe that they use a duty cycle of 5 min signal processing and 15 min of sleep to save battery life. So the system will not detect any interaction or conversation for 15 min. They also state that the performance of the system degrades if the individual utterances are short. This is relevant because short utterances are usual for casual conversations, where the participants only speak short sentences and/or interrupt each other frequently. We also found that most social interactions tend to be very short (see Table 6.1).

3. Problem Setting and Research Approach

As we have described in Section 2.2, previous work mainly focuses on Bluetooth proximity to determine social interaction or signal processing of audio signals for speaker detection.

In contrast to the this, we focus on analyzing the standard smartphone sensors as they are captured by FunF. In this work, we considered following facets of the data:

- Audio
- Bluetooth
- Calendar Data / Appointments
- Social Media Apps
- Location Data (Global Positioning System (GPS))
- Magnetic field values

The main goal is to determine the feasibility of solely using standard sensor data for social interaction detection, without further feature extraction.

The data of the sensors alone is not enough for this work. We additionally need ground truth data to evaluate possible hypothesis. To capture the ground truth and sensor data, we developed an Android[1] app based on the FunF Framework. With this app, the user can easily label his face-to-face social interactions at the moment they take place.

Thus, our overall research approach can be explained by the following three steps:

[1] http://en.wikipedia.org/wiki/Android_(operating_system)

First, we developed an Android app that allows us to collect sensor data from our participants while they can label their social interactions as they take place. See Section 4.1 for further information.

Second, we conduct two user studies with three participants each. The participants install the data collection app and label their social interactions during three days. After the data collection phase, we conduct an interview with every participant to gain additional meta-data. This meta-data consist of semantic labels for their location data and labels for their social interactions that divides them into social and business interactions. See Chapter 5 for details on the collection setup.

As third and last step we analyze the collected data and use it to generate visualizations. We show how the time of day, audio data, calendar appointments, magnetic field values, Bluetooth data and location data relate to social interactions. The goal of this step is to visually explore the data for potential features to use in other data mining algorithms. See Section 4.2 for implementation details about the visualization. We present results of our analysis in Chapter 6.

4. Data Collection and Visualization Prototype to Explore Mobile Sensing for Social Interaction Detection

In this chapter we will detail our data collection app and how we performed our exploratory data analysis.

We will start with a description of our app. This includes an introduction to the Funf Open Sensing Framework[1], followed by an explanation of the application concept and implementation. We will also describe the mined data and how the user can interact with the application to label his social interactions.

In the next section we describe our exploratory data analysis setup. We show how we extract and transform the labeled data from the app into various diagrams and statistics using data processing libraries for Python[1]

4.1. Mobile Sensing and Labeling App

To allow the users to efficiently and easily label their social interactions we developed an app for Android smartphones using the Funf Open Sensing Framework[1].

[1] https://www.python.org/

First we will describe the data mining framework that is used by the implementation, after that we will describe the app itself. This includes a description of select implementation details and an overview of the user interaction model.

4.1.1. Funf Open Sensing Framework

The Funf Open Sensing Framework[1] is a framework that enables researchers to easily build Mobile Sensing apps for the Android OS. The framework consists of following building blocks:

`FunfManager` The `FunfManager` is the central class for the whole FunF Framework. It handles the interaction with the Android OS and manages the data collection schedule and the configuration (see below).

`Pipeline` The `Pipeline` encapsulates the configuration. This includes the schedule for data collection, when to back up and upload the collected data and when to fetch a new configuration, if necessary.

`Probe` A `Probe` performs the actual data collection. It can perform an arbitrary task. Every probes has the lifecycle states `ENABLED`, `DISABLED` and `RUNNING`. The lifecycle is managed by the `FunfManager` according to the configuration in the `Pipeline`

The framework additionally includes a persistence layer to store the collected data in SQLite[2] databases. Figure 4.1 shows a high level overview of the framework and how the components interact with each other.

The primary data format for the FunF Framework is the JavaScript Object Notation (JSON) [6]. JSON is a lightweight and easy to use data format. It is primarily used because of its compact and simple syntax. It is used both for the configuration of the framework and as data storage format of the collected data.

[2] `http://www.sqlite.org/`

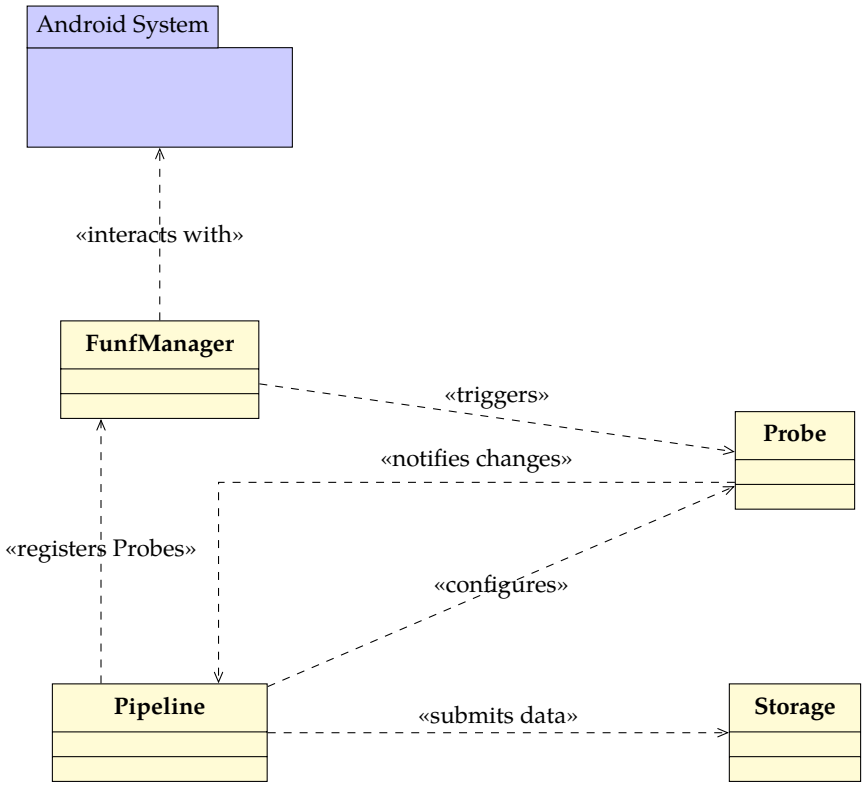

Figure 4.1.: High level architecture overview of the Funf Open Sensing Framework[1]

It is also important to note that all time stamps in FunF are in Unix time. This implies that all timestamps are expressed as seconds, and smaller time units are floating point numbers (as fractions of a full second).

4.1.2. Data Mining App

Our app is devided into two tiers:

User Interface The user interacts with an easy-to-use User Interface (UI) we developed, that allows the user to label the interactions with other people as they happen

Data Collection The app performs the data mining via the FunF framework in the background, and extends the data mining capabilities of FunF to include the label data.

First we will describe the user interface and how it enables the user to easily capture social interactions. We will provide a detailed description of how the user can interact with the app.

After that, we will describe the data mining tier. This includes a description of the data mining architecture and an overview of the collected data.

User Interface

Ease of use was our focus for the UI of the app. The user can label an interaction partner quickly and efficiently, with minimal interruption and distraction of the natural flow of the interaction.

We define a label as a representation of a distinct interaction partner. Or in other words, one label defines exactly one person with whom the user has (potentially multiple) social interactions with. A label can only be in exactly one of the three following states:

new A completely new label. It was not entered into the app before

active A label that is active tracks an ongoing social interaction

inactive When a social interaction is finished, the corresponding label is deactivated and becomes inactive. This state could also be called **unused**

The flow of the app is centered around the concept of *activating* and *deactivating* labels.

We *activate* a label when a new social interaction with the particular person (denoted by the label) starts. We *deactivate* a label as soon as the interaction with the person ends. When we interact with multiple persons during a social interaction, we activate their respective labels at the start of the interaction. We activate and deactivate labels as additional persons join or leave the group, respectively.

The user has several possibilities to activate labels. First, they can use the + symbol in the main screen. This allows the users to either activate a label (via the auto-complete function of the input, see Fig. 4.2) or create and activate a label if it does not exist (see Fig. 4.3). Second, the user can activate labels from the view of unused/deactivated labels. We describe these possibilities in the following paragraphs.

Figure 4.2 shows the user interaction when an existing label is activated via the auto-complete function. The user starts activating a label via the + symbol in the main menu bar. After tapping the symbol, the input box for the label text appears. As soon as the user starts typing, a menu to select unused labels (that contain the entered text), appears. If the user selects one of the entries, the corresponding label is activated.

Adding a new label via the main screen is accomplished in nearly the same way. (Fig. 4.3) After opening the input box via the + symbol in the main menu bar, the user can type in the text for the new label. After a tap on the "Go" action of the on-screen keyboard, the app creates the new label and simultaneously activates it.

The previously described interactions to activate labels assume that the user wants to activate only a few labels at a time or exactly knows what label to activate. The app additionally provides a view for all inactive labels. This allows for browsing and sorting of inactive labels, which in turn enables the user to easily find labels or activate multiple labels at once.

The view of inactive labels is accessed by swiping the main screen from right to left. In this view, the user can see a list of all unused labels. This list can be sorted in three ways:

- by the label text (alphabetically, this is also the default sort option)
- by the date of last use (most recent first)
- by the number of usages of the label (most used first)

Figure 4.4 shows the three different sorting options.

During our development we conducted small test runs for our application. The test runs showed that only activating and deactivating was not enough to allow the users to label their interactions correctly. The app must allow additional use cases, for example:

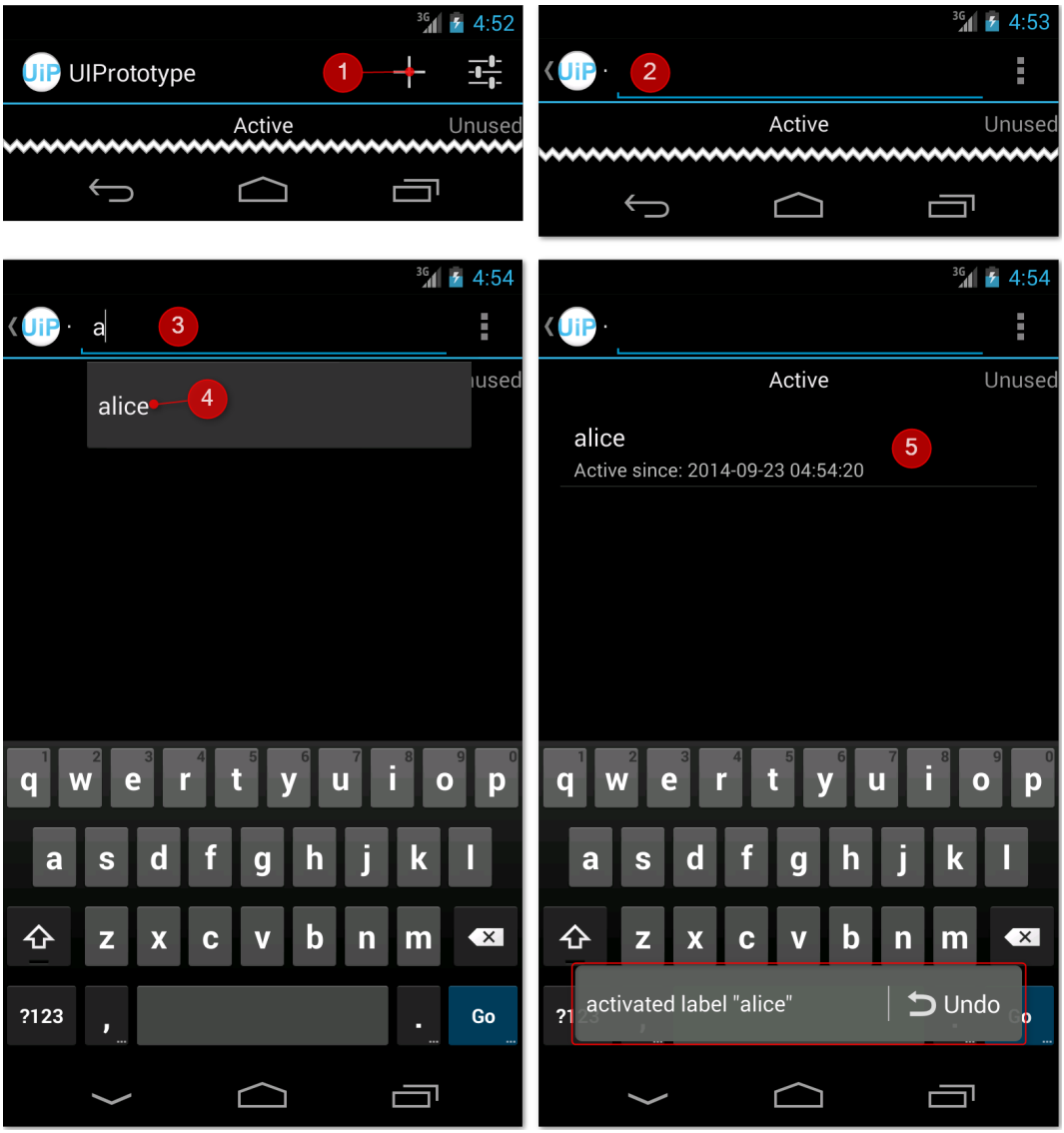

Figure 4.2.: This screenshot series shows how the user can activate an existing label via the + symbol in the main menu bar. After tapping the symbol (1) an input box for the label appears (2). As soon as the user starts typing (3), the autocomplete function suggests labels containing the typed text. If the user taps a suggestion (4), the label is activated (5).
Note the highlighted box at the bottom of the last screenshot. The user can undo every action in the app to avoid data collection errors

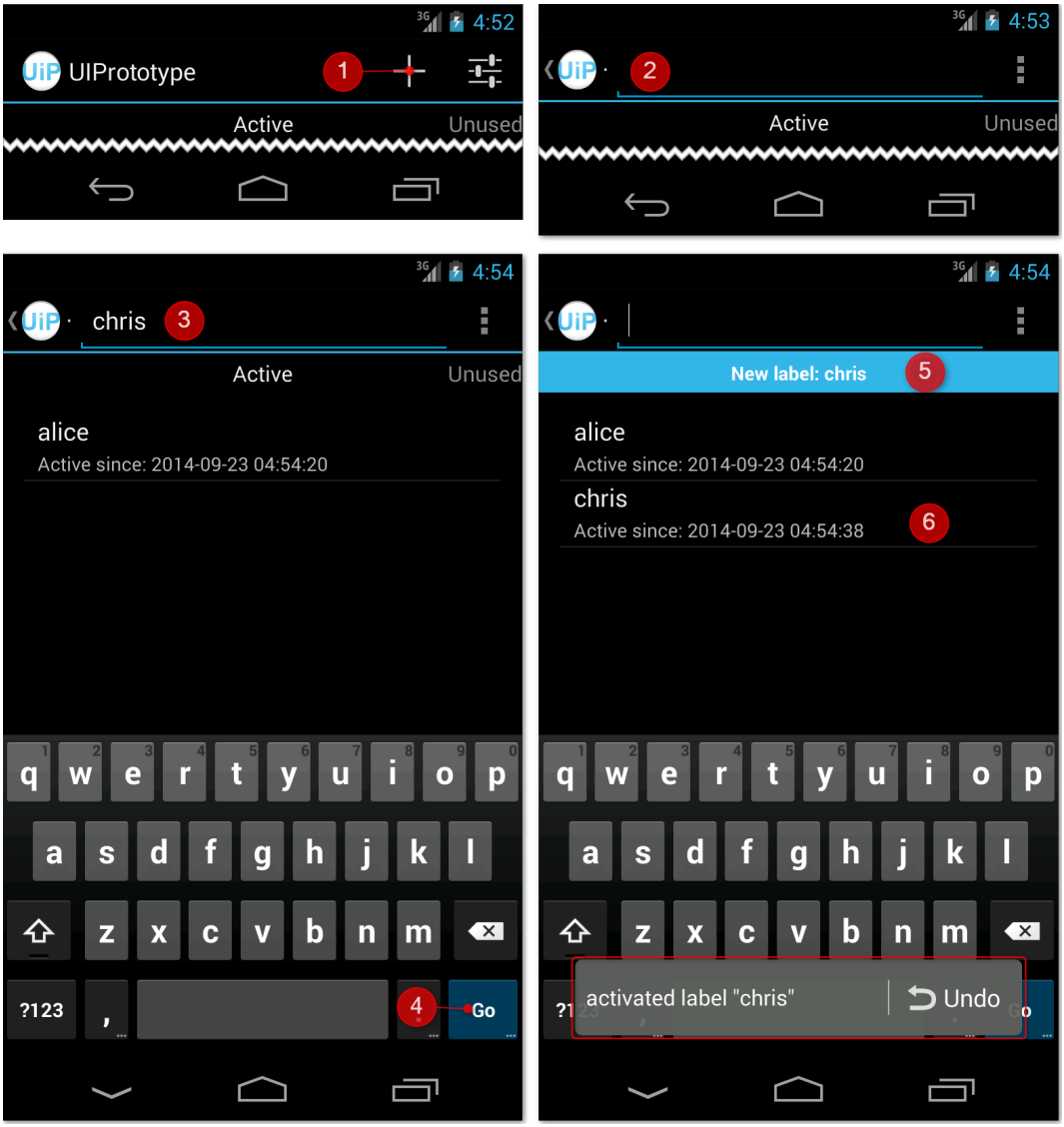

Figure 4.3.: This figure shows the flow for adding a new label to the labeling app After tapping the symbol (1) an input box for the label appears (2). After entering the text for the new label (3), a tap on the "Go" action of the on-screen keyboard creates a new label (5) and simultaneously activates it (6).
Note the highlighted section in the last screenshot. The user can undo every action in the app to avoid data collection errors

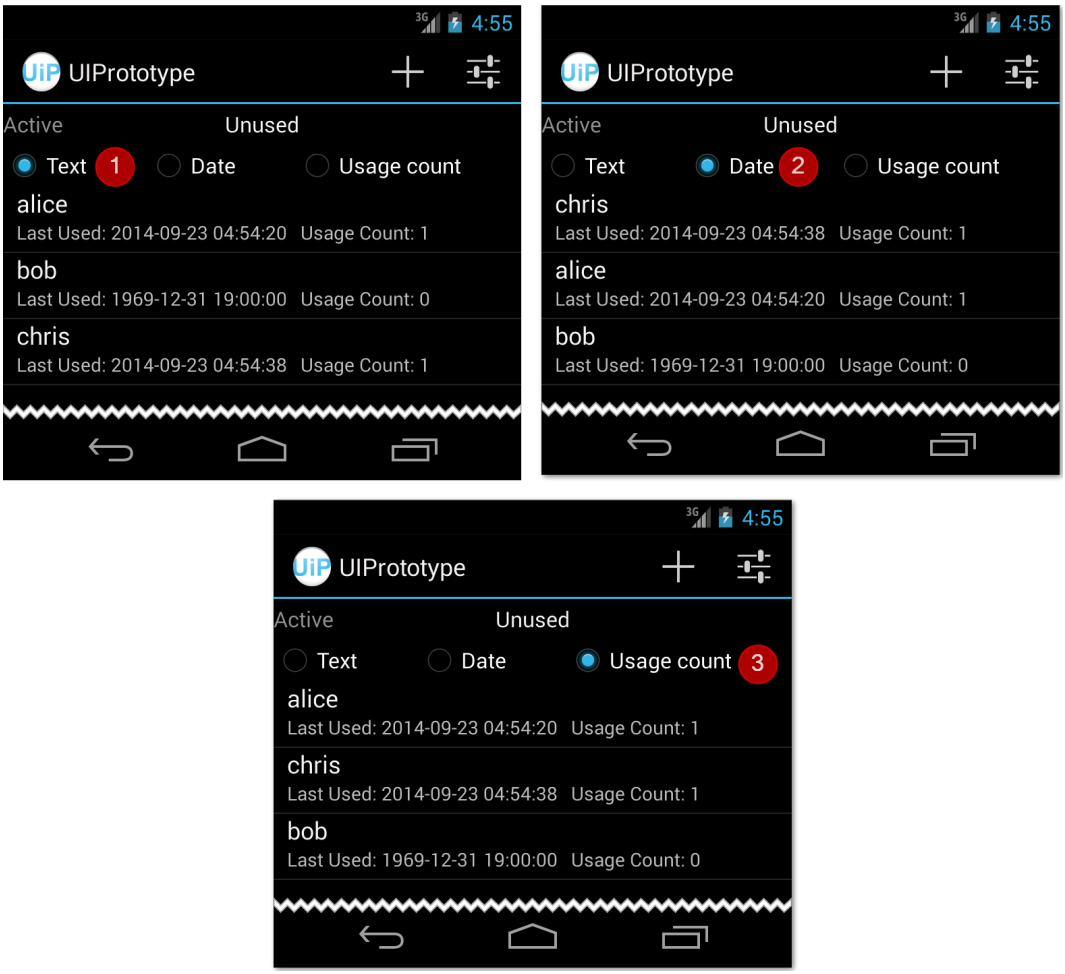

Figure 4.4.: The view of unused/inactive labels can be sorted by text of the label (1) which is the default sorting option, last usage date (2) and number of usages (3).

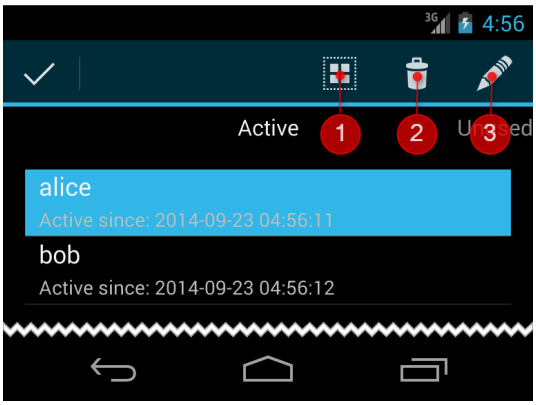

Figure 4.5.: After a long press on a label the app enters the edit mode. Available options are (1) selecting all labels, (2) discarding the label data (this does not record any data) and (3) editing the timestamps of the activated labels

- labeling an interaction that already took place
- edit activation and deactivation time for a label
- discard an activated label (erroneous activation)

We implemented an edit function for active labels to fulfill these use cases. A long press on a label in the list of active labels activates the edit mode (Fig. 4.5). In the edit mode, the user can discard labels or edit label information. If the user discards a label, no data for this activation is recorded.

The edit mode allows the user to label the social interactions after the fact. This means, if the user forgets to to label a social interaction, they can enter the data for the social interaction at a later time. First, the user activates the correct labels for the interaction. Then, the user can set the correct timestamps using the edit mode. Figure 4.6 shows this interaction.

The settings screen is the last element in the UI we explain. The most important setting for the user is probably the option to completely disable all data mining functionality. As soon the checkbox for data mining in the settings screen is unchecked, all data mining is stopped until the user enables the checkbox again. We add this checkbox to address privacy concerns of our users. See Fig. 4.7 for a screenshot of the settings screen.

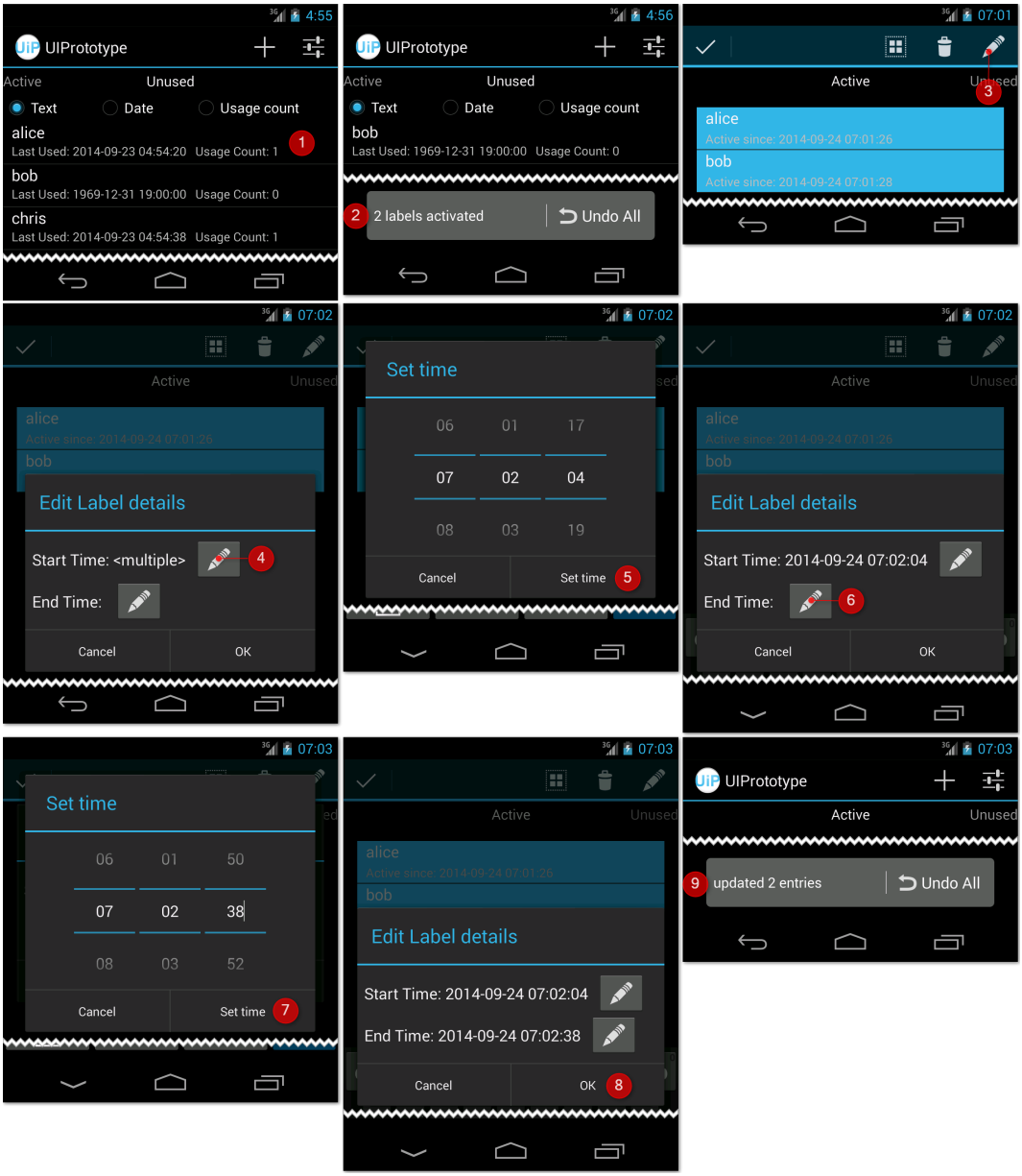

Figure 4.6.: The user can add data for social interactions after they took place. First he activates the corresponding labels (1,2), then he uses the edit mode (3) to set the correct start and end timestamps (4,5,6,7). After the save (8), the app adds the interactions to the dataset (9).

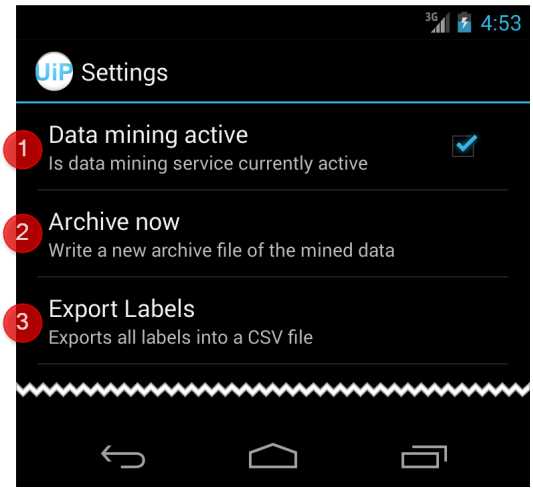

Figure 4.7.: The settings screen allows the user to completely disable the data mining of the app (1), to force a new backup of the mined data (2) (per default, backups are made periodically) and export all labels to a CSV file for further processing (3).

Data Collection

Our data mining app uses nearly all the probes provided by FunF . An overview of the used probes including the structure of the data points and a brief description of the various fields per data point can be found in Appendix A

FunF encapsulates all the data mining logic in so called `Probes` (see Section 4.1.1). This enables the developer of a data mining app to add additional data collection logic without modifying the code of FunF itself. We created a new android module that encapsulates the labeling framework. The module includes a FunF probe that collects the label data for FunF and an Android content provider[3] that holds the mined data for the labels.

Figure 4.8 shows the individual components of the application and how they interact with each other.

To gather data about social interactions, we ask the user to *activate* or *deactivate* labels as described in Section 4.1.2.

[3] `http://developer.android.com/guide/topics/providers/content-providers.html`

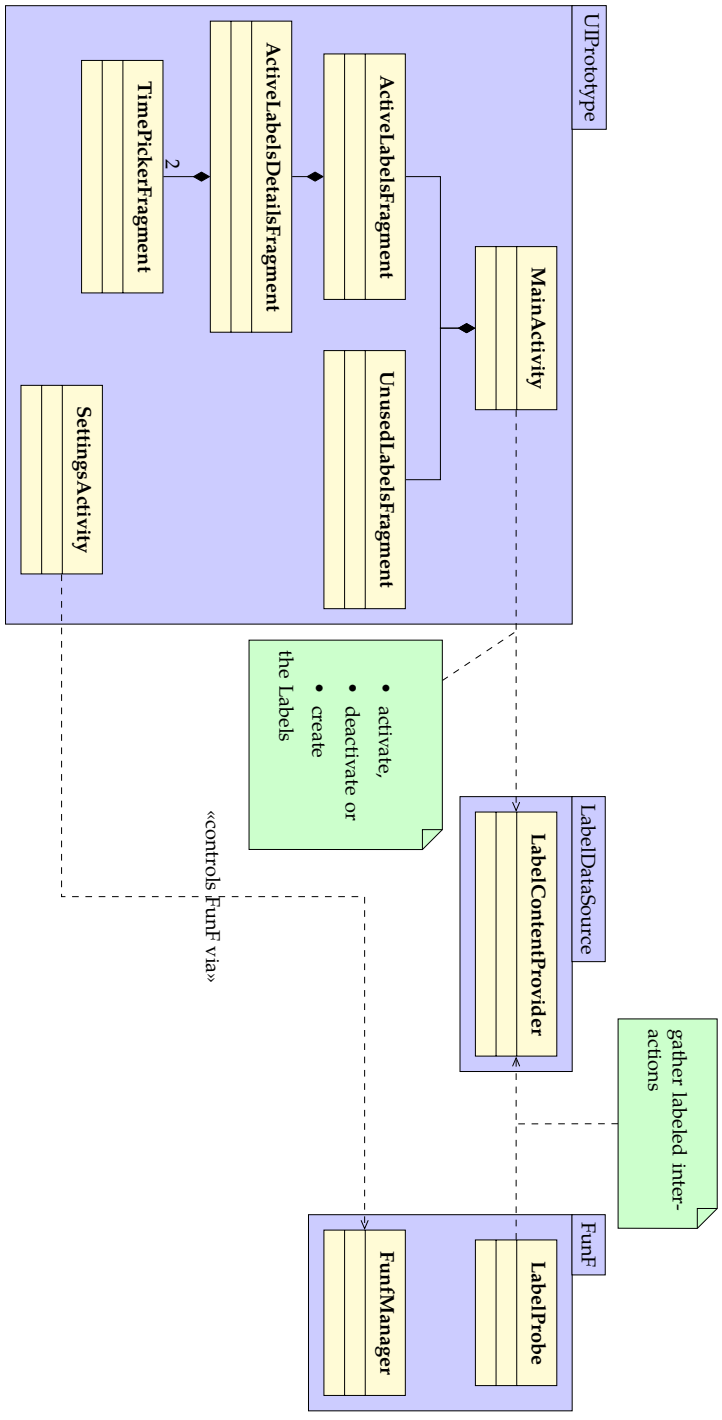

Figure 4.8.: This figure shows an high-level overview of the classes and interactions of the data mining app. Please note that the class diagram is not complete

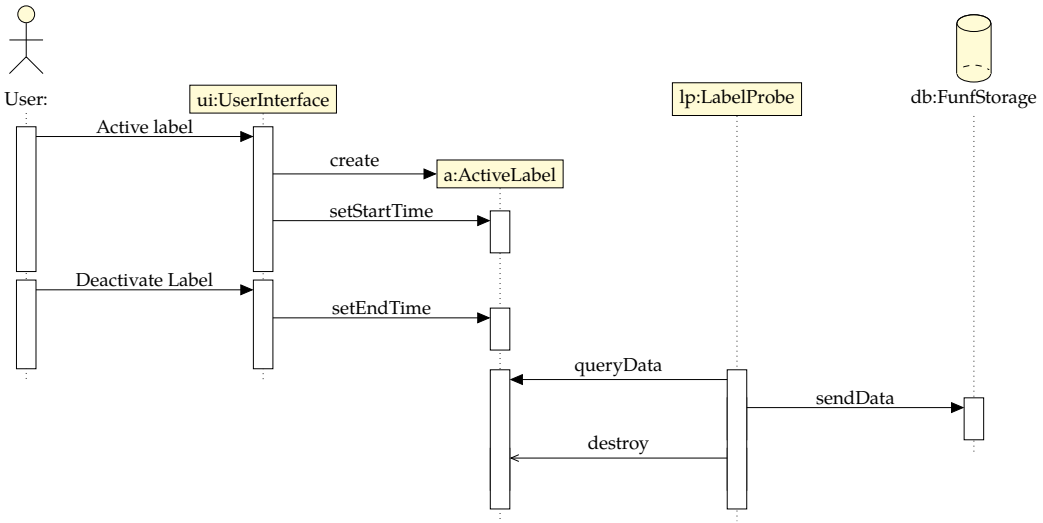

Figure 4.9.: This figure shows the sequence of a labeled interaction through the system

Every time the user activates a label, the app starts tracking a new social interaction for this label. The start timestamp for this particular social interaction is set at the moment the user activates the label. When the user deactivates the label, the system sets the end timestamp for the social interaction. Thus, every activation of the label indicates a new social interaction, every deactivation implies the end of the social interactions.

With this recording method we can track individual social interactions and their flow through time. Even in group interactions, accurate tracking of individual participants is possible and even encouraged.

We gather all social interactions in a Android content provider. The content provider is implemented with a SQLite database, that holds the various labels, currently active social interactions and not yet mined interactions.

The probe for FunF gathers all finished social interactions and exports them to the FunF framework. The framework in turn collects this data in its own database. These databases, with all the data collected by FunF, are then used for data analysis. The social interaction probe additionally deletes the finished interactions after they were exported to FunF. This keeps the database small and ensures that the interaction database does not become a perfor-

mance bottleneck detrimental to the user experience. Figure 4.9 shows the sequence of a captured interaction through the data mining process, starting with the user interaction and ending with FunF, where the data is finally stored.

In this section we described how the app is structured architecturally. We also showed the possible user interactions with the app and explained how the app persists the collected data. Additionally, we showed how the data flows through the app.

4.2. Exploratory Data Analysis

In this section we describe our exploratory data analysis setup. We show how we extract and transform the data from the archives generated by the FunF framework data using Python[4] and the Pandas Library[5] [31].

As mentioned in Section 4.1.1, the primary data format of the FunF Framework is JSON. FunF stores all data collected by the probes in one SQLite table, `data`. The table consists of five columns:

1. `_id` unique identifier of the dataset within the table
2. `name` fully qualified name of the `Probe` that generated the dataset
3. `timestamp` time of the data point
4. `value` the value of the data point as JSON string

Additionally, FunF archives the SQLite database at regular intervals. Every time the framework archives the database, a new database file is created. This means that for a given data collection run, the data will be scattered across many databases. Therefore, we need to combine all archived databases in one dataset to be able to analyze the data. FunF provides some sample data analysis scripts. We reuse this scripts to combine all archive files into one database. This database contains on table that holds the combined data, `data`. The structure of the table is similar to the table in the individual archive files:

[4] https://www.python.org/
[5] http://pandas.pydata.org/

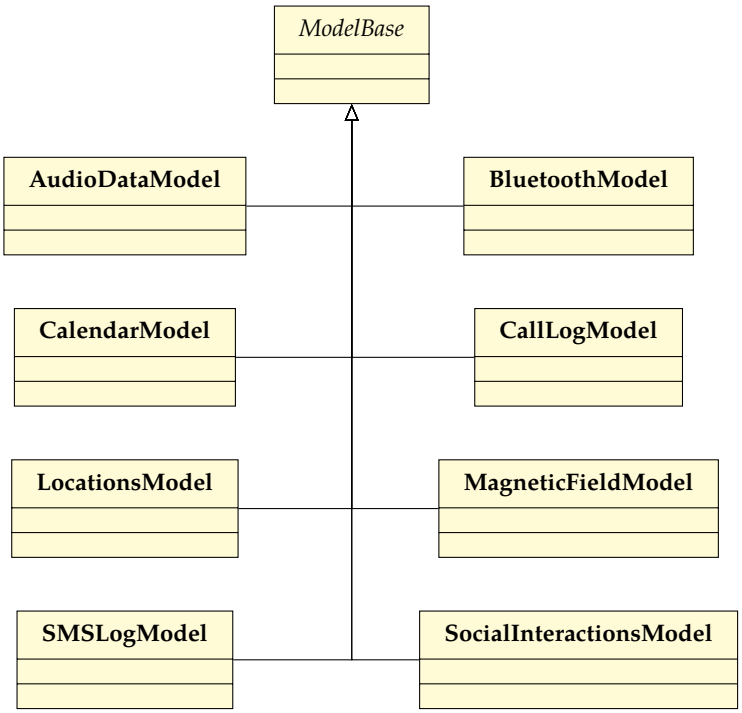

Figure 4.10.: This figure shows the data models developed for this thesis. Every model contains all necessary extraction and normalization logic to populate itself from a FunF dataset

- `id`: unique identifier of the data point within the whole dataset
- `device`: unique identifier of the device the data point was recorded on within the dataset
- `probe`: fully qualified name of the probe that recorded the dataset
- `timestamp`: timestamp of the data point
- `value`: the value of the data point as JSON string

Based on the combined dataset, we extract the data in so called data frames, a data structure of the Pandas Library. A data frame is a two dimensional data structure with labeled rows and columns. We provide a Python module with custom data models for some probes of FunF. This data models have a common interface and contain the necessary extraction and normalization logic to conveniently analyze the data collected by FunF. See Fig. 4.10 for an overview of all the data models used for analysis.

With the data models we generate various diagrams to find correlations between social interaction activity and the sensor data collected by FunF. Additionally, we ask our participants for more meta-data about their activity during the data collection.

In this chapter we described how the app collects the data, how the user can interact with the app and how we prepare and transform the raw data to use it for further analysis. We also provided a detailed overview of the architecture of FunF and of the architecture of the social interaction labeling mechanism. We also showed how we use Python and the Pandas Library for data analysis.

5. Data Collection and Exploration Study Setup

In this chapter we explain how we conducted our data collection. First, we define the goal of the data collection. After that, we describe the setup and how the participants have to collect the data. After that, we define the two groups of participants in term of age, general occupation and further demographic parameters. Then provide a quick overview over the collected data. Lastly, we describe the structure of the interview we hold with every participant after the data collection run.

5.1. Goal

The goal of the data collection run was to collect as many sensor readings as possible while asking the user to label their social interactions.

We want to have a diverse set of sensor readings, because we explore the mined data to find correlations between sensor readings and social interactions. Most of the related work concerning social interactions focuses on the microphone as sensor (for voice or speaker recognition, see Section 2.2) while we want to explore all options provided by FunF.

5.2. Setup

In this section we describe the setup of the data collection study. We explain what the participants need to do and how we structure and conduct the study.

The study is conducted over the duration of three days. Every participant installs the data collection app on their Android device, be it a smartphone or a tablet. We require that all sensor on the device stay active during the study period, this means that the participants have to keep GPS, Bluetooth and WiFi enabled.

We assist the participants during the initial setup of the app and explain how they need to label their social interactions. The participants are asked to label their social interaction during the study period as exactly as possible. To recap the detailed explaination in Section 4.1.2, the participant has to activate (or create a new) label at the start of a social interaction with the person the label indicates, and deactivate the label at the end of the social interaction with the person. At group interactions, every participant of the interaction should be tracked individually.

We do not impose any restriction on the participant's activity during the study period, we want to capture their every day social interactions and social activity.

When the three days are over, we download the mined data from the participants device and conduct a brief interview about their activity during the three days to gain additional meta-data (see Section 5.5).

5.3. Participants

We conducted two data collection runs during his thesis, each with a distinct participant group.

The first group contains three male software development professionals, in the age range of 25 - 35 years. Additionally, they share the same office space and they collected their data simultaneously. The second group contains two male and one female researcher(s) in the age range of 25 - 35 years. They work in the same building. They collected the data sequentially on a device provided by their research institute. Table 5.1 shows the summary of the two participant groups that were so kind to provide data for this thesis.

Group	1	2
Number of participants	3	3
Gender	3 male	2 male, 1 female
Age	25 - 30	25 - 35
Occupation	Software Development	Research
Modus	Simultaneously	Sequential

Table 5.1.: Demographic overview of the study participants. The *modus* describes if the participants in a group collected the data during the same three days or if every participant was collecting the data separately.

5.4. Collected Data

We collect all data provided by the FunF Framework. For an overview and explanation of the collected data see Appendix A.

While we collect all that provided by the FunF Framework, we focus our attention on the following `Probes`:

- `LabelProbe`: The social interactions as labeled by the participants. See Appendix A.2 for details.
- `SimpleLocationProbe`: The GPS data of the user during the data collection study. See Appendix A.32 for details.
- `BluetoothProbe`: This probe reports all discovered Bluetooth devices. See Appendix A.14 for details.
- `CalendarProbe`: Collects all appointments in the Android calendar for a given day. For details see Appendix A.3
- `MagneticFieldSensorProbe`: This probe periodically records the magnetic field values (Appendix A.23)
- `AudioFeaturesProbe`: This probe collects and analyzes the audio sensor readings. Appendix A.11 describes this probe in detail.

We discuss the data and the analysis in detail in Chapter 6.

5.5. Interview Structure

After the data collection, we perform an exit interview with the participants. We try to answer following question with our participants, based on preliminary data analysis results of the data of a participant.

- How do you evaluate the usability of the app?
- How do you rate labeling your social interactions?
- Classify your labeled interactions in following categories:
  - business
  - personal
- Classify your location clusters in the following categories:
  - work
  - home
  - leisure
- What did you like about the data mining?
- What did you dislike about the data mining?

6. Results

In this chapter we present the result of our data analysis. We show how the social interactions correlate with various sensor readings collected by our participants during the duration of three days. We visually explore the data to find potential features to use for data mining or machine learning.

First we provide some descriptive statistics about the collected data, then we present the visualizations we use for data analysis. We will discuss how

- Time of day
- Audio data
- Calendar appointments
- Magnetic field values
- Bluetooth data
- Location data

influence the interaction frequency, interaction duration and other factors.

6.1. Statistics about the Collected Data

As discussed in Section 5.3, we have two distinct user groups four the data collection. Table 6.1 shows descriptive statistics about the social interactions labeled by both groups. This includes the total number of collected data points across all sensor probes, the total number of labeled social interactions, the average number of social interactions per person and the shortest, longest, median and average interaction duration per group.

The demographic statistics of the groups are shown in Table 5.1.

Group	1	2
Total datapoints (all sensors)	approx. 16M	approx. 35M
Total number of social interactions	211	126
Average number of interactions per person	70	42
Shortest interaction	00:00:01	00:00:01
Longest interaction	02:06:19	02:31:53
Median interaction duration	00:02:24	00:03:53
Mean interaction duration	00:09:49	00:17:54

Table 6.1.: Data Collection and labeling statistics for the two user groups

6.2. Social Interactions Versus Time of Day

Our first focus is how the social interactions are spread out during the time of day. For this purpose, we counted the number of social interactions during every hour of a week and generated a heatmap based on the binned social interactions.

Figure 6.1 shows the day-time heatmaps for the first group. This group labeled their social interactions simultaneously. Therefore, the days in th different heatmaps correspond to each other. As can be seen, most of the social interactions of this group take place before noon. This can be attributed to their workplace, where they interact with a lot of different people. Another interesting finding is the spike on Monday between 08:00 and 09:00. We were able to attribute this spike to a meeting taking place during this time.

Figure 6.2 shows the distribution of social interaction over the labeled days for the second group. Note that this group performed their data mining sequentially and independent from each other. Therefore the days shown in

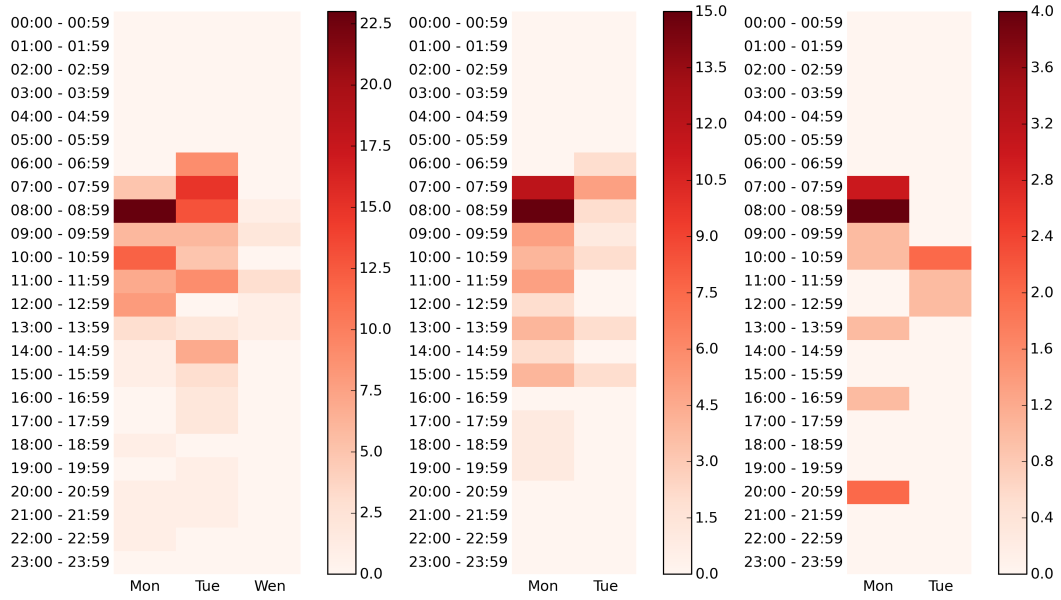

Figure 6.1.: This heatmaps show the distribution of social interactions over the course of the monitored timespan. Most of the social interactions take place in the morning hours, with a spike at Mon from 08:00 - 09:00. This spike is attributed to a meeting taking place at this time.

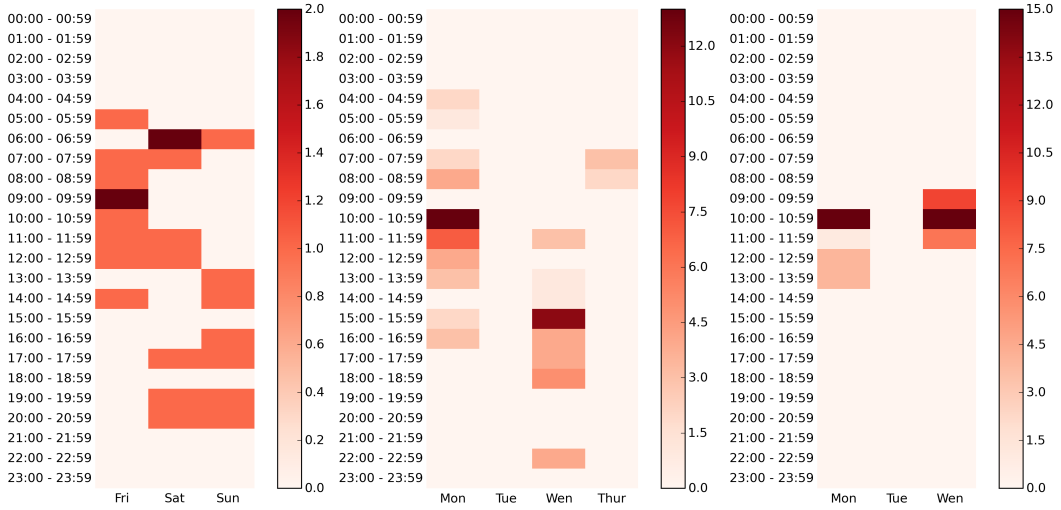

Figure 6.2.: Here the heatmaps of the distribution of social interaction during the week are shown for the second group. Because Group 2 labelled their social interactions sequentially, the days in the different heatmaps do not correspond to each other.

the heatmaps do not correspond to each other. But still there are some similarities, especially for the second and third heatmap, where the majority of social interactions on Monday take place from 10:00 - 14:00. We attribute this to the fact that both subjects work in the same research group

We see that the number of social interactions for most of the subjects in both groups take place during the working hours of the week, with a reduce during the afternoon and the evening hours.

6.3. Social Interactions and Audio Energy

Next, we look at social interactions and if they correlate to audio signal strength, more specifically, how they correlate to Power Spectral Density (PSD). The FunF framework provides a probe that computes the PSD across four frequency bands:

- 50 Hz to 250 Hz
- 250 Hz to 500 Hz

- 500 Hz to 1000 Hz
- 1000 Hz to 2000 Hz

For this purpose, we plot the signal strength over time and overlay it with a step plot of the social interactions. We can then search for specific signal strengths on the frequency bands that correlate with a high social interaction count.

Figure 6.3 shows such a plot. The green line indicates the audio signal over time, will the blue steps show the social interactions. We choose a step plot for the interactions because the label can only be active or not active, and the step edges show the range of how long a specific number of interactions is active very well. We calculated such a plot for every frequency band. During our analysis we also use plots where we subtract the mean or median energy from the individual data points.

We are not able to determine a specific signal band or signal energy that correlates with social interactions. See Chapter 7 for a further investigation of this topic.

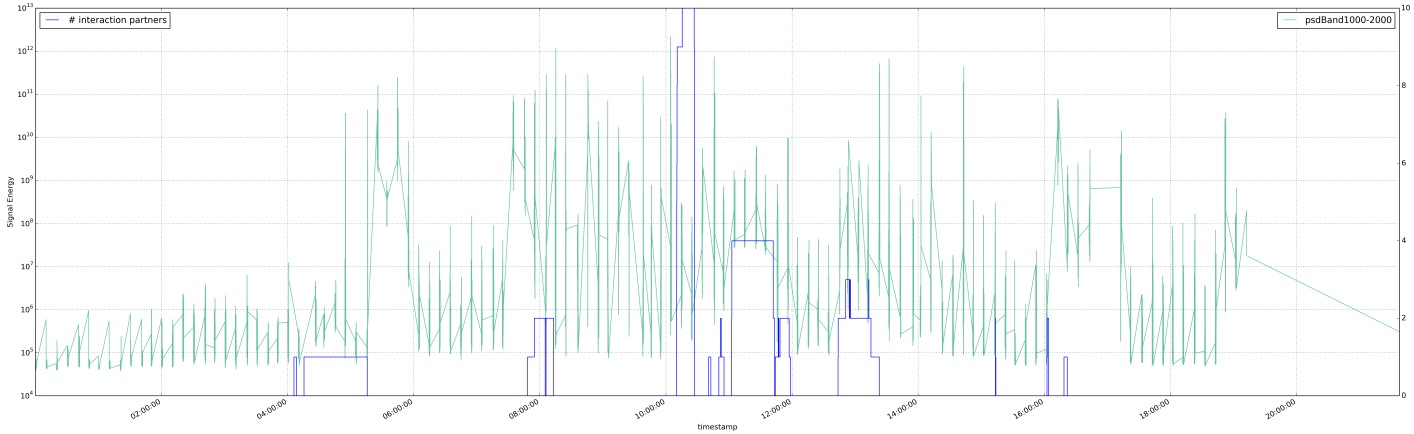

Figure 6.3.: Sample plot we used to find correlations between the audio signal energy and the number of interactions. The green plot shows the signal energy over time. The blue steps indicate the number of currently active social interactions. The height of the step is proportional to the number of currently active labels. The vertical step boundaries indicate the start and end of the social interaction.

6.4. Calendar Data

The next point of our analysis is calendar data. We propose that calendar data can correlate with social interactions, because in our experience most appointments in a calendar correspond to events where interactions occur, for example business meetings or an appointment to meet friends.

This assumption holds true for some of our participants. Figure 6.4 shows the interaction data of one of our participants for whom we got calendar data. As we can see, some appointments correspond to increased social interaction activity, especially the appointment between 09:00 and 10:30 at the first day (shaded orange) and the appointments between 06:00 and 12:00 of the first and second day (shaded in green and blue, respectively).

For further analysis of this correlation we need additional data of multiple participants over a longer period. We also need different professions and personal styles of calendar usage, for example some people might add dates to their calendar as personal reminders as opposed to appointments involving other people.

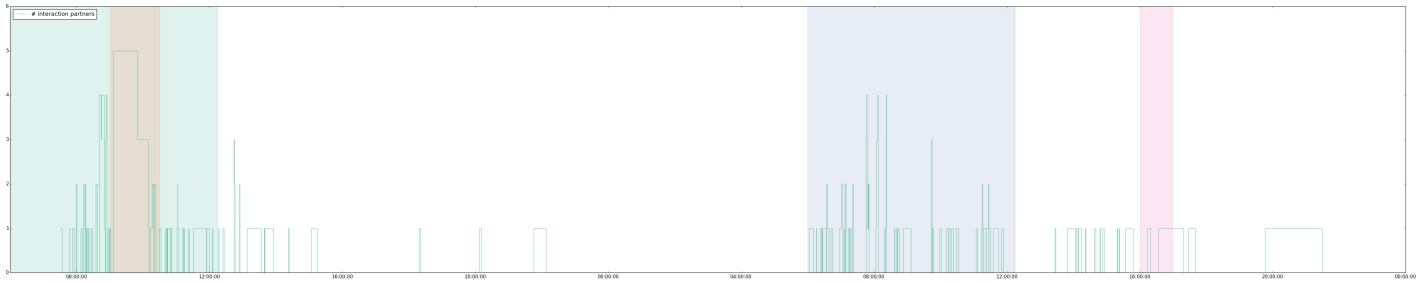

Figure 6.4.: This figure shows the correlation between calendar appointments and social interaction activity for one participant. The shaded areas show the time ranges where an appointment is marked in the calendar of the participant. Note the appointment shaded in orange on the first day, and the appointments between 06:00 and 13:00 of the first and second day (shaded green and blue). These appointments correlated with high social activity.

6.5. Magnetic Field Value

Inspired by IndoorAtlas[1], we try to use magnetic field values to determine if participants are near each other, and in turn, have social interactions. Haverinen and Kemppainen [21] describe a system that performs indoor positioning by comparing magnetic field value readings with an a-priori generated field value map of an indoor location.

For this purpose we plotted the magnetic field sensor readings of our first group, that performed their data mining simultaneously and compared the data. Unfortunately, the magnetic field values are very susceptible to miscalibration and other errors. We find the comparison of raw, uncalibrated field values not viable to determine proximity.

Figure 6.5 shows a slice of the magnetic field value readings for the first group. During the shown slice all participants where in the same office. Unfortunately, their proximity does not translate to similar sensor readings. We guess this is due to the uncalibrated sensors and the unsynchronized sensing intervals of the different data collection apps on the smartphones of our participants.

[1] http://www.indooratlas.com/

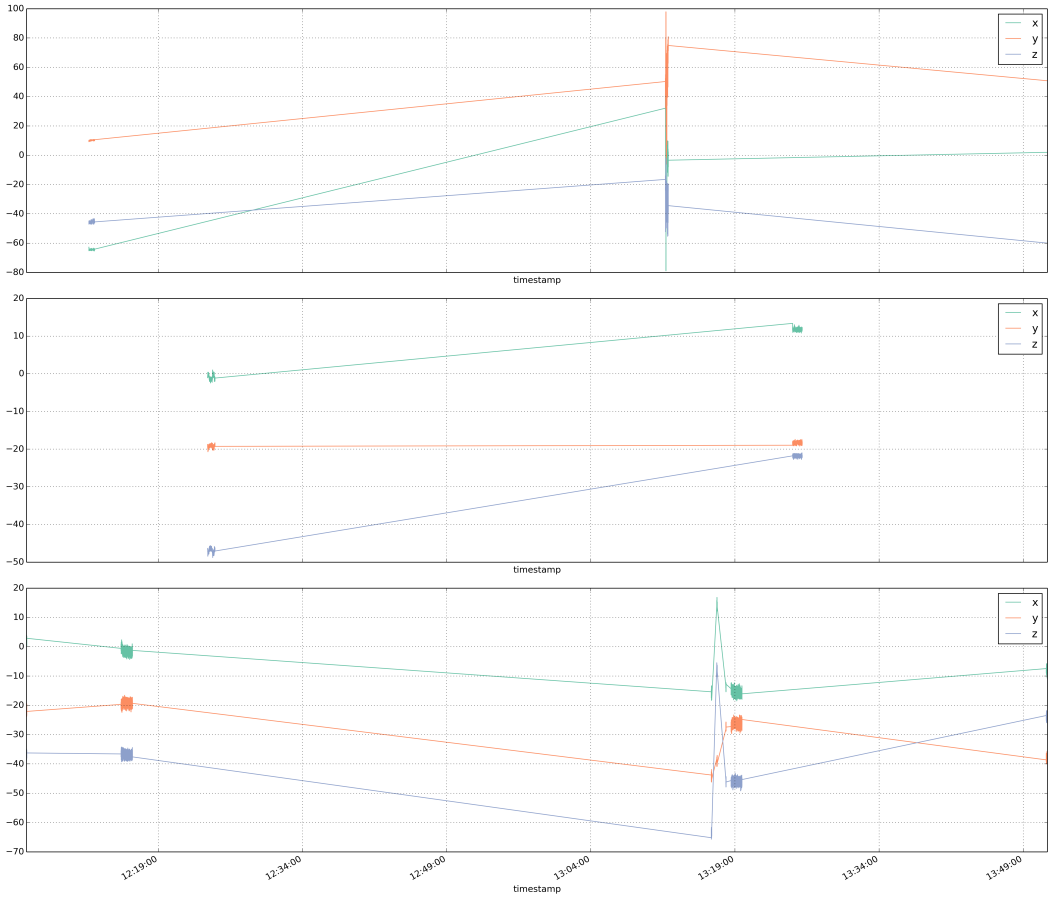

Figure 6.5.: This figure shows the magnetic field values of the first day of the first group. During the shown timeframe all participants where in the same office. Note the different sensor readings and the different times the magnetic field values where sampled

6.6. Bluetooth Proximity Data

Our next focus is on the bluetooth proximity data of our participants. Bluetooth proximity is used very often to infer social relationships of an observed group (for example [18, 1, 5, 32] and many others)

We try to find correlations between specific labels (and thus persons) and the Bluetooth devices discovered by FunF. We build a co-occurrence table of discovered Bluetooth devices and Label-Id, based on their temporal relation. When a Bluetooth device (uniquely identified by its hardware address) is discovered at the same time a label is active, we count one co-occurrence.

We found numerous co-occurrences with a high count for all our participants, but these co-occurrences are not directly related with social interactions. We elaborate this further in Chapter 7.

6.7. Location Data

Lastly, we look at the location data of our participants. We generate a heatmap of the social interactions based on their time-correlated GPS information. We mark a spot on the heatmap at the location the user is at the time of a social interaction.

Additionally, we cluster the location data and count the social interactions taking place at every cluster. For this purpose, we group the raw location data into clusters of 100 m radius. For each cluster we determine the time frames the user was at this location, and count the social interactions taking place during these time frames.

Figure 6.6 shows the result of our location data analysis. The heatmap identifies areas where social interactions occur. The pins donate the centers of the cluster our algorithm calculates. For most of our participants the social ineractions are centered around two distinct locations, their home and their workplace (identified by our participants during the interview). The figure shows an *atypical* result. The participant labeled during a weekend. On this weekend social interactions occurred on various locations, which indicates

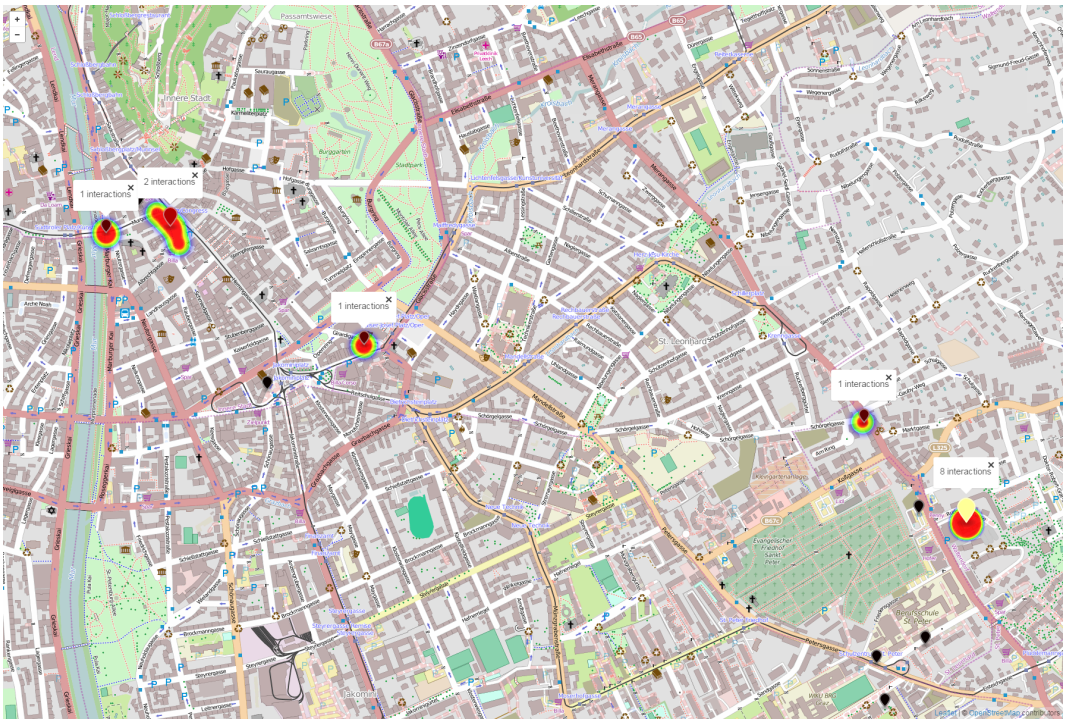

Figure 6.6.: The result of our GPS data analysis. The pins denote the centers of the clusters we calculate with our cluster algorithm. The heatmap shows the areas where most social interactions occur. Black pins donate clusters of location data where the participant did not label any social interactions

the participant was visiting friends, shopping in the city or involved in other social activity outside their immediate family or work environment.

We also found that the geographic location strongly correlates with a specific group of people. Social interactions with colleagues happen mostly at the probands work place, while interactions with their significant others mostly happen at home.

While we hit some roadblocks as described in the previous section, we think that we have found some interesting results that can warrant further exploration.

Additionally, several users told their interest in location heatmap visualization. We think this heatmap would be interesting to get a sense where someones social hotspots are and to provide some interesting feedback for a person if they want to gain some insight in their social interaction patterns.

6.8. Open Source Release

We release all code written for this thesis as open source under the MIT License[2]. Note that the FunF Framework is licensed under the LGPL License[3] The repository is located here:

`https://github.com/mpern/master_thesis`

[2] `http://opensource.org/licenses/MIT`
[3] `http://www.gnu.org/licenses/lgpl.html`

7. Discussion

In this chapter we discuss our findings from Chapter 6. We evaluate how well FunF works as a framework for data collection on mobile phones and discuss some of our results in greater detail. We elaborate how labeling of social interactions on a smartphone impacts our participants. After that, we give recommendations on how infer social interactions on a smartphone and discuss possible future approaches.

7.1. Lessons Learned

In this section, we will show the challenges of our approach and discuss how these influence our result. We will assess the FunF Framework, elaborate our difficulties with Bluetooth proximity and give a summary of the feedback by our participants.

7.1.1. FunF Framework

While the FunF Framework serves as a good starting point for data mining on smartphones, it has also some flaws to consider. First and foremost, the framework itself is not yet stable enough. For this thesis, we used version 0.4, the current version at the time of writing is 0.5RC1. We tried using the new version for our app, but the new version has serious bugs that renders it unusable. We fixed several bugs for 0.4 to make it usable on modern smartphones. The major drawback of FunF is that all probes run on the user interface thread, which makes any app using it prone to so called "Application Not Responding" Errors of Android, especially if the app uses many probes and/or calculation intensive probes.

Label (Hash)	BTID	Co-Occurrences
535e7ed454…	0C:89:10:0B:79:E2	34
bbe83a32be…	0C:89:10:0B:79:E2	33
4094786d50…	0C:89:10:0B:79:E2	33
bbe83a32be…	2C:D0:5A:87:77:E8	33
535e7ed454…	74:E5:43:52:F5:8D	32

Table 7.1.: This table shows the top five co-occurrences of Bluetooth devices with social interactions for one of our participants. As we can see, multiple labels co-occur with the same device, while some labels co-occur with other devices with nearly equal frequency.

7.1.2. Bluetooth Proximity

While numerous publications use Bluetooth proximity to infer social dynamics, we were not able to infer significant correlations (see Section 6.6). This is mainly because for modern Bluetooth devices, the device can not be discovered by other devices per default. This also holds true for all modern smartphones, which in turn means we are not able to associate a smartphone to a label. For all our participants, we were not able to find a device or a set of devices that highly co-occurs with one of the labels. We propose that most devices are more correlated with a specific location (for example, the Bluetooth-enabled television in the living room is discoverable per default. Therefore, all social interactions taking place in this room co-occur with this device.) It could be possible to generate "device fingeprints" that identify a location based on the discovered Bluetooth devices.

Table 7.1 show the top five co-occurences of social interactions with Bluetooth devices for one of our participants. As we can see, several labels co-occur with the same Bluetooth device, while some labels co-occur with different Bluetooth devices. The results for our other participants look similar. We were only able to determine one meaningful co-occurrence that links one specific label to one specific Bluetooth device.

7.1.3. Labeling Social Interactions

We ask all our participants how the labeling of social interactions impacts them. All our participants report that the UI is easy to use, but remark that the labeling itself is a difficult task. The primary reasons that make labeling hard are:

- A lot of social interactions are very brief (for example, a few words at the coffee machine in the office or a spontaneous meeting of a friend on the street). This makes it hard to label the interaction correctly, with regard to activating and deactivating the label and the correct moments.
- Sometimes our participants just forget to deactivate a label. This is to be expected during a busy day. We included a way to manually set the start and end time for a label to mitigate this, but it still happened during our data collection phase.
- Labeling every social interaction during the day is very tiresome. It especially breaks the flow of group interactions because you have to constantly adjust your active labels.

With this section we show that there are some problems with our approach. In the next section we give recommendations on how to avoid these problems and what future works on the topic of our thesis could use to generate better results.

7.2. Recommendations

Based on our results and the points we raise before, we will now give some recommendations and ideas for future attempts to detect social interactions on smartphones.

First and foremost, we think a future approach has to include advanced audio signal processing. Xu et al. [42] show how audio signal processing can be used to infer the speaker count of meeting or similar setting. One approach would be to program a new FunF `Probe` and include it in the data collections. We think the estimated count would be a very good indicator for the social interactions a person has at a given moment.

Another consideration is the FunF Framework. Because of its architecture, one has to carefully choose which and how many `Probes` to use. We think for the next approach has to consider which data to mine and only configure `Probes` for the selected data. The configuration itself needs to be cautiously evaluated. We find that data mining can drain the battery of smartphones at a very high rate. Therefore, every duty cycle for every `Probe` has to be weighted against the energy drain and adapted to the requirements of the next approach.

Additionally, the duty cycles should be synchronized between all participants. Figure 6.5 illustrates the reason for this requirement very well. As we can see, the data points do not occur at the same time, but shifted between the participants. To perform time correlation, the system time of the smartphones and schedules of the `Probes` need to be synchronized. The short duration of most social interactions proposes a challenge for every duty cycle. Make it too long and there will be no data available for many interactions. Make it too short and the energy drain becomes unacceptable.

The next data collection run has to include a greater number of participants that perform the labeling for a longer time period. This allows us to capture a more diverse behavior sample for scenarios like weekends, vacation days and others. We also think that only simultaneous data mining for all participants of a group leads to viable results.

Another point brought up by the user study is how hard it is to label social interactions. The app should assist the user in labeling. One idea would be to use the output of the Crowd++ [42] algorithm to suggest to the user that a social interaction has happened. Based on the output, the app could suggest the count of participants and the estimated start and end times. The user could then activate labels and adjust the timestamps accordingly.

8. Conclusion

With this thesis we tried to determine the number of persons a user has face-to-face social interactions for a given timestamp. Using nearly all data mining capabilities of the FunF Framework, we performed a user study with six participants divided into two groups. We collected a total of more than 50M data points for all our participants.

Using this dataset, we visualized how and if the time of day, bluetooth proximity data, magnetic field values, location data and audio signal energy correlate with the labeled social interactions.

Writing the data mining app was a great opportunity to gain an insight into the Android Software Development Kit (SDK) and into the FunF Framework and learn about the inner workings of the sensors a typical smartphone possesses.

Performing a data collection helped us understand the difficulties that arise when working with persons that know nothing about your research and how to convey information so that everyone can understand it.

During the visualization phase we gained valuable knowledge on how to process, normalize and visualize time-series data using the Pandas Data Analysis Library in conjunction with matplotlib[1]. It also deepened our understanding of the Python programming language.

We think that this thesis provides a good starting point for social interaction detection on smartphones. We show numerous approaches and ideas and how they pan out. We also discuss all our findings and problems, so that further research on this topic can benefit from this information by avoiding

[1] http://matplotlib.org/

our problems and getting ideas on new approaches. Additionally, we provide numerous considerations and ideas for future work in Section 7.2.

While it was unfortunate that we were not able to find some strong correlations between social interactions and our mined sensor data, we think the large data sample can be used by future work to verify their results or as initial training data for other algorithms. We also think that our data set includes some interesting patterns, that may trigger new ideas on this topic. We think the correlation between calendar appointments and social interactions warrants further investigation and could provide an easy way to increase the accuracy of a possible algorithm that detects social interactions in smartphone sensor data.

Additionally, we provide all code that was written for this thesis (this includes the data mining app and all scripts to normalize, analyze and visualize the data) as open source, so that some of it may be used again.

With this thesis we present a first exploration on how to detect social interactions on mobile phones via Mobile Sensing. We implemented a smartphone data mining app, with which we conducted two user studies with three participants each over the course of three days. We analyzed and visualized the connection between time of day, audio data, calendar appointments, magnetic field values, Bluetooth data and location data and discussed theses results and provided several recommendations how future works on this topic could improve on our study.

Appendix

Acronyms

AR Augmented Reality. 13

CS Cosine Similarity. 16

DFT discrete Fourier transform. 14

ECG Electro-cardio gramm. 9

GMM Gaussian Mixture Model. 15
GPS Global Positioning System. iii, v, 19, 38, 39, 51, 52, 104
GSM Global System for Mobile Communications. 3

JSON JavaScript Object Notation. 22, 34, 35

MFCC Mel Frequency Cepstral Coefficient. 82
MFCC Mel Frequency Cepstral Coefficient. 15, 16

NTP Network Time Protocol. 108

OCR Optical character recognition. 12
OS Operating System. 3, 22

PLP Perceptual Linear Predictive. 15
PSD Power Spectral Density. 44, 76, 82

SD Secure Digital. 71
SDK Software Development Kit. iii, 59, 80, 85
SMS Short Messaging Service. 106

UI User Interface. 23, 24, 29, 57

Bibliography

[1] Nadav Aharony et al. “Social fMRI: Investigating and shaping social mechanisms in the real world”. In: *Pervasive and Mobile Computing* 7.6 (2011), pp. 643–659 (cit. on pp. iii, 21–23, 51).

[2] Xuan Bao and Romit Roy Choudhury. “MoVi: Mobile Phone Based Video Highlights via Collaborative Sensing”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’10. San Francisco, California, USA: ACM, 2010, pp. 357–370 (cit. on p. 11).

[3] A. Beach et al. “WhozThat? Evolving an Ecosystem for Context-aware Mobile Social Networks”. In: *IEEE Network: The Magazine of Global Internetworking* 22.4 (July 2008), pp. 50–55 (cit. on p. 11).

[4] Mark Bilandzic et al. “Laermometer: A Mobile Noise Mapping Application”. In: *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*. NordiCHI ’08. Lund, Sweden: ACM, 2008, pp. 415–418 (cit. on p. 10).

[5] Elisa Gonzalez Boix et al. “Flocks: enabling dynamic group interactions in mobile social networking applications.” In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. Ed. by William C Chu et al. ACM, 2011, pp. 425–432 (cit. on p. 51).

[6] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. RFC Editor, Mar. 2014, pp. 1–15 (cit. on p. 22).

[7] Nirupama Bulusu et al. “Participatory sensing in commerce: Using mobile camera phones to track market price dispersion”. In: *Proceedings of the International Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense 2008)*. 2008, pp. 6–10 (cit. on p. 12).

[8] Chih-Yen Chen et al. “A Review of Ubiquitous Mobile Sensing Based on Smartphones”. In: *International Journal of Automation and Smart Technology* 4.1 (Mar. 4, 2014), pp. 13–19 (cit. on pp. 3, 5).

[9] X. Chen et al. “Cellular phone based online ECG processing for ambulatory and continuous detection”. In: *Computers in Cardiology, 2007*. Sept. 2007, pp. 653–656 (cit. on p. 9).

[10] Yohan Chon et al. “Automatically characterizing places with opportunistic crowdsensing using smartphones”. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. New York, New York, USA: ACM Press, 2012, p. 481 (cit. on p. 13).

[11] Sidney Cobb. “Social support as a moderator of life stress”. In: *Psychosomatic medicine* 38.5 (1976), pp. 300–314 (cit. on p. 1).

[12] Sheldon Cohen and Thomas A Wills. “Stress, social support, and the buffering hypothesis.” In: *Psychological bulletin* 98.2 (1985), p. 310 (cit. on p. 1).

[13] Linda Deng and Landon P. Cox. “LiveCompare: Grocery Bargain Hunting Through Participatory Sensing”. In: *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*. HotMobile ’09. Santa Cruz, California: ACM, 2009, 4:1–4:6 (cit. on pp. 12, 13).

[14] Tamara Denning et al. “BALANCE: Towards a Usable Pervasive Wellness Application with Accurate Activity Inference”. In: *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*. HotMobile ’09. Santa Cruz, California: ACM, 2009, 5:1–5:6 (cit. on p. 9).

[15] ThangM. Do, SengW. Loke, and Fei Liu. “HealthyLife: An Activity Recognition System with Smartphone Using Logic-Based Stream Reasoning”. In: *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Ed. by Kan Zheng, Mo Li, and Hongbo Jiang. Vol. 120. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2013, pp. 188–199 (cit. on p. 9).

[16] Wen Dong, Bruno Lepri, and Alex (Sandy) Pentland. “Modeling the co-evolution of behaviors and social relationships using mobile phone data”. In: *Proceedings of the 10th International Conference on Mobile and*

Ubiquitous Multimedia. MUM '11. New York, New York, USA: ACM Press, 2011, pp. 134–143 (cit. on p. 12).

[17] D.L. Donoho. "Compressed sensing". In: *Information Theory, IEEE Transactions on* 52.4 (Apr. 2006), pp. 1289–1306 (cit. on p. 11).

[18] N. Eagle and A. Pentland. "Social Serendipity: Mobilizing Social Software". In: *IEEE Pervasive Computing* 4.2 (Jan. 2005), pp. 28–34 (cit. on p. 51).

[19] Nathan Eagle and Alex (Sandy) Pentland. "Reality mining: sensing complex social systems". In: *Personal and Ubiquitous Computing* 10.4 (Nov. 2005), pp. 255–268 (cit. on p. 1).

[20] Chunming Gao, Fanyu Kong, and Jindong Tan. "HealthAware: Tackling obesity with health aware smart phone systems". In: *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*. Dec. 2009, pp. 1549–1554 (cit. on p. 9).

[21] Janne Haverinen and Anssi Kemppainen. "Global indoor self-localization based on the ambient magnetic field". In: *Robotics and Autonomous Systems* 57.10 (Oct. 31, 2009). 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (5th CIRAS), pp. 1028–1035 (cit. on p. 49).

[22] Zhanpeng Jin et al. "HeartToGo: A Personalized medicine technology for cardiovascular disease prevention and detection". In: *Life Science Systems and Applications Workshop, 2009. LiSSA 2009. IEEE/NIH*. Apr. 2009, pp. 80–83 (cit. on p. 9).

[23] W.Z. Khan et al. "Mobile Phone Sensing Systems: A Survey". In: *Communications Surveys Tutorials, IEEE* 15.1 (2013), pp. 402–427 (cit. on pp. 7, 8).

[24] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity Recognition Using Cell Phone Accelerometers". In: *SIGKDD Explor. Newsl.* 12.2 (#mar# 2011), pp. 74–82 (cit. on p. 12).

[25] N.D. Lane et al. "A survey of mobile phone sensing". In: *Communications Magazine, IEEE* 48.9 (2010), pp. 140–150 (cit. on pp. 6, 7).

[26] Elisabeth Lex et al. “Where am I? Using Mobile Sensor Data to Predict a User’s Semantic Place with a Random Forest Algorithm”. In: *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Ed. by Kan Zheng, Mo Li, and Hongbo Jiang. Vol. 120. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Beijing, China: Springer Berlin Heidelberg, 2013, pp. 64–75 (cit. on p. 13).

[27] Qiong Liu and Chunyuan Liao. “PaperUI”. English. In: *Camera-Based Document Analysis and Recognition*. Ed. by Masakazu Iwamura and Faisal Shafait. Vol. 7139. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 83–100 (cit. on p. 13).

[28] Hong Lu et al. “SoundSense: scalable sound sensing for people-centric applications on mobile phones”. In: *Proceedings of the 7th international conference on Mobile systems, applications, and services*. MobiSys ’09. Kraków, Poland: ACM, 2009, pp. 165–178 (cit. on p. 13).

[29] Hong Lu et al. “StressSense: detecting stress in unconstrained acoustic environments using smartphones”. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp ’12. Pittsburgh, Pennsylvania: ACM, 2012, pp. 351–360 (cit. on p. 9).

[30] Nicolas Maisonneuve et al. “NoiseTube: Measuring and mapping noise pollution with mobile phones”. English. In: *Information Technologies in Environmental Engineering*. Ed. by IoannisN. Athanasiadis et al. Environmental Science and Engineering. Springer Berlin Heidelberg, 2009, pp. 215–228 (cit. on p. 10).

[31] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56 (cit. on p. 34).

[32] Emiliano Miluzzo et al. “CenceMe – Injecting Sensing Presence into Social Networking Applications”. In: *Smart Sensing and Context*. Ed. by Gerd Kortuem et al. Vol. 4793. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 1–28 (cit. on pp. 11, 14, 51).

[33] Emiliano Miluzzo et al. “Darwin Phones: The Evolution of Sensing and Inference on Mobile Phones”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’10. San Francisco, California, USA: ACM, 2010, pp. 5–20 (cit. on p. 15).

[34] Emiliano Miluzzo et al. “Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application”. In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys ’08. Raleigh, NC, USA: ACM, 2008, pp. 337–350 (cit. on p. 14).

[35] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. “Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones”. In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys ’08. Raleigh, NC, USA: ACM, 2008, pp. 323–336 (cit. on p. 10).

[36] Min Mun et al. “PEIR, the Personal Environmental Impact Report, As a Platform for Participatory Sensing Systems Research”. In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’09. Kraków, Poland: ACM, 2009, pp. 55–68 (cit. on p. 11).

[37] Nuria Oliver and Fernando Flores-Mangas. “HealthGear: Automatic Sleep Apnea Detection and Monitoring with a Mobile Phone”. In: *Journal of Communications* 2.2 (2007), pp. 1–9 (cit. on p. 9).

[38] Kiran K. Rachuri et al. “EmotionSense: A Mobile Phones Based Adaptive Platform for Experimental Social Psychology Research”. In: *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*. Ubicomp ’10. Copenhagen, Denmark: ACM, 2010, pp. 281–290 (cit. on pp. 12, 15).

[39] Rajib Kumar Rana et al. “Ear-phone: an end-to-end participatory urban noise mapping system”. In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN ’10. Stockholm, Sweden: ACM, 2010, pp. 105–116 (cit. on p. 11).

[40] Kewei Sha et al. “SPA: A Smart Phone Assisted Chronic Illness Self-management System with Participatory Sensing”. In: *Proceedings of the 2Nd International Workshop on Systems and Networking Support for Health Care and Assisted Living Environments*. HealthNet ’08. Breckenridge, Colorado: ACM, 2008, 5:1–5:3 (cit. on p. 9).

[41] Arvind Thiagarajan et al. “VTrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones”. In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’09. Berkeley, California: ACM, 2009, pp. 85–98 (cit. on p. 10).

[42] Chenren Xu et al. “Crowd++: Unsupervised Speaker Count with Smartphones”. In: *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’13. Zurich, Switzerland: ACM, 2013, pp. 43–52 (cit. on pp. 16, 17, 57, 58).

[43] Zengbin Zhang et al. “SwordFight: Exploring Phone-to-Phone Motion Games”. English. In: *IEEE Pervasive Computing* 11.4 (Oct. 2012), pp. 8–12 (cit. on p. 13).

Appendix A.

Description of Collected Data

Here we describe the data and the configuration of the various probes (for the description of the `Probe` concept see Section 4.1.1) we use in our data collection app. For every probe we will include a brief description, the relevant configuration and a sample data point. Additional descriptions of the fields will be included as needed.

If a field starts with `"ONE_WAY_HASH"` it means that the value is the SHA-1 hash of the original value. The hashing is used to protect privacy sensitive information. We also shortened the hash in the sample data to the first five digits. The remaining hash is indicated by "...".

We shortened some data points if the data structures where particularly complex. This is indicated by "...".

A.1. Archive

Interval	Duration
1 hour	-

Description

The `Archive` action saves all collected data to the external memory (the Secure Digital (SD)-Card if the phone provides one)

Please note that the archive function of FunF is not implemented as probe! We included this section to provide information about the archiving interval.

A.2. LabelProbe

Interval	Duration
30 min	-

Description

This probe adds the collected interaction data to the dataset. All finished interactions, that are not yet archived, are added to the dataset.

Sample Data


```
{
   "_id":15,
   "activated_ts":1390074710,
   "deactivated_ts":1390075274,
   "label_id":13,
   "label_text":"manuel",
   "timestamp":1.39007471E+9
}
```


A.3. CalendarProbe

Interval	Duration
24h	-

Description

This probe syncs all calendar events for the current day.

For a description of the fields see CalendarContract.Events[1], CalendarContract.Instances[2] and CalendarContract.Attendees[3] The keys in the data structure correspond to the constants defined in these interfaces.

Note: The calendar content provider is only available since Android version 4.0, "Ice Cream Sandwich". For all previous versions of Android this probe will report no data.

Sample Data


```
{
    "ATTENDEES": [],
    "_id": 544,
    "begin": 1395826200,
    "end": 1395829800,
    "event_id": 780,
    "hasAttendeeData": true,
    "timestamp": 1395826200,
    "title": "{\"ONE_WAY_HASH\":\"b1079...\"}"
}
```


A.4. WifiProbe

Interval	Duration
5 min	-

[1] http://developer.android.com/reference/android/provider/CalendarContract.Events.html

[2] http://developer.android.com/reference/android/provider/CalendarContract.Instances.html

[3] http://developer.android.com/reference/android/provider/CalendarContract.Attendees.html

Interval	Duration

Description

This probe records data about all WiFi networks currently in range.

For a description of the fields see ScanResult[4]. Note: the original timestamp of the `ScanResult` is preserved in the field `tsf`.

Sample Data


```
{
    "BSSID": "50:9f:27:e2:37:48",
    "SSID": "3WebCube3746",
    "capabilities": "[WPA2-PSK-CCMP+TKIP][WPS][ESS]",
    "distanceCm": -1,
    "distanceSdCm": -1,
    "frequency": 2462,
    "level": -86,
    "timestamp": 1389908808.917,
    "tsf": 1219760760436,
    "wifiSsid": {
        "octets": {
            "buf": [
            51,
            87,
            101,
            98,
            67,
            117,
            98,
            101,
            51,
```


[4] http://developer.android.com/reference/android/net/wifi/ScanResult.html

```
            55,
            52,
            54,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
            ],
            "count": 12
        }
    }
}
```


A.5. AccelerometerFeaturesProbe

Interval	Duration
2m	15s

Description

Absolute central moment[5], Standard deviation, Max deviation, PSD[6] per axis

The default frequency band edges for the PSD are defined as follows in the source code:

```
@Configurable
private double[] freqBandEdges = {0,1,3,6,10};
```

Sample Data


```
{
   "diffFrameSecs":0.000000,
   "numFrameSamples":73,
   "timestamp":1390403180.353338,
   "x":{
      "absoluteCentralMoment":0.20370956292137365,
      "maxDeviation":1.139627123260274,
      "mean":-1.017327929260274,
      "psdAcrossFrequencyBands":[
         10.888577705504122,
         13.104709450845048,
         12.874202781714772,
         6.484905577113416
      ],
      "standardDeviation":0.29490550854615283
   },
   "y":{
      "absoluteCentralMoment":0.0880777692249952,
      "maxDeviation":0.2918075219178071,
      "mean":5.484327521917807,
      "psdAcrossFrequencyBands":[
         0.24340133699038946,
```


[5] http://en.wikipedia.org/wiki/Central_moment
[6] http://en.wikipedia.org/wiki/Spectral_density#Power_spectral_density

```
            0.5365554433102747,
            1.5222832409664304,
            1.8080624684578268
         ],
         "standardDeviation":0.1151088059813906
      },
      "z":{
         "absoluteCentralMoment":0.19776735916682292,
         "maxDeviation":1.2028540753424668,
         "mean":8.279079924657534,
         "psdAcrossFrequencyBands":[
            0.2595266054708534,
            19.818497668580576,
            21.093091502129425,
            17.31872895277045
         ],
         "standardDeviation":0.33603985335287434
      }
}
```


A.6. AccelerometerSensorProbe

Interval	Duration
2m	15s

Description

This probe records the raw accelerometer sensor data as it is emitted by the Android framework.

For a description of the values see Sensor.TYPE_ACCELEROMETER[7]

[7] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ACCELEROMETER

Sample Data


```
{
   "accuracy":0,
   "timestamp":1390063524.900984440,
   "x":-0.28593445,
   "y":4.5034027,
   "z":8.959442
}
```


A.7. AccountsProbe

Interval	Duration
1 hour	-

Description

This probe reports the accounts that are stored on the phone.

Sample Data


```
{
   "name":"{\"ONE_WAY_HASH\":\"87431...\"}",
   "timestamp":1389908498.182,
   "type":"com.beeminder.beeminder"
}
```


A.8. ActivityProbe

Interval	Duration
2m	15s

Description

This probe reports an "activity level" based on the variance sum of the acceleration for the X, Y and Z axis (see Appendix A.6). The thresholds for the three levels (low, medium, high) are defined as follows:

```
if (varianceSum >= 10.0f) {
    data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_HIGH);
} else if (varianceSum < 10.0f && varianceSum > 3.0f) {
    data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_LOW);
} else {
    data.addProperty(ACTIVITY_LEVEL, ACTIVITY_LEVEL_NONE);
}
```

Sample Data

```
{
   "activityLevel":"high",
   "timestamp":1389893298.639
}
```

A.9. AndroidInfoProbe

Interval	Duration
1 hour	-

Description

This probe reports the build number, firmware version, and SDK level of the phone.

Sample Data

```
{
   "buildNumber":"occam-user 4.4.2 KOT49H 937116 release-keys",
   "firmwareVersion":"4.4.2",
   "sdk":19,
   "timestamp":1389925801.197
}
```

A.10. ApplicationsProbe

Interval	Duration
1 hour	-

Description

This probe reports all apps on the phone. This also includes currently *uninstalled* apps.

For a description of the fields see ApplicationInfo[8]

[8] http://developer.android.com/reference/android/content/pm/ApplicationInfo.html

Sample Data


```
{
   "compatibleWidthLimitDp":0,
   "dataDir":"/data/data/com.google.android.ears",
   "descriptionRes":0,
   "enabled":true,
   "enabledSetting":0,
   "flags":8961605,
   "icon":2130837525,
   "installLocation":-1,
   "installed":true,
   "installedTimestamp":null,
   "labelRes":2131230720,
   "largestWidthLimitDp":0,
   "logo":0,
   "nativeLibraryDir":"/data/app-lib/com.google.android.ears-1",
   "packageName":"com.google.android.ears",
   "processName":"com.google.android.ears",
   "publicSourceDir":"/system/app/GoogleEars.apk",
   "requiresSmallestWidthDp":0,
   "seinfo":"default",
   "sourceDir":"/system/app/GoogleEars.apk",
   "targetSdkVersion":17,
   "taskAffinity":"com.google.android.ears",
   "theme":0,
   "timestamp":1389893397.398,
   "uiOptions":0,
   "uid":10024
}
```


A.11. AudioFeaturesProbe

Interval	Duration

Interval	Duration
10min	1 min

Description

This probe calculates and reports PSD[9], Mel Frequency Cepstral Coefficients (MFCCs)[10], L1/L2/L_Infinity Norm[11] of the current audio stream recorded by the phone.

The standard frequency band edges for the PSD are defined as follows in the source code of FunF:

```
private static double[] FREQ_BANDEDGES = {50,250,500,1000,2000};
```

Sample Data


```
{
   "diffSecs":1.0,
   "l1Norm":22.984625,
   "l2Norm":28.85640093636072,
   "linfNorm":9.9498743710662,
   "mfccs":[
      79.89206072209798,
      7.446925849002253,
      2.639189540920471,
      2.290348643340287,
      2.5271848690643544,
      1.2880679340115282,
      1.422433897642004,
```


[9] http://en.wikipedia.org/wiki/Spectral_density#Power_spectral_density
[10] http://en.wikipedia.org/wiki/Mel-frequency_cepstrum
[11] http://en.wikipedia.org/wiki/Norm_(mathematics)

```
      2.092365094262467,
      2.7298077884122636,
      0.8105104856889456,
      -0.7024508446031763,
      -1.3647949765935876
   ],
   "psdAcrossFrequencyBands":[
      15226814.391497899,
      2280275.170662274,
      624375.5262948957,
      146985.84662805777
   ],
   "timestamp":1389893400.6
}
```


A.12. AudioMediaProbe

Interval	Duration
10 hours	-

Description

This probe reports meta-data about audio files stored on the phone.

For a description of the fields see MediaStore.Audio.AudioColumns[12]

Sample Data


```
{
   "_display_name":"hangout_dingtone.m4a",
```


[12] http://developer.android.com/reference/android/provider/MediaStore.Audio.AudioColumns.html

```
    "_id":"23",
    "_size":17294,
    "album":"Notifications",
    "album_id":1,
    "artist":"",
    "artist_id":1,
    "date_added":1360783791,
    "date_modified":1360784724,
    "duration":1045,
    "is_alarm":false,
    "is_music":false,
    "is_notification":true,
    "is_ringtone":false,
    "mime_type":"audio/mp4",
    "timestamp":1360784724,
    "title":"Join Hangout",
    "track":0,
    "year":0
}
```


A.13. BatteryProbe

Interval	Duration
5 min	-

Description

This probe reports statistics about the battery used by the phone.

For a description of the fields and their values see BatteryManager[13]

[13] http://developer.android.com/reference/android/os/BatteryManager.html

Sample Data


```
{
   "health":2,
   "icon-small":17302941,
   "invalid_charger":0,
   "level":91,
   "plugged":2,
   "present":true,
   "scale":100,
   "status":2,
   "technology":"Li-ion",
   "temperature":321,
   "timestamp":1389907380.173,
   "voltage":4245
}
```


A.14. BluetoothProbe

Interval	Duration
5 min	-

Description

This probe reports discovered Bluetooth devices in the vicinity.

For a description of the fields see the Android SDK about BluetoothDevice.ACTION_FOUND[14]

[14] http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#ACTION_FOUND

Sample Data

```
{
   "android.bluetooth.device.extra.CLASS":{
      "mClass":4063500
   },
   "android.bluetooth.device.extra.DEVICE":{
      "mAddress":"E0:2A:82:99:5C:AB"
   },
   "android.bluetooth.device.extra.NAME":"LISA-HP",
   "android.bluetooth.device.extra.RSSI":-82,
   "timestamp":1389907383.415
}
```

A.15. CallLogProbe

Interval	Duration
10 hours	-

Description

This probe reports all call logs (incoming, outgoing).

For the description fo the Fields see CallLog.Calls[15]

The hash of the phone number is calculated from last 10 digits of the actual phone number, where all non-digit character were removed.

Note: The same call can be logged multiple times. A duplicate check of the data is necessary.

[15] http://developer.android.com/reference/android/provider/CallLog.Calls.html

Sample Data


```
{
   "_id":1,
   "date":1360783930571,
   "duration":0,
   "name":"{\"ONE_WAY_HASH\":\"\"}",
   "number":"{\"ONE_WAY_HASH\":\"286dd...\"}",
   "numberlabel":"{\"ONE_WAY_HASH\":\"\"}",
   "numbertype":"{\"ONE_WAY_HASH\":\"\"}",
   "timestamp":1360783930.571,
   "type":1
}
```


A.16. CellTowerProbe

Interval	Duration
1h	-

Description

Data about nearby cellular towers. The value of `type` is either of TelephonyManager.PHONE_TYPE_CDMA[16], TelephonyManger.PHONE_TYPE_GSM[17] or TelephonyManager.PHONE_TYPE_NONE[18]

[16] http://developer.android.com/reference/android/telephony/TelephonyManager.html#PHONE_TYPE_CDMA

[17] http://developer.android.com/reference/android/telephony/TelephonyManager.html#PHONE_TYPE_GSM

[18] http://developer.android.com/reference/android/telephony/TelephonyManager.html#PHONE_TYPE_NONE

The remaining fields are from CdmaCellLocation[19] or GsmCellLocation[20], depending on the `type` of the cell tower.

Sample Data

```
{
   "cid":2445946,
   "lac":59952,
   "psc":45,
   "timestamp":1389893397.823,
   "type":1
}
```

A.17. ContactProbe

Interval	Duration
7 days	-

Description

This probe collects the contact data of the phone.

For a description of how the contact data in android is structured, see ContactsContract.Data[21]

Note that the phone number of the contact is transformed as follows:

1. all non-digit characters are removed

[19] http://developer.android.com/reference/android/telephony/cdma/CdmaCellLocation.html

[20] http://developer.android.com/reference/android/telephony/gsm/GsmCellLocation.html

[21] http://developer.android.com/reference/android/provider/ContactsContract.Data.html

2. only the last ten digits of the result of the previous step are saved

Sample Data


```
{
   "contactData":[
      {
         "_id":47,
         "data1":"{\"ONE_WAY_HASH\":\"b0dec...\"}",
         "data2":3,
         "data4":"{\"ONE_WAY_HASH\":\"\"}",
         "data_version":2,
         "is_primary":1,
         "is_super_primary":0,
         "mimetype":"vnd.android.cursor.item/email_v2",
         "raw_contact_id":7
      },
      ...
      {
         "_id":1994,
         "data3":0,
         "data4":14,
         "data5":0,
         "data_version":0,
         "is_primary":0,
         "is_super_primary":0,
         "mimetype":"vnd.android.cursor.item/vnd.googleplus.profile.comm",
         "raw_contact_id":281
      }
   ],
   "contact_id":7,
   "custom_ringtone":"{\"ONE_WAY_HASH\":\"\"}",
   "display_name":"{\"ONE_WAY_HASH\":\"eba1e...\"}",
   "in_visible_group":1,
   "last_time_contacted":0,
   "lookup":"3063i16e9885f0d92b055.3698ee%3At..scholz%40nonomy..net",
```

```
    "photo_id":0,
    "send_to_voicemail":0,
    "starred":0,
    "times_contacted":0,
    "timestamp":1389893398.94
}
```


A.18. GravitySensorProbe

Interval	Duration
1h	1min

Description

This probe collects the readings of the gravity sensor, if available.

See Sensor.TYPE_GRAVITY[22] for an explanation of the fields and their values.

Sample Data


```
{
    "accuracy":3,
    "timestamp":1390063525.329695378,
    "x":-0.39721924,
    "y":4.516546,
    "z":8.695598
}
```


[22] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_GRAVITY

A.19. GyroscopeSensorProbe

Interval	Duration
30 min	1 min

Description

This probe collects the data of the built-in gyroscope, if available.

See Sensor.TYPE_GYROSCOPE[23] for an explanation of the fields and their values

Sample Data

```
{
   "accuracy":3,
   "timestamp":1390063525.329695378,
   "x":-0.39721924,
   "y":4.516546,
   "z":8.695598
}
```

A.20. HardwareInfoProbe

Interval	Duration
7 days	-

[23] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_GYROSCOPE

Description

This probe reports hardware information of the used phone.

Sample Data


```
{
   "androidId":"d9c3057650b79129",
   "bluetoothMac":"10:68:3F:25:F8:91",
   "brand":"google",
   "deviceId":"353918052579394",
   "model":"Nexus 4",
   "timestamp":1390064288.154,
   "wifiMac":"f8:0c:f3:ff:d1:98"
}
```


A.21. LightSensorProbe

Interval	Duration
5 min	1 min

Description

This probe reports the values of the light sensor, if available.

See Sensor.TYPE_LIGHT[24] for a description of the fields and their values

[24] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_LIGHT

Sample Data


```
{
   "accuracy":0,
   "lux":8.0,
   "timestamp":1390063544.574729086
}
```


A.22. LinearAccelerationSensorProbe

Interval	Duration
1 h	1 min

Description

This probe reports the value of the linear acceleration sensor provided by the Android framework.

Sensor.TYPE_LINEAR_ACCELERATION[25] explains how "linear acceleration" is defined in context of the Android framework.

Sample Data


```
{
   "accuracy":3,
   "timestamp":1390063528.613216563,
   "x":0.84480834,
   "y":-1.0073481,
   "z":-0.5863557
}
```


[25] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_LINEAR_ACCELERATION

A.23. MagneticFieldSensorProbe

Interval	Duration
1 h	1 min

Description

This probe reports the values of the magnetic field sensor, if available.

See Sensor.TYPE_MAGNETIC_FIELD[26] for an explanation of the fields and their values

Sample Data

```
{
   "accuracy":3,
   "timestamp":1390063528.706722423,
   "x":23.278809,
   "y":-18.899536,
   "z":-29.579163
}
```

A.24. OrientationSensorProbe

Interval	Duration
3 min	15 sec

[26] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_MAGNETIC_FIELD

Description

This probe reports the orientation of the device, if the needed sensors are available.

See Sensor.TYPE_ORIENTATION[27] for an explanation of the fields and their values

Sample Data

```
{
   "accuracy":3,
   "azimuth":262.7669,
   "pitch":-47.684673,
   "roll":-12.561616,
   "timestamp":1390063530.688650074
}
```

A.25. PressureSensorProbe

Interval	Duration
1 h	1 min

Description

This probe collects the data of the pressure sensor, if available.

See Sensor.TYPE_PRESSURE[28] for an explanation of the fields and their values

[27] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ORIENTATION

[28] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_PRESSURE

Sample Data

```
{
   "accuracy":0,
   "pressure":963.0907,
   "timestamp":1390063540.148264020
}
```

A.26. ProcessStatisticsProbe

Interval	Duration
5 min	-

Description

This probe reports information regarding currently running processes and additionally parses various data points from the `/proc` file system

For additional information regarding the fields and their values use following table:

Top Level Key	Additonal Information
`RUNNING_PROCESS_INFO`	ActivityManager.RunningAppProcessInfo[29]
`RUNNING_PROCESS_MEMORY_INFO`	Debug.MemoryInfo[30]
`ERRORED_PROCESS_INFO`	ActivityManager.ProcessErrorStateInfo[31]
`CPU_LOAD`	see probe source code
`MEM_INFO`	see probe source code
`NET_DEV`	see probe source code

[29] http://developer.android.com/reference/android/app/ActivityManager.RunningAppProcessInfo.html

[30] http://developer.android.com/reference/android/os/Debug.MemoryInfo.html

[31] http://developer.android.com/reference/android/app/ActivityManager.ProcessErrorStateInfo.html

Sample Data

Note: Sample data is shortened due to the high number of fields reported by this probe.


```
{
   "CPU_LOAD":{
      "BootTime":1388689048,
      "ContextSwitch":758735866,
      "CpuTotalTime":14823899,
      "Processes":762939,
      "cpu":{
         "freq":13.53,
         "nice":0.32657614,
         "system":8.068511,
         "total":18.49101,
         "user":10.095923
      }
   },
   "ERRORED_PROCESS_INFO":null,
   "MEM_INFO":{
      "Active(anon):":681356,
      "Active(file):":481756,
      "Active:":1163112,
      "Buffers:":241264,
      "Cached:":595732,
      "HighFree:":8248,
      "HighTotal:":1286140,
      "Inactive(anon):":4520,
      "Inactive(file):":349560,
      "Inactive:":354080,
      "LowFree:":49920,
      "LowTotal:":592648,
      "MemFree:":58168,
      "MemTotal:":1878788,
      "Mlocked:":0,
```

```
      "SwapCached:":0,
      "SwapFree:":0,
      "SwapTotal:":0,
      "Unevictable:":1116
   },
   "NET_DEV":{
      "rev_rmnet1:":{
         "RxBytes":0,
         "RxPackets":0,
         "TxBytes":0,
         "TxPackets":0
      },
      "rev_rmnet5:":{
         "RxBytes":0,
         "RxPackets":0,
         "TxBytes":0,
         "TxPackets":0
      },
      "rmnet1:":{
         "RxBytes":0,
         "RxPackets":0,
         "TxBytes":0,
         "TxPackets":0
      },
      "rmnet6:":{
         "RxBytes":0,
         "RxPackets":0,
         "TxBytes":0,
         "TxPackets":0
      },
      "rmnet_usb2:":{
         "RxBytes":0,
         "RxPackets":0,
         "TxBytes":0,
         "TxPackets":0
      }
   },
```

```
   "RUNNING_PROCESS_INFO":[
      {
         "flags":4,
         "importance":100,
         "importanceReasonCode":0,
         "importanceReasonImportance":0,
         "importanceReasonPid":0,
         "lastTrimLevel":0,
         "lru":0,
         "pid":1349,
         "pkgList":[
            "com.sand.airdroid"
         ],
         "processName":"com.sand.airdroid",
         "uid":10116
      },
...
   ],
   "timestamp":1389908809.587
}
```


A.27. ProximitySensorProbe

Interval	Duration
5 min	15 sec

Description

This probe reporst the data of the proximity sensor, if available.

See Sensor.TYPE_PROXIMITY[32] for an explanation of the fields and their

[32] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_PROXIMITY

values

Sample Data

```
{
   "accuracy":3,
   "distance":5.000305,
   "timestamp":2779512955.522638
}
```

A.28. RotationVectorSensorProbe

Interval	Duration
5 min	15 sec

Description

This probe records the data provided by the rotation sensor, if available.

See Sensor.TYPE_ROTATION_VECTOR[33] for an explanation of the fields and their values

Sample Data

```
{
   "accuracy":3,
   "cosThetaOver2":0.68528765,
   "timestamp":1390063530.074733897,
   "xSinThetaOver2":0.35262012,
```

[33] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ROTATION_VECTOR

```
   "ySinThetaOver2":0.26114565,
   "zSinThetaOver2":0.58123547
}
```


A.29. RunningApplicationsProbe

Interval	Duration
-	-

Description

This probe emits the applications that are currently open or in use via a polling method.

The probe polls the currently active app. When a new app is active, the probe logs the previously running app, because now the probe can determine the duration the previous app was running.

Sample Data


```
{
   "duration":5.010,
   "taskInfo":{
      "baseIntent":{
         "mAction":"android.intent.action.MAIN",
         "mCategories":[
            "android.intent.category.HOME"
         ],
         "mComponent":{
            "mClass":"com.android.launcher2.Launcher",
            "mPackage":"com.android.launcher"
         },
```

```
            "mFlags":274726912
        },
        "id":1,
        "persistentId":1,
        "stackId":0
    },
    "timestamp":1389909641.554
}
```


A.30. ScreenProbe

Interval	Duration
-	-

Description

This probe records when the screen turns on and off.

Sample Data


```
{
    "screenOn":true,
    "timestamp":1389909641.55
}
```


A.31. ServicesProbe

Interval	Duration
1 hour	-

Description

This probe reports all running services.

For a description of the fields see ActivityManager.RunningServiceInfo[34]

Sample Data


```
{
   "activeSince":237800,
   "clientCount":1,
   "clientLabel":17040560,
   "clientPackage":"android",
   "crashCount":0,
   "flags":0,
   "foreground":false,
   "lastActivityTime":243362,
   "pid":1420,
   "process":"net.dinglisch.android.taskerm",
   "restarting":0,
   "service":{
      "mClass":"net.dinglisch.android.taskerm.MyAccessibilityService",
      "mPackage":"net.dinglisch.android.taskerm"
   },
   "started":false,
   "timestamp":1389907376.846,
   "uid":10094
}
```


A.32. SimpleLocationProbe

[34] http://developer.android.com/reference/android/app/ActivityManager.RunningServiceInfo.html

Interval	Duration
30 min	-

- goodEnoughAccuracy: 50

Description

This probe filters the set of locations reported by the GPS sensor for the most accurate location within a max wait time, ending early if it finds a location that has at most the goodEnoughAccuracy (default=80). For the description of accuracy, see Location.getAccuracy()[35]

For a description of the fields see Location[36]

Sample Data


```
{
   "mAccuracy":24.344,
   "mAltitude":0.0,
   "mBearing":0.0,
   "mElapsedRealtimeNanos":1204282882397069,
   "mExtras":{
      "networkLocationType":"wifi",
      "nlpVersion":2005,
      "noGPSLocation":{
         "mAccuracy":24.344,
         "mAltitude":0.0,
         "mBearing":0.0,
         "mElapsedRealtimeNanos":1204282882397069,
         "mExtras":{
            "networkLocationType":"wifi",
```


[35] http://developer.android.com/reference/android/location/Location.html#getAccuracy()

[36] http://developer.android.com/reference/android/location/Location.html

```
            "nlpVersion":2005,
            "travelState":"stationary"
         },
         "mHasAccuracy":true,
         "mHasAltitude":false,
         "mHasBearing":false,
         "mHasSpeed":false,
         "mIsFromMockProvider":false,
         "mLatitude":47.0744566,
         "mLongitude":15.4501974,
         "mProvider":"network",
         "mSpeed":0.0,
         "mTime":1389893330976
      },
      "travelState":"stationary"
   },
   "mHasAccuracy":true,
   "mHasAltitude":false,
   "mHasBearing":false,
   "mHasSpeed":false,
   "mIsFromMockProvider":false,
   "mLatitude":47.0744566,
   "mLongitude":15.4501974,
   "mProvider":"network",
   "mSpeed":0.0,
   "mTime":1389893330976,
   "timestamp":1389893331.327
}
```


A.33. SmsProbe

Interval	Duration
10 hours	-

Description

This probe reports the Short Messaging Service (SMS) log of the phone.

For the description of the data see Telephony.TextBasedSmsColumns[37]

Note: The probe records the same message multiple times, therefore a duplicate check is necessary.

Sample Data


```
{
   "address":"{\"ONE_WAY_HASH\":\"27cf6...\"}",
   "body":"{\"ONE_WAY_HASH\":\"a8426...\"}",
   "date":1361656254322,
   "locked":false,
   "person":"{\"ONE_WAY_HASH\":\"286dd...\"}",
   "protocol":0,
   "read":true,
   "reply_path_present":false,
   "service_center":"+436990005999",
   "status":-1,
   "subject":"{\"ONE_WAY_HASH\":\"\"}",
   "thread_id":7,
   "timestamp":1361656254.322,
   "type":1
}
```


A.34. TelephonyProbe

Interval	Duration
7 days	-

[37] https://developer.android.com/reference/android/provider/Telephony.TextBasedSmsColumns.html

Description

This probe records the mobile network information.

For a description of the fields see TelephonyManager[38]

Sample Data


```
{
   "callState":0,
   "deviceId":"353918052579394",
   "deviceSoftwareVersion":"06",
   "hassIccCard":true,
   "line1Number":"{\"ONE_WAY_HASH\":\"bad34...\"}",
   "networkCountryIso":"at",
   "networkOperator":"23203",
   "networkOperatorName":"tele.ring",
   "networkType":3,
   "phoneType":1,
   "simCountryIso":"at",
   "simOperator":"23207",
   "simOperatorName":"",
   "simSerialNumber":"89430700001271195285",
   "simState":5,
   "subscriberId":"232072503251308",
   "timestamp":1390066495.261,
   "voicemailAlphaTag":"Mailbox",
   "voicemailNumber":null
}
```


A.35. TemperatureSensorProbe

[38] http://developer.android.com/reference/android/telephony/TelephonyManager.html

Interval	Duration
20 min	10 sec

Description

This probe records the temperature. Implementation depends on the device, therefore the probe is not available on all devices.

Some devices will record temperature of battery, others the temperature of CPU or environment

see Sensor.TYPE_AMBIENT_TEMPERATURE[39]

A.36. TimeOffsetProbe

Interval	Duration
1 h	-

Description

This probe reports the current system time offset compared to a major Network Time Protocol (NTP) server in seconds.

The default NTP server is defined in the source code as follows:

```
@Configurable
private String host = "2.north-america.pool.ntp.org";
```

[39] http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_AMBIENT_TEMPERATURE

Sample Data


```
{
   "localTimeOffset":-9.901,
   "roundTripDelay":0.284,
   "timestamp":1389950854.598
}
```