Native Visualization of Mobile Activity Patterns

Bachelor Thesis in Computer Science

Christian Janßen
302530

Advisor:: Prof. Ulrik Schroeder, RWTH Aachen
Second Advisor:: Prof. Jan Borchers, RWTH Aachen

Supervisor: Dipl.-Inform. Hendrik Thüs

Christian Janßen
"Computer-Supported Learning" Research Group
LuFG Informatik 9
RWTH Aachen University
September 1st, 2013

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Aachen, September 1st, 2013 Christian Janßen

Contents

Part I

Chapter 1 Introduction

Smartphones and their daily usages

Since the introduction of the smartphone and its ongoing boom in sales, those devices getting more and more important for their users. Nearly 30% of the German population own a smartphone and 50% of them use it on a daily basis [6]. Googling a question, making a phone call, sharing your location and status with your friends via facebook or just taking a snapshot of your lunch - smartphones are used in nearly every location and situation. With all these possibilities and potential in usage for everyday situations, it is getting harder and harder to keep track of when one has used its smarthphone, where it was used and most important, for what was it used. Knowing this may have a positive influence in productivity. While on the topic of productivity, another interesting fact is, that 72% of owners state that they use their smartphones at work [6]. The obvious question is, did one use its smartphone for relevant research or emailing, or was it used to chat on whatsapp or checking facebook. It would be desirable with focus on efficiency and productivity if by the end of the day one could check his or her smartphone activity and would see that one may have to improve his or her working or learning behavior, because the smartphone was used in a distracting way or, the more preferable case, that the smartphone was mostly used to get work done.

Need of a structured overview

Another situation in which nine out of ten users utilize their device is while they are on the move [6]. Again, it would be desirable if one had information how he or she got from place A to place B, how much time it took and what has been done in this time.
Not only has the computation power of smartphones significantly improved, their integrated cameras are also quite satisfying and are comparable to low-end digital cameras. Since most people tend to take pictures with their smartphone nowadays, it

would be helpful to get a simple overview in which one can easily see the exact position of the picture's location without taking any further investigation.

The lacking of the possibility to check what has been done with the smartphone may unconsciously lead to a distracting use of it. That is, because, as already mentioned, keeping track of all activities is extremely difficult and thus the total time of the daily usages is normally unknown. There exists a need of a structured overview which provides informations about time, location an d applications to get a self reflecting impression about one's daily mobile activities.

1.1. Objectives

Requirements for the application

Due to the stated need of an application which provides information about one's daily activities the main goal of this Bachelor thesis is to create such an application. Requirements for applications are normally high and the developer has the task to find optimal solutions for them. This application is no exception and thus has its own requirements. On the one hand it should display enough information to grant the ability to draw conclusions about one's daily activities but on the other hand it should also display the data in a visual appealing and intuitive way such that the visual appearance is not too crowded with unnecessary information.
The application should also offer the possibility to take minor adjustments to fit in one's individual needs and thus making the experience with the application feel. In contrast to that those adjustments should not be to detailed and low in number to keep the options structured and understandable.

Applications for partial solutions

As mentioned, the main objective of this thesis is to create an application which provides feedback and a self reflecting view for one's daily activities. There are some existent applications that partially fit in the described situation and offer solutions. For example one can use Google Latitude to see where he or she has been traveled respectively which routes were taken to reach a specific place. But it will not show which applications have been used at a specific place. Also while this thesis was written the support for Google Latitude was discontinued [5]. Another partial solution is facebook. One can share his or her location and photos on facebook but this will not show a route on a map nor does every one want to public his or her whole life on facebook.
To see the most used applications of the day one could take a

look at the operating systems utilization chart which displays the percentage of used battery life and shows the cpu usages in total time. Not only does it not take into account the visited location this is also a very impractical and non intuitive way of gathering information about one's daily activities.

Testing of graphical possibilities

Another objective was to create an application which provides manifold views of the user's daily activities. It should not be limited to a map displaying pins or a list of application names with the total time of usages. Instead the application should show what is possible to develop with the help of already existent graphical views and, what is even more interesting, with creation of new views. All views should differ from each other and show different possibilities in visualizing informations of one's daily activities while each one could be used as a stand alone self reflecting view.

The application of this Bachelor thesis should perform as a central information provider which combines the listed objectives and displays the daily activities of the smartphone. Those information will be presented in various ways and can be filtered by different criteria within a clear, intuitive graphical user interface.

Chapter 2 Background

This chapter provides background information about the main topic of this bachelor thesis. First, the ongoing trend of using a smartphone in nearly every situation and therefore the need of keeping track of one's own mobile activities is discussed. To grant an application the possibility to give self reflecting impressions the application itself needs to be provided with personal data of the user. The second section is about this provided data, its origin and how it is gathered. Once the origin of the information is discussed the next section talks about the representation of this it. In this context the meaning of native visualization is explained and an alternative is presented which involves a short introduction of a currently written Master thesis. The last section lists and explains which hardware and software was used during the implementation.

2.1. Self reflection

ADD REFERENCE TO SELF REFLECTION PAPER!

A Smartphone has many usabilities. It can be used as a camera, newspaper, music player or as a portable gaming device and those are just a few examples. There are a lot more ways to use a smartphone and as mentioned in the introduction, they are used in nearly every situation. It is a modern multitool which was used by more than 23 million people in Germany in 2012 [6].

Smartphones as modern multitools

As advantageous as it may be in everyday situations, the downside is that most people do not know how much time they spend on their phone and thus do not know how distracting it may be. For example, checking new mails may lead the user to also check the newest facebook messages and stay within this application a few minutes longer than expected. At least 50% of smartphone owners access the Internet with their device [6], thus most people are always available through instant messaging services like whatsapp. The result is that people write and receive messages

more often. And because smartphones are capable of running diverting games, one may uses its device to beat the last achieved high-score.

Distraction

But a smartphone can be a great helper too. It is an easy to use digital calendar which reminds the user of all upcoming events, it can be used as a travel guide or a navigation system, it allows to quickly respond to an important email and has many more useful advantages. But the previous short examples demonstrate that a smartphone can also have a distracting influence to its owner.

At this point the the idea of self reflection is needed. The concept of self reflection is the critical reflection on one's own actions and positions and coming to a conclusion. This can be used to assess the distracting influence of smartphones to its owner. But for an accurate assessment one needs to keep track of his or hers own daily activities, which is nearly impossible to do for a smartphone without the help of tools that provide background information.

Provide a self reflecting view

As mentioned, the help of a tool is needed which provides information about the owner's mobile day in a self reflecting manner. This tool in form of a smartphone application should display the information such that the user is able to instantly see where, when and for what the device was used. If one is provided with this data, he or she is able to say, that they used their smartphone in a productive manner or if they used it for entertainment. Further more, one could tell if he or she used the smartphone to divert themselves from working or studying. Although the application can display the needed information, the conclusion must been drawn by the user.

Possible improvements in productivity

With this application and the provided information a user may be willing to rethink his or her work respectively study behavior. This then could lead to less frequent use in distracting applications, thus improving efficiency and productivity in daily tasks.

As described, there is a possible application area for such a smartphone application. It would visualize data and information about the owner's daily activities in such a way that the application could be used as a tool for self reflection.

2.2. Provided Data

In the last section the idea of an application which displays information in a way such that one can use it for self reflection was described. What has not been described is the source of the data to be visualized and how it is gathered.

SmartDay and Big Brother

This thesis' application, namely *SmartDay*, will not gather the data it uses, instead the data is downloaded from a server and stored internally. The mentioned data arises from an external application called *BigBrother*. This application is based on the Master thesis of Thorsten Kammer [9] and was reimplemented and developed by Hendrik Thüs in 2013 [2]. ...
IMPROVE REFERENCES
The data Big Brother gathers, is sent to a server where it can later be downloaded in an aggregated way and used by the application developed for this bachelor thesis. The data contains amongst others information about the user's visited locations, the name of the currently used application, start and end time of used applications. This data is uploaded and stored on a web server, which is then accessed by this thesis' application.

Privacy issues

With the revelations published by Edward Snowden in June 2013 about the U.S. American spy program PRISM one might be concerned about privacy violation by third parties. It should just be said, that this project is still an experimental phase. If it should be published for a larger audience than the developers much work would be put into encryption and ensuring the prevention of unauthorized access by third parties. Another solution would be the release as an open source project. In this case users could host such a centralized service on their own.

Reasons for online stored data

One of the reasons why the data of daily activities is stored online is the limited storage of mobile devices. This way the used size of storage can be minimized and only needed information can be downloaded. The possibility to merge data from other devices the user owns, like PCs, laptops and tablets is another reason to upload the data. Being able to access the data from multiple devices like tablets is also an important point. This method is of course at the cost of data traffic but is needed to centralize data especially in case of merging data and accessing them from different devices.

The idea of a data set which also contains information about other used devices has great potential in granting an even better overview of one's daily activities and thus this would make

the self reflecting view provided by the application even more meaningful. Having the data of the activities of a laptop could be powerful for student who is learning with it, because he or she may not use the smartphone during that time, but instead one could see if he or she was productive, by checking if the laptop was actual used for studying by reading a pdf file or if it was used to browse non important things on the Internet. With additional data available from laptops or PCs, the coverage of one's day would be more complete.

2.3. Native Visualization

Now that the origin of the provided data has been explained, the idea of a native visualization will be explained in more detail.

What does visualization mean?

The term of *visualization* means representing of abstract data or information, like a text, in a visual ascertainable form. Its meaning is not limited to computer science. It can be found in various situations and places, technically anywhere where someone tries to convey information in a visual way. This may be a picture of an artist or a even a movie. The concept is not even limited to modern time. Since the early days of mankind, humans try to express information in form of visual tangible objects. For example the Egyptians did this 2000 years before Christ, by creating pectorals which for instance display a gryphon standing on a kneeling man of different skin color to express the Egyptians position in foreign countries [7].

As mentioned, in computer science, visualization means the representation of data in an illustrative way. For example, creating a pie chart for results of a survey or drawing a graph representing the daily temperatures for a week. For this thesis the visualization has to fulfills the task of displaying one's daily activities in an appealing and easily to understand way, such that one can directly draw conclusion from the information.

Native visualization

Native visualization describes the creating of visual ascertainable objects only by means of resources that a specific system provides without any addition. A visualization would be the generation of a website with with the use of JavaScript and then displaying this website in the application. But this would be non native because of the use of a website to create the view.
In this thesis, the application will be implemented for the mobile operating system Android 4.0.3 and higher. Native visualization under Android means the use of Java and the access to

the standard Android application programming interface. The application will not use JavaScript or anything else as this would infringe the terms of native visualization.

A non native visualization of daily activities

An alternative to native visualization of activities can be found in Thomas Honné's Master thesis "Interactive Visualizations of Activity Patterns in Learning Environments" [8]. The thesis describes, among other topics, the visualization of daily activities in a web browser environment. For comparison an interesting fact is, that the data arises from the same application. For more information please refer to chapter 5.

An advantage over the non native method is that data can easily be stored locally, thus making the application available for offline usage assuming that the needed data has been downloaded at some point in the past.
With a native application the user has tool for self reflection which is not permanently bound to an Internet connection and does not require any additional non native resources.

2.4. Hardware and Software

As mentioned in the previous section the thesis' application will be developed for Android 4.0.3 and higher. To get an impression of the used tools in the development process this section describes the used software, the development environment and Android, as well as the used hardware, the development device.

The operating system of choice

The application was developed for Android 4.0.3 "Ice Cream Sandwich" with the application programming interface level 15. To minimize overhead due to compatibility adjustments the support is only guaranteed for Android 4.0.3 and newer versions, which account for 63% of all Android devices [4].
Android was the operating system of choice because it has a free developer license, great supportive community and with a share of 76.7% in quart 2 of 2013 [1], Android is the largest market share holder of mobile operating systems.

The test device

The testing device was the Motorola tablet Xoom with Android 4.0.3. The tablet with a screen size of 10.1 inches and resolution of 1280 times 800 pixel gave great advantage in the testing process of the application. Its screen is large enough to display all relevant data without minimizing their visual appearance. In addition the screen size allows precise testing of multi-touch gestures. The alternative to the real development device would have been

an Android emulator provided by Google. With the emulator the development of this application would have been nearly impossible due to the fact that it does not support the ability to simulate multi-touch input with a normal mouse and the nonexistent support for GoogleMaps.

The development environment

Google's recommended software development kit Eclipse with Android Development Tools served as development environment. Working and developing with Eclipse was simple and comfortable due to Google's numerous tutorials [3]. Neither installing the software development kit on a Windows 7 computer nor setting up new projects was a problem. The test device connected to the computer was directly recognized by eclipse and executing and testing written code on the it proceeded without problems. Eclipse's debug mode was also very helpful at many points in the development process and helped to track down hidden bugs.

Working with Eclipse and Android was comfortable, a lot easier than expected and straightforward, even for a first time developer. Eclipse's improvement proposals and performance advices and Google's tutorials with helpful examples and background informations made this project a great educational experience.

Part II

Chapter 3 Development and Implementation

"It's done, when it's done"

—An English Phrase

3.1. Outline

The sections in this chapter will describe the various stages of development and implementation of this Bachelor thesis' application. At first the basic idea of the layout and the individual views will be described and explained. To get an even better impression the early paper prototype will be shown. Afterwards a short description of the aspired time schedule for the development as well as the creation of the written elaboration is given.

The implementation of the basic layout which was displayed and explained with the help of the paper prototype, will be the first part describing the actual implementation process. Second, the handling of the data to be visualized is explained. This covers the server based access as well as local storing and loading.

After the preparations for the visualization have been set forth, the different views will be described. Starting with the explanation of the map view, it will be clarified how the loaded data is visualized. This is followed by the section focusing on the chart view. It explains how a library is used to draw pie charts and how the loaded data has to be prepared. The last section describes how the timeline was implemented. It covers, how one can draw its own views in Android and how to make use of gestures.

3.2. Paper Prototype

The first step of the development process was the creation of a paper prototype. It was needed to demonstrate the idea and visual appearance of the project and application. Another advantage of the paper prototype is to be able to show the application to different people and to get feedback without even writing one line of code. This provides the ability to eliminate possible false estimations in the forefront of the application's implementation. False estimations may be the assumption of wrong needs of possible future users or the creation of an non intuitive layout. In the following the paper prototype will be shown and explained.

Basic layout

The first idea was to split the screen into two parts, an option part and a view part. The left part should be the option part, occupying one sixth of the visible screen. It should always display the following, in appearing order listed, options.
In the top left a view selection containing three buttons ordered vertically with titles "Map-View", "Chart-View" and "Timeline" are displayed. Taping one of those buttons causes a view switch to display the respective view.
Under the view selection a list of options should be visible. Those options vary from view to view but should always contain a button at the top of the list displaying the currently selected date. Tapping on this button causes a date-selection-window to pop up. Selecting a date leads to an update of the displayed view, now visualizing the respective data. Other options will be explained together with their respective views.

Map-View

When the application has been loaded, the user will be displayed the map view thus making it the application's start view. The view will display a map with a route, representing the user's visited locations. On start up the selected date will be the current day and therefor the draw route represents the user's latest movements. Tapping on the route should bring up a speech bubble which contains information about the tapped location. Those informations will be the time spend on that location, the used applications and possibly shot photos.

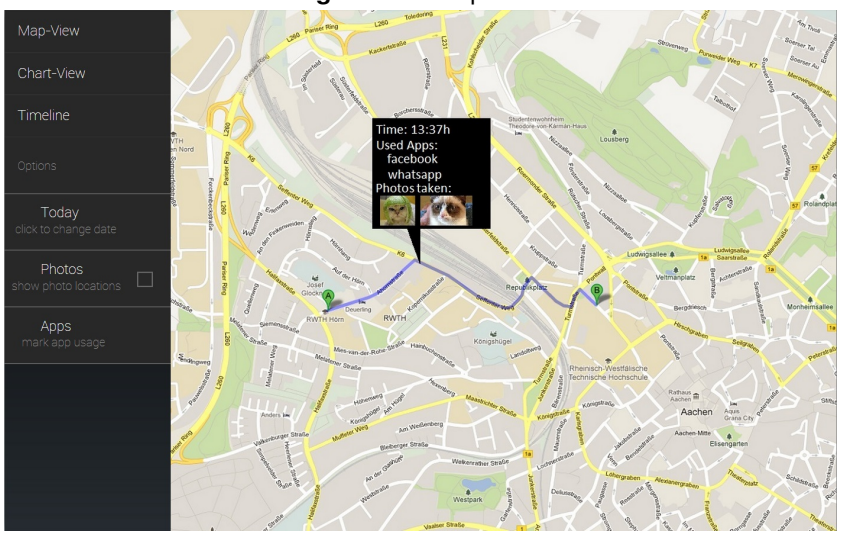
Map-View specific options

The view provides two special options. The first option is represented by a check box and has the title "Photos". The option provides the ability to highlight those locations, where a photo was taken. The second option is called "Apps". It lets the user pick applications from a list and colors all parts of the route where those specified applications were used. Those options would provide a great tool to quickly get an overview over the position data

of specific applications and actions. An actual image of the paper prototype for the map view can be found in figure 3.1.

The map view can mainly be used to answer the question "Where was smartphone used?", since the view focuses on visualize the map and traveled routes rather than displaying numbers and percentages.

Figure 3.1.: Map-View



The second view, the chart view, shows the user his or hers daily activities in form of a pie chart. Its layout can be found in figure 3.2. The idea was to create location based charts that shows for each visited place the percentages of used applications. To break down the number of shown charts and thus to give a better overview, locations would be summarized in an intuitive way. That means that one has a chart for work, home and on the move. Those charts are lined up vertically and have an individual chart on the top of the list. This individual chart can be adjusted by the two extra options described in the following.

Chart-View

The first option lets one choose which places are taken into account for the individual chart. The second option determines which specific applications are displayed in the first chart. With these two options, one has the ability to to customize the first chart to visualize only those data which fit one's individual needs.

Chart-View specific options

To get an even better overview, applications are classified into different groups. "Social" and "Productive" could be two groups, which split the applications apart. Applications like whatsapp

Grouping of apps and locations

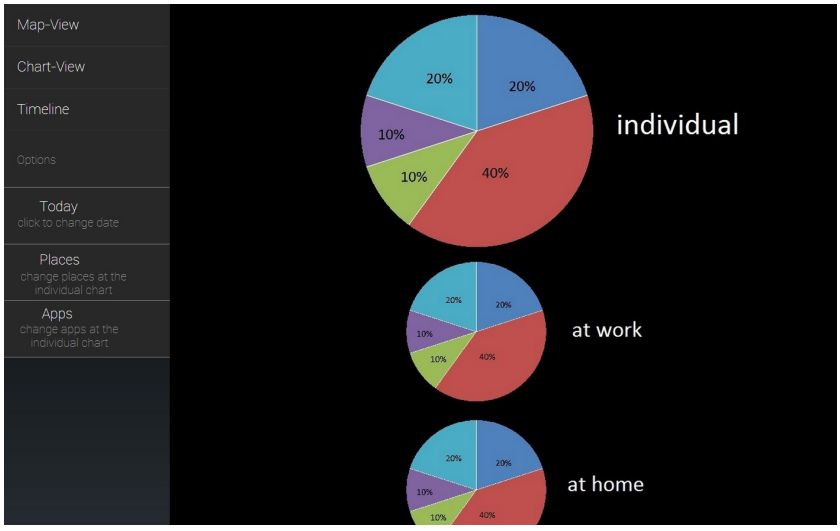
and facebook may be social, while mail programs would be productive. The user would also be able to classify the applications by him- or herself, allowing to personalize the application. Furthermore, home, work and other places are chosen individually and extra places like parent's home can be added by the user, to give the him or her more space for customization.

Interacting with the view

To get an better insight of the application groups, tapping on a slice of the pie chart will bring up a detailed view. In this detail view, one can see which applications are included in the tapped group and how large their percentage of daily activities is. In addition the total usage time of each application is shown, to give the displayed percentages more expressiveness.

The chart view, in contrast to the map view, is able to give a better overview over the percentages of used applications rather than show in which locations they have been utilized. It gives answer to the question "What was the smartphone used for?"

Figure 3.2.: Chart-View



Timeline

The timeline is the third and last view of this thesis' application. For a visual representation of the description of this view, please refer to figure 3.3. This view visualizes the daily activities in a chronological manner with two major parts. The first part is the the timeline itself and is described as follows. At the layout's bottom a horizontal line is draw with markings for every hour, from 0:00 to 24:00. Above this line, colored rectangles are drawn which represent a timespan in which an application was used. At the layout's top one can see markings for the visited location in

dependence of the displayed timespan. One has the possibility to scroll horizontally through the view to observe the consecutively occur activities.

The detailed view

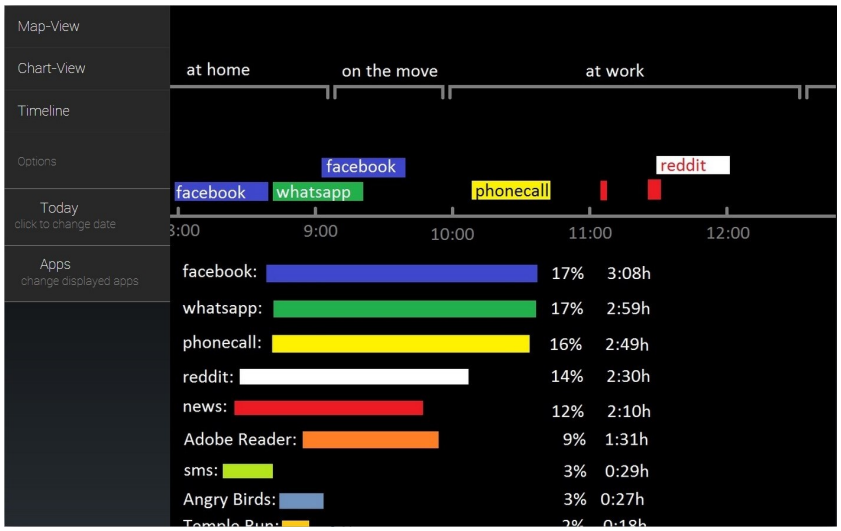
Beneath the actual timeline a detailed view of all applications used on the selected date is found. This second part shows the application in descending order starting with the application mostly used on the chosen date. For each application a rectangle which represents the percentage of daily usage, is drawn. This rectangle will have the same color as the respective rectangles in the timeline and thus the detailed view can be used as a legend to identify the applications displayed in the timeline. Next to the application's bar the respective percentage and total amount of time is displayed.

Timeline specific options

This view offers the extra option "Apps", where one can filter the displayed applications. With that option one is able to hide all other applications and display only those which are relevant for the user and thus giving the application a feel of personality and individuality.

Although it partially acts as a bar chart, the timeline, unlike the other views, focuses on the daily schedule by visualize the activities in a chronological order. With this view the user is able to answers the question "When was the smartphone used?".

Figure 3.3.: Timeline



As mentioned in the first place, this was the basic idea of the application and a few things have been changed during the development process. A summary of changes which differ from the paper prototype, along with the reasons for those changes can be found in the next sections.

3.3. Basic Layout

The actual implementation work started with the creation of the basic layout. It characterizes the general picture of the application and has changed during the process of implementation.

View, Viewgroups and Fragments

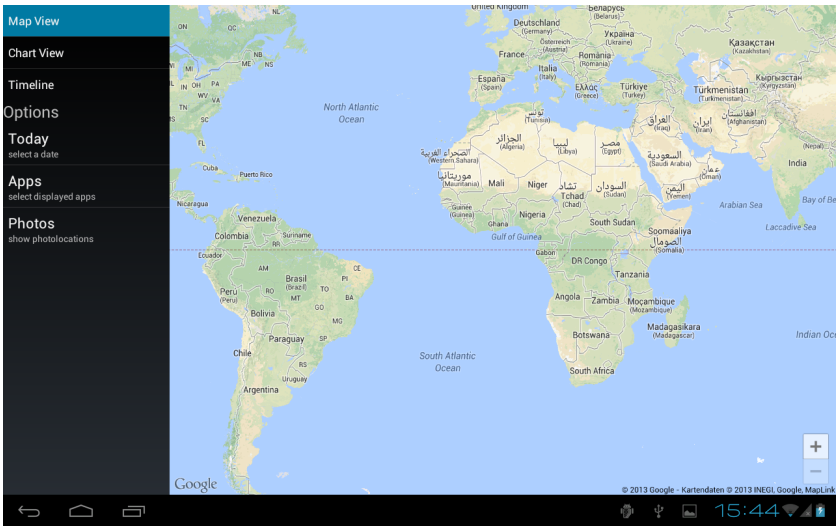
When the application first starts, one has to pass a layout to the main process, the main activity, to create a visible view. Those layouts are defined in xml and contain views and viewgroups. Basically, viewgroups contain views and/or other viewgroups and define their layout. For example one can define, whether visible views are ordered horizontally or vertically. Views contain the actual visible content, for instance a text or an image. Views are not static objects and can change during run time. One is also able to create and delete views and viewgroups during runtime, which will be helpful later. In order to create an application which looks and works like the paper prototype, one has to be able to change or switch whole groups of views in order to switch between map view, chart view and timeline. For this purpose one can use fragments.Fragments are kind of subactivities which handle their own lifecycle and viewgroups and they are necessary for an interactive application, as they can be reused during runtime, thus provide an efficient way for switching views.

The first basic layout can be found in figure 3.4. It resembles the paper prototype in functionality and appearance. Tapping on the view's name causes a view switch and options bring up pop up menus to select applications or a date. To provide the application with the ability to react to user input, one can assign click listeners to views. After creating and assigning a click listener, Android calls it as soon as the user taps on an assigned view and a specified action, for instance a view switch, will be performed.

Visible fragments

As mentioned above, at the start of the application the main activity needs a layout in order to create visible content. In this case the layout consists of three fragments. The first fragment is the view selection fragment. This is a list fragment which contains the three section names as views. The list fragment provides the programmer with the function `onListItemClick` which he or she can override. It gets called if one of the views names get clicked and is provided, among others, with the list position of the clicked view. With this information one is able to determine which view is requested and can then call the main activity to initiate a switch.
The second second fragment is the options fragment which is also

Figure 3.4.: First Basic Layout



a list fragment with the same abilities as the view selection fragment. If the user taps on an option the respective menu pops up and if the user makes adjustments the underlying dataset is updated and the main activity gets called to update the currently visible views.
The last fragment and most important in case of visualization and self reflection is the view section fragment. Those fragments visualize own layouts with different views which are described in their respective section.

Add and switch fragments

In the first layout, the view selection fragment called the main activity in order to switch the view section fragment. To switch fragments the main activity uses the FragmentManager api. To obtain a FragmentManager object, the main activity calls `getFragmentManager()`. With this object one can access a FragmentTransaction object, which is used to add or replace fragments. With the FragmentTransaction's function `add(int, Fragment, String)` one adds a fragment to a specific view. Do do so, one has to specify the container in which the fragment will be placed via an integer id previously assigned, the fragment to add to the container and an optional tag to access the fragment later. With a call of FragmentTransaction's function `commit()` the transaction will be executed.
To switch a fragment, one calls the FragmentTransaction's function `replace(int, Fragment, String)` with the same variables as previously described. It should be mentioned that only

those fragments can be replaced that were added in a programmatically way and were not defined in the xml's layout file. Instead if just adding a new fragment to an occupied container one should use the `replace()` function or otherwise it can not be guaranteed that only one fragment is visible. But because `replace()` destroys the fragment, its rather inefficient if a user switches views very often, due to the fact that the fragment will be recreated every time it is added.
A solution to this problem are the FragmentTransaction's `show()` and `hide()` functions. These functions obviously show and hide fragments without destroying them, therefore they are more efficient in case of power consumption and calculation time. Another advantage is the maintaining of zoom and scroll positions without adding a single line of code.

Dialogs and DialogFragments

To change the fragments appearance, one can tap on the options on the left side to bring up a pop up menu with a calendar or a checklist of application names. These option menus are created with Android's Dialog objects and controlled by the DialogFragment api. The first options menu, the calendar, is displayed by a special dialog, the DatePickerDialog which provides a visual appealing user interface to select a date. Once the option fragment created a DialogFragment, it has to assign itself as a listener to the object and call `show()` to Display the graphical user interface. To be able to handle the choice of a date, the DialogFragment implements DatePickerDialog's `OnDateSetListener`, which gets called with the selected year, month and day if the user approves his or hers chosen date. Afterwards the DialogFragment calls the previous assigned listener and passes the the date to it. The options fragment then displays the selected date and stores it in the dataset, which notifies the main activity to update all visible fragments.
The other pop up options menu is also Dialog managed by a DialogFragment. It displays application names with a check box. To show this dialog, the options fragment creates a DialogFragment object and calls `show()`. The options fragment does not have to assign itself as a listener but has to select a mode for the DialogFragment in order to load, display and store data correctly. Those modes are select_apps, ignore_apps and select_highlight_apps. Each mode builds its dialog identically with the help of a dialog builder. This builder offers different functions to create various dialogs and in this case is used to create a dialog with a multiple choice list and two buttons on the bottom. The builder provides a function called `setMultiChoiceItems()` with the following input variables. A string array of application names, an array of booleans representing checked states of the

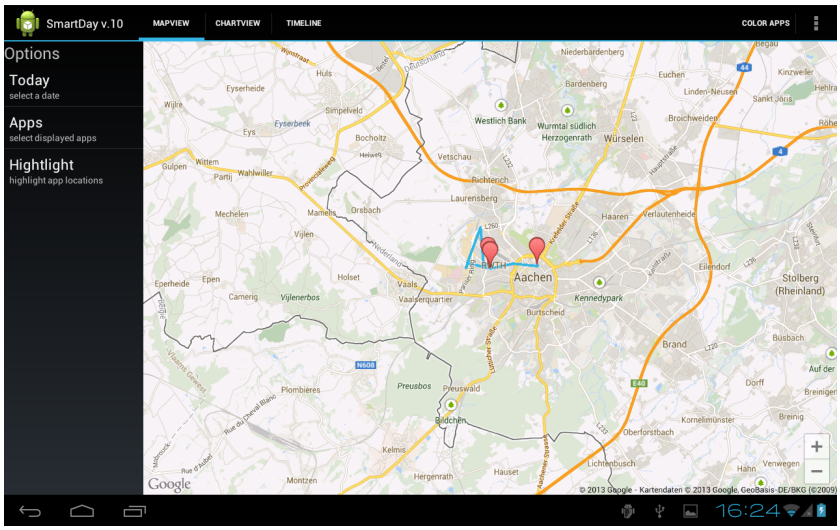
applications and a listener to handle click events. The application names and the boolean arrays are different for every mode, as they are loaded and gathered differently, for example the select_apps mode only displays names of applications used at the currently selected date and forgets the selections after a restart, whereas the ignore_apps mode loads all application names ever used by the owner an stores the selections in the dataset. The assigned listener tracks selection and deselection of list items and stores the application's name and the respective boolean for every change. If the user confirms his or hers selection, the dialog then stores according to the chosen mode, the selections in the dataset which again tells the main activity to update the fragments.

Change of the basic layout's visual appearance

As the process of development progressed, the need of more options than previously assumed raised. Options for colorization of applications, switching the logged in user, deleting local files and permanently ignoring applications needed to be assigned to the layout. The problem with these options were, that they would not be used as much as selecting a date or temporarily hide displayed applications. Adding the options under the existing ones would cause a loss in simplicity and clarity of the application. The solution was Android's ActionBar api.

The ActionBar serves as a navigation bar which is set up at the top of the application and lets the programmer add tabs and option buttons, where option buttons can also be combined in small pop up menu.

Figure 3.5.: Final Basic Layout



The action bar

In order to make full use of the now used ActionBar, the former view selection fragment was deleted and the view's are now selected on the action bar by tapping on the new tab header. This layout is more intuitive as it resembles know patterns of Internet browsers and feels more structured. On the right side of the action bar, the color apps button is visible next to three dots which open a small list with the other new options - switch user, delete saved files and ignore apps. For more details see figure 3.5.

Action bar tab header

The main activity controls and sets up the bar as follows. By default, the action bar is visible in the application's layout but does not contain anything than the application's name in the top left corner. To add tab headers and options, one has to work with the ActionBar api. The main activity calls the function `getActionBar()`, which returns an ActionBar object. With this object one can call the ActionBar's function `newTab()` to construct a new Tab object, which then has to be provided with a title via `setText(String)` and in order to react to taps, with a TabListener. In this thesis' application the TabListener class which implements the ActionBar's TabListener interface and manages its own tab and the respective fragment. The reference to the TabListeners are saved in the main activity, such that it can programmatically select or reselect a specific tab to notify changes or just to switch the tab. Once the data is provided, one can add the tabs to the ActionBar with a call of the ActionBar's function `addTab(Tab)`.

Action bar option buttons

Option buttons for the action bar are declared in an xml file. In this application the xml nodes representing the buttons by four key value pairs. The first entry is an id, which is used to refer to the button from within the programs code. The title of the button is given in the field of the same name. A priority number to define the order of appearance of the buttons and an option to define if the button should be grouped under the three dots list or if it should directly be accessible on the action bar.
The main activity then has to override the function `onOptionsItemSelcted()` which gets the tapped button as input. With the previously assigned ids, one can differentiate between the four buttons and execute their respective functions which will be explained in the following.

Switch user

The "Switch User" button allows the user to switch the profile accessing the visualized data. If this button is tapped, the main activity looks up if the user had stored his or her login information and deletes them. Then the underlying dataset function `createNewUser()` is called and the user is asked to provide new

login data. The mentioned data set and its corresponding functions will be explained in section 3.4.

Delete files

"Delete Files" gives the user the ability to regain occupied memory storage by deleting saved files. The main activity looks up the applications file directory and deletes any stored file, excluding the user login data. Those stored files are downloaded information about days along with colorings of displayed applications and the selection of ignored applications.

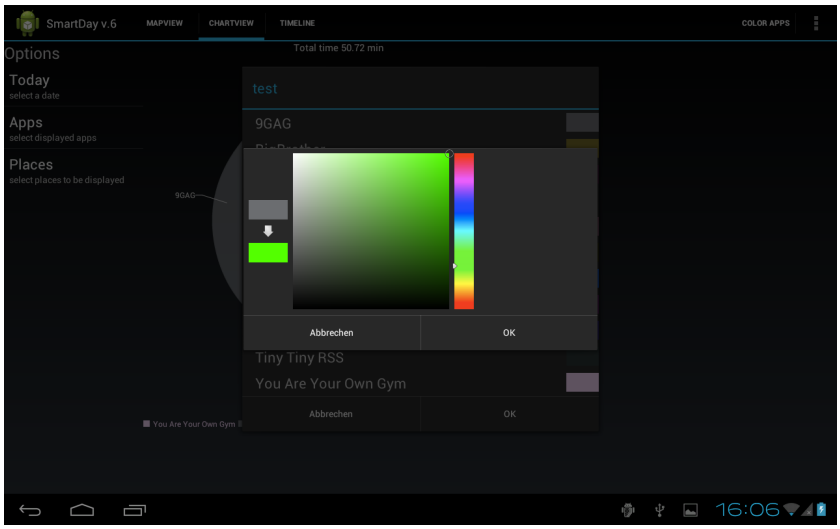
Ignore applications

In order to permanently remove applications from selections and views, one can tap the "Ignore Apps" button. This causes the main application to request every used application from the dataset by calling `getAllApps()`. After the dataset has downloaded all application names from the server, the main activity creates a dialog where the user can pick applications which will be hidden permanently. The selection will then be stored in the dataset and all views will be updated.

Color apps

The button which is visible as long as the screen sizes offers enough space, is the "Color Apps" button. Tapping this button causes the main activity to create a dialog displaying application names with respectively colored rectangles next to them. A tap on the application name or rectangle unveils a new pop up menu which lets the user pick a color as seen in figure 3.6.

Figure 3.6.: Color picker



This new pop up is built with Yuku Sugianto's Android Color Picker or the Indonesian translation Ambil Warna[10]. To use this color picker, one has to include the source code into its own application code and define it as a library project in order to access functions and classes. Once the Ambil Warna was linked

as a library project, the application uses it as follows. As soon as an application name or the respective rectangle is tapped, the `onClickListener` extracts its own title, which is the applications name and creates an `OnAmbilWarnaListener` which overwrites the functions `onOk()` and `onCancel()` which are listeners and get called if their respective button is tapped. If `onOk()` is called, the function stores the color and respective application name and afterwards updates the color of the rectangle.

The AmbilWarnaDialog

After the construction of the `OnAmbilWarnaListener`, the `AmbilWarnaDialog` will be created by passing the current activity, the start color and the listener. With a call of `show()` the dialog is displayed and the user can interact with it. Once the user has picked all colors and hits ok, the dialog stores the new colors in the data set which then notifies the main activity to update its fragments. If the user decides that he or she does not want to apply the changes and taps cancel, all changes are discarded.

The basic layout is an essential part of the application experience. It offers not only a structured layout for view corresponding options and general ones, but it also presents a surrounding structure for views themselves. The possibility to seamlessly attach new views with tab headers and options on the left option fragment without destroying the uniform appearance of the application may be an advantage for possible future projects.
The basic layout was created to be minimal in terms of displayed options in order to prevent distraction from the actual function of this application, the self reflection.

3.4. Data Management

In the last section the term of a data set was mentioned a few times. This data set or the managing class of it will be explained in this section. It is the essential for the whole application, as it provides every fragment with the needed user data. The construction of this class was one of the hardest part of developing SmartDay and pieces of it were rewritten or removed as the development of the different fragments proceeded. Especially notifications for the main activity in order to update views if new data was available have been adjusted a few times.

JSON format

The application's underlying dataset is stored in a *JavScript Object Notation*, in short JSON, format. JSON structures consist of key:value pairs, where the value can be a number, string, object or array. Keys and strings are enclosed by quotation marks and the key is separated from the value by a colon. An object is enclosed by curly braces and contains a collection of key:value pairs, each separated with a comma. An array is a collection of objects, each separated by a comma and enclosed by square brackets. JSON offered an easy way to handle the data provided by the server and was also natively supported by Android. The following sample describes the actual structure of the JSON object which is primarily used by the application and can be seen in figure 3.7.

JSON object in SmartDay

In the top level object, the structure holds informations about download and date time stamp, along with the total time of application usages. In this application, all information about time are given in seconds precision. The JSON object holds an array called result and contains a JSON object for every application used at the specific day. Each of these objects contains a field app with the application's name stored as a string and a field duration, containing the total duration time of the application. Each object also contains an array *usage*. This array contains objects of every timespan in which the application was used. Every usage object contains keys start and end with respective time stamps for start and end time. In addition the field session contains the applications session id and an array *location* is given which contains two objects. These objects contain a key:value pair key which identifies whether the object holds information about the longitude or latitude, by storing a string with content lng respectively lat. The actual position information is stored in the value field which stores it as double.
With these information one has all needed data to create the fragments reflecting one's daily activities.

Figure 3.7.: The Basic JSON Object

```
{ "dateTimestamp": long,
  "downloadTimestamp": long,
  "totalDuration": long,
  "result":[
    { "app": string,
      "duration": long,
      "usage":[
          { "session": string,
            "start": long,
            "end": long,
            "location":[
                 {"key": "lat",
                  "value": long
                 },
                 {"key": "lng",
                  "value": long
                 }
            ]}, ...
      ]}, ...
  ]
}
```

DataSet initialization

The data managing class *DataSet* is designed to be a singleton to guarantee that every view and fragment of the application is provided with the same data. In order to obtain a `DataSet` instance, one can call `getInstance()` which returns an instance of `DataSet` or initializes one and returns it, if none exists. In the process of initialization, the instance requests a new object of the class `UserData` which holds the user name and password. While the `UserData` object is created, it is directly checked, if the data is correct by contacting the server, but this will be explained later in this section. After requesting user data, it is checked, whether data for colorization and for ignoring applications are stored locally and can be loaded or not. If files are available, they are read and respective JSON objects are created and stored. After `UserData` calls `DataSet` to pass a successfully created and tested user object it is stored and the date is initialized. Since the application provides support to display data of multiple days in one fragment, the current date is determined, stored and selected as start and end date, meaning that at the time of initialization only one date, the current day is selected. Storing of the current date is only necessary to check if a chosen date equals the current date and thus label it as “Today”.

Once the initialization part and therefore the creation of the `DataSet` instance is completed, the instance calls its own function `getApps(Listener)` with `null` as listener. This function requests a download of data of the selected day and the reason for the call with `null` is, that in this way the main activity is notified that the data is available for first time and it can set up the fragments for the first time which need the data.

Creating a new user

As mentioned, the a new user object is requested from `UserData`. This is done by calling `getUserLoginData`. This function is provided with the main activity and an optional boolean as parameters. The main activity is needed to for different functionalities like storing data in a local file or accessing the Internet, which is explained later. The boolean can be set to true if one wants to create a new user, ignoring and deleting stored user files. This is needed if one wants to switch the user login data the application uses. In case no stored user data is available, the application shows a dialog in which one can type in user name and password. In addition, one can check a box to tell the application, that his or her should be saved. Once the user hits ok, the data will be stored according to the check box and then the login data will be tested. If the server's reply to the test is positive the user data is passed to the `DataSet`, else the dialog is shown again.

Asynchronous tasks

To communicate with a server one has to note a few things. First, the communication has to be executed on a separate thread. One does not want the main thread to handle the download because waiting for a server response may occupy the thread for a few seconds and thus other actions are not possible during this time. And due to the use of an asynchronous thread one should ensure that at some point the downloaded data gets passed to the requester in order to be able to display the data.
Android provides a good solution to handle short usages of asynchronous tasks. The `AsyncTask` api offers the ability to easily create and maintain a separate thread. To use its functionality, one has to extend `AsyncTask<Params,Progress,Result>`, define the three generic types and overwrite the following functions.`onPreExecute` gets called on the main thread and is used in this application to bring up an dialog, showing that something like the testing of user data is done right at the moment. `doInBackground` is the actual function which runs on the new thread. It gets Params as input and returns Result. In this application the server access and the downloading of data are implemented into this function. The last function`onPostExecute` is called after the asynchronous tasks is finished. It is used to

remove the dialog which was set up in `onPreExecute` and processes Result which was returned by `doInBackground`.
Once the class has been implemented, one can instantiating it and then call `execute(Params)` to execute the asynchronous task.

Now the use of asynchronous tasks has been clarified, Server access

3.5. Mapview

3.6. Chartview

3.7. Timeline

Chapter 4 Evaluation

Part III

Chapter 5 Related Work

Chapter 6 Conclusion

6.1. Review

Appendix A Bibliography

Printed References

[7] Sarah Haller. "Pektorale. Eine Studie zur Form, Bedeutung, Verwendung und Praxis anhand der Dahschur-Pektorale aus dem Mittleren Reich des Grabkomplexes Sesostris' III.". BA thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, 2013.

[8] Thomas Honné. "Interactive Visualizations of Activity Patterns in Learning Environments". MA thesis. RWTH Aachen University, 2013.

[9] Thorsten Kammer. "Capturing Activity Context in Mobile Learning Environments". MA thesis. RWTH Aachen University, 2013.

Online References

[1] Stefan Beiersmann. *Android erreicht 70 Prozent Marktanteil in Europa*. URL: `http://www.zdnet.de/88160639/android-erreicht-70-prozent-marktanteil-in-europa/` (visited on 14/08/2013).

[2] *Big Brother Application*. URL: `http://learning-context.de/text/3/Collectors` (visited on 13/08/2013).

[3] Google. *Development Tutorials*. URL: `http://developer.android.com/training/index.html` (visited on 15/08/2013).

[4] Google. *Distribution of Android Versions*. URL: `http://developer.android.com/about/dashboards/index.html` (visited on 14/08/2013).

[5] Google. *Latitude discontinued*. URL: `https://support.google.com/gmm/answer/3001634` (visited on 12/08/2013).

[6] Google. *Unser mobiler Planet: Deutschland. Der mobile Nutzer*. May 2012. URL: `http://services.google.com/fh/files/blogs/our_mobile_planet_germany_de.pdf` (visited on 06/05/2013).

[10] Yuku Sugianto. *Android Color Picker*. URL: `https://code.google.com/p/android-color-picker/` (visited on 23/08/2013).