MORA - A Sensor Data Analysis Toolkit for Mobile Phones

Heinrich Hartmann
University Koblenz

Christoph Schaefer
University Koblenz

Matthias Thimm
University Koblenz

Abstract

1. Introduction

- Much sensor data from smartphones available.
- Problem: Extract sensible information from raw data. E.g. for - Quantified Self - Internet of Things
- Need tools for gathering test data and evaluation of algorithms: MORA.

2. MORA Toolkit

The MORA toolkit provides a simple mean to record, transfer, store and analyze sensor data from Android mobile phones. In the following sections we given an overview about the overall architecture and discuss the design choices of the individual components.

Design goals of the MORA toolkit:

- Simple codebase. New developers should be able to contribute easily.
- Low CPU usage on the mobile device (Battery efficiency, and allow high frequency recordings).
- Data visualization and export tools included, to enable data mining and analysis in later steps.
- Privacy awareness. Give the users control over their data.
- Provide plugin interface e.g. for mobile data mining.
- Reuse of existing standards and technologies. JSON. Postgres. NodeJS.

The Toolkit consists of three components:

- **Sensor Collection App**. Records great varity of sensors. Flexible event loop/dispatcher - design. Essentially single threaded. Plugins storage, bulk transfer and streaming built-in.
- **Sensor Storage Service**. Receives data over HTTP or ZeroMQ, inserts into Database and File System.
- **Data Inspection Service**. Gives convenient view on data stored in database. Allows users to keep control over their data.

Flexible serialization format **Json Sensor Format**.
* How are the goals achieved?

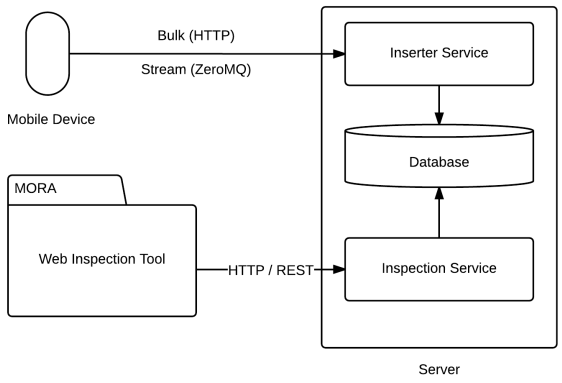

Figure 1. MORA System Overview

2.1. Mobile Sensor Collector

The Mobile Sensor Collector is an Andorid App, that allows the recording of data from a varity of sensors avaialble on modern smart phones. It consists of a background service that performs the actual recording, and a GUI activity (cf. Figure 2) that allows convenient usage of the service. A list of supported sensor types can be found in Table 1. It is also possible to integrate the sensor collection service into other applications using an API.

In the most basic use case the sensor collection service receives sensor samples form the Android system, serializes them to the JSF format described in Section 2.3 and stores the serialized samples on the local file system. On request the stored samples are transferred via HTTP to a server running the Sensor Storage Service described in Section 2.4.

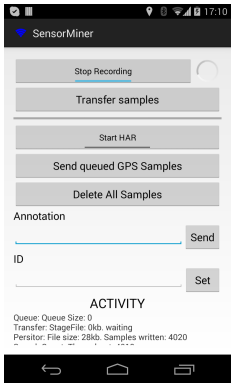

Figure 2. Mobile Sensor Collector GUI

Sensor Name	Type Prefix	Frequency
Accelerometer	ACC	50 Hz
Gyroscope	GYR	50 Hz
Magnetometer	MAG	10 Hz
WiFi	WIFI	every 5 sec.
GSM	GSM	every 5 sec.
Bluetooth	BLT	every 5 sec.
GPS	GPS	every 5 sec.

Table 1. Supported Sensor Types

In addition to this basic functionality the service allows direct streaming using the ZeroMQ messaging library and arbitrary processing of the recorded samples using a plugin interface. As an example, we have used such a plugin to detect human activitites (e.g. running, sitting, standing) from accelerometer data (cf. Section 3.1).

The general arichtecture is inspired by event logging and aggregation frameworks like rsyslog or logstash. We view incoming sensor samples as events that we can persist in a log file, process or stream to a remote location. Simplicity and performance were the main design goals of the application. A slim architecture should avoid the loss of sensor data due to processing overhead and allow new developers to understand the codebase quickly. In particular, we avoid the use a database on the mobile device but write directly to a flat file.

2.2. The JSON Sensor Format (JSF)

Data transfer is one of the key features offered by the MORA toolkit, therefore we have spent considerbale effort in optimizing our serialization format. After experiments with XML, CSV and the Common Log Format we finnaly settled for a JSON based format, which we call *JSON Sensor Format* (JSF).

The file format is human readable, so it can be quickly inspected for debugging. At the same time verbosity is limited, avoiding too much transfer-overhead. The adherence to the JSON standard allows the reuse of existing parsers, validators and scripting tools, which is very important for us. The format is line-based and stateless which allws streaming over sockets or message queues, where lines can be dropped or even interleaved with lines from other recordings, without being invalidated.

An example JSF file looks as follows

```
["ACC", 1410360212231, "Max", [0.1,0.5,-9.8]]
["TAG", 1410360213142, "Max", "Riding bus."]
```

The first field contains a sensor type identifier (e.g. “ACC” referres to the accelerometer sensor). The second field is the unix timestamp of the recorded event in milliseconds. The third field contains a user name. The last field is is a general JSON object, that represents the value of the recorded sensor event.

2.3. Sensor Storage Service

The sensor storage service allows to gather and persist the recorded sensor samples on a central location. To do so it opens an HTTP endpoint that accepts POST requests with “multipart/form-data” attachment, containing sensor data in the JSF format. Valid requests can easily be generated using file upload in HTML forms (cf. RFC1867). The service allows (gzip) compression and (SSL) encryption of the uploaded data. Two implementations of the sensor storage service are included in the MORA toolkit, one as Node.js-application the other Java-Servlet running on an Apache Tomcat 7 webserver.

Once a valid POST request is received, the service stores the file in a dedicated folder on the file system and inserts the data into a PostgreSQL 9.3 database. Only if both insertions completed successfully a *200 OK* response is sent back to the client.

The Node.js implementation of the sensor storage service, comes with a dynamic management of the database scheme. When a JSON file with an unknown sensor type is received a new table is created for this new type which holds the sample value in a JSON column. Note that, the JSON datatype became available only recently with PostgresSQL 9.2, and is currently not supported by any other popular open-source relational database management system (e.g. MySQL, SQLite). This is one of the main reason for our choice of PostgreSQL as storage solution.

2.4. Web Inspection Tool

An important step that should be considered before any data mining task is a manual data exploration phase. For

this purpose, MORA provides a powerful data inspection tool. Using the web-based front end, a first overview of the data integrity can be obtained quickly. Both raw data and mining results can be viewed in the browser. Occasionally appearing transmission errors caused by the interaction of mobile phones and the back end, can be discovered easily by the user. In addition, plots for all sensor types give a rough measure for the quality of the recorded data. With the help of this presentation we could, for example, discover GPS inaccuracies or frequency deviations.

Besides time series plots and map based views of GPS data the inspection tool offers a bar-code-like representation (Figure 3) which is able to display a large number of tags on a time axis.

To get more details all views can be zoomed in and out. On each zoom level the corresponding data of the displayed interval can be exported and downloaded as CSV (Figure 4).

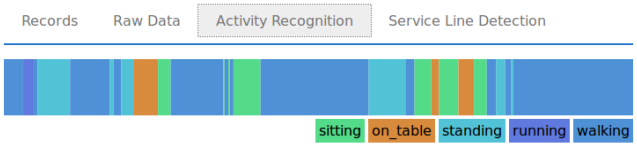

Figure 3. Bar-code-like representation of human activities in the inspection front end.

Via the web interface, all users have access to their recorded data and are entitled to delete their tracks if necessary. Apart from the possibility to carry out small maintenance tasks, this give users the control over their data and thus enhances the privacy.

- Automatic display of available tables - as text (TEXT) or plot (float)

* Mining output has to be stored in db - Can be created on mobile device and inserted into transfer file - Or can be done on the server

3. Practical Examples

3.1. Human Activity Recognition

* Decision tree learning with WEKA * Import classifier as JAVA class into MORA Library

3.2. Service Line Detection

* Web service for SLD * GPS Samples are gathered via MORA lib * Query results are inserted back into MORA * Analysis of classification results via inspection tool

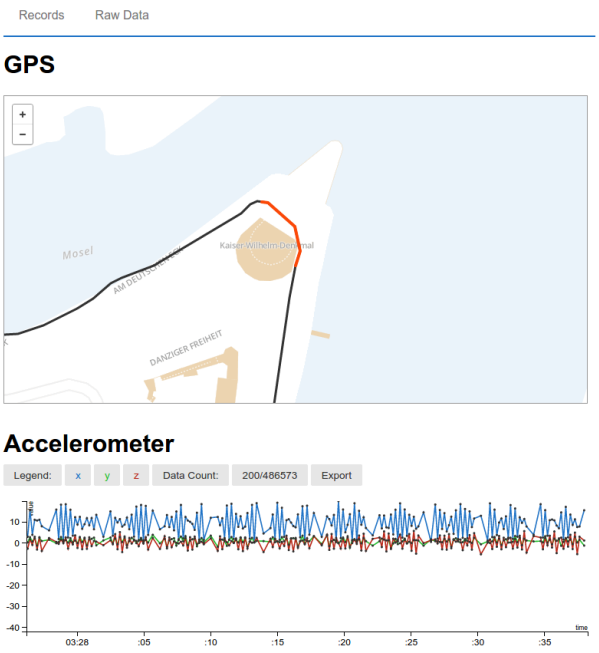

Figure 4. Zoomed in map view and the corresponding accelerometer raw data in the inspection front end.

4. Related Work

* FUNF * SDCF

References

[1] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[2] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.