Context Management for Supporting Context-aware Android Applications Development

https://doi.org/10.3991/ijim.v11i4.6952

Hanan Elazhary
King Abdulaziz University, Jeddah, Saudi Arabia
Electronics Research Institute, Cairo, Egypt
helazhary@kau.edu.sa;hananelazhary@eri.sci.eg

Alaa Althubyani, Lina Ahmed, Bayan Alharbi, Norah Alzahrani, Reem Almutairi
King Abdulaziz University, Jeddah, Saudi Arabia
{aathubyani,lahmed0002,balharbi0001,nalzahrani0084,ralmutairi0017}@stu.kau.edu.sa

Abstract—Building context-aware mobile applications is one of the most ambitious areas of research. Such applications can change their behavior according to context or perform specific tasks in specific contexts. Regardless of the application, all context-aware mobile applications share the need to retrieve and process context information. This paper presents a Context Management tool for the Android platform (ACM). ACM allows easy access to internal on-board mobile sensors and hardware features extracting corresponding raw data. Raw context is processed into higher-level more human-readable context that is provided seamlessly to the mobile applications. Different methods are used for this purpose including fuzzy classifiers. Since different mobiles have different sensors and hardware features, ACM can adapt to the mobile device by deactivating access to unavailable ones. Information regarding the available sensors and hardware features and their specifications can also be queried. Additionally, applications can request notifications regarding context change or specific context values. In addition to providing developers with supporting classes and methods, ACM is accompanied by an application that allows developers to examine its functionality and capabilities before using it. The application can be also used to examine the readings of the different sensors in different situations and thus calibrate them as needed. Additionally, it can be used to modify and personalize default interpretations of raw context values to high-level ones. ACM has been tested empirically and the results show extreme interest of context-aware mobile application developers in its promising capabilities and that it is conducive to facilitating, speeding up and triggering development of many more of such applications.

Keywords—Android programming, context awareness, fuzzy classification, mobile applications

1 Introduction

Smartphones have now become an essential part of our daily life. Thus, numerous applications have been and are being developed for smartphones. Accessing internal on-board mobile sensors and other hardware features is an essential task in most applications. This is especially true in case of context-aware mobile applications whose behavior depend on the context obtained via such sensors and hardware features [1]. The problem is that this hard and time-consuming task has to be repeated with almost every context-aware mobile application. In case of the Android platform, for example relatively long pieces of code have to be written for this purpose using Android SDK. Accordingly, many developers limit the number of manipulated sensors and mobile features merely due to the time constraints [2]. What makes the problem worse is that many classes and methods are deprecated over the years [3] entailing learning to use alternative ones. This paper presents an Android Context Management (ACM) tool that provides a set of classes and methods that facilitate access to raw context values to save developers such trouble so that they can concentrate on the main functionalities of their applications. It is worth noting that we considered the Android operating system in particular due to its popularity in comparison to other smartphone operating systems and because Android itself is Java based and open source.

Since such applications, typically require higher-level context values, raw data extracted from internal mobile sensors and hardware features are processed to be presented in a more-readable form using several methods including fuzzy classifiers. For example, *time* can be presented in terms of the current *hour, day of the week,* or *season*. On the other hand, a specific temperature can be interpreted probabilistically as 40% cold and 50% chilly. Since different mobiles have different sensors and hardware features, ACM can adapt to the mobile device by deactivating access to unavailable ones. Accordingly, information regarding the available ones and their specifications can be queried. Additionally, applications can request notifications regarding specific context values or specific context changes.

ACM is accompanied by a mobile application that allows developers to examine its functionality and capabilities at various levels of granularity before using it. Mobile users can also use the application to retrieve information about and from their mobile sensors and hardware features. The application can be also used to personalize default interpretations of raw context values to high-level ones as needed and to calibrate sensors. The contributions of this research study can be summarized as follows:

- Developing the ACM tool that processes raw context values extracted from internal mobile sensors and hardware features into higher-level ones using various methods including fuzzy classifiers.
- Providing developers with classes and methods that allow fast and easy access to such context values in addition to information about available sensors and features.
- Developing an application that allows developers to examine the functionalities of ACM before using it, examine sensors behavior, calibrate them as needed and personalize the default interpretations of raw context values to high-level ones.

The rest of the paper is organized as follows: Section 2 presents related research in the literature to emphasize the above contributions. Sections 3 and 4 provide background about Android mobile sensors and hardware features respectively. The details of ACM are provided in Section 5. Section 6 presents the results of the empirical evaluation of ACM. Finally, the conclusion and future work are provided in Section 7.

2 Related work

Numerous frameworks have been suggested in the literature for managing context for context-aware mobile applications. For example, the Open Data Kit framework [4, 5] has been developed for the Android platform to allow applications to easily access external mobile sensors connected via Bluetooth or USB. Similarly, the BraceForce middleware [6] has been proposed for helping novice programmers in accessing external mobile sensors data with minimal coding. It also supports model-driven data acquisition in order to reduce expensive communication with external sensors. Nevertheless, those systems are mainly concerned with facilitating access to external mobile sensors and do not take into consideration mobile hardware features.

Relatively few research studies have been concerned with managing context extracted from internal on-board mobile sensors and hardware features. For example, SemaDroid [7] is a management framework intended for monitoring usage of sensors by installed applications and providing security against illegal access to sensitive sensor data. Many other systems have been mainly concerned with processing raw context into higher-level more-human readable context. For example, Yusheng et al. [8] proposed the ComSensor method to compose data of several sensors for context with more semantics. Korpipää and Mäntyjärvi [9] developed an ontology of the constituents of context derived from mobile sensors such that context instances can be easily derived facilitating the development of context-aware mobile applications. The ContextTorrent [10] is an Android-based framework that processes raw context into semantically searchable context by modeling context as an ontology. Fuzzy context information has been derived in case of uncertainty [11]. Naive Bayesian networks have also been used for context classification [12]. Kramer et al. [13] proposed an engine for context acquisition, composition and broadcasting to be shared by Android context-aware mobile applications. CAMF [14] is an Android-based framework that suggests processing raw context into high-level context using machine learning techniques, but it is merely a proposal. It is clear that those systems have limited capabilities and that each one is concerned with one aspect of context management.

A more advanced management framework has been proposed by Korpipää et al. [15]. In this framework, raw data extracted from internal mobile sensors can be processed into higher-level context using fuzzy classifiers, an ontology and a naive Bayes classifier. It also allows subscription for context notifications. Unfortunately, the framework has been developed for the obsolete Symbian platform. Additionally, it did not take into consideration accessing sensor information and accuracy or deactivating access to absent sensors. Besides, hardware features were not addressed. Most of

those disadvantages are probably because it was developed about fourteen years ago when mobiles and their operating systems were relatively less developed than today.

3 Android mobile sensors

Android mobile sensors [3] are intended to capture the context of the mobile and its environment and are broadly classified into three categories: motion sensors, position sensors and environmental sensors (in addition to special-purpose sensors such as heart rate sensors). Sensors may be either hardware-based or software-based. Software-based sensors are virtual sensors that derive their data from hardware sensors. This section provides background about those sensors in addition to a discussion about their accuracy. The section also discusses Android SDK support to access those sensors data.

It is worth noting that the sensors utilize a frame of reference, which is defined in relation to the screen of the mobile when it is positioned in its default orientation. In this coordinate system, the x-axis is horizontal and is directed to the right (of the mobile), the y-axis is vertical and is directed upwards and the z-axis is horizontal and is directed outwards from the screen face. In the world's frame of reference, on the other hand, the X-axis is directed towards the east, the Y-axis is directed towards the north and the Z-axis is perpendicular to the ground directed towards the sky.

3.1 Motion sensors

The motion sensors are responsible for detecting the motion of the mobile with respect to time and space. Those sensors include:

- The *accelerometer* or the *linear acceleration* sensor, which measure the acceleration force (including gravity) applied on the mobile along each of the three axes in m/sec^2.
- The *gravity* sensor, which measures the force of gravity applied on the mobile along each of the three axes in m/sec^2. It is one of the most useful sensors for motion detection.
- The *gyroscope*, which measures the rate of rotation of the mobile around each of the three axes in *rad/sec*.
- The *rotation vector* sensor, which measures the rotation vector component of the mobile along each of the three axes. This unitless sensor is one of the most useful sensors for motion detection (in addition to the gravity sensor).
- The *step counter* returns the number of steps taken by the mobile user since the last mobile reboot and activation of the sensor.

3.2 Position sensors

The position sensors are intended to determine the physical position of the mobile. Those sensors include:

- The *magnetic field* or *geomagnetic field* sensor, which measures the ambient geomagnetic field strength along each of the three axes in *microtesla* (μT). It can be used together with the accelerometer to determine the position of the mobile in the world's frame of reference such as the mobile position relative to the North Pole or the orientation of the mobile in the application's frame of reference.
- The *proximity* sensor, which measures how close an object is relative to the front screen (face) of the mobile in *cm*.

3.3 Environmental sensors

The environmental sensors are intended to detect environmental conditions surrounding the mobile. Those sensors include:

- The *ambient temperature* sensor measures the ambient temperature in ^{o}C.
- The *light* sensor measures the ambient illumination level in *lux* (*lx*).
- The *pressure* sensor measures the ambient pressure in *millibars* (*mbar*) or *hectopascals* (*hPa*).
- The *relative humidity* sensor measures the relative surrounding humidity in *percentage*.

3.4 Sensors accuracy

Sensors accuracy refers to how much the value returned by the sensor is close to the real value and how much it is reliable. One of the following integer numbers represents the accuracy of a sensor:

- A value of -1 indicates a *no contact* value that cannot be trusted because the sensor was not in contact with what it has been measuring.
- A value of 0 indicates an *unreliable* value that cannot be trusted because, for example, the sensor is not calibrated.
- A value of 1 indicates *low*-accuracy.
- A value of 2 indicates *medium*-accuracy.
- A value of 3 indicates *high*-accuracy.

3.5 Android SDK support

Android sensor framework within Android SDK allows easy access to the mobile sensors to obtain different types of information:

- The *SensorManager* class that allows getting a list of all available sensors or sensors of a given type. It provides various methods for listening to sensors.
- The *Sensor* class that allows obtaining full information about the capabilities of a given sensor such as its name, its type and version, its vendor, its resolution and power in addition to its maximum range and minimum delay.

- The *SensorEvent* class that allows obtaining information about a sensor event such as a new value in addition to its timestamp and accuracy.
- The *SensorEventListener* interface that provides the *onAccuracyChanged* and the *onSensorChanged* methods that are called when the accuracy and reading (even with a similar value) of a sensor change respectively.

It is worth noting that extracting high-level context values from raw level context values using Android SDK requires writing relatively long pieces of code.

4 Android mobile hardware features

Android mobile hardware features are the data that can be extracted from the mobile hardware components (other than the sensors) using corresponding Android SDK classes and methods [3, 16]. They include:

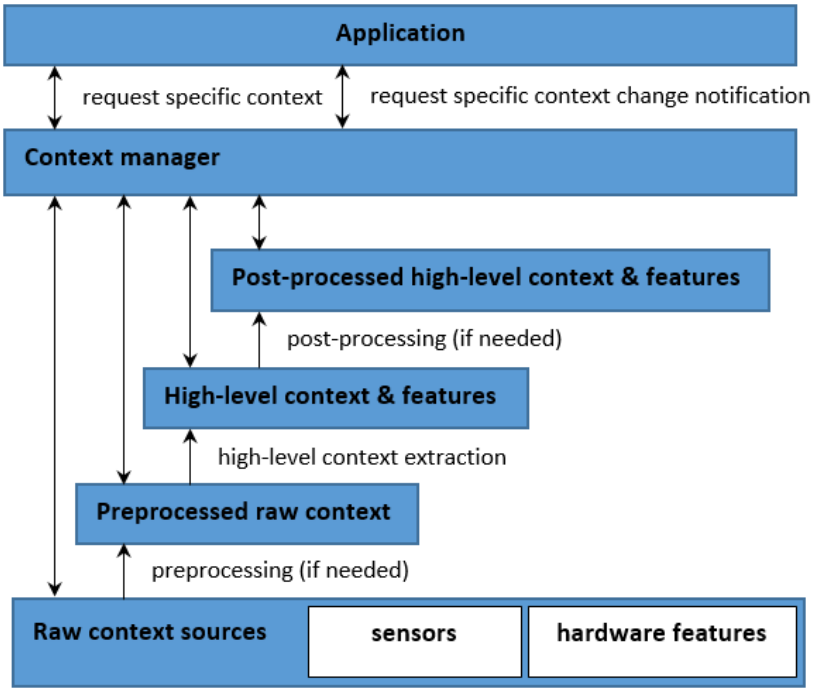

Fig. 1. Block diagram of ACM

- *Bluetooth*: applications can use this mid-range wireless technology for communication within a range of 10 meters at a data transfer rate of about 2.1 *Mbps*.
- *Camera*: applications can control the camera parameters and capture images with embedded Global Positioning System (GPS) coordinates.
- *Device battery*: applications can monitor the battery level and charging state (whether it is charging, discharging or full for example).

- *Device time*: applications can obtain current time in milliseconds since January 1, 1970 and convert it to a meaningful value.
- *Location*: applications can determine the mobile location using the GPS, for example.
- *Near Field Communication (NFC)*: applications can use this technology to sense electronically enabled objects within a certain range in proximity to the mobile and read data from these objects.
- *Wi-Fi*: applications can use Wi-Fi connectivity for communication.

It is worth noting that Android SDK provides classes and interfaces to listen to and access hardware features. Nevertheless, as with mobile sensors, this requires writing relatively long pieces of code.

5 ACM tool

Figure 1 shows a block diagram of ACM. As shown in the figure, ACM has a multi-layered architecture. At the lowest layer, raw context values are extracted from the mobile sensors and hardware features using Android SDK classes and methods. Those raw values may be pre-processed as required in order to be converted into higher-level more human readable values. Such high-level context values may be post-processed into other forms. Context values generated from any of the four layers of ACM can be accessed by the context manager to be provided to context-aware mobile applications upon request. This is explained in more details in the following subsections.

Table 1. Objective default interpretations of different types of sensors data

Sensor	Reading	Interpretation
Gravity sensor (x, y & z in m/sec^2)	$z \geq 9.8/2$ $z \leq -9.8/2$	mobile screen up mobile screen down
Proximity sensor (p *cm*)	$p = 0$ $p \neq 0$	near (object in proximity) far (no object in proximity)
Pressure sensor (p *mbar*)	$p > 1013$ $p = 1013$ $p < 1013$	High pressure Normal pressure Low pressure
Heart rate sensor (b *beats/min*)	$b < 60$ $60 \leq b \leq 100$ $b > 100$	low (for adult) normal (for adult) high (for adult)

5.1 Extracting high-level context values

ACM processes raw context values in order to generate high-level more human-readable and more usable context values. Interpretations of raw context values into high-level context values may be objective or subjective. Examples of objective interpretations of different types of sensors data are provided in Table 1. The table provides an example of motion sensors, which is the gravity sensor. As shown in the

table, we can use the gravity sensor to figure out whether the mobile is face up or down. If the mobile were face up, for example the reported gravity along the z-axis is expected to be equal to the standard gravity (≈ 9.8 m/sec^2). Nevertheless, due to noise a threshold of half this value is typically used [16]. This allows detecting whether the mobile is face up without having to be perfectly parallel to the ground. Using ACM, the programmer can question whether the mobile is face up or down without having to go through all this trouble and can query alignment directly.

Table 2. Subjective default interpretations of environmental sensors data

Sensor	Reading	Interpretation
Ambient temperature ($t\ ^oC$)	$t \leq 0$ $0 < t < 5$ $5 \leq t < 10$ $10 \leq t < 15$ $15 \leq t < 20$ $20 \leq t < 25$ $25 \leq t < 30$ $30 \leq t < 35$ $35 \leq t < 40$ $t \geq 40$	Freezing Cold Chilly Cool Mild Warm Very warm Hot Very hot Extremely hot
Light sensor (l *lux*) – based on Android SDK	$l < 0.001$ $0.001 \leq l < 0.25$ $0.25 \leq l < 100$ $100 \leq l < 400$ $400 \leq l < 10{,}000$ $10{,}000 \leq l < 20{,}000$ $20{,}000 \leq l < 110{,}000$ $110{,}000 \leq l < 120{,}000$ $l \geq 120{,}000$	Extremely dark No moon (dark) Full moon Cloudy Sunrise Overcast Indirect sunlight (shade) Sunlight Maximum sunlight
Light sensor (l *lux*) - based on Microsoft interpretations	$0 < l \leq 10$ $10 < l \leq 50$ $50 < l \leq 200$ $200 < l \leq 400$ $400 < l \leq 1{,}000$ $1{,}000 < l \leq 5{,}000$ $5{,}000 < l \leq 10{,}000$ $10{,}000 < l \leq 30{,}000$ $30{,}000 < l \leq 100{,}000$ $l > 100{,}000$	Extremely dark Very dark Dark inside Dim inside Normal inside Bright inside Dim outside Cloudy outside Under sunlight Maximum sunlight
Relative humidity sensor (h %)	$h < 30$ $30 \leq h < 40$ $40 \leq h < 50$ $50 \leq h < 60$ $60 \leq h < 70$ $h \geq 70$	Uncomfortable dry Dry Comfortable wet Uncomfortable wet Extremely uncomfortable wet

The table also provides an example of position sensors, which is the proximity sensor. Some proximity sensors provide a zero value when an object is close to the mobile and so this has to be interpreted as shown in the table. On the other hand, some other binary sensors provide *near* and *far* discrete values. Those differences are han-

dled in ACM and programmers can easily question proximity of objects to the mobile without having to deal explicitly with such differences.

An example of environmental sensors is the pressure sensor. Since average sea-level pressure is about 1013 *mbar*, values higher than this level are interpreted as high pressure and those lower than this level are interpreted as low pressure as shown in the table. The pressure sensor reading is used to compute relative altitudes between floors in a building for example. To account for drifts in pressure, typically altitude is obtained relative to pressure obtained from a local reporting station.

The heart rate sensor is an example of a special purpose sensor. A value of the heart rate sensor between 60 and 100 beats/min inclusive indicate normal heart rate for a typical resting adult as shown in the figure. Of course, there are exceptions to this rule. This problem can be handled using ACM as explained in the following sections.

Examples of subjective interpretations of some environmental sensors data are provided in Table 2. The temperature between 0^{o}C (freezing) and 40^{o}C (very hot) are subjectively divided into eight ranges. Of course, those ranges can be altered and the temperatures below 0^{o}C and above 40^{o}C can be similarly divided into ranges. This is also handled in ACM as explained below.

Light sensors provide light intensity in *lux*. Interpretations of some discrete light values are specified in Android SDK by a set of constants such as LIGHT_NO_MOON, which specifies 0.001 *lux* as light intensity when there is no moon and LIGHT_SUNLIGHT, which specifies 110,000 *lux* as light intensity in sunlight [3]. Based on those interpretations, we can divide light intensities into ranges as shown in the table. Alternative interpretations based on Microsoft interpretations [17] for creating light-aware user interfaces are also provided in the table.

Values of relative humidity are also divided into ranges between 30 % and 70 % as shown in the table. The 30% to 60% range has been frequently reported as being most comfortable [18]. According to the engineering toolbox [19], human comfort typically lies between 25% to 60%. Humidity above 70% causes corrosion and mold and is extremely uncomfortable and wet.

5.2 High-level context values post-processing

High-level context values may be post-processed into other forms. For example, raw temperature context is normally processed using crisp membership functions such as those shown in Figure 2 to identify the ranges shown in Table 2. Such membership functions can be replaced by fuzzy membership functions such as those shown in Figure 3. In other words, crisp quantization may be replaced by fuzzy quantization so that interpretation of raw context into high-level context is probabilistic. For example, a temperature value of 5^{o}C can be interpreted as 50% cold and 50% chilly. ACM allows programmers to replace the default crisp quantization by fuzzy quantization. Nevertheless, programmers should be aware of the essence of fuzzy membership functions in the first place in order to be able to benefit from this functionality.

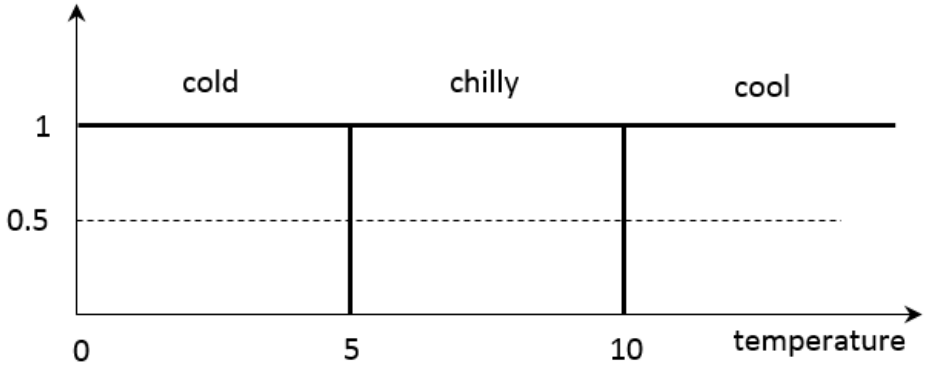

Fig. 2. Crisp quantization (default membership functions)

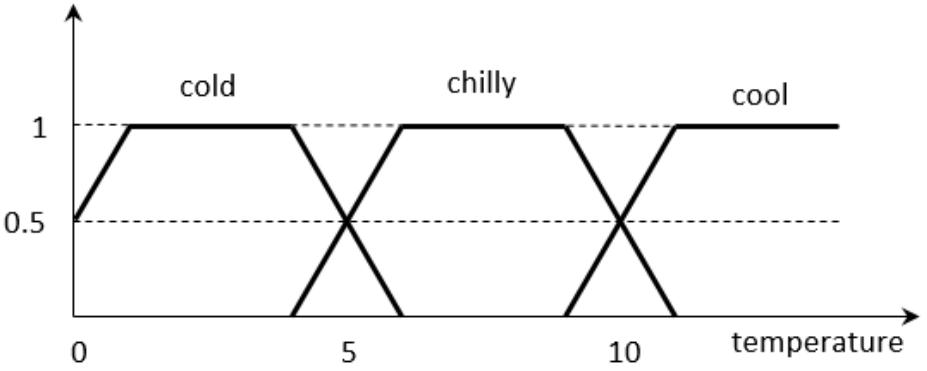

Fig. 3. Fuzzy quantization (alternate membership functions)

5.3 Applications support

For applications to benefit from ACM, they may directly access or request notifications regarding specific context values or when some of those values change. ACM provides classes and methods for application programmers for this purpose. It is worth noting that although raw and some high-level context values can be accessed directly using Android SDK, it is not unusual that constants and methods in Android SDK are deprecated and replaced by others. Additionally, as previously noted, extracting high-level context values from raw level context values using Android SDK requires writing relatively long pieces of code. So accessing context values through ACM would save developers such trouble. In other words, ACM provides raw and several forms of high-level context values conveniently to context-aware mobile application developers so that they can concentrate on the logic of their applications and save the time wasted in context processing with every new application.

Examples of some of the methods provided by ACM are shown in Table 3. Those are examples of a comprehensive set of methods provided by ACM. The first set of methods are responsible for directly accessing sensors information and data. Each

Android mobile sensor has an Id, type and name. The *accessSensor*, *getSensorId* and *getSensorName* methods can provide type (and Id), Id and name of a given sensor respectively. In addition to providing this basic information, ACM also supports developers by providing a definition of each sensor using the *getSensorDefinition* method. The resolution, maximum range and unit of each sensor can also be obtained using the *getSensorResolution*, *getSensorMaximumRange* and *getSensorUnit* methods respectively. Information about the *GP2A proximity sensor* are shown in Figure 4.

Table 3. Example ACM sensors and hardware features methods

Method	Functionality
accessSensor getSensorId getSensorName sensorSensorDefinition getSensorResolution getSensorMaximumRange getSensorUnit getSensorRawData getSensorReadableForm getSensorAccuracy getSensorTimeStamp isSensorExist(Id) isSensorExist(name)	Get sensor *Id* and *type* Get sensor *Id* as an integer Get sensor *name* as a string Get sensor *definition* as a string Get sensor *resolution* (and unit) Get sensor *maximum range* (and unit) Get sensor *unit* Get sensor *raw data* Get sensor *readable form* Get sensor *accuracy* during last event as an integer Get sensor *timestamp* when last event occurred as a long number Return *true* if sensor (accessed by its Id) exists and *false* otherwise Return *true* if sensor (accessed by its name) exists and *false* otherwise
getPlugType getHealth getStatus	Get battery *plug type* Get battery *charging health* Get battery *status*
getCity getCountryCode getZip getCountry	Get the *city name* of the current location Get the *country code* of the current location Get *Zip code* of the current location Get *country name* of the current location
getSeason getYear getMonth getDay getHour	Get the *current season* Get the *current year* Get the *current month* Get the *current day* Get the *current hour*

The most important methods, though are the *getSensorRawData*, *getSensorReadableForm*, *getSensorAccuracy* and *getSensorTimeStamp*. Those methods help developers extract context of a given sensor in a raw form or in a high-level readable form in addition to the timestamp at which this context was extracted and the accuracy of the extracted context. The *isSensorExist* is used to check the existence of a given sensor using its Id or name.

It is worth noting that hardware features are also supported by a number of similar methods. For example, the *accessFeature* method can provide type (and Id) of a given feature. This method is equivalent to the *accessSensor* method of the mobile sensors.

Sensor and hardware feature-specific methods are also provided to obtain additional information or specific high-level context values. For example, the *getPlugType*, *getHealth* and *getStatus* methods are used to obtain the plug type (such as USB or

wireless), health (such as good or overheat) and status of the device battery (such charging, discharging or full) respectively. Another example, instead of using the *getFromLocation* method from Android SDK to convert a geographic location into detailed address information, ACM allows convenient access to the components of the current address, which are the city, country code, Zip code and country name seamlessly using the *getCity*, *getCountryCode*, *GetZip* and *getCountry* methods respectively. Similarly instead of using the *GregorianCalendar* class from Android SDK for detailed information about current time, ACM provides, for example the *getSeason*, *getYear*, *getMonth*, *getDay*, *getHour* methods to obtain the current season, year, month, day and hour respectively.

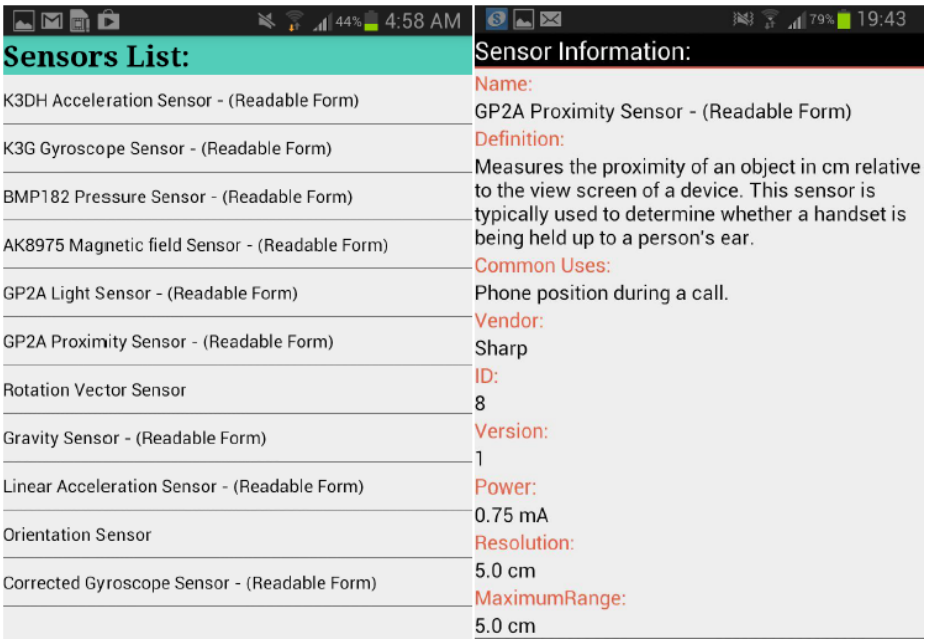

Fig. 4. ACM application snapshots

5.4 ACM mobile application

ACM also provides a mobile application that can be used by developers to visually test the functionality and capabilities of ACM before adopting it. Figure 4 shows example snapshots of ACM application. The snapshot on the left hand side depicts a list of all available sensors on a *Samsung Galaxy Note 1* mobile while that on the right hand side depicts the details of a selected proximity sensor. The term *Readable Form* appears next to sensors with high-level values.

The application can be also used to experiment with sensors. For example, the orientation of the mobile can be changed while visually examining the readings of the different motion sensors to quickly and conveniently examine how their readings change with the change of the orientation.

The application is also useful in calibrating sensors. For example, the linear acceleration sensor typically has an offset. The developer can place the device on a table for example and request calibration. The application would then record the offset along each of the three axes to be subtracted from the corresponding readings of the linear acceleration sensor to get the correct values.

Moreover, since default interpretations of raw context values (into high-level context values) have exceptions, applications should be capable of effecting modifications as needed. For example, the normal heart rate of a resting trained athlete is usually close to 40 beats/min [20]. Accordingly, the default values shown in Table 1 (ranging from 60 to 100 beats/min inclusive) should be overwritten when the application is used by such an athlete. The application allows adjusting such default values. In the above example, the default normal values of the heart rate sensor can be adjusted from the 60-to-100 range to the 35-to-45 range.

Table 4. Results of the satisfaction questionnaire

Evaluation indicator	Average
Efficient and easy to use	5
Does not have a long learning curve	4.7
Conducive to increasing the number of developed context-aware mobile applications	4.1
Promising to speed up development	5
Would like to use it in developing applications in the future	5

6 Evaluation

In this section, we present the results of the empirical evaluation of ACM. ACM has been tested by fifteen Android developers, who were given the chance to use it for a month in their applications. Afterwards, we prepared a questionnaire targeting evaluating their reflection about ACM based on their experience. They were asked to respond to the questionnaire based on a Likert scale in which 1 represents extreme dissatisfaction and 5, on the other hand represents extreme satisfaction. The results are provided in Table 4. As shown in the table, the developers believe that ACM is efficient and easy to use, does not have a long learning curve and thus they would readily use it in developing context-aware mobile applications in the future. Additionally, they believe that such a tool would trigger and speed up development of more such applications. The reliability and consistency of the questionnaire results were estimated using *Cronbach's alpha*. We obtained a value of α equal to 0.87 indicating a very high degree of reliability and acceptance of the questionnaire results.

7 Conclusion

This paper presented ACM, an Android context management tool. ACM provides support to context-aware mobile application developers and to mobile users. Application developers are provided with a set of classes and methods allowing easy access to raw and high-level context values and information about sensors and hardware features. High-level context values are extracted using several methods including fuzzy classifiers. Applications can request notifications regarding specific context values or specific context changes. ACM is accompanied by a mobile application allowing developers to test its functionalities and allowing mobile users to get information about and access sensors and mobile hardware features extracting their raw data in addition to corresponding higher-level context values. Since different mobiles have different features and sensors, the tool can adapt to the mobile device by deactivating access to unavailable sensors and hardware features. ACM also allows personalizing interpretations of raw context values to high-level ones and facilitates sensors calibration via the mobile application.

ACM has been tested empirically and the results show extreme satisfaction of Android application developers with its capabilities. As future work, we will consider additional methods for extracting more and other forms of high-level context values. For example, we can compose high-level context from multiple sources and develop ontologies [21] and naïve Bayes classifiers. We also intend to extend ACM to accept context values from external sources and sensors.

8 References

[1] Elazhary, H. (2015). A cloud-based framework for context-aware intelligent mobile user interfaces in healthcare applications. Journal of Medical Imaging and Health Informatics, 5(8):1680-1687. https://doi.org/10.1166/jmihi.2015.1620

[2] Alnanih, R., Ormandjieva, O. & Radhakrishnan, T. (2013). Context-based and rule-based adaptation of mobile user interfaces in mHealth. 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare, Niagara Falls, Ontario, Canada, pp. 390-397. https://doi.org/10.1016/j.procs.2013.09.051

[3] Android O Developer Preview, https://developer.android.com/index.html [Online; accessed: 2017-03-01].

[4] Brunette, W., Sodt, R., Chaudhri, R., Goel, M., Falcone, M., VanOrden, J. & Borriello, G. (2012). Open Data Kit sensors: A sensor integration framework for Android at the application-level. 10th International Conference on Mobile Systems, Applications and Services, Low Wood Bay, Lake District, UK, pp. 351-364. https://doi.org/10.1145/2307636.2307669

[5] Chaudhri, R., Brunette, W., Goel, M., Sodt, R., VanOrden, J., Falcone, M. & Borriello, G. (2012). Open Data Kit sensors: Mobile data collection with wired and wireless sensors. 2nd ACM Symposium on Computing for Development, Atlanta, GA, USA. https://doi.org/10.1145/2160601.2160614

[6] Zheng, X., Perry, D. & Julien, C. (2014). BraceForce: A middleware to enable sensing integration in mobile applications for novice programmers. 1st International Conference on

Mobile Software Engineering and Systems, Hyderabad, India, pp. 8-17. https://doi.org/10.1145/2593902.2593907

[7] Xu, Z. & Zhu, S. (2015). SemaDroid: A privacy-aware sensor management framework for smartphones. 5th ACM Conference on Data and Application Security and Privacy, San Antonio, Texas, USA, pp. 61-72. https://doi.org/10.1145/2699026.2699114

[8] Yusheng, X., Zhixin, M., Xiaoyun, C. & Lian, L. (2008). A composite sensor-based context modeling method for context-aware pervasive computing. International MultiConference of Engineers and Computer Scientists, Hong Kong, pp. 1163-1168.

[9] Korpipää, P. & Mäntyjärvi, J. (2003). An ontology for mobile device sensor-based context awareness. 4th International and Interdisciplinary Conference on Modeling and Using Context, Stanford, California, USA, pp. 451-458. https://doi.org/10.1007/3-540-44958-2_37

[10] Hu, D., Dong, F. & Wang, C. (2009). A semantic context management framework on mobile device. International Conference on Embedded Software and Systems, Hangzhou, Zhejiang, China, pp. 331-338.

[11] Mäntyjärvi, J. & Seppänen, T. (2002). Adapting applications in mobile terminals using fuzzy context information. 4th International Symposium on Human Computer Interaction with Mobile Devices, Pisa, Italy, pp. 95-107. https://doi.org/10.1007/3-540-45756-9_9

[12] Korpipää, P., Koskinen, M., Peltola, J., Mäkelä, S. & Seppänen, T. (2003). Bayesian approach to sensor-based context awareness. Personal & Ubiquitous Computing, 7:113-124. https://doi.org/10.1007/s00779-003-0237-8

[13] Kramer, D., Kocurova, A., Oussena, S., Clark, T. & Komisarczuk, P. (2011). An extensible, self contained, layered approach to context acquisition. 3rd International Workshop on Middleware for Pervasive Mobile and Embedded Computing, Lisbon, Portugal. https://doi.org/10.1145/2090316.2090322

[14] Wang, A. & Ahmad, Q. (2010). CAMF - Context-aware machine learning framework for Android. Software Engineering and Applications Conference, Marina del Rey, California, USA, pp. 388-395. https://doi.org/10.2316/p.2010.725-003

[15] Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H. & Malm, E. (2003). Managing context information in mobile devices. Pervasive Computing, 2(3):42-51. https://doi.org/10.1109/MPRV.2003.1228526

[16] Milette, G. & Stroud, A. (2012). Professional Android Sensor Programming. John Wiley & Sons, Inc., Indianapolis, Indiana.

[17] Understanding and interpreting lux values, https://msdn.microsoft.com/en-us/library/windows/desktop/dd319008(v=vs.85).aspx [Online; accessed: 2017-03-01].

[18] Indoor humidity levels, http://www.sensitivechoice.com/indoor-humidity/ [Online; accessed: 2017-03-01].

[19] Recommended relative humidity levels, http://www.engineeringtoolbox.com/relative-humidity-d_895.html [Online; accessed: 2017-03-01].

[20] What's a normal resting heart rate? http://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979 [Online; accessed: 2017-03-01].

[21] Elazhary, H. (2016). CAL: A controlled Arabic language for authoring ontologies. Arabian Journal for Science and Engineering, 41(8):2911–2926. https://doi.org/10.1007/s13369-015-2016-z

9 Authors

Hanan Elazhary earned her B.Sc. and M.Sc. degrees from the Department of Electronics and Communications Engineering, Cairo University. She earned her Ph.D. degree in Computer Science and Engineering from the University of Connecticut, USA. Currently, she is an associate professor in the Computer Science Department, Faculty of Computing & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia and the Computers and Systems Department, Electronics Research Institute, Cairo, Egypt. Her research interests include distributed systems, software engineering and intelligent tutoring systems.

Alaa Althubyani, Lina Ahmed, Bayan Alharbi, Norah Alzahrani and Reem Almutairi were undergraduate students in the Computer Science Department, Faculty of Computing & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia at the time that this work was conducted.

Article submitted 27 March 2017. Published as resubmitted by the authors 13 May 2017.