Distributed Systems – Assignment 1

Frederik Rothenberger
frothenb
frothenb@student.ethz.ch

Simon Wehrli
siwehrli
siwehrli@student.ethz.ch

ABSTRACT
This report describes the two Android apps programmed by Simon Wehrli and Frederik Rothenberger. The first app reads all sensor data available on the running Android device and presents it graphically to the user. The second apps intent is to detect a theft that takes the phone when the owner leaves it somewhere temporarily. This is solved by tracking the movements and warning the user with a sound and an optional text message if a deliberate movement occurs.

INTRODUCTION

Both apps are divided into two subcomponents. A graphical user interface let the user choose his preferred settings and a show a graph of the result of the calculation with the sensor data. A second subcomponent is the actual sensor logic. In the Anti-Theft app this logic is separately running in a Service.

We first present the different components with a focus on important design decisions and then discuss the enhancements and problems encountered during the development process.

VS_SIWEHRLI_SENSORS (APP 1)

Activities

Because this app is relatively simple, it only has three activities. There is the main activity, which presents you with the list of sensors to read the values from and a button to switch to the actuators activity.

The actuators activity is there because of point six of the first task of assignment one and its only purposes are to present two buttons, one for vibration and one to play a sound. Additionally there is also a seek bar to set the duration of the vibration.

Main Activity

The main activity consists of a *ListView* and a button. The *ListView* gets its content from an *ArrayAdapter* which in turn is provided with the list of sensors from a *SensorManager*.
To figure out how to fill a *ListView* was basically the only difficulty that we encountered while implementing this activity.

This activity of the first App was the hardest to implement, mainly due to the fact that we decided to plot the values in a graph (so that we could reuse the code for task 3).

The activity itself consists of 6 *TextViews* and one *SurfaceView*. Filling the text views with actual sensor data was done via a *SensorListener* in the *onSensorChanged* method and did not pose a problem. The tricky part was getting the *SurfaceView* to draw a graph or rather how to get to the Canvas used for drawing onto said *SurfaceView*. We needed quite some time to realize that we first had to get the *SurfaceViewHolder* and could then get the Canvas from there. Continuing from there was a lot easier again.

One additional problem did show up later: We first (mis)used the *onSensorChange* method for initiating a call to the draw method used to draw a new picture because we figured the normal refresh rate of the accuracy sensor was enough for us. We later then realized that this refresh rate was neither consistent (it was much slower in the second app) nor did every sensor refresh at the same speed. The proximity sensor was the most extreme in this regard: it only has two discrete values. So it only called our draw method if you waved your hand in front of the phone.

So we then decided to start another Thread dedicated to drawing on the *SurfaceView* which solved these problems.

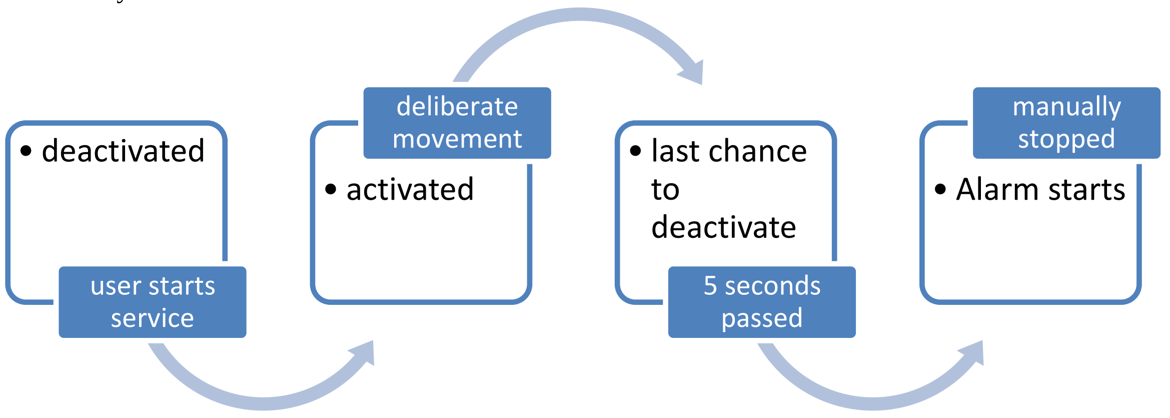

Figure 1. State transition diagram of the Anti-Theft service

VS_SIWEHRLI_ANTITHEFT (APP 2)

Main Activity

The main activity provides a user interface to enter the settings before activating the tracking. The settings get stored via *SharedPreferences* to provide a smooth user experience when returning back to the main activity, either by restarting the app or via the notification displayed during tracking. The settings get passed to the *AntiTheftService* via extra data fields attached to the intent.

Anti-Theft Service

We decided to use an unbounded service which lives, once started with the necessary user settings, in the background and continues working if the main activity gets destroyed by the OS.

The Service can be in different states, shown in Figure 1. The transitions are guided by the handlers reacting on sensor measurement and the passed time. Note that it's still possible to stop the service when the alarm started, because this is a requirement of the exercise definition.

The actual sensor logic is build stable regarding measurement failures, different device hardware and accidental movements. Let's look a bit more closely at the implementation details. The following list motivates the choices for the different steps of the algorithm.

- After each measurement, we calculate the absolute difference to the previously measured norm of the three component accelerometer sensor data. This is less sensitive than using the accelerometer values directly, which we noted, are slightly hardware dependent.
- Now we would like to have an algorithm that checks, if a certain percentage of the norms calculated in the last five seconds are over a certain threshold. Because this would require quit some space to save all the time points, we made an approximation of this.
- We try to detect when the phone is not moved or just accidentally by the owner. We conclude this, if more than a certain amount *n* of not deliberate movements occurs. Then we set a so called checkpoint there, simply a point in time. We introduced this amount *n*, because we don't want to set the checkpoint if we detect only a single norm close to zero, for example when the thief stops walking from time to time. The alarm starts, if we reach a point, where the last checkpoint is older than the current time minus the five seconds tolerance.

We tested this algorithm with real people and made the observation that it worked like we intended.

Enhancements

In the main view, the sensitivity can be adjusted and be checked through a graphical representation of the measured sensor data in real time to check if the chosen sensitivity is appropriate for the environment (e.g. if you're currently in a train). In addition, a user-defined timeout to still quietly disarm the device and an optional phone number to inform a friend can be entered to ensure that you notice if your phone gets stolen.

GENERAL DIFFICULTIES

Throughout this assignment we mostly encountered difficulties arising from not fully understanding the android framework rather than program complexity. This was at times frustrating and in most cases due to the many abstraction layers of the Android Framework (e.g. using a *SurfaceViewHolder* to draw onto the Canvas of a *SurfaceView* or using an *ArrayAdapter* to fill a *ListView* with elements).

One example is worth writing down in more detail. During solving this problem, we found a rather surprising design decision made by the android developers. When reading an array of values out of a sensor, say by something similar to *event.values*, then the way the android framework pass you the value array changes depending on the screen state of your device! If the screen is off, it accidentally does not pass a reference on a copy of the framework intern value array anymore, but a reference on the mutable data array. Hence the values suddenly changes under your reference to newly measured sensor data. In our first version of the sensor logic, this led to a change of zero in the norm of the values, as indicated in Code 1.

```
float[] oldValues;
public void
       onSensorChanged(SensorEvent
       event) {
  // calculate the norm of the change
  float change = (event.values[0] -
         oldvalues[0])^2 + ...;
  change = Math.sqrt(change);
  // bug! change == 0
  oldValues = event.values;

}
```

Code 1. Values passed to handler are not copied if screen of device is off and lead to unexpected dependencies

WHO DID WHAT

Frederik developed the Sensors app and the graphical visualization for both apps while Simon developed the rest of the Anti-Theft app. All important algorithms were discussed and evaluated as a team.

CONCLUSION

We really dived into the Android platform and liked its mostly clean design and well documented high-level objects. But there are often many levels to understand which adds a lot of complexity and makes the learning curve steep. But finally we managed to get everything as we wanted it.