Sensordata

An android-application to read, store and display compressed sensordata

Florian Kraemer

Contents

1 Objective

Smartphones offer a wide range of use. We want to use their fair processing power, storage capacity and attached sensors to allow easy handling of sensor networks. Thus the task is to develop an android application that is capable of receiving data from sensor nodes via the telecommunication network, processing, storing and displaying them in a significant way. Furthermore managing the metadata of each node on a phone will make it user-friendly to find nodes, alter positions in an existing sensor-network or install new nodes. Challenges are to keep the application as modular as possible, to allow changes to the supported sensor-network, like for example the number of sensors, the number of platforms or switching between compression algorithms. The smartphone is also not intended for storing the data forever. Possibly having smartphones in the field and their hardware capabilities requires to backup the data regularly. But connecting to a web-service would also allow for using for example a tablet just for displaying data. It is also important to make the application usable on different devices featuring different screen sizes and processing power or telecomunication features.

1.1 The supported sensor-platform

The current release of the application supports a platform with four sensors, each having two sub-sensors, one for moisture- and one for temperature measurements. An additional sensor is used to collect battery data. The data is then compressed with a Huffman-Code, a method developed by Rachel Cardell-Olivier, and transmitted in a SMS-message.

TODO insert footnote

2 Architecture

2.1 Overview

The application features several activities for the different types of user-interactions.
Then there is an underlying service, that handles the background data-processing. It reads, decompresses and stores the sensordata from the phone's SMS-inbox. It also handles which numbers are interpreted to be platforms and to which platform in the database they should correspond Besides this it also provides the connection between the activities and the database.

2.2 Data model

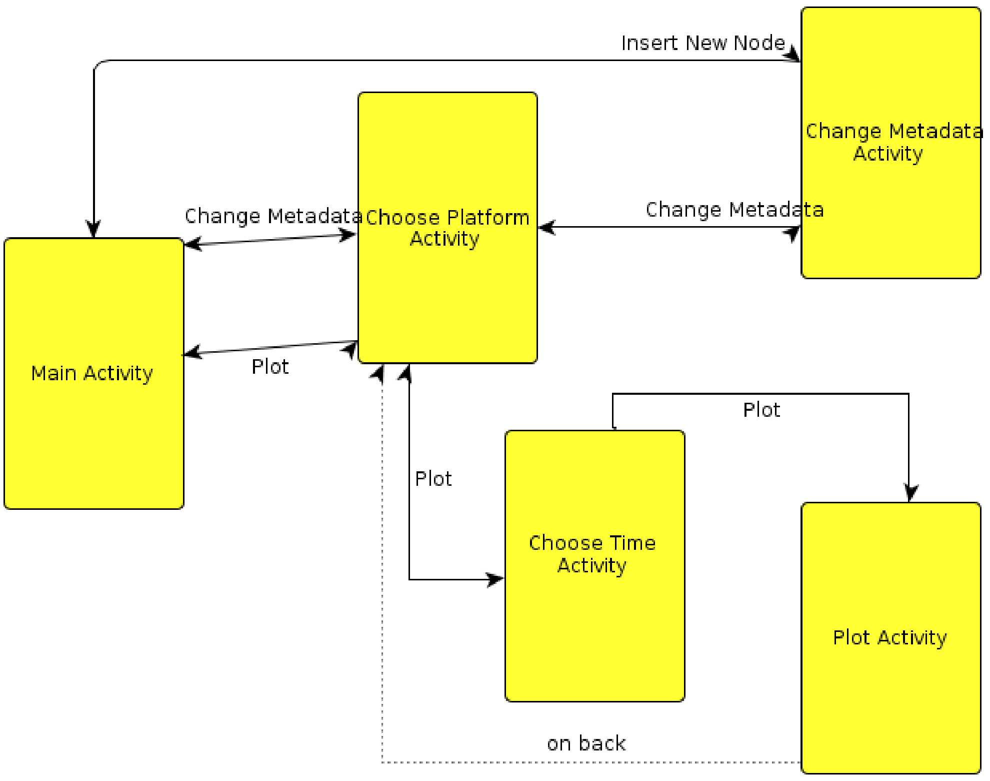
An existing data model FOOTNOTE was reduced to fit the specifications of a smartphone application. This reduces the amount of choices the user has to make in order to operate and keeps the complexity to a certain level. It also makes the database as lightweight as possible. The measurement entity-type features no dedicated id, but instead the timestamp and subsensor id are used as a unique primary key.

Entity-type	Attributes
platform	id,description,period,latitude,longitude,mobile number
sensor	id,latitude offset, longitude offset, elevation offset, platform id
subsensor	id, phenomena id, sensor id
measurement	timestamp, measurement, subsensor id
phenomena	id, description, unit, maximum, minimum

3 Implemention

3.1 Activities

The current release uses just five activities. This is possible due to recycling of two of them: Depending on the intent they are started with, the choose-a-platform-activity and the change-a-platform-activity react differently. This makes sense, since very similar actions like altering and inserting new metadata for a platform require the same layout and it prevents duplicate code.



Graph of activities

3.1.1 Main

3.1.2 Choose Platform

3.1.3 Change Platform

3.1.4 Choose Time

3.2 Plot data

3.3 Plotting

To visualize the data the library androidplot is used. It offers a wide range of preferences, but a rather sparse API. The biggest challenge here is to find the limits for displaying as much data as possible while still making it wel-readable and handable for the user. Thus it might just be a well-sized extract from the chosen data. Since the plots are implementing a touch functionality, the shown ranges are limitted with maximum and minimum values, so one can not get lost in the depth of the graph. To make the touch experience a bit smoother, a thread was implemented, that slows the scrolling and zooming exponentially down after the touch gesture, instead of stopping immediately. To not allow the view to be scrolled on smaller screen sizes a custom scrollview was implemented, that allows to block scrolling the view while gestures are made on the plots.

3.4 Thread safety

The android platform comes with a mannerism, that necessitates special handling at times. The most important constraint is that changes to the views can only be done from the UI-thread. When using multithreading thus attention must be paid to this. The android API provides the AsyncTask, its onPostExecute is executed in the UI-thread and can therefore be used to meet this challenge. In the current release CountdownLatches are used to prevent from problems upon binding to the underlying service. This process takes some time and when put for example in the OnStart-method of an activity, it is not reliably executed at the expected time.