Distributed Systems – Assignment 1

Robin Guldener
ETH ID 11-930-369
robing@student.ethz.ch

Nico Previtali
ETH ID 11-926-433
pnico@student.ethz.ch

Lukas Bischofberger
ETH ID 11-915-907
lukasbi@student.ethz.ch

ABSTRACT

We developed two mobile applications for the Android platform from the ground up for the HTC Desire Nr. 25. We completed all of the tasks and our apps worked without crashes on the device. We also completed the enhancements in the following way: (1) After the device alarm has gone off, we start logging the path the thief walks. (2) For suppressing the alarm we created the possibility to save a friends phone number and continuously send GPS data to it. If no phone number is specified we try to acquire to phone's default email account and send the GPS data to this.

1. INTRODUCTION

For this assignment we implemented two mobile applications on the Android platform using the Android Developer Tools based on Eclipse[3]. For two out of three team members this was the first time working with Android and subesequently the majority of the time spent on the project was devoted to reading the Android API Guides[2] and Android Reference[4]. Whilst the documentation material is mostly very well written, there are a few corner cases where different documents describe methods in mutual disagreement and the provided GUI editing tools of the Android Developer Tools have not always worked to our satisfaction.

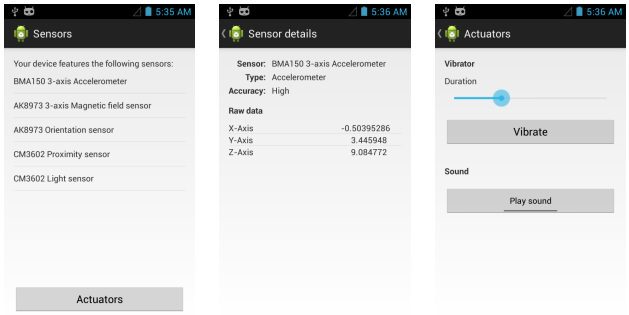
Our team of three was split into two, with Lukas Bischofberger and Nico Previtali implementing the Anti-Theft Alarm and Robin Guldener implementing the Sensing with Android application. The reporting task was also split accordingly with Robin Guldener additionally covering the Introduction and Conclusion parts whilst Nico Previtali and Lukas Bischofberger also described their efforts for the enhancements of the Anti-Theft Alarm.

2. SENSING WITH ANDROID

The Sensing with Android application (SwA) allows any Android device owner to quickly get an overview of all the sensors available on her device and to read raw data from any sensor in realtime. Additionally, SwA also enables the user to quickly explore the actuators of their device using the builtin Actuators Activity, which supports activating a device's builtin vibrator and playing a predefined, aurally pleasant jingle of bells.

In the following paragraphs each Activity will be presented in detail.

The Main Activity (cf. Figure 3(a)) is the sole entry point of the Application and thus also the only activity that is listed in the launcher. It is the heart of the SwA application and displays a ListView containing all the names of the available sensors. This allows for the efficient selection of any particular sensor. To make the interaction with the list easier for humans with thicker fingers or any potential feline users, we have increased the height of a single row in the list beyond the default value. First tests have shown a particular increase in user-satisfaction, which we could directly link to this design decision. One of the main challenges of the Main Activity is how to pass on the information which sensor was selected to the sensor details activity. We have resolved



(a) MainActivity (b) SensorsActivity (c) ActuatorsActivity

Figure 1: Activities of the SwA application. Figure (a) shows an example of the sensors available on the lab-provided HTC desire. Figure (b) shows the readings for a particular time of the device's builtin accelerometer. Figure (c) displays the Actuators Activity.

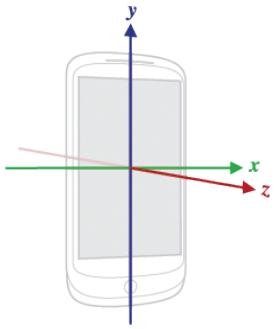
this issue by using the Java provided hashCode function[5] as detailed in Listing 1. We consider this a particularly elegant solution as it does not depend on any, not necessarily unique, sensor names and leverages existing language design features.

```
1 // s holds a reference to the selected Sensor object
2 Intent intent = new Intent(this, SensorActivity.
      class);
3 intent.putExtra("sensor", s.hashCode());
4 this.startActivity(intent);
```

Listing 1: Passing the selected sensor using Java's hashCode method

The Sensors Activity (cf. Figure 3(b)) provides detailed information on the selected sensor such as the sensor type and provides realtime access to both the raw sensor data as well as the current accuracy of the measured data. To populate the ListView displaying the raw data we have implemented a SensorAdapter, which is a sensor aware implementation of the abstract Adapter class that adjusts the data source exposed to the ListView to the particular sensor currently selected. This provides a clean solution to the heterogenous data we receive from the SensorEvent class[6] and allows for maximum compatability even with future sensor types.

Finally the Actuators Activity (cf. Figure 1(c)) employs a simplistic interface to allow the user to interact with the device's builtin vibrator and play a predefined sound file. The duration of the vibration can be conveniently adjusted using a SeekBar and allows vibration durations ranging from 0ms up to 1000ms, enabling the user to explore different kinds of tactile feedback in a minimal amount of time. Great care was also taken when choosing the builtin sound file and we finally settled on a pleasing, yet very well noticeable jingle of bells.



(a) Accelerometer

Figure 2: Accelerometer of the device.

3. THE ANTI-THEFT ALARM

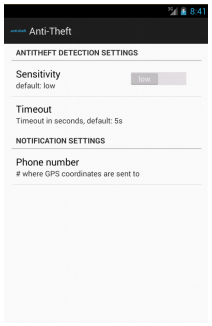
The Anti-Theft Alarm application allows to secure your device against thiefs. In the application you can activate the sensor logic which is then supervising if the device is moved within a certain amout of time. If the device is conitiously moved within this amount of time, the sensor logic triggers the alarm. A sound file then is played and in the background, the application starts to periodically send out GPS coordinates. This GPS coordinates are sent by SMS or otherwise, if the user hasnt denoted a telephon number by mail. All the necessarily settings can comfortably set by click on the settings button of the device.

To detect when the device is stolen we used the accelerometer sensor to recognize movements. The accelerometer measures the acceleratoin of the three dimensions of space (cf. Figure 2(a)). For each of the dimension, we measured the acceleration and compare them to some threshold. This was necessarily because the sensor registers very small changes in acceleration. Certainly we do not want to monitor very small changes and trigger fake alarm.

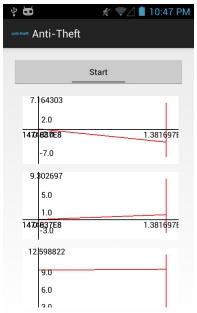
```
1  float threshold = 0.2;
2  float x = event.values[0];
3  float y = event.values[1];
4  float z = event.values[2];
5
6  float diffX = Math.abs(Math.abs(oldX) - Math.abs(x))
       ;
7  float diffY = Math.abs(Math.abs(oldY) - Math.abs(y))
       ;
8  float diffZ = Math.abs(Math.abs(oldZ) - Math.abs(z))
       ;
9
10 // only issue when the the sensor values aren't
       within a certain threshold
11 if (diffX >= threshold || diffY >= threshold ||
       diffZ >= threshold) {
12   ...
13 }
```

Listing 2: Threshold the retrieved data to only detect wilful movements.

The main part of the sensor logic is the service which runs in background. This service registers the movements and triggers the alarm even if the application is closed or the device is locked. The service is started when clicking on the togglebutton and stopped when disarming the device or deactivating the Anti-Theft application. Movements are recognized by the MovementDetector class. This class holds all the methods invoked by the system when the observed sensor is being changed. To avoid an alarm upon non-deliberate movements we defined a timestamp after a sensor change and waited for a second change within five seconds. Then if a third change at least five seconds after the first occurrence was detected, the alarm gets started. Obviously the user is able to disable the alarm for a self-specified amount of time after the detection.



(a) PreferenceActivity



(b) MainActivity

Figure 3: Activities of the AT application. Figure (a) shows the preference activity where all settings are saved. Figure (b) shows the graph with the sensor data after an alarm has gone off.

```
1 // register accelerometer listener
2 movementDtr = new MovementDetector(this, this);
3 sensorManager.registerListener(movementDtr,
      accelerometer, SensorManager.
      SENSOR_DELAY_NORMAL);
```

Listing 3: MovementDetector

The settings of the application are passed to the service by using the Android standard way of passing data between intents. This extra data is then retrieved by the service in the onStartCommand method. When the service triggers the alarm an ongoing notification is notified. This notification can not be cleared by the user. If the users clicks on it, it will forward him to the MainActivity where he can disarm the alarm.

```
1 antitheft.putExtra("sensitivity", sharedPrefs.
      getBoolean("sensitivity", false));
2 antitheft.putExtra("timeout", sharedPrefs.getString(
      "timeout", "5"));
3 antitheft.putExtra("number", sharedPrefs.getString("
      phone", null));
```

Listing 4: Passing data to the service

4. ENHANCEMENTS

The application visualizes each dimension of the recorded data. The simple curves in the two dimensional space. The x-axis is the time and the y-axis is the sensors dimension. This will mean, that the first plot is the plot of the recorded sensor data in the X dimenson, the seconds plot the one of the Y dimension and the last one in the Z dimension. To pass the data around the application we used a class instantiated with the Singleton design pattern. This class, called AccelDataSet stores all the recorded data. The plots are done using a class open source provided by Ankit Srivastava[1]. This is a simple class with takes arguments for the x and y axis and then plots the data with a curve.

```
1 plot2d graphX = new plot2d(this, time, xValues, 1);
2 plot2d graphY = new plot2d(this, time, yValues, 1);
3 plot2d graphZ = new plot2d(this, time, zValues, 1);
4 viewgroup.addView(graphX, params);
5 viewgroup.addView(graphY, params);
6 viewgroup.addView(graphZ, params);
```

Listing 5: Using the plot2d class provided by Ankit Srivastava

Furthermore we implemented a possibility to alarm the owner of the device through someone else's phone. That means that if an alarm goes off, the app sends a text message to a specified number. Or if there is no number specified

we tried to implement the possibility to send an email to the phone owner's email address.
Simultaneously in the background when the alarm starts the application starts to read out the localization data of the device. This is continiously done defined by some interval. This messages containing the GPS data are issued asynchron so that the user can interact with the device meanwhile sending SMS in the background. The service will stop sending the GPS coordinates when the users disarms the alarm or stopps the application. Unfortunatly we couldn't test that implementation due to lack of a sim card or missing account on our phone.

5. CONCLUSION

Finally we would like to summarize the main challenges we encountered when implementing assignment 1 and reflect on our key learnings. Clearly the main challenge was getting familiar with the Android platform and understanding its key underlying principles. Whilst we were quickly able to understand the distinction between processes, activities and services, getting used to the GUI layering and figuring out the connection between Layouts, Widgets and their data providers such as Adapters proved much more time consuming than originally anticipated. Additionally the asynchronous nature of the Anti-Theft alarm meant we also had to understand asynchronous function calls and Threading, concepts which can usually be ignored by beginners. Overall we feel we now have a good understanding of the platform and look forward to deepening our knowledge in future assignments.

6. REFERENCES

[1] Android 2D Plot Class. `http://www.ankitsrivastava.net/2012/03/a-simple-2d-plot-class-for-android/`. Accessed on 13 Oct 2013.

[2] Android API Guides. `http://developer.android.com/guide/components/index.html`. Accessed on 13 Oct 2013.

[3] Android Developer Tools. `http://developer.android.com/sdk/index.html`. Accessed on 13 Oct 2013.

[4] Android Reference. `http://developer.android.com/reference/packages.html`. Accessed on 13 Oct 2013.

[5] Java Documentation for Object. `http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html`. Accessed on 13 Oct 2013.

[6] SensorEvent API Reference. `http://developer.android.com/reference/android/hardware/SensorEvent.html`. Accessed on 13 Oct 2013.