frontiers in
ICT

TECHNOLOGY REPORT
published: 20 April 2015
doi: 10.3389/fict.2015.00006

AWARE: mobile context instrumentation framework

***Denzil Ferreira*[1]*, *Vassilis Kostakos*[1]* and *Anind K. Dey*[2]**

[1] *Community Imaging Group (COMAG), Department of Computer Science and Engineering, University of Oulu, Oulu, Finland*
[2] *Ubicomp Laboratory, Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, USA*

Edited by:
Javier Jaen, Universitat Politècnica de València, Spain

Reviewed by:
Carlos Duarte, Universidade de Lisboa, Portugal
Rubén San-Segundo, Universidad Politécnica de Madrid, Spain

***Correspondence:**
Denzil Ferreira and Vassilis Kostakos, Community Imaging Group (COMAG), Department of Computer Science and Engineering, University of Oulu, Erkki Koiso-Kanttilan katu 3, Door E, Oulu FI-90014, Finland
e-mail: denzil.ferreira@ee.oulu.fi; vassilis@ee.oulu.fi

We present a mobile instrumentation toolkit, AWARE, an open-source effort to develop an extensible and reusable platform for capturing, inferring, and generating context on mobile devices. Mobile phones are sensor-rich but resource-constrained, and therefore several considerations need to be addressed when creating a research tool that ensures problem-free context collection. We demonstrate how AWARE can mitigate researchers' effort when building mobile data-logging tools and context-aware applications, with minimal battery impact. By encapsulating implementation details of sensor data retrieval and exposing the sensed context as higher-level abstractions, AWARE shifts the focus from software development to data analysis, both quantitative and qualitative. We have evaluated AWARE in several case studies and discuss its use, power consumption, and scalability.

Keywords: context-aware systems, ubiquitous data capture framework, mobile toolkit, mobile sensing, user studies, mobile questionnaires, ESM

INTRODUCTION

Mobile phones have become miniaturized computers that fit in a pocket. They are inherently personal and their potential to sense the user's environment, i.e., context, is appealing to researchers. The convenience and availability of mobile phones and application stores make it easier for a researcher to reach thousands of users. More importantly, mobile phones have increasingly more built-in sensors (e.g., accelerometer, gyroscope, luminance). They are primarily used to enhance the user experience, such as application functionality or mobile phone user interaction (e.g., vibration feedback, screen orientation detection), but are nowadays being leveraged for research purposes.

Despite the many contributions in mobile computing research, there are still challenges to overcome. First, it still remains laborious for scientists to conduct human subject studies that involve mobile devices. Second, developers who wish to create context-aware applications on mobile phones typically need to start from scratch. Lastly, end-users need to use multiple and often isolated (i.e., not sharing data) applications to make their device more context-aware. We argue that these three important limitations are due to the same reason: there is a lack of open and reusable software for creating context-aware applications on mobile device.

To address this need, we designed and created AWARE (accessible at http://www.awareframework.com, since 2011), an open platform for context-aware mobile computing research, application development, and for the end-user. AWARE allows creating new mobile research tools for data mining, visualization, and analysis that builds on previous development. More importantly, it has the potential to enable access to the wider range of interrelated sources of context information and their relationships, including the user's individual and social behavior.

This manuscript includes what were AWARE's requirements as a mobile platform for context and data collection, supported by the related work. We also include an empirical evaluation of AWARE's impact on mobile power consumption and server scalability. More importantly, we demonstrate AWARE's use in several use-cases (laboratory, small and large-scale deployments), by the authors and fellow scientists.

RELATED WORK

The Context Toolkit (Dey et al., 2001) is the reference conceptual framework for developing context-aware applications. It separates the acquisition and representation of context from the use of context by an application. In other words, context should always be present, independently its use in applications. However, since the Context Toolkit was introduced in 2001, computing has become increasingly mobile and so has the user's context. Research in the field of mobile computing is challenging due to the mobile devices' limited storage, power, and network capabilities. Over the years, several research tools were developed that weight these challenges in order to gain a better insight into users' mobile context. We categorized contextual functionalities that are desirable on mobile context middleware based on Context Toolkit's guidelines on use of context and referring to previous work's most notable features.

EVENT-BASED, MANUAL, SEMI- OR FULLY AUTOMATED CONTEXT GENERATION

It is challenging to fully automate actions based on sensed context alone. Context can be generated by user rules (i.e., *manual*), and triggered by *events* (i.e., system, user activity) from *semi-* or *full-automated* algorithms. *CORTEX* (Biegel and Cahill, 2004) uses the concept of a sentient object model for the development of mobile context-aware applications. By combining sentient objects and an event-based communication protocol for *ad hoc* wireless environments, *CORTEX* allows researchers to define inputs and outputs, contexts, fusion services and rules using an inference engine. On the other hand, *Context Studio* (Korpipää et al., 2004) takes into account users' mediation and accountability in context inference.

Users can combine the existing contextual probes to incrementally add context-awareness to the mobile phone.

USER-GENERATED AND USER-MANAGED CONTEXT

In fact, a human can be a context sensor, e.g., as a user he provides the data and manages his current context. For example, *Momento*'s (Carter et al., 2007) mobile client displays questions to the user about their location, nearby people, and audio. The researcher has a desktop client to configure and oversee a remote deployment. Similarly, *MyExperience* (accessible at http://myexperience.sf.net, since 2007) captures both sensor- and human-based data to understand the user's motivation, perception, and satisfaction on mobile technology. The human-based data collection [e.g., surveys and experience sampling methods (ESM)] is triggered off sensor readings and pre-established researcher's rules, and later synchronized to a remote server. *EmotionSense* (accessible at http://www.emotionsense.org, since 2013) probes the user for social psychology context, inquiring about individual emotions, activities, and verbal as well as proximity interactions among friends using the mobile phone's microphone.

VISUALIZATIONS OF CONTEXT INFORMATION

Presenting and managing contextual information is imperative for the end-users. *ContextPhone* (Raento et al., 2005) emphasizes context as an understandable resource to the user. Using application widgets, users had control over the sensors data collection. Similarly, *AWARENESS* (van Sinderen et al., 2006) prioritizes users' privacy concerns. It applies the concept of quality of context (QoC). Users' privacy concerns would increase or decrease QoC, depending on how much context is shared at any given time (e.g., disabling GPS would reduce the QoC for the context of location). The context is then shared with previously trusted devices and the mobile phone user is the sole controller of privacy aspects. "Self-tracking" users may use *Funf* (accessible at http://www.funf.org, since 2011) to collect their personal mobile data.

AD HOC CONTEXT EXTENSIBILITY VIA COMPONENTS (I.E., PLUGINS)

Context sources keep emerging and changing, thus it is important that a middleware may be adapted and extended, ideally without further development. *OpenDataKit* (accessible at https://opendatakit.org, since 2012) simplifies the interface between external sensors and mobile phones by abstracting the application and driver development from user applications and device drivers, by management of discovery, communication channels, and data buffers. It is component-based, allowing developers to focus on writing minimal pieces of sensor-specific code, enabling an ecosystem of reusable sensor drivers. Integration of new sensors into applications is possible by downloading new sensor capabilities from an application market, without modifications to the operating system.

EXCHANGE OF CONTEXT EVENTS AND DATA ACROSS DEVICES AND PLATFORMS

Also, desirable is a cross-platform data collection middleware. Leveraging the browser, *Ohmage* (accessible at http://www.ohmage.org, since 2012) is a smartphone-to-web toolkit designed to create and manage experience sampling-based data collection campaigns, accessible in multiple platforms, in support of mobile health pilot studies. Similarly, CenceMe (Miluzzo et al., 2008) infers physical social context and shared information through social network applications to other devices.

REMOTE CONTROL OF CONTEXT ACQUISITION (I.E., DASHBOARDS, STUDIES, COLLABORATION)

To enable multidisciplinary collaboration, web-based dashboards have been created to orchestrate large-scale and distributed studies. Emotional Monitoring for PATHology (*Empath*) (Dickerson et al., 2011) allows to remotely monitor emotional health for depressive illness. *Ginger.io* (accessible at http://www.ginger.io, since 2011) is another behavioral analytics web-based dashboard that turns mobile data (e.g., movement, call, and texting patterns) into health insights.

REMOTE DATA OFFLOAD FOR ASYNCHRONOUS CONTEXT INFERENCE

Mobile context may also be inferred externally to the device, asynchronously. For example, CenceMe (Miluzzo et al., 2008) infers the social context detected locally on the device, then transfers processing to a backend server to match common shared social contexts to raise social awareness. *Momento*'s (Carter et al., 2007) integrated with a Context Toolkit server to analyze audio segments to detect proximity of people.

AWARE

AWARE CLIENT

AWARE is inspired by **Table 1** references. We would like to emphasize that AWARE's novelty lies not on its functionalities individually, but instead on their collective use. More importantly, it is a toolkit that builds on previous work's core functionalities and makes them available to researchers, developers, and end-users.

AWARE is available as a regular Android mobile application. The collected data is primarily stored locally on the device and it is not shared remotely, ideal for small scale and local user-studies. For distributed and large-scale studies, the client uploads sensor and plugin data to the cloud. The user interface was designed to allow users to control the sensor (**Figure 1** – Sensor Manager) and plugin functionality (**Figure 1** – Stream), thus supporting functionality 2.2. The client also allows the user to further install plugins (**Figure 1** – Plugin Manager), which extend AWARE's capabilities (supporting 2.4).

By default, the users are presented with the Sensor Manager interface, where they can manage the collected sensor data on the mobile phone. Here, they can see currently active sensors, the sensor's power-consumption estimation (provided by Android's API), the client's current version, their AWARE device identifier [a universal unique identifier (UUID) is generated randomly upon every installation] and a toggle for debugging messages from AWARE's sensing.

By pressing to top-left corner icon, i.e., the navigation button, the users can access the Plugin Manager. Here, they may install any publicly available plugins (as soon as they are added to the repository). Moreover, for researchers, AWARE supports study-specific plugins, thus only the study participants are able to view and install them. If the client or a plugin is updated on the repository or missing from the device, the users are prompted to install it.

Table 1 | Overview of supported context-related functionalities.

Frameworks	Functionalities						
	2.1 Generation	2.2 User-managed	2.3 Visualizations	2.4 Extensibility	2.5 Exchange	2.6 Remote control	2.7 Offload
CORTEX	√		√	√			
Context Studio	√	√		√			
ContextPhone	√	√	√		√		
AWARENESS	√	√	√		√	√	
Momento	√	√			√	√	√
MyExperience.sf.net	√	√				√	
CenceMe	√		√		√		√
EmotionSense.org	√	√	√				
Empath	√	√	√			√	√
Funf.org	√	√	√	√			
Ginger.io	√	√	√			√	
Ohmage.org	√	√	√			√	
OpenDataKit.org	√			√			

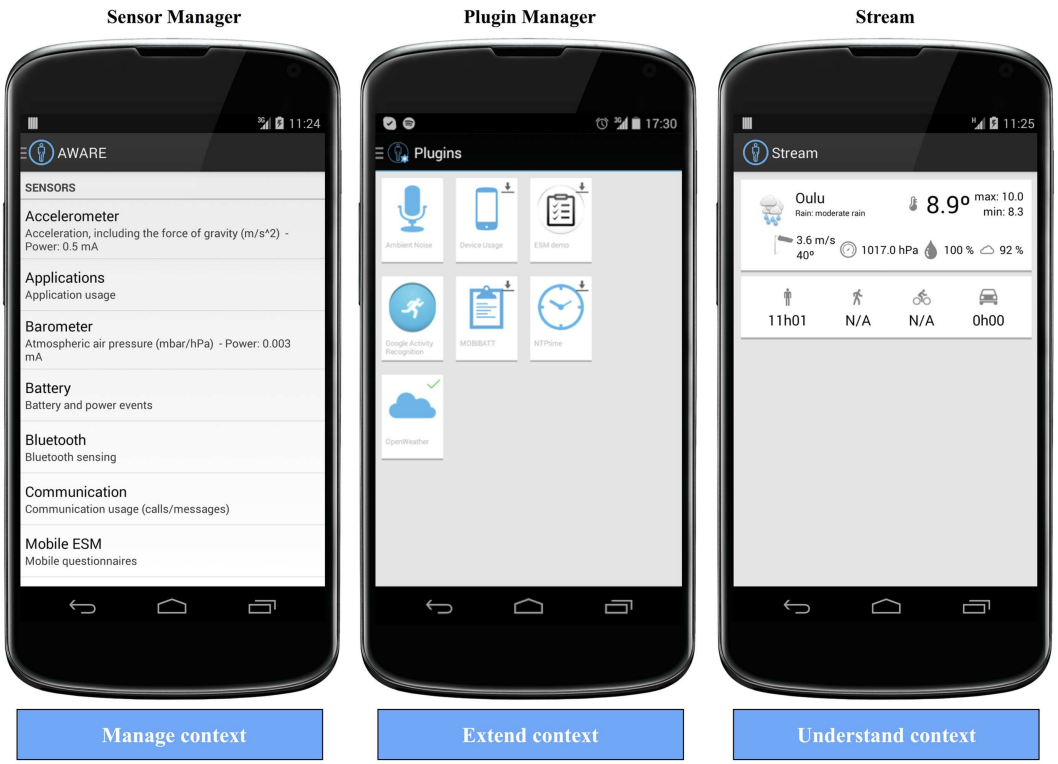

FIGURE 1 | Overview of AWARE client's interfaces: sensor manager, plugin manager, and stream.

The Stream interface's purpose is to present contextualized information from the collected data. These visualizations may be interactive, present information in real-time and reused for faster future application development. They also act as a shortcut to each plugin's settings. This interface adapts by hiding or showing more information according to the active sensors and plugins.

AWARE's context sensors

AWARE records data from available hardware-, software-, and human-based sensors as the users carry their smartphones, thus supporting 2.1 and 2.2. Hardware sensors can be the accelerometer, magnetometer, and photometer, just to name a few. The client supports the majority of device's available sensors. New sensors are constantly added to the framework as they become available

on newer Android devices. Software sensors include, for example, the user's calendar, email, social activity, and other logs (e.g., calls, messages). Human-based sensors are mobile questionnaires (i.e., for ESM), voice, or gesture input, where users' actions are the inputs to algorithms, providing answers for disambiguation, or supervised data labeling, for example.

AWARE's context plugins

The locally stored data (in SQLite databases) can then be abstracted as context using AWARE plugins by means of data analysis. AWARE plugins' primary goal is to collect and abstract sensor data to generate context. They may reuse other sensor and plugin data and context to create new higher-level contexts using data mining and machine learning techniques (implemented within the plugins' code), and may provide support for a new – either external or internal – sensor, thus collecting and analyzing data. Second, AWARE plugins may also present and explain context data to the user using context cards, thus supporting 2.3. These cards allow end-users to use, explore, and interact with – and application developers to *reuse* – context for their smartphone applications. These plugins extend *ad hoc* AWARE's functionality, thus supporting 2.4.

Context data sharing

The collected context is shared (thus, supporting 2.5) among AWARE sensors, plugins, and applications using three strategies simultaneously: *broadcasts*, *providers*, and *observers*:

- *Broadcasts* quickly update other sensors, plugins, and applications with user's context. Each receives a brief description of user's current context (i.e., as a string), regardless of the data captured with such context. Multiple broadcasts can be received simultaneously from different sources (e.g., sensors, plugins, applications).
- *Providers* store sensor data and plugin context data. The data are stored locally on the device, and if required, remotely on a MySQL server. Sensors and plugins may request (i.e., pull) the data by querying the data directly, or subscribe to it using Observers. Using AWARE's web API, cloud services, such as Google's Cloud and App Engine, Amazon's EC, and Microsoft's Azure may also access and use the context data.
- *Observers* monitor changes to the sensor data and plugin context data in real-time, sharing updates to other remote devices using message queue telemetry transport (MQTT) message callbacks. Observers provide active (i.e., push) and event-based access to context. They are energy efficient and support real-time data labeling.

AWARE SERVER

The client may upload sensor and plugin data to the cloud (**Figure 2**), if the user signed-up for a study. A study can be managed on AWARE's dashboard (accessible at https://api.awareframework.com, supporting 2.6). For application developers and researchers, new plugins can be shared by uploading the compiled package (e.g., a digitally signed.APK file) to their dashboard. The AWARE server offers the following data handling functionalities: data replication, issue remote commands to AWARE devices and exchange context's data, and visualize the data online.

Remote data synchronization

The data replication takes place within the sensor and plugin data synchronization code, and is transparent to the users and developers. The replication process can be scheduled, triggered locally with an event, or remotely on-demand. The clients' data are sent as JavaScript Object Notation (JSON) objects via hypertext transfer protocol secure (HTTPS) POSTs, replicating the devices' data into a MySQL database. Different strategies are in place to attempt to minimize data collection loss, such as the use of processor wakelocks to keep the sensors alive even when the phone is idle, multithreading to reduce delays in context storage and allow cooperation between multiple sensors and exception fallbacks to decide what to do if one sensor is unavailable or is, for some reason, faulty.

AWARE uses MQTT for exchanging context messages in a publish/subscribe approach between mobile phones and other servers and devices. MQTT is designed for constrained devices and low-bandwidth, high-latency or unreliable networks, and supports persistence, both on the server and mobile phone, i.e., if the device or the server is unreachable, the data are queued locally for delivery at a later time. The built-in MQTT infrastructure provides support for distributed context and provides real-time context data exchange to available AWARE devices. MQTT servers can be clustered to support load-balance. For added security, MQTT also supports secure and authenticated connections. Using MQTT as a delivery mechanism, the dashboard can issue commands that are delivered to other AWARE devices and servers (supporting 2.7). On the other hand, AWARE devices can also issue commands to other devices and exchange context data as MQTT messages.

Human-based sensing

AWARE supports a flexible ESMs questionnaire-building schema (defined in JSON) for *in situ* human-based context sensing. Diverse ESM questions can be chained together to support a step-by-step questionnaire. ESM can be triggered by context events, time, or on-demand, locally (with broadcasts) or remotely from the dashboard and servers (with MQTT). Although it collects subjective user input, it allows us crowdsourcing information that is challenging to collect through physical sensors, as follow:

- *Free text*: allows the user to provide free text input as context. This can be leveraged to capture sensor-challenging context, such personal opinions, moods, and others;
- *Radio*: allows the user to select a single option from a list of alternatives. One of the options can be defined as "Other," which will prompt the user to be more specific, replacing "Other" with the users' input;
- *Checkbox*: allows the user to select one or more options from a list of alternatives. Similar to the Radio ESM, one of the options can be defined as "Other";
- *Likert*: allows the user to provide ratings, between 0 and 5/7, at 0.5/1 increments. The Likert-scale labels are also customizable. The default rating is no rating;
- *Quick*: allows the user to quickly answer the ESM by pressing a simple button.

Privacy and security

AWARE has been approved for several research projects by the Institutional Review Board (IRB) of our university (Europe), and also in US. Keeping users' privacy in mind, AWARE obfuscates and encrypts the data using a one-way hashing of logged personal identifiers, such as phone numbers. Increased security is achieved with application permissions, certificates, user authentication, and the use of secure network connections to access and transfer the logged data between the client and the dashboard.

CREATING CONTEXT-AWARE APPLICATIONS

AWARE follows Android architecture and development guidelines. In other words, AWARE is available from MavenCentral repository[1] ("compile com.awareframework:aware-core:*@aar") as an open-source (Apache 2) library, which can be added to any Android development application or plugin using Android Studio's Gradle mechanism. The source code of the client is also available in GitHub[2] and we encourage fellow researchers to report any issues, discuss strategies, and further extend our work.

We offer more detailed tutorials and documentation on AWARE's website. In summary, once added to the application's Gradle, a developer has full access to AWARE's API public methods [e.g., Aware.startPlugin(), Aware.stopPlugin(), Aware.setSetting(), Aware.getSetting(), and many others], Providers (i.e., ContentProvider databases), Broadcasts (i.e., Intents), Services, and Observers. Since we followed Android architecture, a Provider is a ContentProvider, a Plugin is a Service, and an Observer is a ContentObserver, and so on. For an Android developer, using AWARE should is an extension to Android's native classes and API, with increased number of functionalities. As an example, if the user is available (i.e., not in a call and not using the device), a high-level context broadcast (e.g., ACTION_AWARE_USER_NOT_IN_CALL)[3] is automatically broadcasted. An application simply needs to listen to this broadcast and may act upon it.

AWARE's OVERVIEW

In summary, AWARE's infrastructure (**Figure 2**) allows:

- *Researchers* to:
  - Self-host or use AWARE-hosted databases;
  - Manage multiple, concurrent studies;
  - Publish study-specific or public plugins;
  - Specify and manage active sensors and plugins for a study;
  - Enroll participants via a QRCode or a direct link;
  - Collaborate with registered co-researchers;
  - Oversee studies in real-time (e.g., amount of data, user participation);
  - Label and group sets of participants' devices;
  - Remotely create and request a mobile questionnaire (i.e., ESM);
  - Remotely request or clear a participant's study data from the databases (on the client and server).
- *Developers* to: embed AWARE and plugins as libraries, thus facilitating the creation of a context-aware application;
- *Users* to: to collect personal data and visualize contextualized information of their daily lives directly on their smartphones.

CASE STUDIES AND EVALUATION

By encapsulating implementation details on sensor retrieval and exposing the sensed data as higher-level abstractions, we shifted our focus from software development to research and analyzing the collected data, both quantitative and qualitative. For brevity, please refer to the case studies references for further

[1]http://search.maven.org - replace * with the updated version code (e.g., 3.3.3 as of 19/3/2015).

[2]http://github.com/denzilferreira/aware-client

[3]http://www.awareframework.com/communication/

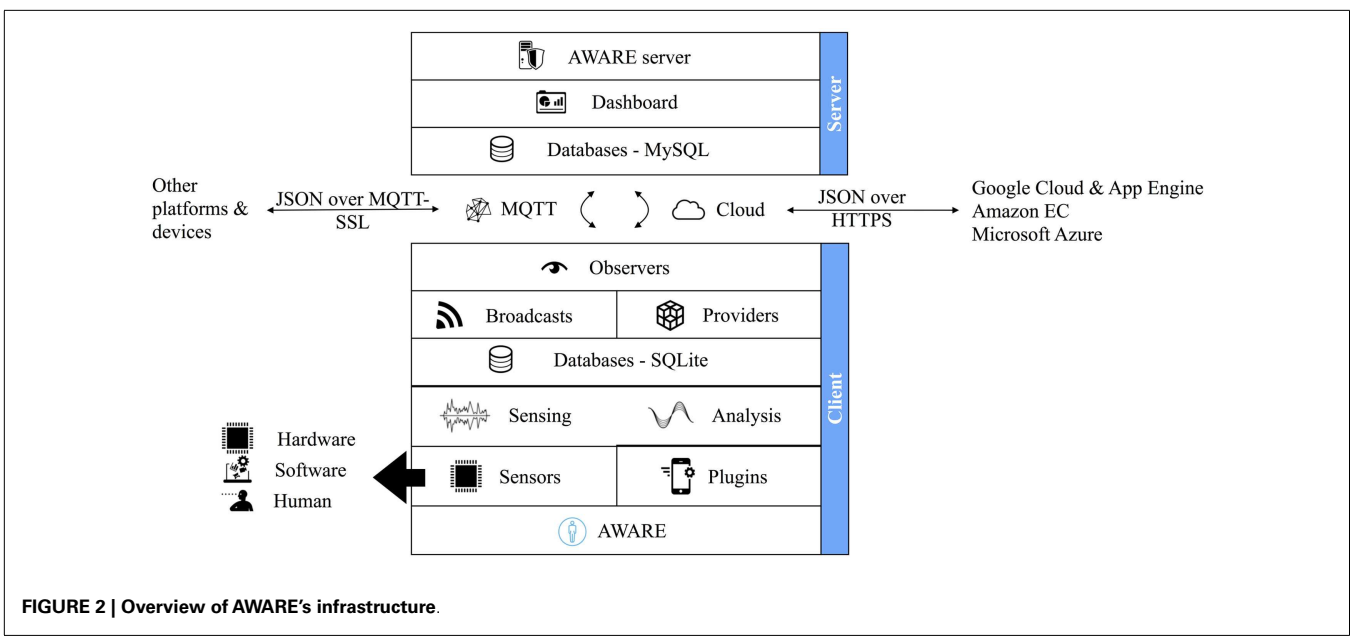

FIGURE 2 | Overview of AWARE's infrastructure.

details on AWARE's active sensors and plugins combinations and usage.

In Dey et al. (2011), AWARE was used to abstract several built-in mobile sensors together (e.g., location, Wi-Fi, network, and more) and external Bluetooth sensors to capture users' proximity habits to their mobile devices. For each data source (e.g., sensor), several metrics of user proximity were created. For example, the Bluetooth phone-sensor RSSI readings and distance calibrations allowed us to investigate the amount of time a phone was at arm, and/or room reach; the accelerometer's axial forces allowed us to determine device's on-body/free moments. Using a decision tree classifier and a time window of 1 min, we were able to create a predictive model accurate up to 83% on whether the device is close/far from the user. The main objective was to understand how often context information can be visually presented to the user. Contrary to our intuition, the device is not really with the user all the time (approximately only 50% of the time). In Ickin et al. (2012), AWARE is used to investigate the quality of experience (QoE) of commonly used mobile applications depending on user's current location, social, and mobility context. AWARE's ESM functionality allowed users to rate *in situ* multiple applications' QoE using the mean opinion score (MOS) ranking, obtain a high-level description of the users' current location (e.g., home, work), social (e.g., alone, with someone, with a group), and mobility context (e.g., sitting, standing, walking, driving). By data mining network and phone performance data (i.e., network signal, available battery), several metrics were created to support the best applications' QoE.

In Ferreira et al. (2011), we used AWARE as a library in a battery user-study application on the Google Play application store, downloaded, and used by thousands of users, where we studied users' concerns regarding battery life and charging behavior. In Ferreira et al. (2013), and reusing Ferreira's sensors (Ferreira et al., 2011) (e.g., battery and application usage sensors), we identified applications' battery impact in its depletion (using linear regression), thus creating a new interactive battery interface for power management on smartphones. In Ferreira et al. (2014), AWARE's screen, application usage, and ESMs functionalities provide insight on the context in which applications are micro-used (i.e., within 15 s) and in Van den Broucke et al. (2014), AWARE's device, location, and network sensors were used to investigate the users' struggles when using mobile cloud computing on their smartphones and tablets.

BATTERY IMPACT AND RESOURCE MANAGEMENT

As a toolkit, it is challenging to evaluate AWARE for all possible scenarios. However, to evaluate AWARE's mobile client battery impact, we accounted for three sensing, storage, and networking scenarios: sensing only (S); sensing and local storage (SL); sensing, local, and remote storage (SLR). In S, we simply enable the sensors but do not record data, thus isolating any storage operations' overheads; and we account for them with SL. In SLR, we also capture the network operations' overheads by uploading the collected data to our AWARE's server over 3G every 30 s. We conducted our experiments on an off-the-shelf LG Nexus 4 (2100 mAh, 4.2 V battery). To reduce a potential measurement bias, we used a minimal version of Android (e.g., AOSP) with no third-party applications (i.e., Google, appstore applications) or background synching. Before deploying AWARE, this reference device's battery on average depletes at a rate of 18.45 mAh (Min = 4.64; Max = 92.66; SD = 12.62). With AWARE, and monitoring sensors' and plugins' statuses (i.e., check if active or updated at 5-min intervals), battery depletion increases to 19.74 mAh (Min = 4.86; Max = 118.2; SD = 13.87), however, not statistically significant [Welch $t(11841) = -0.753$, $p = 0.45$], i.e., a negligible difference in battery life when AWARE is not in active use.

During the tests, we consistently kept the device's display off, while 3G, Wi-Fi, GPS, and Bluetooth were active as to better simulate device's every day use. With Qualcomm's Trepn Profiler, a diagnostic tool for evaluating in real-time the performance and power consumption of Android applications, we measured the power consumption (mA) and processor load (%, normalized to account all cores) of each high-performance sensor (i.e., capable of several samples per second – accelerometer, gyroscope, etc.) when in individual use and at two sampling rates: 5 and 100 Hz, in 10-min long tests. For low-performance sensors, i.e., that require frequent polling for updates – applications (refresh background processes), Bluetooth (scanning), locations (update requests), network (amount of traffic), processor (processing load), and Wi-Fi (scanning) – we tested them at 5-, and at 60-s intervals. The remaining sensors are only event-based and therefore it is challenging to measure their power consumption as these events are triggered exclusively by the operating system (**Figure 3**).

Across all the sensors and sampling rates, AWARE's client overall battery impact is, on average, 19.69 mA (SD = 23.34) for sensing only; 24.69 mA (SD = 23.88) with local storage; and 138.03 mA (SD = 29.92) if also connected to AWARE's server. In other words, if a researcher would run a study where participants upload their data automatically (SLR condition) every 30 s, and if they used the same reference device, they could theoretically provide 15 h and 21 min worth of sensor data. More importantly, in our real-world deployments, our participants did not report a significant decrease in their device's battery life or performance.

We have built-in several strategies to reduce power-consumption of AWARE: event-based sampling and opportunistic analysis (e.g., only when charging or the battery is above a certain amount) and scheduled synching. Similar to the literature review (Biegel and Cahill, 2004; Falaki et al., 2011), this suggests that AWARE's event-based sensing approach is more efficient than periodic polling. We also expect that novel battery technology and power-efficient sensors will reduce even further AWARE's battery impact in newer devices. During all the tests, AWARE consumed on average approximately 32.9 MB of internal memory (Min = 27.4; Max = 33.19; SD = 1.53), regardless of the sampling rate, as Android's memory management reclaims unused resources automatically. Noteworthy, the data are scheduled for weekly or monthly space maintenance (i.e., remove data that is already replicated on the server) if required, releasing the storage space back to the device.

SCALABILITY AND PERFORMANCE

We also evaluated AWARE's server dashboard scalability and performance with an increasing number of participants (e.g., 1 thousand, 10 thousand, 100 thousand, 1 million, 10 million), and

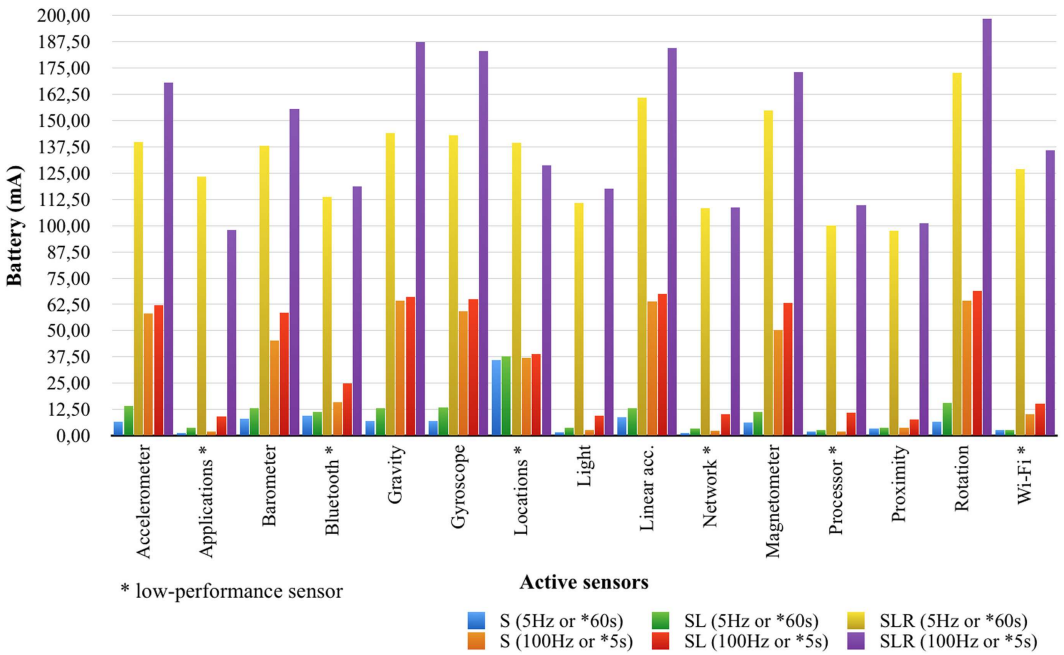

FIGURE 3 | Battery consumption per active sensor and sampling condition.

Table 2 | AWARE's server dashboard performance per amount of devices.

	Amount of devices				
	1 Thousand	**10 Thousand**	**100 Thousand**	**1 Million**	**10 Million**
Functionalities					
Page load	1.43 s	1.66 s	1.75 s	2.82 s	10.23 s
Devices	0.12 s	0.13 s	0.29 s	0.81 s	7.09 s
Visualization	0.19 s	0.22 s	0.59 s	0.68 s	2.07 s
Sort	0.32 s	0.44 s	0.99 s	1.73 s	14.16 s
Pagination	0.35 s	0.42 s	0.91 s	1.59 s	9.17 s
Search	0.31 s	0.29 s	0.68 s	1.67 s	9.23 s

measured the amount of time of: page load (load the whole dashboard's interface); devices (load the devices' data); visualization (visualize the devices' data); sort (e.g., order by device ID, by amount of data); pagination (switch to next group of devices); and search (lookup query for specific data or device). To test these conditions, we used a Python script that mimics the client's JSON object synching mechanism to the server and inserts 100 thousand data points per device in all the sensors' databases. Our evaluation was conducted on a modest researcher server machine: Linux-based, 4-core central processing units (CPUs), 8 GB of random access memory (RAM), and 300 GB of storage (**Table 2**).

We found that up to 1 million participants, the dashboard's overall performance takes on average 0.85 s (Min = 0.12; Max = 2.82; SD = 0.71) to be fully operational, and degrades upon 10 million participants, with an acceptable average of 8.66 s (Min = 2.07; Max = 14.16; SD = 3.98). We optimized the dashboard's performance by only loading the data if it is requested by its user, pagination is limited to groups of 50 devices, or only request the data for a given day. Using cloud-computing services such as Amazon EC or Google Cloud, one could run multiple instances of AWARE servers and do load-balance.

CONTRIBUTIONS

As a research tool, AWARE encapsulates and reuses sensors and plugins for different research questions. More importantly, it is a toolkit that supports collaboration with other researchers. Because it can be packaged as a library, application developers can request context data from AWARE's sensors and plugins, and, with the users' consent, have access to high-level context inferences provided by the context sharing mechanisms. The AWARE API follows the reference conceptual Context Toolkit (Dey et al., 2001) context-aware application development requirements:

- *Separation of concerns*: each sensor and plugin collects data, independently of where it is used or how it is used;
- *Context interpretation*: abstractions of multiple layers of context are transparent to the researcher and developer, using the Providers and the Broadcasts;
- *Transparent, distributed communications*: the context sharing mechanisms (e.g., Provider, Observer, Broadcast, MQTT) offer transparent communication between context sensors, plugins, applications, and devices;
- *Constant availability of context acquisition*: the context sensors operate independently from the applications that use them;
- *Context storage*: context is stored locally, and optionally remotely, and can be used to establish trends and predictions of context;
- *Resource discovery*: for an application to communicate with a provider, it needs to know where the context is stored, provided by the content URI.

The plugins allow developers to work independently – or to collaborate with others – by isolating plugins' core functionality from their ultimate research and application development use. The toolkit abstracts sensor data acquisition and processing into reusable context components, from local and external sensors. Moreover, AWARE's context sharing mechanisms offer transparent context data exchange between other plugins and applications, both locally and remotely. AWARE plugins provide the building blocks to extend the toolkit to support novel contexts and sensors, on an *ad hoc* basis. The AWARE server supports server clustering for load-balance and scaling up the number of instances of the servers as required. Furthermore, we note that mobile phones remain resource-constrained environments, in terms of network, storage, battery life, and processing power. Thus, AWARE relies on MQTT messaging protocol resilience mechanisms to exchange context data between different devices while being network efficient. This allows it not only to overcome storage limitations but also support redundancy: context data can be stored locally and synchronized remotely.

LIMITATIONS

Despite our best efforts, we cannot guarantee a problem-free mobile data collection tool. AWARE is designed as an Android accessibility service to increase its importance to the OS task manager and reduce termination likelihood. AWARE supports Android 2.3 or higher; however, sensor deprecation might limit or replace, as needed, some functionality as the toolkit and Android evolves. Also, more robust security procedures must be considered to protect context data, since encryption and obfuscation can be circumvented. As a fallback, AWARE does not collect personal data and hashes personal identifiers to protect users' privacy. Nonetheless, this alone might not be enough for other more privacy strict domains and procedures, such as in health-care. Lastly, for the moment, the AWARE mobile application is only available for Android devices (e.g., tablets, Google Glass, smartphones, Android Wear). However, the AWARE server and MQTT functionalities allow exchange and use of context information across platforms with the support of JSON and MQTT.

CONCLUSION

Making the transition from mobile phones to "smartphones," in the true sense of the word, requires more tools that offer programing and development support. However, the development of context-aware applications still remains challenging because developers have to handle with the raw sensor data, analyze it to produce context, and often are forced to start from scratch. There is a lack of a coherent and modular repository of relevant tools, which motivated AWARE.

We must acknowledge that AWARE is no single ready-to-use, fits-all, "silver bullet" solution that would meet the requirements of all possible researchers. Research fragmentation is the biggest challenge for such toolkits. As such, we argue that a mobile instrumentation toolkit must support multidisciplinary research and collaboration from the ground-up. We believe that AWARE plugins provide a mechanism for extending, adapting, and evolving on an *ad hoc* basis, potentially withstanding the test of time.

Some of the key characteristics of AWARE include the ability to

- combine sensor data from multiple plugins to create new context abstractions in real-time;
- dynamically configure plugins, update plugins, or install new plugins from an online repository;
- store data locally and/or remotely;
- provide an API that any application can use to access high-level context (e.g., "is the user sitting?") and take action.

However, having access to context information is just a first step. Context-aware applications must do a lot more: present information, acquire and store context information, and relate new information to other captured information. AWARE not only collects, manages, shares, and presents context data but also reuses context information. Naturally, there are still open challenges for AWARE, but we leave that for future work and invite fellow researchers to contribute at http://www.awareframework.com.

ACKNOWLEDGMENTS

Funded by the Academy of Finland project 137736, 276786, and TEKES project 2932/31/2009.

REFERENCES

Biegel, G., and Cahill, V. (2004). "A framework for developing mobile, context-aware applications," in *PerCom* (Orlando, FL: IEEE), 361–365.

Carter, S., Mankoff, J., and Heer, J. (2007). "Momento: support for situated ubicomp experimentation," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, CA: ACM), 125–134.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum. Comput. Interact.* 16, 97–166. doi:10.1207/S15327051HCI16234_02

Dey, A. K., Wac, K., Ferreira, D., Tassini, K., Hong, J.-H., and Ramos, J. (2011). "Getting closer: an empirical investigation of the proximity of user to their smart phones," in *Ubicomp* (Beijing: ACM), 163–172.

Dickerson, R. F., Gorlin, E. I., and Stankovic, J. A. (2011). "Empath: a continuous remote emotional health monitoring system for depressive illness," in *Proceedings of the 2nd Conference on Wireless Health* (San Diego, CA: ACM), 5:1–5:10.

Falaki, H., Mahajan, R., and Estrin, D. (2011). "SystemSens: a tool for monitoring usage in smartphone research deployments," in *Proceedings of the Sixth International Workshop on Mobiarch* (Washington, DC: ACM), 25–30.

Ferreira, D., Dey, A. K., and Kostakos, V. (2011). "Understanding human-smartphone concerns: a study of battery life," in *Pervasive* (Berlin: Springer-Verlag), 19–33.

Ferreira, D., Ferreira, E., Goncalves, J., Kostakos, V., and Dey, A. K. (2013). "Revisiting human-battery interaction with an interactive battery interface," in *Ubicomp* (Zurich: ACM), 563–572.

Ferreira, D., Goncalves, J., Kostakos, V., Barkhuus, L., and Dey, A. K. (2014). "Contextual experience sampling of mobile application micro-usage," in *MobileHCI* (Toronto: ACM), 91–100.

Ickin, S., Wac, K., Fiedler, M., Janowski, L., Hong, J.-H., and Dey, A. K. (2012). Factors influencing quality of experience of commonly used mobile applications. *IEEE Comm. Mag.* 50, 48–56. doi:10.1109/MCOM.2012.6178833

Korpipää, P., Häkkilä, J., Kela, J., Ronkainen, S., and Känsälä, I. (2004). "Utilising context ontology in mobile device application personalisation," in *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia* (Maryland: ACM), 133–140.

Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., et al. (2008). "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems* (Raleigh, NC: ACM), 337–350.

Raento, M., Oulasvirta, A., Petit, R., and Toivonen, H. (2005). ContextPhone: a prototyping platform for context-aware mobile applications. *IEEE Pervasive Comput.* 4, 51–59. doi:10.1109/MPRV.2005.29

Van den Broucke, K., Ferreira, D., Goncalves, J., Kostakos, V., and De Moor, K. (2014). "Mobile cloud storage: a contextual experience," in *MobileHCI* (Toronto: ACM), 101–110.

van Sinderen, M. J., van Halteren, A. T., Wegdam, M., Meeuwissen, H. B., and Eertink, E. H. (2006). Supporting context-aware mobile applications: an infrastructure approach. *IEEE Comm. Mag.* 44, 96–104. doi:10.1109/MCOM.2006.1705985

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 25 February 2015; accepted: 31 March 2015; published online: 20 April 2015.
*Citation: Ferreira D, Kostakos V and Dey AK (2015) AWARE: mobile context instrumentation framework. Front. ICT **2**:6. doi: 10.3389/fict.2015.00006*
This article was submitted to Human-Media Interaction, a section of the journal Frontiers in ICT.