SenSocial: A Middleware for Integrating Online Social Networks and Mobile Sensing Data Streams

Abhinav Mehrotra
School of Computer Science
University of Birmingham, UK
axm514@cs.bham.ac.uk

Veljko Pejovic
School of Computer Science
University of Birmingham, UK
v.pejovic@cs.bham.ac.uk

Mirco Musolesi
School of Computer Science
University of Birmingham, UK
m.musolesi@cs.bham.ac.uk

ABSTRACT

Smartphone sensing enables inference of physical context, while online social networks (OSNs) allow mobile applications to harness users' interpersonal relationships. However, OSNs and smartphone sensing remain disconnected, since obstacles, including the synchronization of mobile sensing and OSN monitoring, inefficiency of smartphone sensors, and privacy concerns, stand in the way of merging the information from these two sources.

In this paper we present the design, implementation and evaluation of SenSocial, a middleware that automates the process of obtaining and joining OSN and physical context data streams for the development of ubiquitous computing applications. SenSocial enables instantiation, management and aggregation of context streams from multiple remote devices. Through micro-benchmarks we show that SenSocial successfully and efficiently captures OSN and mobile sensed data streams. We developed two prototype applications in order to evaluate our middleware and we demonstrate that SenSocial significantly reduces the amount of programming effort needed for building social sensing applications.

Keywords: Mobile sensing, social sensing, ubiquitous computing, mobile middleware.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Wireless Communications; D.2.11 [Software Architectures]: Domain-specific architectures.

General Terms: Design, human factors, performance.

1. INTRODUCTION

The smartphone revolution has marked the beginning of the twenty-first century. Today more than a billion people carry with them a device capable of always-on connectivity, high-speed data processing and advanced sensing [11]. Besides being a technically advanced device, the smartphone is also a highly personal item, interwoven with the everyday life of its user. These affordances of the smartphone opened up a market for context-aware, personalized applications, covering domains such as healthcare [19], safety [43], environment monitoring [35], and transport [41].

Middleware'14 December 8-12, 2014, Bordeaux, France.
ACM 978-1-4503-2785-5/14/12 ...$15.00.
http://dx.doi.org/10.1145/2663165.2663331.

Data collected through smartphone sensors comes in high resolution and spans over multiple modalities (context types): location, movement, audio environment, proximity with other objects, collocation with other devices, to name a few. In addition, sensor data collection can be done on an unprecedented scale with millions of users in parallel.

Access to mobile sensor data can revolutionize numerous fields, from Web-based applications, where using such data can lead to increased application relevance and improved user experience [26, 45], to psychological and sociological research, where access to fine grain context provides a glimpse into the everyday life of an individual [18, 34]. However, mobile context-awareness is still not adopted on a large scale, and to the full extent of the available modalities. In fact, today's Web-based applications mainly exploit only location information from phones [6]. In addition, there is a limit to what a physical sensor of a device can infer about the user, and a rich set of contextual data, especially that related to a user's emotional state, thoughts and opinions, remains untapped. These, however, can be mined through OSNs. OSNs enable users to establish and maintain social connections, express themselves through text, images, sounds and videos, therefore, can capture sophisticated information such as users' interests, relationships, event attendance, and much more. As such, OSNs represent a rich source of contextual information that is complementary to the aspects captured by physical sensors [14, 22].

As proposed by Beach et al. [14], OSNs serve as another source of users' contextual information. The OSN data streams can be further coupled with physical sensor streams in order to provide a holistic picture of users' behavior, habits and interests. Possible examples include a social science research application that captures emotions through the sentiment analysis of OSN posts, senses the physical context as the relevant posts are made, and maps the data to the social network in order to not only examine single user's emotions, but also analyze large-scale emotion propagation, and various factors that might drive it. However, developing such conceptually simple applications is often challenging due to the intricacy of implementing the means of obtaining, refining and managing streams of coupled social and physical context in real time.

To address these challenges in this paper we present SenSocial, to the best of our knowledge, the first middleware that lifts the burden of building and binding social and physical context data streams from an application developer. SenSocial abstracts the means of obtaining linked OSN and sensor information in real time, and lets the developer concentrate on high level functionalities of rich ubiquitous computing applications. Our middleware offers remote management of streams and filters to refine contextual (physical and social) data streams, so that relevant parts of the data are isolated and delivered to the overlying application. Important prac-

tical aspects of middleware functioning, such as energy efficiency, data transmission burden, memory allocation, real-time triggering, and privacy management, were all considered during the design stage. Moreover, for the privacy concerns SenSocial gives the practitioners a clear indication of all the sensing modalities used by the application, as well as the location and the granularity at which such data are stored.

The contributions that SenSocial brings to the field of mobile social sensing middleware can be summarized as follows:

- **Close coupling of OSN and mobile sensing data streams in real time.** In SenSocial, OSN actions such as comments, posts, and likes, can immediately trigger remote sensing of the physical context on the user's mobile. This captures the relationship between the activities that the user performs on OSNs and the physical context extracted by means of the mobile phone sensors.

- **Remote management of data stream.** SenSocial allows the application developer to remotely create, subscribe to, and filter sensor data streams on the mobile clients. Such a stream can be managed dynamically from the server. In addition, the middleware enables group stream management through the multicast stream – a collection of streams originating from geographically collocated or OSN-interconnected users.

- **Filtering of data streams.** SenSocial supports both topic-based and content-based streams. Content-based streams enable the specification of modalities of interest, and conditions under which these modalities should be sampled. Such a condition can be based on a time interval of a day, a certain value of the physical context, e.g. "when the user is running", and the OSN actions of the user, e.g. "when the user likes a page". Moreover, the SenSocial server component supports logical conditions that involve sensor streams generated by multiple users who are related to the target user thorough their geographical location and/or OSN links.

- **Privacy management control.** SenSocial allows the developer to manage the type and the level of granularity – raw or classified to a high-level description – of sensed contextual data that are be stored and shared among the middleware components. Note that the OSN side privacy is left to be handled by the OSN platform as per the users' requirements (such as visibility of Facebook posts defined by users).

SenSocial is envisioned as a distributed middleware: we implement it as an Android smartphone library and a Java library residing on a centralized server. We design SenSocial as a light-weight and easy-to-use general purpose middleware for mobile social sensing. By means of micro-benchmarks we evaluate SenSocial's battery consumption, memory and processing overheads, and context sensing latency. We show that an application built on top of the middleware performs well with respect to the above metrics, as compared to a similar application built without using the middleware. Also, the middleware allows fine tuning of data sampling, transmission and privacy control parameters in order to achieve the desired trade-offs, such as data granularity versus energy efficiency. We build two sample applications based on social and sensor information streams, which demonstrate how SenSocial lifts the burden of the OSN actions and physical context integration and remote stream management from the developer. In particular, we show that by using SenSocial the amount of code needed for the implementation of the applications is greatly reduced.

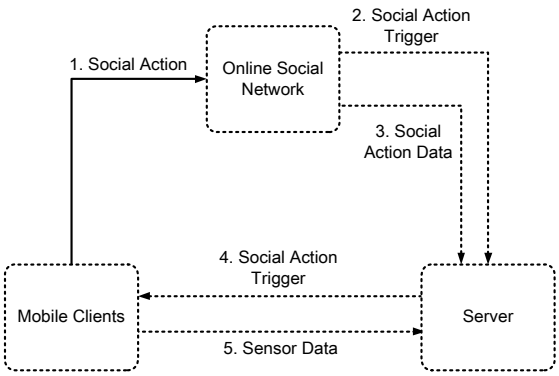

Figure 1: SenSocial architecture overview. SenSocial is distributed over a server and participating mobiles. In addition, the middleware taps into OSN data of the users. The server-side of the middleware remotely controls, via triggers and data aggregation, mobile sensing according to observed OSN actions. An API is exposed to developers for building both mobile and server-side applications. In particular, sensor data streams can be created and manipulated on both mobiles and the server.

2. SENSOCIAL AT A GLANCE

Figure 1 shows the high-level architecture and the flow of information in our system. The middleware is distributed over two components, one residing on mobiles and the other on a centralised server. In addition, SenSocial implements necessary plug-ins for accessing OSN information. A plug-in registers actions that SenSocial users perform on an OSN *(1)*, irrespective of the device (desktop, laptop or smartphone) and the means of OSN access (standalone application or online website). When an action is registered by the plug-in, action presence *(2)* and the related OSN content *(3)* are sent to the server. The server side of SenSocial allows the application developer to define desired modalities that will be sensed on the mobiles, and the conditions that need to be satisfied for the sensing to commence. These are encapsulated in filters that the application can modify dynamically. If the detected OSN action satisfies the conditions, a trigger is sent to selected mobiles to commence sensing *(4)*. When a mobile receives a request for sensing, it samples relevant sensors, and sends the stream of data to the server *(5)*. Thus, SenSocial allows remote, dynamic management of sensor and OSN data streams coming from multiple mobiles.

Figure 2 shows an example application that can easily be built on top of SenSocial. The application notifies a user when one of his/her OSN friends visit his/her home town. In the figure, there are five users of the application: users A and B who live in Paris, and users C, D and E living in Bordeaux. User A has social network links with C and D, and the server keeps track of these connections. The context of each of the users is periodically sensed by the mobile application and transmitted to the server. Let us assume that user C travels from Bordeaux to Paris. As user C enters Paris, the new sensed context is transmitted to the server. The server identifies Paris, the home town of user A, as the current location of one of A's OSN friends (user C in this case), and triggers the mobile application installed on A's phone that in turn notifies A about the presence of a friend in Paris. To summarize, SenSocial enables applications that not only collect a richer context, but also bridge mobile sensing and OSN information.

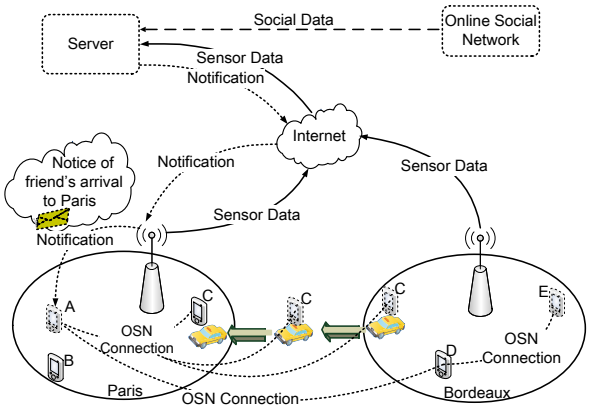

Figure 2: Geo-aware social notifications built on top of SenSocial. User A gets notified when one of her friends, in this case user C, enters Paris.

3. SYSTEM DESIGN

SenSocial consists of a middleware layer on both mobile devices and a centralized server. The overlying application, that can also be distributed over the mobiles and the server, interacts with the middleware through a publish-subscribe API. The mobile component of SenSocial manages data sensing from mobile phone sensors so that the users physical context can be acquired. Contextual data can be mined in either its raw state (e.g. accelerometer x-axis intensity values), or can be classified to high level inferred states (e.g. activity classified as "running"), as requested by the overlying application developer. The server component of SenSocial communicates, through specialised plug-ins, with OSNs, and gathers information on the users OSN actions and connections. The key feature of SenSocial is its ability to manage the sensing of the user's physical context once an OSN action is detected, and to couple the sensed physical context with the OSN-related context. Moreover, SenSocial allows the developer to define sophisticated filters that will determine the conditions under which the context will be sensed, the granularity of the acquired data, and a subset of users that will be involved in the particular data collection task. In addition, due to its ability to remotely manage sensing, SenSocial allows the context data to be sent to a register application listener that resides either on a mobile or on a server. Finally, SenSocial exposes an API for privacy management that sets policies in place so that only data from pre-approved sensors, and only data of pre-defined granularity (raw or classified) can be delivered to the application.

3.1 Abstractions

Today, OSNs attract users by allowing them to perform numerous activities on a single platform. At the same time, such an OSN becomes a great source of collecting data about users' social actions. However, the main goal of our work is focused towards the integration of OSNs and mobile sensing data streams in real time. It becomes very difficult to trace the users' OSN actions, transmit these actions to the relevant client's mobile where they are coupled with the physical context data in real time. Refining and managing the streams of such data are additional challenges of this work. Therefore, we design and implement the middleware that hides the complexity and reduces the coding efforts required to implement such features. The above functionalities of SenSocial are exposed through an intuitive API, based on the following abstractions:

- **Publish-Subscribe Interaction Paradigm.** SenSocial middleware's interaction scheme supports asynchronous communication with the applications. It is based on the publish-subscribe paradigm where the middleware acts as the publisher and the application residing on top of the middleware is the subscriber. Such an interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events, in order to be notified subsequently of any event, inferred by the middleware, that matches their registered interests. In SenSocial, the subscriptions made by applications can be categorized as topic-based and content-based. Topic-based subscriptions are the ones that allow the specification of modalities of interest (e.g. user's location). Whereas, more complex content-based subscriptions enable the developer to express the interest in modalities under certain filtering conditions (e.g. get user's location when the user posts about football on his/her Facebook wall).
- **Streams.** The mobile middleware offers two kinds of streams: (i) continuous: sensor data (raw or classified to high level descriptions) are sampled periodically with a given rate; (ii) social event-based: sensor data are pulled from the sensors and streamed when social activity is detected.

  We expose continuous streams to alleviate the effort of obtaining and managing sensor data from the developer. Continuous streams are highly flexible and the developer can configure streamed data granularity, a stream's duty cycle and sample rate. Moreover, through filters, discussed later in the text, the developer can impose conditions under which sampling will take place.

  In case of social event-based streams, sensor data are sampled only when the middleware traces a new OSN action by the user. By offering such streams, SenSocial hides the complexity of obtaining and linking OSN actions with sensor data in real time. Just like a continuous stream, a social event-based stream can also be filtered and classified on the mobile. Additionally, any stream can be transmitted to the central server, where it can be further merged with streams coming from other mobiles. On the server, such streams can be filtered with conditions based on the users' social network links and geographical location.
- **Stream Filters and Aggregators.** Filters are used to refine stream data so that only the information of interest is captured. Moreover, by restricting sensor sampling and data transmission, stream filtering on a mobile can reduce the phone's energy consumption and the data plan usage. The distributed filter is a concept that allows the configuration of stream filters both on the mobile application, as well as on the server application. It consists of a set of conditions where each condition comprises of a modality, a comparison operator, and a value (specific context). For example, a filter can be set to obtain data from GPS only when a user is walking. In this case, the filter modality is `physical_activity`, the operator is `equal`, and the value is `walking`. The stream that is being filtered is the GPS stream. Note that an unrelated stream, the accelerometer stream, has to be sensed in order to infer the activity. Filters can be conditioned on time interval, physical context, and OSN activities, but need not be conditioned on streams coming from the same device. Thus, one can create a filter that sends user's GPS data only when another user is walking. In this case, the server component of SenSocial manages the stream by filtering the user's GPS stream with respect to the accelerometer stream that is created on the instance of SenSocial running on the mobile device of another user.

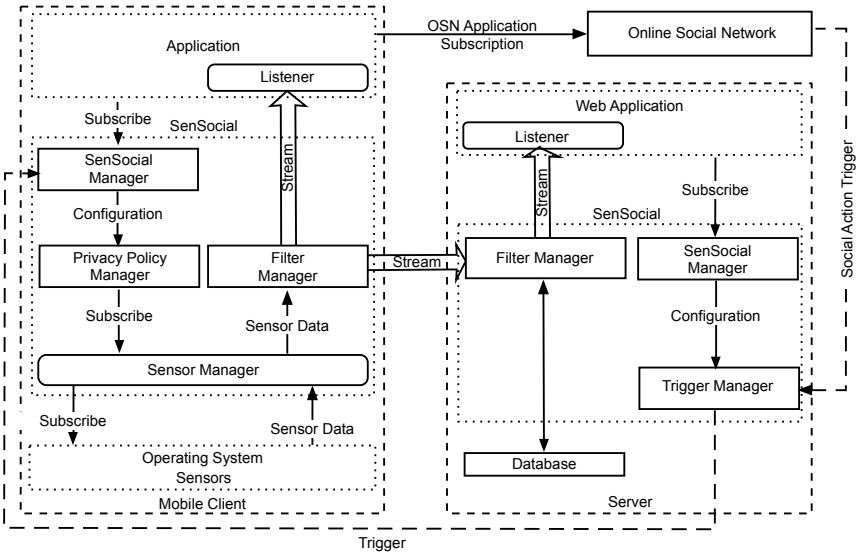

Figure 3: SenSocial components of mobile client and server-side middleware. The server can control multiple clients; we show a single client for clarity.

Aggregators manage multiple streams received by the server by wrapping them into a single aggregated stream irrespective of the streams' sources. In an aggregator, data from individual streams is multiplexed to the same join stream, which can further be processed as any other stream in the system.

- **Multicast Stream.** The multicast stream abstracts related streams of multiple clients into a single entity. The main feature of the multicast stream is its ability to manage remote data collection from a large number of clients. Instantiated on the server, the multicast stream can tap into the information about the geographic location of the users, or their OSN interconnectivity, and through a query that takes geo or OSN attributes into account, select a subgroup of users whose data will be collected. Furthermore, filters set upon a multicast stream are transparently distributed to all the users encompassed by the multicast stream.

3.2 System Components

In Figure 3 we show the components of SenSocial middleware, their location within the architecture, and types of data that are exchanged among the components. The goal of these components is to efficiently support the the abstractions we explained earlier. The middleware exposes a concise set of API calls that are sufficient for controlling these components.

The SenSocial Manager is the core component of the middleware and represents a point of entry for the overlying application – the SenSocial Manager exposes `registerListener()` a method for registering a stream data listener with the middleware. The listener has to implement SenSocial Listener and can be a component in the mobile application layer, or on the server side, in which case the stream data coming from the mobile are forwarded to the server where they can be further processed either as an individual stream or as a part of an aggregated stream.

The SenSocial Manager exposes methods to create and manage contextual streams. A stream is created by specifying a certain context modality (i.e. location, Bluetooth environment, accelerometer) and the required granularity of the data (i.e. raw samples or high level classified data). A filter containing a set of conditions can be added directly to this stream. The SenSocial Manager passes the requests for a new sensor stream, along with that stream's filter, to Privacy Policy Manager. Here, the request is screened for compliance with the privacy descriptor that defines the granularity and the type of data that is allowed to be sampled from a user's phone. The privacy descriptor comprises of privacy policies. These can be dynamically defined by the developer or exposed as settings to the users. Note that, in order to allow the sensing of a stream, Privacy Policy Manager screens for both the modality required by the stream and its filtering conditions. If the request is cleared, it is forwarded to the *Sensor Manager* that taps into raw sensor data. As discussed before, the middleware offers two modes for sensor data streaming: social event-based and continuous. In case of the former, sensing is triggered by the *Trigger Manager* located on the server side of our middleware when a user performs an OSN action and the sampled sensor data is integrated to the OSN action data.

The data streams are forwarded to the *Filter Manager*, where stream filtering takes place in case the stream entails any filters. Streams whose data satisfies developer-defined filtering conditions are passed to the registered listener. Additionally, the SenSocial Manager enables the application to dynamically define the duty cycle and sample rate of a stream by passing a configuration object.

In SenSocial we are interested in supporting large scale integration of OSN and physical sensor data streams, thus we pay specific attention to the management of multi-user sensing. This is particularly important for applications where the set of mobiles whose data we are interested in changes frequently, as is the case with location-dependent mobile sensing applications. Such features are supported through the SenSocial server-side stream creation and management. Again, the SenSocial Manager (server side) represents the first point of contact with the server application layer. It allows stream creation and subscription in a way similar to the SenSocial Manager (client side). A stream configuration file is generated by the SenSocial Manager (server side) and the respective mobile clients are notified to download the configuration file and merge it with the existing stream file. The Trigger Manager maintains a controlled communication link between the server and

mobile devices. Triggers can carry either stream configuration information or signals to start sensing based on an OSN action. The SenSocial Manager (server side) exposes API calls to create a multicast stream (i.e. a stream on the given set of users) based on the users' geographic location and social network links. As a result, the incoming streams on the server can originate from numerous devices and with variable frequencies.

Filter Manager on the server oversees the incoming streams with respect to the filters defined by the server application. These filters can include data from multiple users, as streams coming from one user can be conditioned on data coming from another user (e.g. report user's location only when her friends are posting positive things about her on Facebook). In case of multiple related streams, Filter Manager ensures that the data are consolidated as an aggregated stream with the help of an Aggregator, that wraps streams into a single aggregated one, irrespective of the devices that these streams come from. For any other purpose, filtering included, such streams can be treated as any plain data stream.

An important feature of the server component is the ability to dynamically create and destroy streams on the remote clients. The server keeps track of users' location and OSN links and can adapt stream sampling based on the user movement and OSN network fluctuations. For example, SenSocial supports sensor data gathering from users who are collocated with a specific person. In such a case, every time the person moves, a new geo-fenced location stream is created on the mobile devices of all the users who are currently nearby, and the previously created streams are removed.

4. IMPLEMENTATION

The SenSocial mobile middleware is implemented as an Android Java library and released as an open-source project[1]. The server component consists of a Java library and PHP server-side scripts. In addition, to tap into OSN data we implement plug-ins for two popular OSNs: Facebook and Twitter. All the manager components – SenSocial, Privacy Policy, Sensor and Trigger Managers – are implemented as singleton Java classes to secure the uniqueness of the global state. Also, the factory design pattern is used to create object of the classes – `Stream`, `Filter`, `Aggregator` and `MulticastStream` – for the ease of access. As far as the implementation of the mobile middleware is concerned, we follow the best practice of Android programming by ensuring that heavy processing tasks are performed on their individual background threads.

Sensor Sampling. Data stream from sensors are accessed via the `Stream` class that can be instantiated on a mobile or on the server, through the SenSocial Manager, that in turn communicates with the Sensor Manager. To implement Sensor Manager, the SenSocial mobile middleware relies on the third party *ESSensorManager* library for adaptive sensing [30].

SenSocial supports all five types of sensor modalities that can be pulled from the ESSensorManager library: GPS, accelerometer, microphone, WiFi, and Bluetooth. SenSocial relies on two modes of sampling provided by ESSensorManager library: one-off sensing and subscription-based sensing. One-off sensing is used for streams that are conditioned on the OSN action trigger. In this case, in order to save the energy, sensing is triggered once, remotely, only if an OSN action is observed. Subscription-based sensing continuously samples sensor data for the streams that are not conditioned on OSN actions. Note that, if a stream is conditioned on other modalities then the conditional modalities are sampled continuously and the stream's required modality is sampled only when the conditions are satisfied. The SenSocial Manager exposes the API calls to define the duty cycle and sample rate of a stream in a key-value object. These settings are later passed to the ESSensorManager library, which adjusts the actual sensing parameters.

Remote Stream Management. Streams of sensor data are accessed through the `Stream` class that can reside on both mobiles and the server. If instantiated on the server, a stream transparently controls sensor sampling on the associated mobile(s). This is performed by encapsulating a stream configuration in an XML file, which is pushed from the server to mobile devices. Stream configuration contains details about the required context modality, granularity of the required data, filtering conditions, and the identification code of the device on which the stream is to be created. Stream configurations can be remotely created and modified according to application developer's requirements. Similarly to a locally managed stream, server-side stream can also be passed a settings key-value object that defines sensor sampling rate and duty cycle.

To notify the mobile device about a new/modified stream configuration, SenSocial uses the Mosquitto broker [10] through SenSocial and Trigger Manager. The Mosquitto broker contacts the mobile via the MQTT protocol. We use MQTT over HTTP protocols due to the fact that MQTT is based on the push paradigm, thus, unlike HTTP-based solutions, does not require continuous polling from the mobile side, resulting in a lower battery consumption. For example, in the case of downloading the specification of new streams, the XML definition is received by the `MQTTService` class on the mobile, and if needed, a stream filter is downloaded from the server by the `FilterDownloader` class. Then, the `FilterMerge` class merges this newly downloaded XML file to the existing set of filter configurations that are stored in the mobile device as an XML file.

Ensuring Privacy Compliance. Privacy Policy Manager exposes API calls for the developers to define the application's privacy policies. These policies can be defined based on the type and the level of granularity - raw or classified - of sensed contextual data that will be stored and communicated among the middleware components. Whenever a stream is created or modified, or the privacy settings are changed, Privacy Policy Manager is invoked to compare all the stream configurations with the latest privacy policies that are stored in the `PrivacyPolicyDescriptor` file. These policies can be dynamically defined by the developer or exposed as settings to the users, and restricts the type, as well as the granularity of sensor data that can be sampled on the mobile device (i.e., raw samples or high-level classifier output). In case a stream does not clear this privacy check, it is automatically paused by the Privacy Policy Manager. Such a stream is moved back to the working state later when it clears the privacy check according to the modified privacy policies.

Sensor Data Classification. With respect to the implementation of the machine learning classifiers, the current version of SenSocial provides a few classifiers that can classify raw sensed data into higher level context classes. Thus, SenSocial can classify raw accelerometer data into user's physical activity, such as "still", "walking" and "running", or infer from the raw microphone data if the audio environment is "silent" or "not silent". Note that we implemented these classifiers as proofs of concept, and did not focus on maximizing the classification accuracy, since it was not the focus of this work. However, the design of the middleware is very flexible in that respect. SenSocial offers the possibility for developers to integrate their own classifiers with the mobile middleware. The integration of external classifiers is possible by registering listeners on the mobile application to receive raw sensor data. It is also possible to add classifiers for data coming from the OSN services.

[1] **https://github.com/AbhinavMehrotra/SenSocial-Library**

Integration with OSNs. Accessing OSN data in SenSocial is enabled via plug-ins for Facebook and Twitter. A mobile user needs to add the Facebook plug-in to his Facebook profile, so that actions such as posts, comments and likes are captured and forwarded to a PHP script on the server. On the other hand, the Twitter plug-in comprises of PHP files that completely resides on the server and periodically queries data from the Twitter server for each user that has authenticated SenSocial via OAuth to access his Twitter data. In both cases, once the OSN data are received, the SenSocial Manager (server side) ensures that the sensor data are fetched from relevant remote clients. It keeps the list of `User` instances containing users' registration information, `Device` instances comprising of the device identification information, and the associated `Stream` instances. The relevant client(s) are selected and the Trigger Manager compiles the OSN action and the relevant device information in a JSON-formatted string passed to the Mosquitto broker that sends triggers to the selected clients. On receiving such a trigger, the SenSocial Manager (mobile side) initiates the one-off sensing for the social event-based streams. The sampled sensor data is coupled with the OSN action data received with the trigger, and delivered to the registered listeners for the respective streams.

Data Storage and Querying. The server component uses a MongoDB database [12] to store the information about user registration, user's OSN friendship and geographic location information. To maintain the updated information about user's OSN links (friendship), the server component classifies OSN actions to infer any change in the OSN. Whereas, the user's geographic location is updated periodically at a time interval that can be configured via the SenSocial Manager (client side). Complex queries can be created on the basis of this information, and multiple related streams can be initiated as a result. The `MulticastStream` class enables seamless management of streams on multiple devices from the server. Filters can be instantiated on top of a multicast stream and distributed via the MQTT broker to all involved devices.

In the example depicted in Figure 2 we show how the above parts of SenSocial integrate into a fully-functional middleware. First, all the devices in the system run an application that registers with an instance of the SenSocial Manager. In addition, the users of the devices need to authenticate with the SenSocial OSN plug-in so that the application can access their OSN relationship information. This information is stored in a MongoDB on the central server. The server side application queries the database for all the OSN friends of user A, and creates a `MulticastStream` instance that abstracts location data streams from the devices that belong to users C and D. The MQTT broker notifies these users to download an XML file that describes the modality that is being sensed (location) as well as the filtering condition (location equals Paris). Each of the phones downloads the file and merges it with its `Filters` file, after which the `Filters` file gets checked for the privacy policy compliance with the `PrivacyPolicyDescriptor` file (predefined by the developer). If it complies, the data are periodically sensed from each of the mobiles and raw GPS coordinates are classified to a descriptive address, i.e. the name of the city that the user is in. If the filtering condition is satisfied, meaning a user is located in Paris, the data are transmitted to the server. On the server, a listener within the application processes the data and notifies user A that user C has arrived to Paris.

5. EVALUATION

In this section we evaluate the performance of the SenSocial platform, mainly focusing on the aspects related to resource efficiency and system scalability.

Table 1: SenSocial source code details.

Counter	Mobile middleware	Server component
Java files	77	46
PHP files	0	2
Source code lines	2635	1185

Table 2: Memory footprint for sample SenSocial application and Google's Activity Recognition (GAR) application.

Application	Heap-size allowed (MB)	Heap-size allocated (MB)	Objects
SenSocial	13.508	12.342	51419
GAR	12.945	11.126	46210

5.1 Evaluation Settings

We evaluate SenSocial on a Samsung Galaxy N7000 phone with 1 GB of RAM, and a dual-core 1.4GHz ARM Cortex-A9 CPU, running a clean slate Android 4.0.1 (Ice-cream sandwich) operating system. We load and run SenSocial as a part of various applications: from a simple stub application to a purpose-built context-sensing and content-adaptation applications explained in the Prototype Applications section. We use third-party measurement tools, such as Count Lines of Code (CLOC) [5], Power-Tutor [46], Android TraceView [4], and Android Dalvik Debug Monitor Server (DDMS) [2] to quantify the middleware performance.

5.2 Source Code and Memory Footprint

SenSocial is based on two main components, one on the mobile side, and the other the server side. The Android-based mobile component comprises of 77 Java classes, while the server is implemented by means of 46 Java classes and 2 PHP scripts. We use CLOC to obtain code count statistics summarized in Table 1.

In the Android OS, paused applications are retained in the phone's memory, and the OS maintains the state of the application's instance. Also, the application process remains attached to the Android window manager for quick retrieval. However, the application instance can be destroyed by the system in case of extremely low memory [1]. Consequently, if an application uses a large amount of memory, it may cause other applications to be killed. Thus, it is essential, especially for a middleware platform, to occupy as little memory space as possible.

We evaluate the memory footprint of a stub application built on top of SenSocial. The application creates continuous sensor streams with each of the five supported sensor modalities (i.e. accelerometer, microphone, GPS, WiFi and Bluetooth), and subscribes to the sensed data by registering a listener to these streams. We compare the memory footprint of a stub SenSocial application to the footprint of an application we term Google Activity Recognition (GAR) that is built on top of the Google's Activity Recognition Library API [8]. It streams high-level physical activity information, obtained through Google Play Services, to the server. The memory footprint obtained via the Android DDMS tool is shown in Table 2. Compared to the stub GAR application, the fully functional SenSocial application uses only 1.216 MB of extra memory. It is important to note that the GAR memory footprint does not include accelerometer sensor sampling, as GAR outsources this functionality to Google Play Services. Google Play Services do not reside in the user space, thus cannot be profiled by DDMS. SenSocial, instead, by relying on the lightweight SensorManager library, consumes just slightly more memory while delivering a much broader set of functionalities.

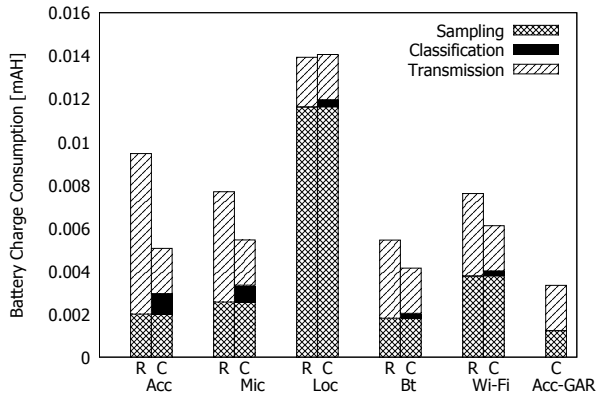

Figure 4: Average battery charge consumed per sensing cycle. We show SenSocial with each of the sensor data stream types and the Google Activity Recognition (GAR) application. The abbreviations in the figure are: R – Sampling and transmission of raw sensor data; C – Sampling, classification and transmission of classified sensor data.

5.3 Energy Management

One of the key challenges for context-aware mobile sensing applications is their energy use and the resultant battery life impact. Continuous sensing of GPS for example can lead to a twenty-fold reduction in the battery lifetime [13]. The total energy consumption depends not only on the type of sensor, but also on the sensor sampling duty cycle, sensed data processing, and transmission rate, in case the data is transferred to a server.

We examine the energy consumption of the SenSocial mobile middleware, and identify the energy requirements of each of the key tasks that the library performs on the mobile: sensing, data classification and data transmission. For that, we develop an application with a background service instance that samples sensors, optionally classifies the data, and transmits either raw or classified data to the server. We also investigate energy requirements of each of the sensing modalities supported by SenSocial: accelerometer, microphone, GPS location, Bluetooth and WiFi. Sensing is performed every 60 seconds for each of the streams during an hour interval. The rate of sensing depends on the modality, and we use the default sensing configuration values from the ESSensorManager library [30]. We measure energy consumption with the frequency of 1 second and average the recorded values, in order to include the extra energy-tails due to the wireless interfaces being prevented from switching to sleep mode [40].

Figure 4 shows the energy consumed with different sensor streams. The energy readings were obtained with PowerTutor and are averaged over an hour. As expected, and in accordance to [13], different sensor modalities are characterized by remarkably different energy costs. The transmission energy is high for accelerometer data as it contains a vector of acceleration values for the three axes, sampled every 20 ms for eight seconds, thus, accumulating a significant amount of raw readings. However, this opens up an opportunity for energy consumption reduction through classification. Indeed, classification of raw accelerometer values to a high level activity description (i.e. “running", “walking", “sitting") halves the total energy consumption of a SenSocial-based application that uses an accelerometer stream.

To put the results in a perspective, we also investigate the energy consumption of the GAR application. Note that GAR relies on GooglePlayServices for activity sensing and inference, and merely establishes links with GooglePlayServices. Still, the energy consumption is only 25% lower than in the case of classified SenSocial data streaming. Note, however, that in the above analysis we use simple classifiers to get high-level data. More sophisticated classification is likely to consume more energy; this can be considered as a baseline in order to estimate the energy performance.

Table 3: Time delay in receiving OSN notifications.

Notification type	Average time [s]	Standard deviation
OSN to Server	46.466	2.768
OSN to Mobile	55.388	2.495

Table 4: Average battery consumption with varying number of OSN actions (within 20 minute time period) that trigger remote sampling of all five supported sensor modalities.

OSN actions	1	2	3	4	5	6	7
Charge consumed [μAH]	51.7	97.1	142.5	187.8	233.2	278.5	324.3

5.4 Time Delay

SenSocial can successfully capture the relationship between OSN actions and a user’s context only if the OSN-based triggering initiates mobile sensor sampling as soon as an action is performed on the OSN. We measure the delay between the time of an OSN activity and the time when the mobile starts sampling. OSN-based triggering also involves centralized server querying, thus we measure two values: *i)* time needed for an OSN action to be reacted upon on the server, and *ii)* time for an OSN action to trigger sensing on the mobile. Note that the latter includes server-side querying.

The measurements were taken when the mobile was connected to an uncongested WiFi network, in order to avoid the effects of poor connectivity on the measured delay. The server is connected to the public Internet via a high speed link, and does not host any other publicly available services. To account for the time difference between the SenSocial server, where a part of the measurement app resides, and the Facebook server, where the OSN actions take place, we built a mobile application on top of the middleware that registered the time when a post is made and a notification about the post is received. The results for the time delay to receive the OSN notifications include the transmission time to make a post. Table 3 shows the response delay averaged over 50 OSN actions. The OSN notifications reach the SenSocial server in 47 seconds on average, and it takes 56 seconds to reach the mobile. The small difference between the OSN to server and OSN to mobile delays indicates that the SenSocial notification mechanism is very efficient as it takes only nine seconds to process the event and notify a mobile. The overall delay is limited by the time Facebook takes to notify SenSocial about OSN actions. This delay varies for different OSNs: our Twitter plugin, which actively scans for new tweets, allows arbitrarily short delay.

5.5 System Scalability

In this section we discuss the scalability of our middleware with respect to the number of active sensor streams, the complexity of stream filters, the number of users and the OSN actions.

Impact of Multiple Streams. The increasing number of sensor streams initiated by a mobile application may influence the mem-

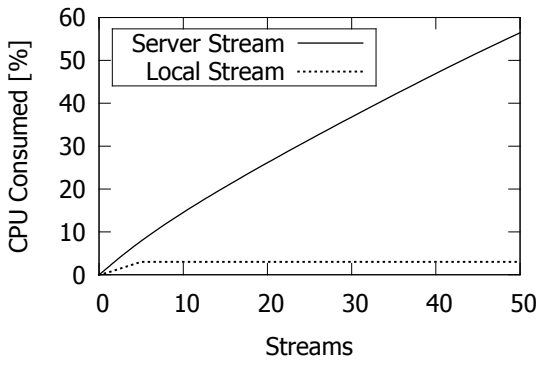

Figure 5: CPU load with increasing number of sensor data streams. Streams are either consumed within the mobile (local streams) or transmitted to the server (server streams).

Figure 6: Screenshot of Facebook Sensor Map application. Each marker corresponds to a user's OSN action, and merges geographic, audio and physical information with the type and content of the OSN action.

ory consumption and CPU load. We measure these factors in an application that iteratively increases the number of active streams until it crashes. The measurements of CPU load and memory consumption were carried out using PowerTutor and DDMS respectively. We find that the number of streams does not affect the memory consumption of the application. However, the increasing number of streams puts more load on the CPU. We experiment with server streams which include mobile sensor data transmitted to the centralized server and local streams (i.e., streams of data that are consumed locally on the mobile by an application layer listener). As shown in Figure 5, the CPU load grows significantly only for streams transmitted to the server. Still, the CPU load is less than 10% even with five streams (i.e., the number of modalities supported by SenSocial).

Impact of Filter Complexity. SenSocial stream filters allow sophisticated pruning so that only the information we are interested in is streamed to the application. Filters of a certain sensor data stream can be conditioned upon the value of different modalities. For example, we can request a stream of microphone data that will be sampled only when the accelerometer data are classified as "walking". Filtering can also include conditions based on the user's OSN activity. In such cases, the stream is transmitted only when the user performs a certain OSN action, e.g. posts, comments or "likes" a page. The use of filtering rules can increase energy efficiency of sensor streaming as the streams are transmitted only when the conditions are satisfied. At the same time, sensing orthogonal modalities and performing complex machine learning classification upon the data consumes energy through increased CPU load. However, the use of filtering rules can also help to save battery by sampling energy-costly sensors only on satisfaction of the conditions based on a less energy consuming sensor. For example, sampling location via GPS is far more demanding in terms of energy than sampling the accelerometer data. Therefore, it might be worth creating a filter that allows location data sampling only if the accelerometer data indicates movement.

Impact of Multiple Users. SenSocial is highly distributed and each additional user merely adds the energy and processing cost of a lightweight local library to his own mobile phone. With more users, the centralized server has to manage notifications to a larger number of end devices. In our implementation, the server relies on the MQTT broker to broadcast notification messages to mobile clients. Due to the broadcast nature of the transmission, the actual number of recipients does not impact server resources. The server component also stores the OSN graphs, and the distributed stream configurations along with the reference of relevant listeners, in a MongoDB database. As a non-relational database, MongoDB allows simple storage of large unstructured datasets. However, due to its non-relational nature querying from MongoDB can be inefficient. This limitation can be addressed by building indices for commonly used queries. In addition, MongoDB natively supports geospatial querying. This translates to fast return of nearby users or those located within a certain area.

Impact of the Number of OSN Activities. We evaluate the ability of SenSocial to cope with a burst of OSN activity that triggers remote sensing on mobiles. We create a mobile application that samples and transmits to the server data from all five supported sensors, and iteratively increases the number of OSN actions in a 20 minute time window. With PowerTutor we measure the energy consumption in each time window. Table 4 shows the energy consumption for up to seven OSN actions in a 20 minute window which is the maximum, as each trigger takes approximately 120 seconds to complete (60 seconds for sampling all the sensor and 60 seconds for receiving the trigger from the Facebook). We see that the energy consumption increases nearly linearly. Due to the finite time for trigger completion that bounds the energy consumed in a unit of time, we conclude that the scalability of SenSocial is not limited by the number of OSN actions.

6. PROTOTYPE APPLICATIONS

In order to show the effectiveness of the abstractions implemented by the middleware, in this section we discuss how the SenSocial API can ease the development of two prototype applications. The first application, *Facebook Sensor Map*, demonstrates SenSocial's ability to trace users' social activity on Facebook and link it to the physical context data acquired through mobile sensing in real-time. This information is then displayed on a map. The second application is *ConWeb*, a contextual Web browser that dynamically modifies rendering and content of a Web page on the server according to users momentary context extracted from the sensors via SenSocial. In other words, SenSocial starts collecting sensor data on the mobile and transmits this information stream to the Web server when a page is requested by the user. The Web server then dynamically generates a page according to the contextual information that has been received from the device. In this paper we examine Facebook Sensor Map and ConWeb purely as demonstrators of SenSocial's utility for mobile social sensing application development.

```
SenSocialManager manager = SenSocialManager.getSenSocialManager(getApplicationContext());
String uid = manager.getUserId();
User user = manager.getUser(uid);
Stream s1 = user.getDevice().getStream(SensorUtils.Sensor_Type_Accelerometer, "classified");
Stream s2 = user.getDevice().getStream(SensorUtils.Sensor_Type_Microphone, "classified");
Stream s3 = user.getDevice().getStream(SensorUtils.Sensor_Type_Location, "raw");
/*----------- Create list of filter condition(s) ----------------*/
ArrayList<Condition> conditions = new ArrayList<Condition>();
Condition c = new Condition(ModalityType.facebook_activity, Operator.equals, ModalityValue.active);
conditions.add(c);
/*----------- Add condition list to the filter -----------*/
Filter filter = new Filter(conditions);
/*----------- Set filter to the streams -----------*/
s1 = s1.setFilter(filter);
s2 = s2.setFilter(filter);
s3 = s3.setFilter(filter);
```

Figure 7: Code snippet of the implementation of the Facebook Sensor Map mobile application using the primitives provided by the SenSocial middleware.

6.1 Facebook Sensor Map

To demonstrate the potential of merging OSN and physically sensed information we develop Facebook Sensor Map. This application, screenshot of which we show in Figure 6, displays information on a geographic map about an individual and his/her social circle, including Facebook activity with the associated contextual data streams extracted from the mobile phone sensors.

Facebook Sensor Map comprises of three parts: a mobile-side application built on top of SenSocial mobile client, a server-side application built on top of SenSocial and a Facebook application added to the user's Facebook profile. The mobile part incorporates `FacebookSensorMapService`, a long-running background service that uses SenSocial to subscribe to the streams of sensor data filtered on the user's Facebook activity.

The Facebook application detects OSN activities and notifies the `FacebookReceiver` script, hosted on the server, every time a Facebook activity (such as posting a status) occurs. Since Facebook activities get captured directly by the Facebook application, Facebook Sensor Map works even if the user interacts with Facebook from another device. Once a new OSN activity is detected, the server notifies `FacebookSensorMapService`, via a trigger from the MQTT broker, to sample the current physical context of the user.

In addition, the trigger includes the JSON-formatted information with the content of the OSN action, e.g. the text of a Facebook post. Captured sensor and OSN data are stored locally in an SQLite database running on the mobile, and displayed on a Google map instantiated within the Facebook Sensor Map mobile application. Moreover, the data are sent to the server where the information is stored in a database, allowing complex OSN and context-based multiuser querying. The information stored in the database is then presented as a set of navigable maps that are updated in real-time.

Figure 7 presents the code snippet from the Facebook Sensor Map mobile application that is implemented inside the background service. We create three streams s1, s2 and s3 of classified accelerometer data, classified microphone data and raw location data respectively. To integrate these streams with Facebook action data, we set a filter on all the streams. This filter holds a condition where the modality-type is `facebook-activity`, the operator is `equal`, and the modality-value is `active`. This condition permits the middleware to sample sensor data only when the user performs an OSN action. The code snippet uses SenSocial's `Stream` and `Filter`, to obtain refined streams of integrated physical context and OSN action data. It subscribes to the streams of required physical context, and requests to sample the stream data only when the user performs a Facebook activity and couple this activity content with the sampled physical context.

6.2 ConWeb – Contextual Web Browser

World Wide Web content has been traditionally served in a user's physical context agnostic way. In the last decade researchers have focused on the problem of adapting the content layout to better suit mobile devices where screen real-estate is limited [27], and to adapt media files to low-connectivity conditions [21]. Iwata et al. developed a system that adapts displayed content depending on user's mobility [25], while applications, such as Google search, adapt the content to the location. Despite these advances, in our opinion, the concept of context-aware browsing is not fully exploited yet.

ConWeb is an application, built on top of the SenSocial middleware, that delivers dynamically generated Web pages based on the context sensed by the mobile devices and activities perform by the users' on the OSNs. ConWeb supports run-time adaptation according to the current physical context and social (OSN actions) data of a user: a page is automatically refreshed every T seconds in order to download a version of the Web page from the Web server adapted to the most recent context information of the user. The value T can be set by the user. Thus, ConWeb enables highly adaptable browsing in which page content, as well as appearance are modifiable. Pages served with ConWeb can adapt not only to the physical content, e.g., by displaying higher contrast colors when it is sunny and a user is outside, but also to the social context, e.g., by showing gift suggestions to a user who is about to attend a birthday, as indicated by information automatically retrieved from OSNs. We present this application to demonstrate SenSocial's potential to remotely manage the streams of integrated physical context and social (OSN actions) data. The application relies on SenSocial's streams to abstract the means of obtaining physical and social context data from the users' mobile phones in real time.

Figure 8 illustrates the architecture of the ConWeb system. ConWeb consist of the following components: mobile application built on top of SenSocial mobile middleware, Web server to host Web pages, server application built on top of the SenSocial server component, and OSN plug-in (optional). The ConWeb application can

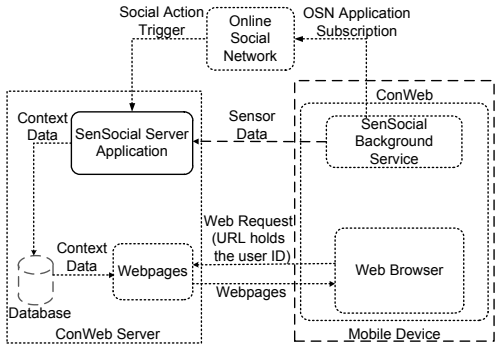

Figure 8: ConWeb architecture.

be configured to receive data streams only related to physical context or the OSN actions associated to it as well. The OSN plug-in is required only in the case of latter.

The ConWeb mobile application comprises of a Web browser and an Android background service. The Web browser uses the `WebView` class provided by the Android API to access and display Web pages. This user identifier helps in linking the user with their latest contextual data stored in the database. When a Web page is requested through the ConWeb browser, a background service, implemented by the `ConWebService` class, starts transmitting the context (physical context or the OSN action integrated with physical context) data to the server in real time. ConWeb can be dynamically configured to present Web pages based on the context chosen by the user. In such a case, ConWeb's server application leverages SenSocial's remote stream management to dynamically destroy the current SenSocial's streams and then subscribe to the streams of relevant context data. Additionally, the streams remain active until the ConWeb browser is running, and goes to the paused state once the ConWeb browser is killed by the user.

To generate personalized Web pages for a user, the SenSocial server component directs the incoming data streams to the database where it overwrites the latest context information of the relevant user. Once the server receives a Web page request, the ConWeb server application extracts the user identification code from the request and modifies the page attributes according to the relevant user's context information. Finally, the "context-aware Web pages" are transmitted back to the ConWeb browser.

6.3 Programming Effort Evaluation

We now quantify the benefits of using SenSocial for OSN-aware mobile sensing application development. Specifically, we analyze SenSocial's potential to reduce application programming effort by implementing the same applications with and without SenSocial.

In Table 5 we show the number of lines of code (LOC) needed for the implementation of ConWeb and the Facebook Sensor Map application both with and without using SenSocial. All the measurements were conducted using the CLOC tool. For a fair measure of programming efforts between the two versions of both applications, we use a third-party sensing library (ESSensorManager [30]) in both cases, and do not include the library in the LOC computation. In total, SenSocial reduces the LOC nine times, from 3423 to 316 in the case of Facebook Sensor Map, and twenty four times, from 3223 to 130 LOC, in the case of the ConWeb application.

Table 5: Lines of code (LOC) programming effort comparison (M – mobile app, S – server app).

Application name	Files	LOC
Facebook Sensor Map (M) (with SenSocial)	8	103
Facebook Sensor Map (S) (with SenSocial)	2	213
Facebook Sensor Map (M) (without SenSocial)	68	2419
Facebook Sensor Map (S) (without SenSocial)	42	1004
ConWeb (M) (with SenSocial)	3	23
ConWeb (S) (with SenSocial)	1	107
ConWeb (M) (without SenSocial)	61	2278
ConWeb (S) (without SenSocial)	38	945

7. LIMITATIONS

The main limitation of the current implementation of SenSocial is its inability to run as a single instance on a device, while supporting multiple overlaying concurrent applications. The limitation stems from the fact that SenSocial runs in the user space of the OS, and is imported as a library to each individual application that uses it. Should the middleware run as a kernel service, such as Google Play Service, multiple applications could concurrently subscribe to a single middleware service. In the user space, we could still develop and deploy a stand-alone SenSocial service, albeit with a high chance of the service being terminated by the OS, and interact with it via Android Intent passing [3]. In this model, however, the user is required to download and run this service beforehand. Without being tied to a particular application, this service would require access to all the sensors that the overlaying application might use, and would be running in the background at all times. However, such a service would resemble malware, and users would likely be reluctant to install it.

Another limitation of the middleware is that the time needed to complete successive sensor sampling cycles on the mobile limits the granularity at which the OSN action - context pairs can be captured. In case a user will perform more than one OSN action between two sampling cycles, the contextual data that were previously sampled will be mapped to these OSN actions. This is a trade-off between accuracy and energy consumption: continuous sensing would be impractical with respect to the actual usability of the system for a final user. In fact, the impact on the battery usage has to be minimal.

8. RELATED WORK

Research in the area of context-aware computing has gained popularity with a dramatic pace. Built in accelerometers, camera, microphone, and GPS sensors have been used as a basis of numerous applications covering a wide range of topics including environment monitoring [35], transport [9], health monitoring [28], stress detection [32], behavior intervention [29], social psychology studies [38] and many others. However, the popularity of smartphone sensing research lead to a great deal of squandered effort as every project has been built from the ground. Researchers proposed several middleware systems that aim to relieve developers from the burden of interacting and managing low-level sensors, delivering energy-efficient sensing on battery-constrained mobile phones, and ensuring the privacy compliances. Energy Efficient Mobile Sensing System (EEMSS) [44], Jigsaw [33] and Acquisitional Context Engine (ACE) [36] are examples of the middleware systems that focused on the energy efficient sensing. AnonySense [20], CITA [39], and Pogo [17] focused on the privacy compliances.

Later, the paradigm shifted towards sophisticated sensor data queries and the remote configuration of the sensors. The idea was to support data collection from a geographically limited area and/or within a certain time interval. BubbleSensing [31] supports geo-

graphic filter-based queries, enabling applications that, for example, sample a phone's microphone every time a user visits a certain location. Similar functionalities are provided by APISense [24], a middleware that also supports temporal querying and just like Funf [7] allows the remote configuration of sensor settings and data collection behavior. SenSocial supports geographic and temporal queries, as well as queries that are based on an online social network graph. Moreover, SenSocial remote stream management is not limited to sensing parameter reconfiguration, but also supports dynamic sensor stream creation and destruction.

All the above work is geared towards the collection of data via in-built mobile sensors, overlooking a rich set of information that can be mined from a user's social environment. Mobile Social Networking (MSN) relies on mobile sensing to dynamically determine social communities, and harness this knowledge for applications ranging from delay-tolerant routing to friend recommendation systems [15]. Examples of MSN middleware include the SAMOA framework [16], the MobiSoC middleware platform [23], MobiClique [37], and Yarta [42]. Communities in these solutions are detected through user's contacts, visits to the same place, or through joint expressed interests. While some of the middleware taps into OSNs [42, 37], none of these solutions considers OSNs as a live source of contextual data comparable to a physical mobile sensor. SocialFusion [14], comes closes to that as it treats OSN data as yet another sensing modality and enables context inference built upon the fusion of sensor and social data. SenSocial shares the above idea of SocialFusion, yet improves it by considering dynamic user-created data – user's post, likes and comments – as they flow in the OSN. At the same time through OSN-triggered remote sensing SenSocial allows closer binding of the physical and OSN context. In addition, SenSocial supports dynamic remote sensor data querying over the user OSN and geographic context.

Recently, a number of commercial applications has been built on top of online social networking and mobile sensing. Google Latitude, Facebook Places and Foursquare are some of the examples. These applications themselves do not expose a middleware API, nor enable sophisticated querying of mobile sensor data over a geographically bounded or OSN-defined group of users. However, their popularity hints the potential of SenSocial to revolutionize mobile sensing applications. All of the above applications use simple location sensing to augment OSN services. SenSocial enables further integration of multiple aspects of the physical context, sensed over a large number of carefully selected mobiles, and OSNs. Moreover, OSNs can be used not only as a source of data, but also as a controller of data sensing. We believe that both current and future mobile social sensing applications can benefit from such tightly bound rich context information.

9. CONCLUSIONS

In this paper we have presented the design, implementation and evaluation of SenSocial, a middleware for the integration of online social networks and mobile sensing data streams, designed to simplify the implementation of richer ubiquitous computing applications. SenSocial fuses user's OSN actions and sensed context data streams, and enables a join consideration of contextual data coming from a large number of geographically or OSN-related users.

SenSocial design involved the definition of key abstractions we used for data handling. We opted for streams as they capture the continuously changing behavior of both physical sensor data, such as user's location, as well as OSN data, such as a Twitter feed. We then developed filters that enable the extraction of the data of interest from a stream. Through publish-subscribe interaction, SenSocial delivers filtered information to the application. We devised components such as the SenSocial, Filter, Trigger, and Sensor Manager to consolidate the above abstractions into a functional middleware. The middleware is distributed over a server and multiple clients, so that the direct access to the sensor data and on-device filtering is supported. However, the server side of SenSocial supports remote data stream management, as well as OSN action-based remote sensing triggering. The practicality of the SenSocial design was demonstrated through a full-fledged implementation, which includes an Android client middleware, Java-based server middleware, interaction with MongoDB database, MQTT triggering and, Facebook and Twitter OSN plug-ins. Through micro-benchmarks we showed that SenSocial manages mobile and OSN-sensing in a resource-efficient and scalable manner. Two pilot applications – Facebook Sensor Map and ConWeb – demonstrate that SenSocial reduces the amount of coding effort up to 24 times.

As future work, we plan to develop components, including text mining tools, that can process data coming from online social networks. In particular, our plan is to develop classifiers that are able to extract OSN post topics and emotional states of the individuals, and link them to the users' physical context. Moreover, we plan to develop machine learning algorithms that exploit the linked information provided by the SenSocial middleware, such as the association between sensor readings and social activities, and infer higher level descriptors of human behavior, such as the user's health state.

Acknowledgements

The authors thank Steve Pillinger and Jack Uttley for their assistance in setting up the server infrastructure. This work was supported by the EPSRC grants "UBhave: ubiquitous and social computing for positive behaviour change" (EP/I032673/1) and "The Uncertainty of Identity: Linking Spatiotemporal Information Between Virtual and Real Worlds" (EP/J005266/1).

10. REFERENCES

[1] Android Activity Lifecycle. `http://developer.android.com/guide/components/activities.html`.

[2] Android DDMS. `http://developer.android.com/tools/debugging/ddms.html`.

[3] *Android Sharing Simple Data*. `http://developer.android.com/training/sharing/index.htmll`.

[4] Android Traceview. `http://developer.android.com/tools/help/traceview.html`.

[5] CLOC – Count Lines of Code. `http://cloc.sourceforge.net`.

[6] Facebook `http://www.facebook.com`.

[7] Funf open sensing framework `http://www.funf.org`.

[8] Google's Activity Recognition Application. `http://developer.android.com/training/location/activity-recognition.htmll`.

[9] Mobile Millennium Project. `http://traffic.berkeley.edu`.

[10] Mosquitto MQTT Broker. `http://mosquitto.org/`.

[11] *ITU World Telecommunication/ICT Indicators Database*, 2013. `http://www.itu.int/en/ITU-D/Statistics/Pages/publications/wtid.aspx`.

[12] *MongoDB*, 2013. `http://www.mongodb.org`.

[13] F. B. Abdesslem, A. Phillips, and T. Henderson. Less is More: Energy-Efficient Mobile Sensing with SenseLess. In *MobiHeld'09*, Barcelona, Spain, August 2009.

[14] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada. Fusing Mobile, Sensor, and Social Data to

Fully Enable Context-Aware Computing. In *HotMobile'10*, Annapolis, MD, USA, February 2010.

[15] P. Bellavista, R. Montanari, and S. K. Das. Mobile social networking middleware: A survey. *Pervasive and Mobile Computing*, 9(4):437–453, 2013.

[16] D. Bottazzi, R. Montanari, and A. Toninelli. Context-Aware Middleware for Anytime, Anywhere Social Networks. *IEEE Intelligent Systems*, 22(5):23–32, 2007.

[17] N. Brouwers and K. Langendoen. Pogo, a Middleware for Mobile Phone Sensing. In *Middleware'12*, Montreal, Canada, December 2012.

[18] T. Choudhury and A. Pentland. Sensing and Modeling Human Networks using the Sociometer. In *International Symposium on Wearable Computers (ISWC'03)*, White Plains, NY, USA, October 2003.

[19] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, et al. Activity Sensing in the Wild: a Field Trial of UbiFit Garden. In *CHI'08*, Florence, Italy, April 2008.

[20] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. Anonysense: Privacy-Aware People-Centric Sensing. In *MobiSys'08*, Breckenridge, CO, June 2008.

[21] O. Davidyuk, J. Riekki, V.-M. Rautio, and J. Sun. Context-Aware Middleware for Mobile Multimedia Applications. In *MUM'04*, College Park, MD, USA, October 2004.

[22] P. S. Dodds, K. D. Harris, I. M. Kloumann, C. A. Bliss, and C. M. Danforth. Temporal Patterns of Happiness and Information in a Global Social Network: Hedonometrics and Twitter. *PLOS ONE*, 6:e26752, 2011.

[23] A. Gupta, A. Kalra, D. Boston, and C. Borcea. MobiSoC: a Middleware for Mobile Social Computing Applications. *Mobile Networks and Applications*, 14(1):35–52, 2009.

[24] N. Haderer, R. Rouvoy, C. Ribeiro, and L. Seinturier. Apisense: Crowd-sensing made easy. *ERCIM News*, 93:28–29, 2013.

[25] M. Iwata, H. Miyamoto, T. Hara, D. Komaki, K. Shimatani, T. Mashita, and K. K. et al. A content search system considering the activity and context of a mobile user. *Personal and Ubiquitous Computing*, pages 1–16, 2012.

[26] T. Laakko. Context-aware web content adaptation for mobile user agents. In *Evolution of the Web in Artificial Intelligence Environments*, pages 69–99. Springer, 2008.

[27] T. Laakko and T. Hiltunen. Adapting Web Content to Mobile User Agents. *IEEE Internet Computing*, 9:46–53, 2005.

[28] N. D. Lane, T. Choudhury, A. Campbell, M. Mohammod, M. Lin, X. Yang, A. Doryab, H. Lu, S. Ali, and E. Berke. BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing. In *Pervasive Health'11*, Dublin, Ireland, May 2011.

[29] N. Lathia, V. Pejovic, K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. Smartphones for Large-Scale Behaviour Change Intervention. *IEEE Pervasive Computing*, 12(12):66–73, July 2013.

[30] N. Lathia, K. K. Rachuri, C. Mascolo, and G. Roussos. Open Source Smartphone Libraries for Computational Social Science. In *MCSS'13*, Zurich, Switzerland, September 2013.

[31] H. Lu, N. Lane, S. Eisenman, and A. Campbell. Bubble-sensing: A New Paradigm for Binding a Sensing Task to the Physical World Using Mobile Phones. In *Workshop on Mobile Devices and Urban Sensing, IPSN'08*, St Louis, MO, USA, April 2008.

[32] H. Lu, G. T. C. Mashfiqui Rabbi, D. Frauendorfer, M. S. Mast, A. T. Campbell, D. Gatica-Perez, and T. Choudhury. StressSense: Detecting Stress in Unconstrained Acoustic Environments using Smartphones. In *UbiComp'12*, Pittsburgh, PA, USA, September 2012.

[33] H. Lu, J. Yang, Z. Liu, N. Lane, T. Choudhury, and A. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *SenSys'10*, Zurich, Switzerland, November 2010.

[34] G. Miller. The Smartphone Psychology Manifesto. *Perspectives on Psychological Science*, 7:221–237, 2012.

[35] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In *MobiSys'09*, Krakow, Poland, June 2009.

[36] S. Nath. ACE: Exploring Correlation for Energy-Efficient and Continuous Context Sensing. In *MobiSys'12*, Lake District, UK, June 2012.

[37] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: middleware for mobile social networking. In *WOSN'09*, Barcelona, Spain, August 2009. ACM.

[38] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas. EmotionSense: A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research. In *UbiComp'10*, Copenhagen, Denmark, September 2010.

[39] L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden. Code In The Air: Simplifying Sensing and Coordination Tasks on Smartphones. In *HotMobile'12*, San Diego, CA, USA, February 2012.

[40] A. Sharma, V. Navda, R. Ramjee, V. Padmanabhan, and E. Belding. Cool-Tether: Energy Efficient On-the-fly WiFi Hot-spots using Mobile Phones. In *CoNEXT'09*, Rome, Italy, December 2009.

[41] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In *SenSys'09*, Berkeley, CA, USA, November 2009.

[42] A. Toninelli, A. Pathak, and V. Issarny. Yarta: A Middleware for Managing Mobile Social Ecosystems. In *International Conference on Grid and Pervasive Computing (GPC'11)*, Oulu, Finland, May 2011.

[43] T. Wang, G. Cardone, A. Corradi, L. Torresani, and A. T. Campbell. WalkSafe: A Pedestrian Safety App for Mobile Phone Users Who Walk and Talk While Crossing Roads. In *HotMobile'12*, San Diego, CA, USA, February 2012.

[44] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, and B. Krishnamachari. A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In *MobiSys'09*, Krakow, Poland, June 2009.

[45] D. Zhang. Web content adaptation for mobile handheld devices. *Communications of the ACM*, 50(2):75–79, 2007.

[46] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *CODES/ISSS'10*, Scottsdale, AZ, USA, October 2010.