Open Source Smartphone Libraries for Computational Social Science

Neal Lathia, Kiran K. Rachuri, Cecilia Mascolo
Computer Laboratory
University of Cambridge
Cambridge CB3 0FD, UK
firstname.lastname@cl.cam.ac.uk

George Roussos
Dept. of Computer Science
Birkbeck College
University of London
London WC1E 7HX, UK
g.roussos@bbk.ac.uk

UbiComp '13 Adjunct, Sept 8-12, 2013, Zurich, Switzerland.
ACM 978-1-4503-2139-6/13/09...$15.00.

Abstract
The ubiquity of sensor-rich and computationally powerful smartphones makes them an ideal platform for conducting social and behavioural research. However, building sensor data collection tools remains arduous and challenging: it requires an understanding of the varying sensor programming interfaces as well as the research issues related to building sensor-sampling systems. To alleviate this problem and facilitate the development of social sensing and data collection applications, we are developing a set of open-source smartphone libraries to collect, store and transfer, and query sensor data. Furthermore, we have also developed a library that can trigger notifications based on time or sensor events to assist experience sampling methods. This paper presents these libraries' architecture, initial feedback from developers using it, and a sensing application that we built using them to study daily affect.

Author Keywords
Android, Data Management, Open Source, Smartphone Sensing, Social Psychology

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous.

Introduction

The smartphone is increasingly being viewed as one of the protagonists of future interdisciplinary research that crosses between the information and social sciences [9]. Smartphone applications promise researchers access to large populations of participants who can contribute both sensor, application interaction, and survey response data as they go about their daily lives: this opportunity has led psychologists to hail smartphones as more transformative to their field than desktop computers and brain imaging technology [11]. Beyond data collection, smartphones are now being researched as tools that can implement and test behaviour change theories [5] and, ultimately, promise to become a central mediating point between people and their therapists [7].

While smartphones will undoubtedly permeate throughout interdisciplinary research, their widespread adoption by researchers is still hindered by the technical challenges of building sensing applications. Designing applications that appropriately balance between energy efficiency, data collection, storage, and transmission continues to be both a non-trivial task as well as an item on the sensor-research community's agenda [12].

To address this concern, we have developed a number of independent, open-source Android libraries that seamlessly support (a) collecting sensor data, (b) formatting, storing and asynchronously transmitting the data to remote servers, and (c) configuring notifications to be triggered based on time or sensor-data events. The libraries take a very broad definition of "sensor" encompassing both those that need to be actively queried for data (e.g., an accelerometer) as well as those that give signals of social interaction via the smartphone (e.g., making a phone call); most notably, we have condensed collecting smartphone sensor data down to two lines of code, and have used well-established design patterns to implement a consistent interface to all of these libraries.

In this paper, we describe the architecture of these libraries, and how they may be used by researchers who are building social or sensing applications. We also discuss an application that was built using them to study daily affect [8], which included collecting sensor and survey response data and remotely reconfiguring the experiment after it was deployed, and preliminary feedback that we have received about the libraries by making them available to postgraduate students. We close by discussing related systems, applications, and future directions.

Overview and Design Patterns

In this section, we describe our high-level goals, why we implemented separate libraries, and the design patterns that we used across the libraries.

1. **Data Collection**. Currently, accessing data from different sensors requires writing code that is tailored and variant for each sensor, and there is little support for continuous sensing applications, barring a few sensors such as the accelerometer. Our first goal was therefore to provide uniform and easily configurable access to sensor data, supporting both one-off and continuous sensing scenarios.

2. **Data Management**. To encapsulate the ideas of formatting, storing, querying, and transmitting data, we built a *data manager* that acts as a data sink for sensor-enhanced applications. While the manager directly supports formatting sensor data, it can also store and transmit any data that is identified with an appropriate tag. Since applications may collect

large amounts of data, the manager can also be reconfigured to transmit data asynchronously.

3. **Triggering Interaction**. Building applications that do more than collect data may require notifying the user about appropriate events. For example, studies of daily life may trigger notifications randomly [2] or based on sensor-events [6]. Our final goal was therefore to provide a uniform, consistent, and configurable means of adding triggers to any application.

We opted to implement these requirements as three separate libraries in order to make them lightweight and minimise inter-dependencies. Therefore, for example, an Experience Sampling Method (ESM) application that does not require sensor data from the device does not require importing the data collection and management libraries, whereas an application that focuses purely on the sensing aspect will not need to import triggering methods.

Since a key aspect of designing these libraries was to facilitate using them, by making them as consistent as possible, we decided to use the following design patterns:

1. **Singleton Managers**. To enforce uniformity, control global state (e.g., configuration parameters), and allow threads to service more than one data request, our libraries use a central singleton that mediates all interaction with the library.

2. **Publish-Subscribe Interaction**. Tasks that require background processing were designed as a publish-subscribe message passing system, where applications only need to subscribe to the relevant component and react when data or events are pushed to them. Doing so allowed us to both separate each library's function from the application that requires it, as well as reduce the amount of computation that each library needs to do in those cases where multiple subscribers need the same functionalities.

3. **Key-Value Configuration**. All of the libraries' configuration settings can be set and retrieved as key-value pairs, regardless of their type. Moreover, when non-critical configuration parameters are *not* explicitly set, the libraries fall back on a set of default values.

Given these three design decisions, we now describe each library's unique features.

Sensing Library

As introduced above, we took a very broad definition of "sensor," taking it to mean *any signal that can be unobtrusively captured from the smartphone device*. We decomposed the available sensors into two generic groups:

- **Pull Sensors**: this set includes all sensors that the Android OS does not capture data from until requested to do so by an application. They currently include the accelerometer, location (both coarse, from the nearest cell-tower, and fine, from the GPS), and microphone amplitude levels. They further support querying for Wi-Fi, Bluetooth, and active application scan results, as well as querying the SMS and call logs.

- **Push Sensors**. The Android OS publishes data about particular events that applications can receive: this set of sensors receives this information

```
public class AccelerometerDataSampler
{
  public ArrayList<float[]> getAccelerometerData(long ws)
  {
    ESSensorManager sm = ESSensorManager.getSensorManager(context);
    sm.setSensorConfig(SensorUtils.SENSOR_TYPE_ACCELEROMETER, PullSensorConfig.SENSE_WINDOW_LENGTH_MILLIS, ws);
    AccelerometerData aData = (AccelerometerData) sm.getDataFromSensor(SensorUtils.SENSOR_TYPE_ACCELEROMETER);
    return aData.getSensorReadings();
  }
}
```

Figure 1: An example code to configure and then capture data from the phone's accelerometer sensor using the sensor library. The equivalent code that directly samples from the Android OS would be approximately 30 lines of code. Querying from data from a different sensor would only require changing the `getDataFromSensor()` parameter and return type.

on behalf of an application, and includes: the battery (energy levels), connection state (whether the phone is connected to a Wi-Fi or mobile network and if it is in roaming mode), the proximity sensor (near/far events), screen on/off events, phone calls starting/ending, and SMS sent/received events.

All pull sensors share the fact that continuously sampling from them would entail looping between querying from the sensor (either for a configurable number of cycles or a pre-defined time) and sleeping; continuously polling push sensors cannot be restricted in this way. Most sensors require a particular permission to be added to the application, which users must agree to when installing the app. Naturally, the sensing library cannot and does not capture data from those sensors that the application has not requested permission for.

All control of sensors data collection is done via the `ESSensorManager` singleton; the full library is available online[1]. This instance manages all the sensing threads and starts, stops, and modifies them as required by the application: this ensures that data is only collected when it is required. The singleton supports two kinds of data collection. The first is one-off sampling, which is implemented as a blocking call, and reduces collecting sensor data to two lines of code (Figure 1). For example, a sample of data with the default configuration can be retrieved by getting the singleton `getSensorManager(...)` and then requesting data with `getDataFromSensor()`.

The second is the publish-subscribe model: subscriptions are given a unique identifier which can then be used to pause and unsubscribe them. When collecting data in this way, application developers need to implement two methods. The first is `onDataSensed(...)`, which implements what do to with the data when it has been pushed to the application; the second is `onCrossingLowBatteryThreshold(...)`. Most crucially, this latter method directly exposes and, in doing

[1] https://github.com/nlathia/SensorManager

so, guides the application developer to implement actions that should be taken when the battery is low.

Once sensor data has been captured it is processed. In its most basic, this step converts the data into objects that are passed on to any subscription. However, while doing so, the library also anonymises the data: it puts all telephone numbers through a one-way hash and only returns features of SMS messages (e.g., whether it was sent or received, and how many words it contained). While this may not be suitable for researchers who need non-private data, we set the default to anonymised data.

Finally, researchers may be interested in sampling sensor data more intelligently. Recent research has shown [10] that non-static configuration parameters can be used to maintain data collection quality while saving energy. The sensing library therefore implements the methods proposed in [13], and we aim to extend it to support future research into diverse sensor sampling strategies.

Data Manager Library

The second library, the `ESDataManager`[2], implements all aspects related to formatting, storing, and transmitting the data. In particular, this library differentiates between three kinds of data: *sensor* data, application *error* data (e.g., exception messages), and *extra* data of the form $(tag, data)$. Under the hood, the library maintains a structured file system where data of different types are written to files as lines of `JSON`-formatted data. The manager stores all this data on the phone's USB storage (or SD card). The three key functions of this library are as follows:

- **Data Formatting**. We opted to format sensor data as JavaScript Object Notation (or `JSON`) documents. This allowed for the flexibility of storing a variety of different sensors in a uniform, extensible, and open format.
- **Data Queries**. Given that data is stored in files, querying it could have potentially required a linear traversal of all stored data. To alleviate this, we use both the structure of the file system as well as a unified file-naming policy to indicate when sensor data was created. The only query function we currently support is to retrieve data from a given sensor that is more recent than a given query time stamp.
- **Transfer Policy**. Sensor-enhanced applications may collect increasingly large amounts of data from their users. We implemented three initial policies for data transfer. The first two are, simply: do not transfer (data is only stored on the phone), and transfer immediately. However, since researchers may not need real-time access to what they collect, and transmitting data over 3G connections may bear a cost on participating users, we included a means for asynchronous data transfer. More specifically, all files older than a given, configurable, time span are transferred to an upload directory and compressed. The library then waits for the phone to connect to Wi-Fi and transmits the data; should a Wi-Fi connection not appear in another configurable timespan (by default, above 30 hours), then data is transmitted by any connection available.

This initial implementation does not supporting a full range of queries, transfer policies, or data formats. The library is also currently mainly suited for those

[2] https://github.com/nlathia/SensorDataManager

applications which will transfer the data remotely at some point: a clear future direction includes augmenting it with storage policies for applications that will not do so (i.e., dealing with storage limits). However, it is a first step towards abstracting away from requiring researchers to repeatedly implement these functions, or indeed requiring them to manually transfer data to/from participants' smartphones.

Trigger Library

Finally, the trigger library[3] allows application developers to add tailored alarms to their application. The triggers are grouped into two broad categories:

- **Time-Based**. The Android OS currently supports one-off and interval-based alarms: this library builds on these to support a wider variety of clock-based alarms that can be used in social research applications. In particular, it allows an application to subscribe to an alarm that is instructed to fire N times, selected randomly with particular conditions (i.e., time between and daily bounds on alarms).
- **Sensor-Based**. The trigger manager builds above the sensor library (which directly collects sensor data) for applications that only seek for alarms to be fired in a more configurable way: (a) the data is passed through a binary classifier (e.g., for triggers that fire only when accelerometer data is non-stationary), (b) the alarm may be fired at a given interval and with a given probability (e.g., 50% of the time a user hangs up the phone, fire this alarm), and (c) the alarm may be configured to fire after a particular delay, or when the binary classifier's state has changed (e.g., 5 minutes after non-stationary data, or when the user is stationary again).

We added a number of further configurable constraints on the application's ability to notify the user to complete surveys. For example, survey notifications were not allowed to be sent within two hours of the previous notification. Most importantly, however, is that all triggers must comply with a set of user preferences, which users could edit via the application's menu. These settings include the maximum number of surveys that can be triggered in one day, and the times when users were available to respond (by default, set to 08:00-22:00).

Putting it Together: Emotion Sense

We used the above libraries to build Emotion Sense[4], an Android application that merges the experience sampling method with smartphone sensor data collection, and gives users feedback about how their reported mood compares to the sensor data that their phone can collect about their behaviours. The application uses the sensor library to collect data from the full spectrum of available sensors, asynchronously transfers the formatted data to our servers with the data manager library, and gives the user notifications to complete surveys using the trigger library.

The version of Emotion Sense used in [8] also reconfigured the sensor parameters, surveys and triggers by checking with remote configuration files stored in our server, to allow the app's behaviour to automatically update without the user's involvement. Specifically, the application periodically downloads the configuration files of the ongoing experiment from the remote server and updates each of the previous components. The main advantage of using such a service is that a variety of ESM

[3] https://github.com/nlathia/TriggerManager

[4] http://emotionsense.org/

protocols can be tested by remotely updating the surveys and triggers without the user's involvement.

Preliminary Feedback

To gather some preliminary feedback about the design of the `ESSensorManager`, the sensing library was offered as an optional tool that postgraduate students in a part-time mobile sensing course could use for their final project. These students are, for the most part, studying alongside their careers and so have substantial experience with software development and the Java programming language. The course itself included introductory material on Android programming, and material on mobile and wireless communication, RFID/NFC and location sensing: the students were exposed to the internal functions of these technologies and not just the APIs.

The project entailed building an Android application that samples audio data from the environment, measures the noise pressure, and posts the results alongside location data to COSM (now Xively[5]), an online database that links sensor data to the Web. The sensing library was offered as an optional tool: 12 of the 16 project submissions opted to use it. Further, a feedback survey was offered; we received 7 complete responses. This survey is part of ongoing work to evaluate the quality of the library; in this section, we briefly recount the reported experiences, and how this reflects on the design of the sensing library.

Three respondents did not use the library. They reported having a low level of familiarity with Android programming (on average 2.6 out of 5—where 5 is "very experienced").The two reported reasons for not using the library were: (a) lack of experience, and (b) a lack of support for the Android emulator. For the former reason, learning to use the library was viewed as an additional burden on top of learning about the other APIs required for the project. The four respondents who did use the library reported slightly higher levels of familiarity with Android programming (3.25 out of 5). They further rated how easy it was to include the library in their app (4.75/5), using the library's API (4.5/5), and the overall usability (4.25/5). While, overall, the experiences were positive, this indicates that the libraries remain tailored to those developers who have experience with Android programming.

We also asked both groups to rate their impression about the amount of code they had to write. The average ratings from those using the library was marginally smaller than those who did not use it (2.75 vs. 3). We note that since the students only had to implement one version of their application, either with or without the library, these ratings for code length may not be representative. From the perspective of those marking the projects (note: not the developers of the `ESSensorManager`), those applications using the library had substantially less sensor sampling code than those that did not and, more importantly, their code appeared to be of higher quality, in terms of being designed to handle potential error exceptions that the library can throw when sensing errors are encountered.

Finally, those who used the library were asked if they would like any additional features to be added to them. The requests ranged from adding further data processing (e.g., audio spectrum filters), explicit support, simulated data, or error handling when developers try to use the library with the (currently unsupported) emulator, providing more example code, and further documentation

[5] http://en.wikipedia.org/wiki/Xively

on formatting the data.

Related Projects

There are a limited number of related open-source smartphone sensing frameworks for computational social sciences. Most prominently, the Funf Open Sensing Framework[6] [1] is a sensing and data processing framework for smartphones. Similar to our libraries, it also supports collection, uploading, and configuration of a range of signals (called "probes") from smartphones. However, there are a number of differences between our libraries and Funf. For example, our sensor libraries implement an adaptive sensing scheme [13] to efficiently query data from the phone's pull sensors, as well as the poll model which Funf is based on, with configurable static duty-cycling intervals. Further, our sensor library supports a low battery threshold notification that provides an opportunity to the applications to act sensibly when the battery level is critical. In addition, our libraries support more sensors: microphone, proximity, connection state. Finally, funf does not support a trigger framework to enable notifications, where as the proposed framework supports it through the trigger library.

MyExperience [4] is a prominent and widely known experience sampling tool for smartphones. They too provide libraries[7] for collecting objective and subjective data from users through smartphones. The libraries are only supported on Microsoft Windows Mobile devices. Our proposed libraries, instead, are designed for the Android platform, which currently has much larger user base than that of the Windows Mobile platform. Further, the proposed framework supports more features than MyExperience such as adaptive sensing, data management, and also more sensors: proximity, microphone, Bluetooth, etc. Some other open sensing frameworks include the Mobile Sensing Platform (MSP)[8] [3] and PACO[9]: we leave a full comparison of our system to others as future work.

Conclusions and Future Directions

This paper has introduced the architecture and use of three Android smartphone libraries that have been designed to support social science research applications. The development and evaluation of the libraries is ongoing: using them in a postgraduate course has given initial feedback into their usability and features. Furthermore, we have used the library to build Emotion Sense: a sensor-enhanced experience sampling application which allows users to compare their reported mood to their smartphone's sensor data.

Our goals in developing these libraries have been to facilitate the design and implementation of new sensor-enhanced research applications. By open-sourcing the libraries, we also hope that they will be useful tools for smartphone sensor researchers who may like to contribute their own methods to the research community; naturally, this includes discussing what functionalities may best serve varying research communities. For example, the sensor library does not currently support conditionally querying sensors (e.g., sensing location only after the accelerometer has sensed movement); these functionalities remain in the application-level domain. Moreover, the libraries continue to offer raw sensor data, rather than allowing programmers to directly code with *inferences* (about contexts or activities) on the data. Indeed, finding an appropriate intersection of needs shared between computer

[6] http://funf.org/
[7] http://myexperience.sourceforge.net/
[8] http://seattle.intel-research.net/MSP/
[9] http://code.google.com/p/paco/

and social scientists, to guide future developments for these libraries, requires further discussion and research.

The libraries are now being used to develop a further 4 different applications by researchers in various institutions. In particular, these libraries are being used as part of the *UBHave* project which aims to merge smartphone sensing and behavioural interventions [7, 14].

Acknowledgements

This work was funded by the EPSRC Ubhave: "Ubiquitous and Social Computing for Positive Behaviour Change" project (EP/I032673/1).

References

[1] Aharony, N., Pan, W., Ip, C., Khayal, I., and Pentland, A. The social fmri: measuring, understanding, and designing social mechanisms in the real world. In *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp '11)*, ACM (New York, NY, USA, 2011).

[2] Bolger, N., Davis, A., and Rafaeli, E. Diary Methods: Capturing Life as it is Lived. *Annu. Rev. Psychology* (2003).

[3] Choudhury, T., Borriello, G., Consolvo, S., Haehnel, D., Harrison, B., Hemingway, B., Hightower, J., Klasnja, P. P., Koscher, K., LaMarca, A., Landay, J. A., LeGrand, L., Lester, J., Rahimi, A., Rea, A., and Wyatt, D. The Mobile Sensing Platform: An Embedded Activity Recognition System. *IEEE Pervasive Computing 7*, 2 (2008), 32–41.

[4] Froehlich, J., Chen, M. Y., Consolvo, S., Harrison, B., and L:w anday, J. A. MyExperience: A System for In situ Tracing and Capturing of User Feedback on Mobile Phones. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys '07)*, ACM (2007).

[5] Hekler, E., Klasnja, P., Froehlich, J., and Buman, M. Mind the Theoretical Gap: Interpreting, Using, and Developing Behavioral Theory in HCI Research. In *ACM CHI* (Paris, France, 2013).

[6] Intille, S., Rondoni, J., Kukla, C., Ancona, I., and Bao, L. Context-Aware Experience Sampling. In *ACM CHI Extended Abstracts*, ACM (2003).

[7] Lathia, N., Pejovic, V., Rachuri, K., Mascolo, C., Musolesi, M., and Rentfrow, P. Smartphones for Large-Scale Behaviour Change Interventions. *IEEE Pervasive Computing* (May 2013).

[8] Lathia, N., Rachuri, K. K., Mascolo, C., and Rentfrow, P. J. Contextual dissonance: Design bias in sensor-based experience sampling methods. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (ACM UbiComp '13)*, ACM (2013).

[9] Lazer, D., Pentland, A., Adamic, L., Aral, S., Barabasi, A., Brewer, D., Christakis, N., Contractor, N., Fowler, J., Gutmann, M., Jebara, T., King, G., Macy, M., Roy, D., and Alstyne, M. V. Computational Social Science. *Science 323* (Feb 2009).

[10] Lu, H., Yang, J., Liu, Z., Lane, N., Choudhury, T., and Campbell, A. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys'10)*, ACM (2010).

[11] Miller, G. The Smartphone Psychology Manifesto. *Perspectives on Psychological Science 7*, 221 (2012).

[12] Rachuri, K., Efstatiou, C., Leontiadis, I., Mascolo, C., and Rentfrow, P. METIS: Exploring Mobile Phone Sensing Offloading for Efficiently Supporting Social Sensing Applications. In *IEEE PerCom* (San Diego, USA, 2013).

[13] Rachuri, K. K., Mascolo, C., Musolesi, M., and Rentfrow, P. J. SociableSense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom '11)*, ACM (2011).

[14] Weal, M., Hargood, C., Michaelides, D., Morrison, L., and Yardley, L. Making Online Behavioural Interventions Mobile. In *Digital Research* (Oxford, UK, 2012).