Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

A Smart Situational Reminder

Semester Thesis

Steffen Schmidt

steffsch@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zurich

Supervisors:
Barbara Keller, Jara Uitto
Prof. Dr. Roger Wattenhofer

January 12, 2015

Acknowledgements

At this point, I would like to thank my supervisors Barbara Keller and Jara Uitto for their great support during the last 14 weeks.

Working on this thesis has been a great experience, not only, because I personally enjoy having a smart phone assistant now, who reminds me of important events depending on my environment, but also, because it was a great opportunity to dig deep into a very interesting area of engineering and computer science.

Abstract

Obliviousness is one of the few flaws in human nature, for which technology does not yet provide a rather satisfactory solution. Current reminder systems for mobile devices only offer reminders based on time or location, or only provide static information, which is shown to the user upon request, leaving the discovery of an almost forgotten event up to chance rather than reason. This thesis introduces the notion of reminders, which are based on a user's social surrounding. It also covers combinations of these reminders with location and time based solutions. In addition, it covers the establishment and use of a reminder community in order to enable reminder push notifications, whenever they are really necessary, offer the opportunity to share reminders between community members and embed the new functionalities in already existing reminder solutions and infrastructures.

Contents

Chapter 1

Introduction

1.1 Motivation

Remembering important events at the right time is a crucial skill for well-functioning social systems. Disruptions in one's private relationships can be caused by such simple things as a forgotten birthday and personal schedules can be completely messed up, if forgotten actions (such as shopping everyday goods in the local supermarket) have to be caught up upon later, even though they could have been conducted very easily at an earlier point in time (e.g. when the given individual was waiting for the tram in front of a supermarket, anyway).

Therefore, there is a demand for applications, which simplify the current solutions for setting up reminder assistance and triggering reminder notifications not only based on a configured time or location, but rather based on the social and physical environment at a given point in time.

1.2 Related Work

1.2.1 Related Android Applications

Google Now

Google Now[1] is Google's solution for a digital personal assistant. It offers a lot of information based on preconfigured settings upon opening the application, such as stock price monitoring, sports events etc. In addition, it features reminders with time and location triggers, so that, for example, the user can be reminded of his or her shopping duties, if he or she is near a supermarket. However, it neither allows direct reminder sharing via the Google Now application, nor can social surroundings be taken into account.

Geobells

Geobells[2] is an Android application that offers location based reminders. Not only does it send push notifications, but it can also be configured to take certain actions (e.g. muting the user's phone) upon reaching a certain location. However, the reminders set in Geobells are not visible in the user's Google Calendar and reminder sharing and the combination with any other reminder type is not possible.

Wifi Alarm

Wifi Alarm[3] offers its users the possibility to relate reminders to certain wifi networks and therefore offers a way to provide location based reminders without depending on the user's location coordinates. Yet, it does not feature a *wifi categorization* and therefore retrieving reminders on entering or leaving a certain group of wifis is only possible upon configuring reminders for all of them. In addition, wifi alarm reminders are also not visible in other calendars or reminder systems.

Friend Reminder

Friend reminder[4] features location and time based reminders, which can be connected to a friend's name. Despite its name, however, it does not feature any reminders based on a user's social surrounding, therefore basically only adding another piece of information for the user himself to the reminder, which could have been also done with the standard Google Calendar application.

IFTTT

IFTTT[5] enables users to easily program the phone to conduct actions once a certain event has occurred. Reminders based on location, time or events on the web can be easily configured. It also enables users to share the so called *recipes* which contain the if-then-else logics for these reminders. Again, reminders based on the user's social environment are not possible, and the large scope of possible configurations makes it hard for users with non-technical background to effectively use the offered functionalities.

Finde Meine Freunde

Finde meine Freunde[6] is not really a reminder application. However, it is the application closest to an social environment aware application, since user's have the possibility to view their friends' current position on a map and can get

notifications, when their friends reach certain locations. In addition, location based to-do lists can be shared among friends. Yet, it does not feature reminders based on the social surrounding of a user. In addition, like all other presented solutions it is not compatible with any other standard solution for calendars or reminders offered in Android.

1.2.2 Related Scientific Work

In general, scientific work on situational reminders which has been conducted so far focuses on two main fields.

The first one is to build systems that help people, suffering from diseases like Alzheimer, to finish an activity successfully once they started it [7] or to build reminder systems that help nursing elderly people [8].

The second main focus lies on using location information to remind users of certain events. This work yielded outcomes that are very similar to the commercial android applications, presented in Section 1.2.1. Some present enhanced solutions, e.g. *iReminder* [9], which pushes reminders based on forecasted future user locations or *RemindU* [10], which offers location based reminders and enables cloud servers to trigger these reminders, while keeping the user location secret to any third party (servers, etc.). In addition, *NAMA* [11], *MemoClip* [12] and *A location based task reminder for moblie users* [13] present enhanced solutions for location based reminders.

An exception is *CybReminder* [14], which has already been developed in 2000 and, in addition to location based reminders, also offers reminders based on user co-location and enables reminder sharing. However, the system has been built over a decade ago, prior to the smart phone era. Reminders need to be set using a computer client and are pushed to users via SMS, printer or e-mail. Therefore it is no a solution for today's smart phone systems, since no Android or iOS versions of *CybReminder* exist. In addition, nowadays, smart phone operating systems provide means that enable developers to create reminder applications, which are way more user friendly than *CybReminder*. However, one needs to consider, that these means were not available at the time, when *CybReminder* was created.

1.3 The Goal of this Thesis

As described above, applications and approaches for situational reminders have already been deployed. However, the presented solutions for Android are all limited to reminders based on physical surroundings and time and therefore, social environments are not taken into account, when deciding whether or not to remind the user. Using available information about friends surrounding the

user can possibly lead to a reduced number of false positives and empower the application to remind users of non-static events, that cannot be related to a certain location, but rather to the proximity of a friend or peer. In addition, none of the presented Android applications offers the opportunity to actually cooperate with the user's standard calendar, but rather establishes an individual reminder environment of its own, leaving space for duplicates and inefficiencies. Further reminders in current Android solutions cannot be shared with other members of a *reminder community*, who seek to help each other by sharing important reminders (with *Finde meine Freunde* being an exception here). This thesis is an approach to combine the described, already known reminder methods for Android with so far neglected social components, to enable a reminder sharing community and to offer an environment designed to cooperate with calendar solutions, which are already in use.

Chapter 2

The SmartR Android Application

2.1 Registration

After having installed the SmartR Android Application, users are prompted to choose a SmartR user name (Figure 2.1). The user then has to give SmartR the right to manage her calendar information, in order to synchronize every reminder set via SmartR with the Google Calendar of the Google account, with which her Android smart phone is connected. If two or more Google accounts are connected with the respective android phone, the user has to choose, which one she would like to use in combination with SmartR (Figure 2.4). *Important*: Please note that SmartR always uses the calendar on the user's Google account, labelled as *primary*, which is also the default label for the first calendar used on a Google Account. In case a user does not have a calendar with this label, she needs to create one in her Google account. However, this should only happen rarely, since very few users decide to alter the standard configuration as far as calendar names are concerned.

In case, the chosen user name is already taken, a pop-up window appears informing the user about this problem. Users can then change their selected user name by pressing *Create/update Account* (Figure 2.3).

In case, the Google Play Services, which are needed for the location retrieval and geofencing functionalities of the SmartR Android application, are not up to date, a dialogue appears, prompting the user to update the phone's play service application (Figure 2.4). After having conducted this step, the whole registration procedure has to be repeated. Again, this is achieved by pressing *Create/update Account*.

After the user registered without any complaint dialogues appearing on your screen, the user profile has been set-up and the application can be used. Via the *Go to main menu* option, users are led to the main user interface of SmartR.

Users are not able to go beyond the registration process interface (e.g. open

the main user interface), as long as they are not registered.

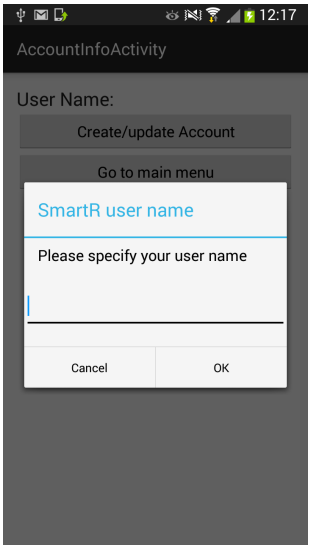

Figure 2.1: Dialogue to specify the SmartR Username

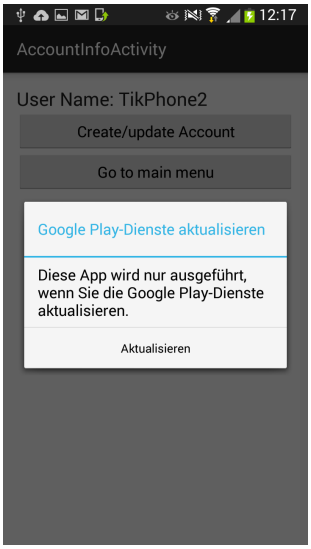

Figure 2.2: Prompt to update Google play services with direct link

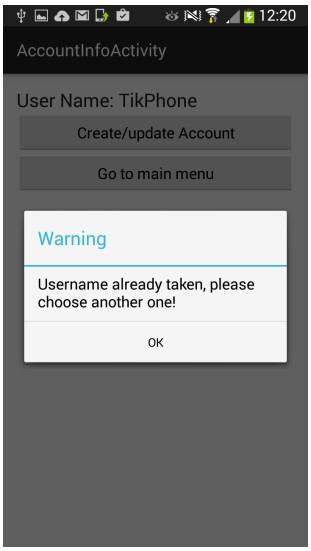

Figure 2.3: Notification of user name unavailability

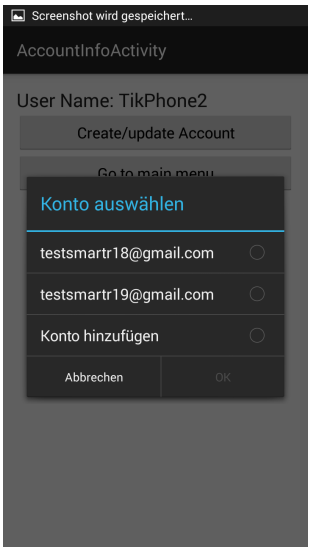

Figure 2.4: Prompt to choose one of the Google account

2.2 The Main User Interface

In the main user interface (Figure 2.5), which is shown upon opening the application (if the user is already registered), the user is presented with several options that represent the core functionalities (adding and editing reminders as well as retrieving reminders) and auxiliary configuration options, which are needed for the evaluation of a situation (e.g. friendship relations, location, wifi-options). The following sections will give an introduction on how to use the main functionalities of SmartR and how they can be accessed from the main user interface.

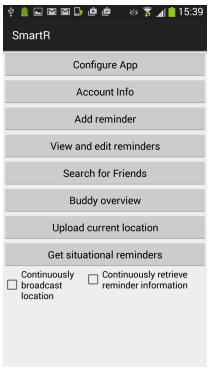

Figure 2.5: SmartR main user interface

2.3 Managing Friendships

Since SmartR offers reminders based on the social surrounding in addition to location, POI and wifi-zone based reminders, it makes sense to connect to some other members (e.g. family, friends) of the SmartR community in order to use this functionality properly.

This can be done by choosing the *Search for Friends* option in the main user interface. This allows users to search for a friend by entering her respective SmartR user name. Search results are shown below the input upon their receipt (Figure 2.6). Upon clicking onto a found name, a dialogue appears asking if the user would like to add this SmartR community member as a friend. If a friend request has already been sent by the searching user or the user is already friends with this member, she is told so and no further actions can be taken. Existing friendships and friend requests can be edited using the *Buddy overview* (Figure 2.8), at which we will have a closer look at in a moment.

In case no friendship exists, no friend request has been sent by the user yet, and the user decides to request or answer a friendship, a window with options to specify and characterize the friendship from your side opens (Figure 2.7). The characterisation parameters are the *buddy level*, which specifies depth of

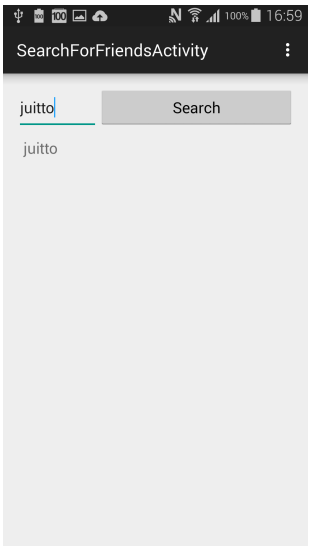

Figure 2.6: User interface for searching members of the SmartR community

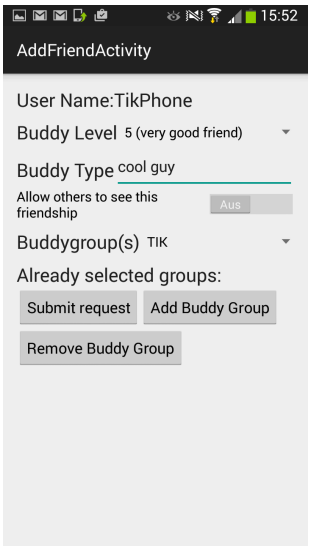

Figure 2.7: User interface for specifying friendship characteristics

the friendship, the *buddy type*, which is an informal textual description of the nature of your friendship and the number and name of different *buddy groups*, which embed the friendship into a larger social context. In addition, the user has the possibility to specify whether the given friendship is visible to other SmartR members, however, this parameter is not yet quite meaningful, since currently friendships are not visible to anyone but the two concerned friends in SmartR.

Users can add buddy groups by choosing the *Add Group* option, which appears, when clicking on the *Buddy Group* Spinner. A dialogue appears prompting to give a name to the new buddy group (Figure 2.9). Upon clicking *OK*, the buddy group is created and added to your local memory. Buddy group memberships are saved in the SmartR MYSQL database, which is discussed later on in this thesis.

After submitting a friend request, the given request can be monitored and edited in the *buddy overview* (Figure 2.8).

2.4 Managing and Editing Wifi Zones

SmartR also features reminders which are based on wifi zones, meaning that those reminders are pushed to the user, once she enters or leaves a zone, in

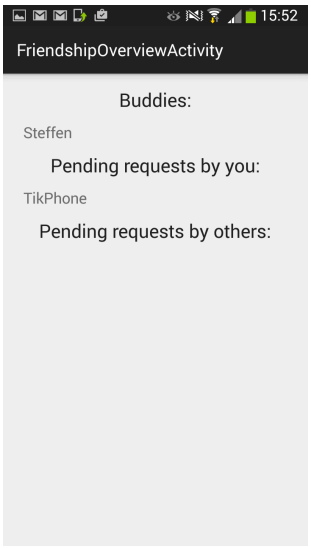

Figure 2.8: Overview over friendships and friend requests

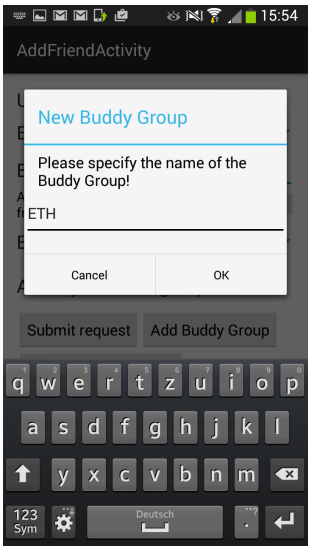

Figure 2.9: Dialogue for creating a new buddy group

which a wifi network with the specified SSID is available. In order to enable this service, wifis have to be added to a list of relevant wifis and associated to a wifi group. This is done by choosing the *Configure App* option in the main user interface and then pressing *Stationary Wifis* in the newly opened window (Figure 2.10). The user then gets presented a dialogue, which offers her the possibility to choose a wifi network SSID from the ones, which are within her reach or to type the SSID of the wifi network she would like to add, manually (Figure 2.11).

In case the user decides to take the *Choose* option, she can select the desired wifi network SSID from a spinner and then has to assign it to a wifi group (Figure 2.13). In case the *Type* option is chosen, the user types the wifi network SSID manually and then, allocates a wifi group to this SSID (Figure 2.12). Pressing the *Add Wifi to list* button will add this SSID-wifi group pair to the list of known pairs and can be chosen later, when creating a reminder based on wifi zones.

In addition, users have the possibility to view all existing SSID-wifi category pairs via the *Show Wifi groups* button and to clear all existing pairs with the *RESET* button.

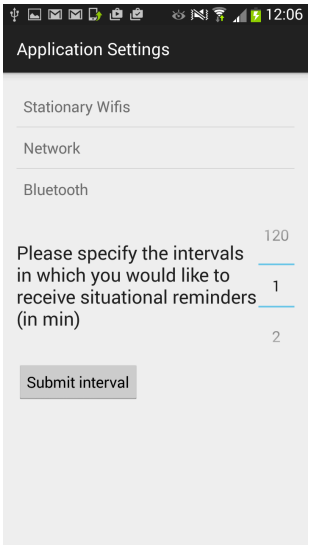

Figure 2.10: App configuration user interface

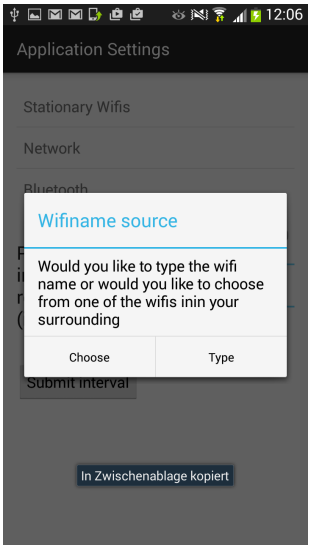

Figure 2.11: User can choose different options for adding wifis

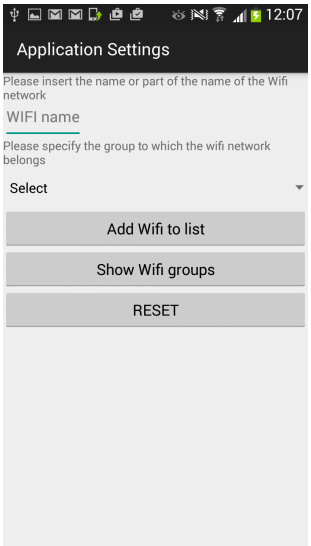

Figure 2.12: Wifi setting user interface *Choose* option

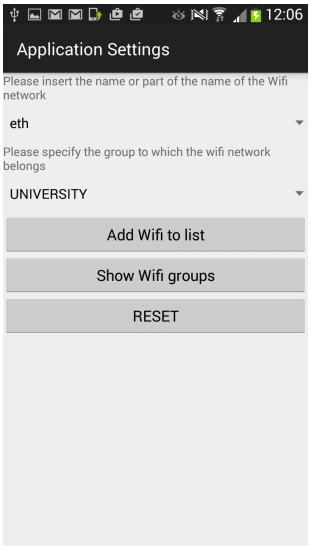

Figure 2.13: Wifi setting user interface *Type* option

2.5 Adding Reminders

Pressing the *Add reminder* button in the SmartR main user interface, opens a dialogue, asking the user which type of reminder she would like to add. The user has the possibility to choose between reminders based on physical surrounding (i.e. POI (point of interest) based reminders), social surrounding, wifi zones and geofences (i.e. geographic locations with a specified radius around them). Please consult Figure 2.14 for a screen shot of the reminder option dialogue.

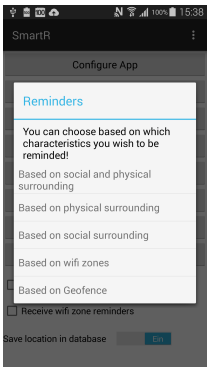

Figure 2.14: Reminder options

2.5.1 Reminders Based on Physical Surrounding

Reminders based on physical surrounding (Figure 2.15) feature the possibility to specify a description and a venue, which are not directly used to evaluate whether a reminder should be triggered or not and therefore can be left blank (which of course does not make too much sense in the case of the description). In addition, there is the possibility to specify a radius, which will be used to search for relevant POIs around the user. This radius marks the border of the circle around the user, in which POIs should be searched. After this, an expiration date or a time frame (start and end date) for the given reminder can be specified (see Figure 2.16 for the a user interface screen shot, in case the *time frame* option was chosen). Based on which case is chosen, the reminder is only evaluated, if the evaluation takes place before the expiration date, or within the specified time frame. Users then have to choose a reminder type, which specifies the kind of POI that should trigger a reminder. At first, a general category (e.g. public transport, daily goods shopping, etc.) is chosen. Following this, a second spinner (field from which to choose an option), appears with more specific options based on the more general type selection. The chosen detailed type will be used in the reminder evaluation process to check for relevant POIs around the user's location.

The reminder could now be submitted, unless the user would like to add

a *social component*. Choosing this option (Figure 2.17) enables the reminder to also be aware of a user's social surrounding and take it into account, when SmartR assesses whether or not to trigger a reminder notification. The user has the opportunity to either specify one or several *buddy groups* and *buddy level*, add one or more *buddies* directly or to do both. These parameters define, which social surrounding is necessary to trigger a push notification for a given reminder. If the user chooses to specify a *buddy group*, she needs to also indicate a *buddy level*. At the time of evaluation, all members of a *buddy group* specified in the evaluated reminder, who have a lower *buddy level* than the one stated in the evaluated reminder are not considered for assessing a user's social surrounding. *Buddies* that have been added directly by the user, are always taken into consideration at evaluation time, so there is no need to specify a *buddy level*, if a user exclusively chooses this option. Additionally, users can set a *social reminder radius*. This radius specifies the area around a user, in which friends have to be located in order to be seen as relevant for triggering the reminder. The default value of this field is set to 100 meters.

Once a reminder has been submitted, users have the possibility to share the reminder with some of their friends, if they wish. They can choose the friends, they would like to share the reminder with, from the list of all SmartR friends and add them to the list of friends, who are sent the reminder (Figure 2.18).

Important: In the reminder setting interface, every *buddy* or *buddy group*, which has been chosen, is added to the list of parameters which are evaluated later, no matter if the *Add Buddy Group* or *Add another buddy* buttons have been pressed. When sharing reminders, you have to explicitly add friends to the *Sharing buddies so far* list by pressing the *Add Another Buddy* button.

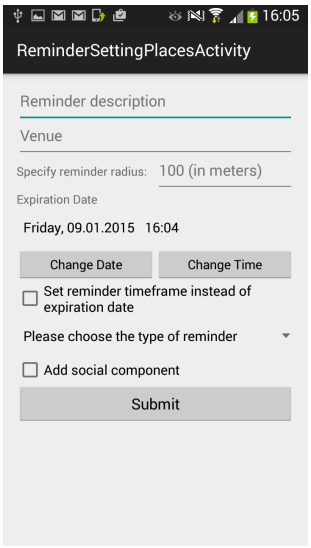

Figure 2.15: User interface for setting physical surrounding reminders

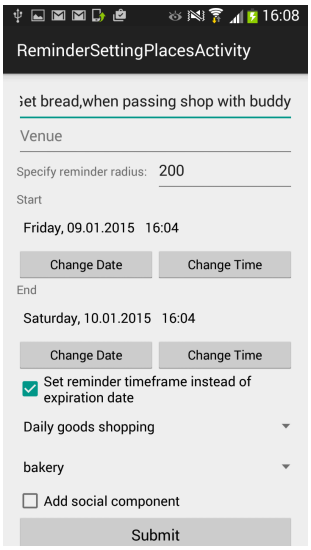

Figure 2.16: User interface for setting physical surrounding reminders for time frame option

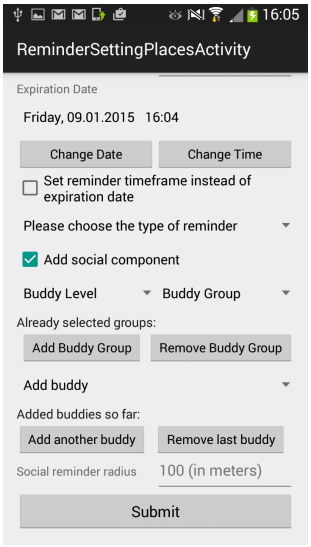

Figure 2.17: User interface for setting physical surrounding reminders with social component

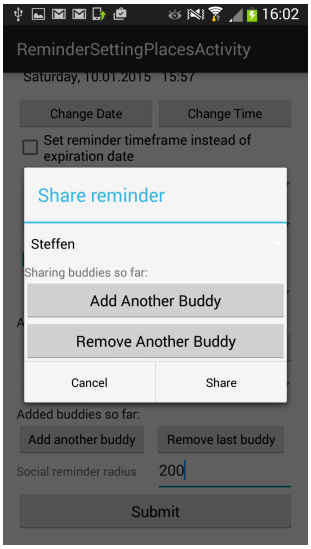

Figure 2.18: Reminder sharing dialogue

2.5.2 Reminders Based on Social Surrounding

The procedure of adding reminders based on a user's social surrounding (for the user interface, please see Figure 2.19) is quite similar to the one for those based on physical surrounding with the *social component* check box being checked. The user interface, again, features fields for a reminder description, a venue description, and the possibility to set a time frame or expiration date for the reminder. Further, the configuration possibilities for *buddy groups*, the *buddy level*, the *social reminder radius* and *buddies* are the same as for the physical environment based reminders with a *social component*. In addition, users have the same reminder sharing opportunities as with physical surrounding reminders.

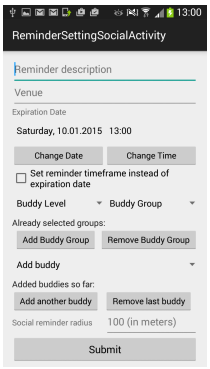

Figure 2.19: Social reminder user interface

2.5.3 Reminders Based on Wifi Zones

For reminders based on wifi zones (Figure 2.20), the configuration process, again, is very similar to that of physical surrounding reminders, as far as *description*, *venue*, *time settings* and adding a *social component* is concerned.

However, users now have the possibility to specify a wifi group and whether they would like to get reminded of an event when they leave or enter a wifi network, which belongs to this group. Furthermore, upon choosing a wifi group, a check box appears, giving users the opportunity to indicate whether they would like not to get reminded, when they reach any wifi from a group, but only if they enter one wifi network of this group (Figure 2.21 and Figure 2.22). This network can be chosen from a spinner, which appears, once the *Would you like to further specify your wifi zone?* checkbox is checked.

In case users enter the wifi zone reminder setting user interface for the first time, a dialogue appears, informing users about the necessity to configure the list of wifis known to the SmartR application and offering the opportunity to open the application settings, in order to directly configure some SSID-wifi group pairs (Figure 2.23).

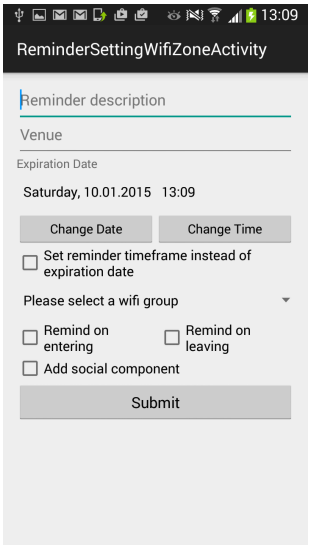

Figure 2.20: Wifi zone reminder user interface

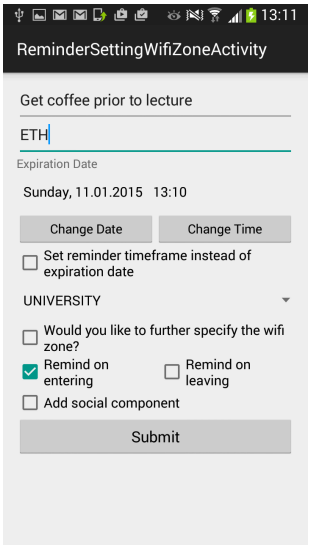

Figure 2.21: Wifi zone reminder filled in for category only

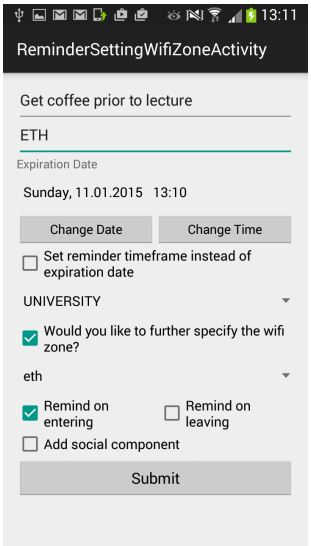

Figure 2.22: Wifi zone reminder filled in for specific wifi network

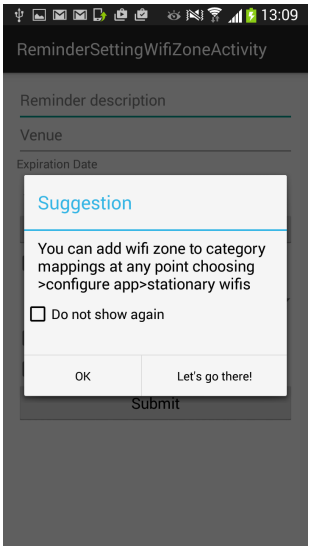

Figure 2.23: Info dialogue for wifi configuration

2.5.4 Reminders Based on Geofences (Geographic Location)

Again, the configuration process regarding *description*, *time settings* and adding a *social component* are similar to the previously presented modes of reminders.

Reminders based on geofences additionally enable the user to specify a location on the shown map and a radius around this location. The so defined area is used to trigger reminders upon users entering or leaving it, based on which option the user decided to choose in the user interface (Figure 2.24).

The shown map is initialized with the location that was last uploaded to the SmartR database by the user's phone. In case no locations have been uploaded yet, the map is initialized to show the default value, which is the location of ETH. User's can easily navigate to their current location by tapping the button in the upper right corner of the map.

When users enter the configuration interface for the first time, they are presented information, about which location settings should be enabled on their phone and offered to either return (*No, thanks*), alter their settings (*Open settings*) or indicate that they do not need (or want) to change anything and go on (*That's already the case)* (Figure 2.25).

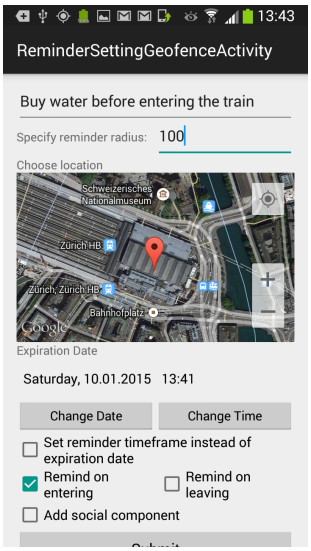

Figure 2.24: Geofence based reminder user interface

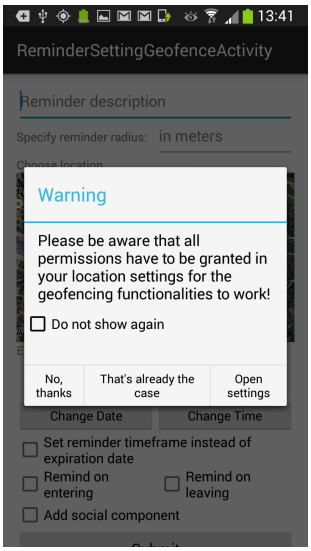

Figure 2.25: Information about appropriate location settings

2.6 Managing Reminders

After having added a reminder, users can view and edit their reminders by simply choosing the *View and edit reminders* option in the main user interface. Hereafter, a list of all reminders, whose start times are still in the future, is presented (Figure 2.26) and clicking on one of the reminder titles, will open the respective reminder configuration environment with the previously chosen configuration. Users can then simply edit the previously made configurations, update and re-share their reminders, if they wish to do so.

In addition, all reminders, which were added, edited or shared via the SmartR Android application are also visible in any Google Calendar application on the user's smart phone and also on the internet, since SmartR uses Google Calendar as container for its reminder data. Therefore, all changes made by the user to an event via one of these applications, directly influence the evaluation of the altered reminder event, once a SmartR evaluation process is initialized. Reminders can also be shared via these interfaces, if the user prefers to do so. Figure 2.27 shows the interface of the *S Planner* application, which comes pre-installed with the phone's Android OS. It can be seen, that the reminders set with SmartR are also visible in this calendar environment.

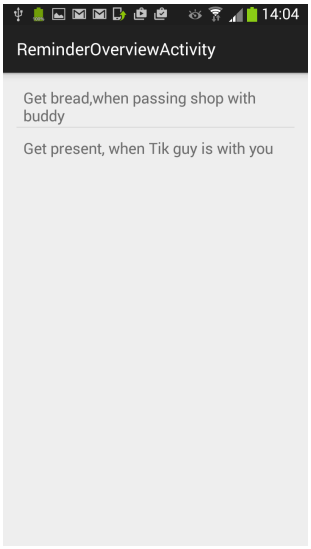

Figure 2.26: SmartR reminder overview

Figure 2.27: Events created with SmartR shown in the S Planner app

2.7 Reminder Retrieval

Now that the user has added and possibly edited her reminders, it stays to explain how she can control and initiate the process of evaluating reminders. This procedure differs a bit between different kinds of reminders. The wifi zone and geofence based reminders are always triggered, when the specified conditions are met and the application is running in the fore- or background, even if they contain a social component.

In case of the social reminders and physical surrounding reminders, the user has the choice to either directly check for current situational reminders by pressing the *Get situational reminders* button or to be presented with a reminder when the app deems appropriate by checking the *Continuously retrieve reminder information* check box in SmartR's main user interface (Figure 2.5). For the second option, the SmartR Android application regularly polls the server back-bone for its analysis of the current situation (more on this later).

The user can set the polling interval by pressing the *Configure App* button, choose an interval on the shown spinner and then pressing the *Submit interval* button (Figure 2.28). The default polling interval is set to one minute.

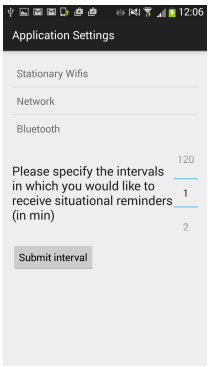

Figure 2.28: SmartR app configuration interface; use spinner to alter polling interval

2.8 Location Upload

Users have three possibilities to upload their current location onto the SmartR database. First of all, they can choose the *Upload current location* option in the main user interface, which triggers the upload of the current location. Secondly, the *Continuously broadcast location* check box can be checked, which tells the SmartR Android application to upload the current geographic position every minute. Last but not least, if the *Continuously retrieve reminder information* check box is checked, with every call to the server back-bone for a reminder

evaluation, the current location is also added to the SmartR MYSQL database.

Chapter 3

Reminder Architecture

The key part of SmartR are the reminders, which can be set using the SmartR Android application. SmartR offers four basic reminder options: Social environment reminders, physical environment reminders, wifi-zone reminders and geofencing reminders. SmartR enables users to combine any of the non-social environment reminders with a social environment reminder.

3.1 General/ Shared Properties

3.1.1 Reminder Parameters Shared by All Reminder Categories

Even though the presented reminder options are evaluated quite differently, they all share certain parameters, which are enlisted and described in Table 3.1.

Reminders are stored as a Google Calendar Event. Parameters, which are not directly supported by these Events, are saved in a JSON-String in the *description* field of the respective Event. This makes it possible to create a key-value mappings upon receiving a request to analyse an event.

It is important to notice that the *description* field in the SmartR reminder configuration user interfaces equals the *summary* or *title* fields in the Google Calendar Event, since the *description* field is already used to store the SmartR reminder parameters for future evaluations.

3.1.2 General Remarks on Reminder Retrieval

Before the actual evaluation process of the relation between the current user environment and a previously configured reminder takes place, it is verified that either the expiration date of a reminder has not been passed or that the current time lies in the time frame, which the user configured upon submitting the reminder. Which of these constraints are assessed depends on whether or not the *time frame set* field is set to *true*.

Reminder category	Contains the category description of a reminder (e.g. geofencing, wifi_zone, etc.), to enable a correct decision, on what to evaluate
Time frame set	Contains true or false based on whether the reminder should be evaluated only within a certain time frame or only before an expiration date specified by the user
Social component	Contains true or false based on whether the reminder is also evaluated based on the user's social environment in addition to another reminder modus; please note that for reminders solely based on the social environment, this field is set to false, since the need for social surrounding evaluation is already indicated by the *reminder category* in this case

Table 3.1: Reminder parameters shared by all reminder categories

3.2 Reminders Based on Social or Physical Surrounding

The triggering process for evaluating reminders based on social or physical surrounding starts on the user's smart phone by either turning on the polling functionality with a certain polling frequency or choosing to directly retrieve all currently relevant situational reminders. Since the evaluation of these reminders needs data from the SmartR MYSQL database anyway, which is accessed through the SmartR server back-bone, it makes sense to perform the complete reminder evaluation on the server in order to minimize computational effort (and thereby battery usage) on the mobile device.

These two reminder categories also share the *last reminder* parameter, for the detailed specification please see Table 3.2.

The *last reminder* parameter is used to be able to compare the current time, to the last time, that this reminder has been triggered. This evaluation is necessary, since users would receive the same reminder over and over again, if their social or physical surrounding stays the same and they have checked the *Continuously retrieve reminder information* check box in the SmartR application's main user interface (Figure 2.5). At the moment, a reminder notification for social or physical surrounding reminders is only triggered, if a given reminder has not been triggered for the last two hours. If the reminder has the *last reminder* parameter set to -1 (value set, when initializing a reminder), the SmartR evaluation process senses that the given reminder has never been triggered before and continues with the evaluation process of the user's surrounding.

Last reminder	Contains the unix timestamp of the last time, that the respective reminder was triggered. It is initialized with -1, when the reminder is created.

Table 3.2: Social and physical surrounding shared reminder parameters

3.2.1 Social Reminders

The social environment visible to SmartR consists of a user's friends who regularly upload their location. The data structure described in Section 5.1.1 makes it possible to quickly assess the user's social environment.

Social Reminder Parameters

Since the evaluation of social reminders needs internet connectivity, they are directly uploaded into the Google Calendar cloud and are only visible in the local calendar instance upon synchronization (which usually happens quickly and automatically). Parameters exclusively used by social reminders, and their respective use, are listed in Table 3.3.

Buddy groups	JSON-String containing group(s) of friends relevant for this reminder ; a reminder is triggered, if one of the group members with a sufficient buddy level is near
Buddies	JSON-String containing user ids of friends relevant for this reminder; a reminder is triggered, if one of specified friends is near
Buddy level	Scale from one to five; reminders are only triggered, if the buddy level of surrounding buddy group members are higher than this level
Social radius	Radius, defining the area around the user's geographic position, in which friends need to be located in order to be seen as near enough to trigger a reminder. The default value is one hundred meters.

Table 3.3: Social reminder parameters

Social Reminder Evaluation

Upon an evaluation request, the SmartR Server Backbone firstly assembles a list of all friends that are relevant for triggering the reminder based on the submitted list of *buddies*, *buddy groups* and the *buddy level*. Friends that have been directly added to the list of *buddies* by the user for a given reminder, are always deemed as relevant. Friends, who belong to a *buddy group*, which is contained in the list of *buddy groups* for a given reminder, but whose friend level with the requester is not above the one asked for by in the reminder's *buddy level* parameter, are not added to the list of relevant friends.

Thereafter, it is checked whether any of the relevant friends are within a certain area of the requesting user. The *social radius* marks the edges of this area. It is important to note that the *social radius* is actually not a radius, but

rather indicates the edge length of a square, in which friends need to be located in order to be considered close enough to trigger a social reminder. This is due to the chosen data structure for storing user locations as described in Section 5.1.1. The process of retrieving surrounding friends is described in great detail in Section 5.1.2.

Once a relevant friend has been spotted in the so marked surrounding area, it is verified that the last location update by the requesting user and the last location update by the relevant friend are at maximum five minutes apart, to avoid triggering an outdated reminder, which would have been relevant and correct some hours ago, when the relevant friend last updated her location. Since the user location is always updated, when calling reminder evaluation requests on the server, and it is also checked (for resiliency reasons) that the last user location update is no older than five minutes, before even starting the evaluation, this guarantees to trigger no false positives from time shift effects.

If there are relevant friends left after this filtering process, the user receives a push notification which contains the names of the nearby SmartR friends and the title of the reminder.

Use Cases

Possible use cases of social reminders range from being reminded that the friend a user is about to meet has her birthday today, to being reminded that someone the user still owes money is currently around, so that she has the possibility to pay back her debt.

3.2.2 Physical Surrounding Reminders

Another important reminder type are physical surrounding reminders. These reminders are triggered once a certain type of place is spotted near the user. The search of nearby places is conducted via a nearby search, which is featured by the Google Places API.

Google Places API

The Google Places API [15] enables developers to make use of Google's database on point of interests (POIs). These POIs are categorised according to Google's list of supported types for searches conducted with the Google Places API[16]. These categories equal the possible *reminder types* that the user can choose from in the SmartR Android application, when configuring a reminder based on the physical environment. SmartR uses the *nearby search* and *place details* requests possible via this API to retrieve information on relevant POIs around the user and to retrieve further information, once a POI has been chosen to be eligible to

Reminder radius	Specifies the radius around a user's last location, in which a nearby search shall be conducted via the Google Places API
Upper reminder type number	Specifies the position of the chosen general reminder type category, within the list of general reminder types, visible to the user in the physical reminder user interface; used to show the previously configured general reminder type, when the user opens a reminder for editing
Detailed reminder type number	Specifies the position of the chosen detailed reminder type category, within the list of detailed reminder types, visible to the user in the physical reminder user interface; used to show the previously configured detailed reminder type, when the user opens a reminder for editing
Reminder type	Specifies the detailed reminder type, for which a nearby search is conducted; the reminder type represents a value from the list of supported place types in the Google Places API [15]

Table 3.4: Physical reminder parameters

trigger a push notification, indicating the respective reminder to the user. This information comes in JSON format and is parsed in the SmartR server back-bone in the course of the evaluation process.

Physical Reminder Parameters

The physical reminder parameters are mainly used to conduct nearby searches for the specified reminder type and to reconstruct the reminder setting view, when the user wishes to edit an existing physical reminder. An overview can be found in Table 3.4.

Physical Reminder Evaluation

First of all, a Google Places *nearby search* is conducted for the user's last uploaded location and the radius specified by the user (its default value in the SmartR Android application is set to 100 meters), upon receiving a request to evaluate a physical reminder. If the search delivers a set of possible results, it is checked, whether the results contain an *opening hours* field, which features a field *open now*, specifying whether or not a place is currently open. Unfortunately, not all POIs, for which information on the opening hours would be interesting and necessary to save the user from false positive reminder notifications, offer this information. Therefore, at the moment, being close to these POIs might trigger reminder notifications, even if they are currently closed.

A possible solution for this would be to assume these POIs to be open, if the majority of places, discovered during the nearby search, who feature an *open now* field have *true* assigned to this field. Otherwise, these results would be assumed to be closed. Another possible solution would be to choose the opening hours of the place with the most customer friendly (e.g. longest opening hours), meaning that if one place has set the *open now* field to true, all places without a *opening hours* field are assumed to be open, leading to a minimization of false negatives. Of course the opposite decision of using the most customer unfriendly opening hour policy would be possible, leading to a minimization of false positives. However, this filter is not yet implemented, since, for now, the given solution suffices for most use cases at the moment.

The opening hour evaluation is done by iterating through the result set, returned by the nearby search in the order of the returned ranking and stopped as soon as an open place is found.

After this, if there has been at least one result that has been found to be open, the name and place id of the place are retrieved and a Google Places *place details* request is performed, to retrieve website and address data for the relevant place. The user is then sent a push notification containing the event title of the triggered reminder, the name of the relevant place and its address.

Use Cases

Physical surrounding reminders can be used to remind users of their shopping duties, whenever they pass a store, which fits these duties. In addition, these reminders are rather convenient in foreign environments. Users could for example specify that they need to draw some cash and wish to receive a reminder, if an ATM is located in a certain area around them. As mentioned above, this reminder will contain the address of the ATM, so that users can easily find it. This makes the presented use case very attractive in foreign places, where the user has no knowledge about the location of important infrastructure components.

On in	Contains whether a reminder is triggered on entering an area (if set to true) or on leaving it (if set to false).

Table 3.5: Wifi zone and geofence shared reminder parameters

3.3 Reminders Based on Wifi Zones and Geofences

These two reminder options do not need any server support, when they are not combined with a social surrounding reminder functionality. Therefore, they are firstly stored locally in the user's Google Calendar, and uploaded onto the cloud in the next synchronization cycle between the users smart phone calendar and the Google Calendar cloud. As in the case of social and physical surrounding reminders, the parameters that can not be saved in fields of a Google Calendar Event are stored using a JSON string in the *description* field of the respective Google Calendar Event. These two types again share a parameter for evaluating the necessity to trigger a reminder notification to the user. Further specifications on this parameters can be found in Table 3.5.

3.3.1 Wifi Zone Reminders

Wifi zone reminders are triggered, if a user enters or leaves the area covered by a specific wifi network. Users can specify whether reminders should depend on a single wifi network or whether they would like to get reminded, when entering or leaving a network from a group of networks.

Wifi Zone Reminder Parameters

In addition to the already mentioned shared parameters, reminders based on wifi zones feature three further parameters to correctly evaluate whether or not a reminder should be triggered and to enable a smooth reminder editing process via SmartR for the user. For further documentation on these parameters, please consult Table 3.6.

Wifi Zone Reminder Evaluation

SmartR periodically scans the environment for available wifi networks, when it is running. In order to sense, whether there are wifi networks, which have been added in the time between two scans, the SmartR Android application updates a list of all available wifis everytime a wifi scan result is returned. Upon receiving a new result of a wifi scan, SmartR compares the list from the previous

Wifi zone	Contains the SSID or the name of the wifi group for which users would like to receive reminders
Upper wifi zone number	Contains the position of the chosen wifi group in the spinner containing the wifi groups
Detailed wifi zone number	Contains the position of the chosen wifi zone in case the user chose to get reminded, when he enters a specific wifi network

Table 3.6: Wifi zone reminder parameters

scan with the currently detected wifis. In this process, a list of wifis that have been removed and one of wifis that have been added, is created. The broadcast receiver (a class instantiation called when scan results are available) then calls an intent service, which initiates the evaluation process, if the lists of removed or added wifi networks contain at least one member.

Following this, the set of events saved in the users local Google Calendar partition is retrieved and the time constraints are checked as described in Section 3.1.2. After that, the reminder category is retrieved and all reminders that do not contain the keyword *wifi_ zone* in their *reminder category*, field are filtered out. If a reminder survives this filter it is evaluated, whether the specific reminder should be triggered on entering or leaving a wifi zone and the corresponding list of *relevant wifis* is retrieved accordingly (the list of added wifis, in case the user specified to be reminded on entering a zone and the list of removed wifis, in case the user specified to be reminded on leaving a zone).

If the wifi zone or group specified in the reminder equals one of the zones in the list of *relevant wifis* or one of the wifi groups, the *relevant wifis* belong to, a notification is send to the user. However, if the field *social component* of the reminder is set to true, the notification is not yet sent and an evaluation of the social surrounding is started. If one of the social contacts specified in the reminder is around the user, a the reminder notification is pushed to the user. The evaluation of the social surrounding itself in the case of wifi zones is similar to the assessment of pure social surrounding reminders and explained in further depth in Section 3.3.3.

Use Cases

A possible use case for wifi-zone reminders is to be reminded of important household duties, like taking out the trash, or feeding a pet, upon leaving a *HOME* wifi-zone. In addition, in combination with a *social component*, this reminder

category could, for example, be employed to remind students of getting a coffee, if the smart phones senses the wifi of the user's favorite coffee shop (e.g. when the user passes it on the street) and one of her friends from the *buddy group* with the name coffee shop is around her.

3.3.2 Geofence Reminders

Geofence reminders are, very much like their wifi zone equivalents, triggered once a user enters or leaves a previously defined geofence.

Geofences

Geofences are geographic constructs, which are based on a geographic position and a radius around this position. Android offers an API [17] to register and monitor geofences, meaning that it is possible for an application to implement and register a service that is triggered by this API, if a user leaves or enters a geofence. Geofences are registered with an expiration date, which in the case of SmartR equals either the reminder's *end date* or the reminder's *expiration date*, depending on whether or not the *time frame set* field is set to true. In addition, each geofence is given an ID, which in the case of SmartR equals the *title* of the corresponding Google Calendar Event, created upon the reminder submission. At this point, please remember that the *title* of the Google Calendar Event equals the *description* given by the user in one of the SmartR reminder configuration user interfaces.

In addition, geofences are no longer monitored by the corresponding Google API client, after the application, for which they where registered, is terminated and need to be re-added to the Google API client responsible for monitoring geofences. Therefore, geofence reminders only work, if the application is running (either in the fore- or background) and the geofence data has to be stored locally, in order to be able to re-add geofences, which have not yet passed their *expiration date*/*end date*, to the Google API client, upon the re-start of the SmartR Android application.

In addition, the SmartR Android application senses, whether a reminder that has been shared with a user by a friend is based on a geofence and adds the geofence needed for the shared reminder to the set of geofences monitored by the Google API client. Along with adding the geofence, the Google API client is given the specifics of the SmartR service handling the geofence transitions, necessary to call it, once a transition happens. This service then deals with the reminder evaluation.

Radius	Radius around the chosen geographic position, which together with this position defines the area of the geofence

Table 3.7: Geofence reminder parameter

Geofence Reminder Parameters

In the special case of a geofence reminder, the coordinates of the chosen circle center are not saved in the JSON string like all other parameters that are usually not specified, when adding a normal Google Calendar Event entry, but instead directly written into the location field of the respective event. Further details on the only additional reminder parameter for geofence reminders can be found in Table 3.7.

Important: Please note that the ID of the geofences registered together with the reminders, which the depend on them, equals the title of these respective reminders.

Geofence Reminder Evaluation

Every time a geofence is entered or left, the SmartR service, handling these transitions, is called. This service then retrieves the ID of the geofence(s), which triggered the intent service from the given intent and compares it to the ids of all events which have *geofencing* specified as their reminder category. If there is a match between the titles of the retrieved events in the calendar and the geofence id, a notification containing the reminder information is pushed to the user. However, again, if the *social component* field is set to true, the same restrictions as for the reminders based on wifi zones to pushing the notification apply.

Use Cases

Geofence reminders can be employed similar to physical surrounding reminders, however, they should be used for event reminders, which should not be triggered, if a certain point of interest category is passed, but which are linked to a specific place. For example, the case of a student, who wishes to be reminded to get some coffee prior to joining a lecture in the morning fulfils these constraints. Even better, in combination with a *social component* the user could be reminded to grab some more cups of coffee, in order to serve the coffee wishes of her co-students.

3.3.3 Social Components for Wifi Zone and Geofence Based Reminders

In case wifi zone and geofence reminders have been complemented with a *social component*, the SmartR Android application sends a request to the SmartR server back-bone to assess the current social environment of the user. This request is only sent for a given reminder, if all conditions for meeting the requirements for notifying the user are satisfied, as far as the geofence or wifi zone component of the reminder is concerned. In addition to the standard user information (e.g. Google account name and registration ID of the device for Google Cloud Messaging), which is sent to the server, when analysing social surrounding reminders, the title of the assessed reminder, the value of the *on in* variable and the reminder category to which the given reminder belongs to, are uploaded. The title is used to enable the server to retrieve the given event from the user's Google Calendar, in order to retrieve the social reminder parameters for this reminder. Due to this, the SmartR Android application does not have to upload the complete event data in order to enable evaluating the social surrounding of the reminder. Thereafter, the same evaluation process as for reminders based on social parameters applies (see Section 2.19). If a friend, who is relevant for triggering the reminder, is sensed within the area of the user in by this process, she is notified with a push notification. The *on in* variable and the reminder category of the reminder, uploaded together with the request, are used to determine, which message to send to the user (e.g. informing the user about whether she left a geofence or a wifi zone).

3.4 Reminder Sharing

Reminder sharing in SmartR is implemented using the possibility to add a list of attendees to a Google Calendar Event. These attendees are identified within the Google Calendar Event by their mail addresses associated with their Google account. As shown in the *user table* section of Table 4.1, a users SmartR identity consists of both, their unique SmartR user name and their Google Account e-mail address in the *user table* of the SmartR MYSQL database. Due to the bijective nature of this mapping (considering only the set of Google Accounts, whose users are also SmartR users and knowing, that all e-mail addresses are unique), the Google e-mail addresses of the friends, who have been added to the list of recipients of the shared reminder, can be easily retrieved from their SmartR names and then used to create an attendee object. This object is then added to the list of attendees for the Google Calendar Event associated with the reminder. Recipients can not only see the shared reminder in their SmartR application and any other Android application using the user's Google Calendar, but also receive a notification e-mail with the specifics of the shared event, if this functionality is turned on in their Google Calendar configuration. At the moment, receiving a

shared reminder means that this reminder is automatically evaluated along with all evaluations of the reminders set by the receiving party herself. In order to stop the received reminder from being evaluated, users need to delete it via a Google Calendar application.

3.4.1 Use Cases

Use cases for reminder sharing, include virtually all reminders with a social component, where the sender would like the receiver to remember something, when both parties are near to each other. This might be the case at a large event like the *Oktoberfest*, where the sender, who is still on her way to one of the tents, would like to share a reminder with a friend, who is already inside, that reminds this friend of already ordering food and beverages, once the sender has reached the outer layers of the festivity area.

Chapter 4

Application Infrastructure Backbone

This section describes the data storage system, user authentication processes and how SmartR is embedded into the existing calendar and reminder infrastructure of the Google/Android system.

4.1 Data Storage

The data provided by users is stored in different environments. The decision on where to store a piece of data depends on whether the specific piece contains data of a calendar event (reminders), whether it contains information about the user (e.g. location, friends,etc.), which is saved on the central SmartR MYSQL database or whether data is only needed locally.

4.1.1 Calendar Data Sets

Definition 4.1 (Calendar data sets)**.** Data sets, which specify characteristics of an event for which the user has submitted a reminder via SmartR, are called calendar data sets.

As already mentioned in Section 2.1, SmartR uses the Google Calendar within the user's account, which is labelled as *primary*. If no such calendar exists, the user has to add a calendar with this label. This, however, has not been a problem with test users so far, since *primary* is the default label given to the first calendar created in a user's Google Account and this setting is changed rather rarely by users.

Calendar data sets are basically modified Google Calendar Events, which are created once a reminder has been submitted and are automatically stored in the users Google Calendar. These data sets contain characteristics, ranging from the title of the event to the reminder type or friends, whose current location should

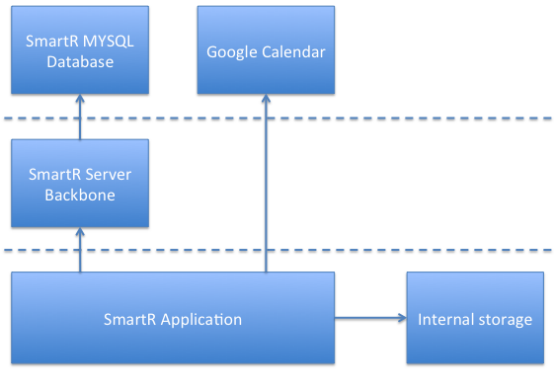

Figure 4.1: SmartR database scheme

be taken into account when triggering a reminder. Characteristics, that will be used by the SmartR server back-bone later, in order to evaluate, whether or not a reminder should be triggered, are stored as a JSON string in the description field of a Google Calendar Event. The user can update or change this data directly via the SmartR Android application or via the Google Calendar interface. The use of Google Calendar Events also facilitates the implementation of the reminder sharing functionality by using the existing Google Calendar infrastructure. In addition, users are offered a visible representation of the time and date configurations of a given reminder within their standard Android calendar application. For more information on available fields in a Google Calendar Event and the Google Calendar API in general, please refer to the documentation listed in the bibliography[18][19].

4.1.2 User Data Sets

Definition 4.2 (User data sets)**.** Data sets, which contain specific information about the user, which need to be known by the SmartR server back-bone and are independent of any reminders that have been submitted, are called user data sets.

User data sets are stored in a MYSQL database. These data sets range from general user information like the user id and the user name over information about current user locations to friendship relationships between various users. This data is used to create a picture of the users current social and geographic environment and can be updated by the user at any time via the SmartR Android

application.

The SmartR MYSQL Database

The SmartR MYSQL database contains seven tables, which store user information, needed for authentication and evaluating reminders. In addition, it covers a table containing word associations. For further specifications of the SmartR MYSQL database, please refer to Table 4.1.

4.1.3 Local Data Sets

Definition 4.3 (Internal data sets)**.** Data sets, which contain information which is only saved locally to enable services in a non-connected environment, are called user data sets.

Local data sets are mostly used for information, which might not be consistent across different devices, used by a user, or which is needed to evaluate reminders, which do not depend on dynamic data, saved as user data sets in the SmartR MYSQL database. The most important local data sets are the wifi zones and geofences, which are used as an invisible fence to check whether a user is within a certain area. While wifi zones are saved in a local SQL database, all other local datasets are shared as key-value mappings in the applications *shared preferences*.

In addition, to the already mentioned kinds of local data sets, all important state parameters (e.g. check state of the *Continuously retrieve reminder information* check box in the main user interface) of the SmartR application are saved, in order to be able to reconstruct the application in a user friendly manner, after a possible restart. In addition, preferences like the polling interval for the continuous reminder information retrieval are saved locally. Last but not least, also the buddy groups, which have been used by a user in the past to categorize friends, are stored locally, to avoid time consuming network operations for retrieving the already used group. The locally saved groups are offered to the user as choice, when he wishes to add relevant buddy groups to a reminder, or when she seeks to add a friend to a buddy group.

4.2 Data Modification and User Authentication

4.2.1 User Authentication

In order to authenticate the user with Google Calendar, SmartR needs to acquire an access token for a user's calendar. The specifics of the actions necessary to acquire such a token can be read in Google's documentation for calendar authentication [20] and the general Google OAuth2 documentation [21]. SmartR

User table	Contains a user's unique SmartR name and the e-mail address of the Google Account connected to SmartR
Buddy table	Contains all friendship relationships between SmartR users and stores the respective characterization parameters of a friendship (e.g. buddy level, buddy group, etc.)
Location table	Contains the last uploaded location of each user in terms of coordinates and the membership of a certain square (see Section 5.1.1)
Cell user table (Deprecated)	Also contains the last uploaded location of each user, but only in terms of the square membership (see Section 5.1.1). Location updates only still happen due to historical reasons
Location mapping	Contains the squares laid over the earth surface in terms of their edge coordinates; is used to map the user's uploaded geographic position to one of these squares
Pending friend requests table	Contains all still pending friend requests sent from one user to another
Word assoc table	Used to store association relationships between two words. Each time a social reminder is uploaded containing more than one buddy group, the given buddy groups are registered pairwise and inserted as a row into this table, if this pair has not appeared before. If the given pair has already appeared, the counter field of this row is incremented (see Section 5.3)

Table 4.1: SmartR MYSQL Database Tables

continuously refreshes access tokens, especially before important commits or data retrieval actions to guarantee a working data transmission. These access tokens are used in different ways depending on the wished action.

4.2.2 Modifying Calendar Data Sets

Reminder submissions and the following creation of a Google Calendar Event are handled by the Google Calendar API[19] for Java, more specifically by an instantiation of the CalendarService class to which the access token is handed. This instantiation is handed the event, which contains the information about the reminder the user would like to submit and the calendar, to which it should be submitted. It then handles the build and execution of the according http request for uploading the payload into the Google Calendar cloud with the correct access token.

4.2.3 Retrieve Calendar Data Sets

Since calendar data sets are evaluated by the SmartR server backbone, the server has to somehow retrieve the information about the calendar events, which store the reminder information. Therefore, it also needs the access token in order to retrieve the relevant data. To avoid saving the user's Google account password or a refresh token in the SmartR database for the access token retrieval, the access token is handed to the server by the SmartR application on the user's phone. This is possible since all evaluation rounds are triggered by the application on the user's phone, which knows the needed access token at any point in time. The server can then send a request to the Google Calendar Cloud to ask for the needed information.

4.2.4 Modifying and Retrieving User Data Sets

As mentioned earlier, the SmartR MYSQL database is accessed via the SmartR server back-bone, which creates and handles the SQL queries. For this purpose, the SmartR server back-bone makes use of prepared statements[22], which offer security against SQL injection attacks.

However, even with prepared statements in use, a malicious party could access the database by just knowing the URL of the corresponding php script and specifying the GET and POST parameters of his http request to support his evil motives and goals. In order to prevent such actions, access to the SmartR server back-bone is subject to a user authentication process.

Algorithm 1 presents the way a session is started. The php script in the server back-bone checks, whether the user is indeed the person, she claims to be by comparing the SmartR user id, which, for every SmartR user, equals the

google mail address, given in the request cookie, to the id that is returned, when meta data for the Google Calendar is requested with the access token. If the user ids match, the access token is saved into the super global $_SESSION variable.

Algorithm 1 SmartR database user authentication set-up

```
1: procedure User authentication set-up(user id, access token)
2:     retrieved google id ← retrieveGoogleCalendarMetaData(access token)
3:     if retrieved google id == user id then
4:         startSession()
5:         Session.accessToken ← access token
6:     else return
```

Every further call of a script that accesses the database asks for the access token first and only continues to the requested service if the provided token matches the one in the $_SESSION variable as shown in *Algorithm 2*. Therefore, the user authentication set up is called, every time, the user receives a new access token. Since the correct combination of access token and user id can only be provided, if a specific user knows the Google account password, and every further call asks for the access token, this method is safe.

Algorithm 2 User authentication set-up

```
1: procedure User authentication(access token)
2:     if access token == Session.accessToken then
3:         continue to functionality
4:     else return
```

4.3 Embedment in Existing Infrastructure

All reminders set with SmartR are saved as events in the users Google Calendar account and can therefore be viewed and accessed by any Android calendar application, which also make use of this account.

In order to provide as many functionalities as fast as possibilities, SmartR uses two different ways to write events to the calendar account.

Reminders that require internet connectivity (e.g. social and physical surrounding reminders) for their situational evaluation (because for example data from the SmartR MYSQL database is needed) are directly uploaded to the Google Calendar Cloud, and are visible in local calendar instances upon synchronization with the Google Calendar Cloud. Their situational evaluation takes place on the SmartR Server.

Reminders that do not require internet connectivity for the situational evaluation (e.g. geofencing and wifi-zone reminders) are directly written into the local

calendar instance of the user. The situational evaluation takes place directly on the user's phone in order to provide connectionless functionality. These reminders are directly visible in the local calendar instance and become visible in the Google Calendar cloud upon the first synchronization of the local Google Calendar instance, following the setting of the respective reminder. Please note, that geofencing and wifi-zone reminder are connectivity dependent, if they contain a *social component*. However, also these reminders are directly written into the local calendar instance.

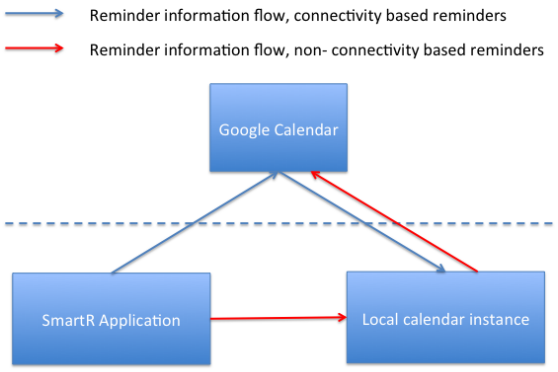

Figure 4.2: SmartR reminder setting information flow

Chapter 5

SmartR Utility Structures

5.1 Location

Providing reminders based on the social environment depends heavily on the availability of geographic location data of SmartR users. Therefore it is crucial, to use data structures that minimize the algorithmic effort necessary to verify if a friend is currently near the user.

5.1.1 Location Data Storage

The naive approach to storing a user's location data would be to just write her position in terms of geographic coordinates into the database. However, this would make it necessary to calculate the distance between any two relevant friends upon the evaluation of a situational reminder, which offers the social reminder functionality. To save time and computational effort, SmartR uses a data structure (similar to the one in RemindU [10]), which splits the earth into squares. Currently these squares have an edge length of 50 meters. These squares are saved in the SmartR MYSQL database in terms of there edge coordinates. Once a user uploads her location, the uploaded coordinates are used to determine the square, in which the user is currently located. The square is saved in terms of a square id, a row id and a column id. This enables an easy comparison of a set of friends, relevant for triggering a reminder to a set of friends, who are currently nearby, for retrieving information, about whether or not to trigger a reminder.

In addition to the location storage in the SmartR MYSQL database, the last uploaded location is stored on the phone, as already mentioned earlier. This offers the possibility to initialize the map in the geofence reminder configuration user interface to this position. Since the map also enables users to quickly navigate to their current position by pressing the upper right button on the map, this reduces the risk that a user has to navigate through the map manually, by basically offering her two different map initialization positions.

5.1.2 Location Retrieval and Use

Now, knowing that user locations are saved in terms of square memberships, one could ask whether it is necessary at all, to store a user location in terms of geographic coordinates. However, since a user's physical surrounding is analysed by performing nearby searches via the Google Places API [15], which needs geographic coordinates to check the surrounding area for points of interest, this is still a necessity.

The user location in terms of the square, in which a user is located, is used to check for neighboring friends, which are relevant for a given reminder. This check is conducted using the *social radius* as specified in Table 3.3. The algorithm for nearby friends is described in *Algorithm 3*. It is important to notice, that the *social radius* is converted into a relevant sequence of squares around the user which should be searched (including the square in which the user is currently located) for friends relevant to a given reminder. In the end, a list of reminder relevant buddies in the desired area is generated, which is then further used as described in Section 3.2.1.

Algorithm 3 Relevant Buddy Location Retrieval

```
1:  procedure RELEVANT BUDDY LOCATION RETRIEVAL
2:      relevantSurroundingBuddies ← newList()
3:      widthInSquares ← ⌊socialRadius / 50⌋
4:      if widthInSquares < 2 then
5:          widthInSquares ← 2
6:      for i = −⌈widthInSquares/2⌉; i < ⌊widthInSquares/2⌋ + 1; i++ do
7:          for j = −⌈widthInSquares/2⌉; j < ⌊widthInSquares/2⌋ + 1; j++ do
8:              curRowId ← userRowId + i
9:              curColumnId ← userColumnId + j
10:             relevantBuddiesInSquare ← retrieveRelevantBuddiesInCell(curRowId, curColumnId)
11:             for relevantBuddyInSquare : relevantBuddiesInSquare do
12:                 if locationUpdatedRecently(relevantBuddyInSquare) then
13:                     relevantSurroundingBuddies.add(relevantBuddyInSquare)
        return relevantSurroundingBuddies
```

5.2 Friendship Structure

In order to be able to analyse the current social environment of the user, it is necessary, that she provides SmartR with information about her social relationships by adding friends and associating them with certain characteristics, such as the level of friendship, the membership in a certain group of friends and an informal description of the friend as already described in previous sections.

Friendships in SmartR are only possible, if both parties mutually agree to become friends. This is necessary, since sensitive location data of friends is retrieved for the reminder evaluation, which should only be used, if these friends approved SmartR of using this data by agreeing to be friends with the requesting party.

In SmartR, friendship characteristics can be changed without consent of the respective friends, since such changes only affect the individual reminder evaluation of the changing party and do not have effects on the reminder evaluation of the respective SmartR friend.

5.3 Word Association

In order to be able to offer more meaningful reminder suggestions in the future, SmartR already features a word association mechanism. Every time, a reminder, based on social environment, with more than one buddy group specified, is submitted by a user, SmartR records all buddy groups submitted with this reminder pairwise as rows in the *word assoc table*. If a pair does not yet exist in the table, a new row is added to the table, otherwise, the appearance *counter* of this pair is incremented. The so gained information, of which words or buddy groups are very much linked is not yet used, but will be available for future use, in order to offer new and better reminder options or suggestions to users.

5.4 Google Cloud Messaging

In order to be able to send notifications to the user, once the evaluation of reminders, which are evaluated on the SmartR server back-bone (e.g. all reminders with a *social component*, physical surrounding reminders, etc.), is done, SmartR makes use of the Google Cloud Messaging For Android[23] service. This service enables servers to send data to a specific android device. The Google Cloud Messaging service needs a registration id, in order to be able to forward the notifications sent by the server to the right smart phone. SmartR creates this registration id, on the first call of the SmartR main user interface. The registration process is conducted by calling the register function of an instance of the Google Cloud Messaging class and provide it with the project number of SmartR given in the Google Developers' Console (see Section 5.5). This routine registers the users phone with the service and then returns the registration id for the given Android device. Every time, the SmartR Android application polls for an evaluation of all social and physical surrounding reminders, or a wifi-zone or geofence reminder triggers the *social component* evaluation process for a single reminder, this registration id is uploaded and needs to be handed over to the Google Cloud Messaging service, in case a notification is sent from the server to the smart

phone. The message send by the SmartR server back-bone and forwarded by the Google Cloud Messaging server, is sensed by a broadcast receiver in the SmartR application. This broadcast receiver calls the SmartR service, which is responsible to build and push the desired notification to the user. Further specifications on how to use the Google Cloud Messaging service can be found in the relevant documentation[23].

5.5 Google Developers' Console

In order to be able to retrieve Google Calendar data for a specific user, conduct nearby searches via the Google Places API, show the (Google) map in the geofence reminder user interface and use the Google Cloud Messaging service, an application needs to be registered with the Google Developers' Console. For functionalities with a great need for authentication like retrieving a user's calendar data, the application must be registered as an OAuth client.

For functionalities like Google Maps or the Google Places API, it suffices to register an application for *Public API Access*.

Further documentation on this can be found in the official Google documentation[24].

Chapter 6

Outlook and Conclusion

6.1 Future Work

SmartR enables users to combine a variety of situational reminder options and connect to a community of technology aware peers, who enjoy the possibility to configure reminders suitable for any situation. However, in order to reach a large scale audience, SmartR needs to be further tested and developed.

In order to qualify for regular use, the effects of running SmartR on a phone's battery need to be further studied. Smart phone owners will only use an application, if this does not mean that they need to recharge their phone within a short time interval. In addition, SmartR is currently still rather dependent on internet connectivity and can only offer a limited functionality, once the user enters a region without or limited connectivity. This problem can be solved by further enlarging the pool of cached data and more buffers, to milden the consequences of short disruptions of network connections.

In addition, users should have the possibility to upload and store their location data encrypted. This might be realised by using a protocol that keeps the user's location a secret to everyone, but her friends, or by using a master secret in the SmartR server back-bone, to encrypt the uploaded locations upon receiving them. Choosing the first method would imply, that the complete reminder evaluation process would need to take place at the user's Android device, since the SmartR server back-end would have no means to decrypt the uploaded, already encrypted location. This might lead to rather heavy computational effort on the user's smart phone. Therefore, option two should be preferably implemented first, before option one has been tested thoroughly. A possible solution has been presented in *RemindU* [10], however, this system would have to be adapted for the use with reminders based on physical and/or social surrounding.

Furthermore, the word and buddy group associations saved in the *word assoc table* should be used to provide a better reminding service to the user. This should be done by using the stored data to assess, whether it might make sense to trigger a social reminder, even though the user has not specified the buddy

groups, to which a currently nearby user belongs, as relevant for the given reminder.

Once SmartR has taken all hurdles for being used by all sorts of people on a daily basis, it should be possible to offer professional services in reminder form. This means that users could be reminded that a plumber is sitting right next to them on the tram, if they had trouble with their sanitary equipment in the morning and set a reminder to search for a plumber later that day.

6.2 Conclusion

This thesis has presented a convenient way of implementing a smart situational reminder, covering a variety of categories of situational reminders and combining them, where these combinations maximize the user's possibilities to use such a smart situational reminder application. In addition, it has been shown that it is possible to create new reminder applications, which can be used harmonically with existing reminder and calendar solutions, thereby minimizing the hurdles for users to apply the offered service on a daily basis.

Bibliography

[1] : Google now; URL: https://www.google.com/landing/now/.

[2] : Geobells; URL: https://play.google.com/store/apps/details?id=com.patil.geobells.lite.

[3] : Wifi alarm; URL: https://play.google.com/store/apps/details?id=com.jdr_software.wifi_alarm&hl=de.

[4] : Friend reminder; URL: https://play.google.com/store/apps/details?id=com.arconsis.friendreminder&hl=en.

[5] : Ifttt; URL: https://ifttt.com/wtf.

[6] : Finde meine freunde; URL: https://play.google.com/store/apps/details?id=com.fsp.android.friendlocator&hl=de.

[7] Chaminda, H.T., Klyuev, V., Naruse, K.: A smart reminder system for complex human activities. 14th International Conference on Advanced Communication Technology (February 2012)

[8] Phyo Wai, A., Foo, S., Biswas, J., Donnelly, M., Parente, G. nad Nugent, C., Yap, P.: Smart phone reminder system for managing incontinence at nursing home. Proceedings of the International Symposium on Consumer Electronics, ISCE (2011)

[9] Tu, Y., Chen, L., Lv, M., Ye, Y., Huang, W., Chen, G.: ireminder: An intuitive location-based reminder that knows where you are going. International Journal Of Human-Computer Interaction (December 2013)

[10] Zhao, X., Li, L., Xue, G.: Remindu: A secure and efficient location based reminder system. IEEE ICC 2014 - Communication and Information Systems Security Symposium (2014)

[11] Kwon, O., Choi, S., Park, G.: Nama: a context-aware multi-agent based web service approach to proactive need identification for personalized reminder systems. Expert Systems With Applications (2005)

[12] Beigl, M.: Memoclip: A location-based remembrance appliance. Personal Technologies (2000)

[13] Lin, C.Y., Hung, M.T.: A location-based personal task reminder for mobile users. Personal and Ubiquitous Computing (2014)

[14] Dey, A.K., Abowd, G.D.: Cybreminder: A context-aware system for supporting reminders. HUC '00 Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (2000)

[15] : Google places api; URL: https://developers.google.com/places/documentation/.

[16] : Supported types for google places api; URL: https://developers.google.com/places/documentation/supported_types.

[17] : Geofencing api; URL: https://developer.android.com/reference/com/google/android/gms/location/Geofence.html.

[18] : Google calendar api event documentation; URL: https://developers.google.com/google-apps/calendar/v3/reference/events#resource.

[19] : Google calendar api documentation; URL: https://developers.google.com/google-apps/calendar/.

[20] : Google calendar authentication; URL: https://developers.google.com/google-apps/calendar/auth.

[21] : Google oauth2; URL: https://developers.google.com/accounts/docs/OAuth2.

[22] : Prepared statements in php; URL: http://php.net/manual/de/mysqli.quickstart.prepared-statements.php.

[23] : Google cloud messaging for android; URL: https://developer.android.com/google/gcm/index.html.

[24] : Google developers' console; URL: https://developers.google.com/console/help/new/.