Easy Implementation of Sensing Systems for Smart Living

Paolo Bellagente, Claudio Crema, Alessandro Depari, Alessandra Flammini,
Giovanni Lenzi, Stefano Rinaldi, Angelo Vezzoli
Department of Information Engineering, University of Brescia, Via Branze 38, 25123 Brescia, Italy.
e-mail: alessandro.depari@unibs.it

Abstract—**Nowadays system architectures based on Internet of Things (IoT) are becoming a pervasive topic in several applications. Smartphones integrate a growing number of high quality sensors and often they are used as gateways for external sensors, wearable sensors or sensors for ambient assisted living. Several recent projects about smart cities and smart living are experiencing sensory data management by means of smartphones, resulting in large data flow and battery power concerns. In these architectures, using sensors embedded within the smartphone is trivial, and several reliable Applications (APPs) are available; on the contrary, APPs managing external sensors have high Source Line Of Code (SLOC), especially if sensors of different vendors are considered and if the APPs need a high level of personalization. In this work, a solution to easily develop Android APPs handling remote sensors, even of different type and manufacturers, is employed in the context of a smart living application. The proposed approach is based on a framework that allows achieving an easy development system architecture, exploiting the Bluetooth 4.0-4.1 technologies. First implementations focused on safety in a Smart City projects are discussed.**

Keywords—Sensing Systems; System Architecture; Easy Development; Hardware/Software Integration; IoT; Android

I. Introduction

Despite the lack of a unique definition, Internet of Things (IoT) is usually addressed as "the combination of distributed information processing, pervasive wireless networking and automatic identification" [1]. In other terms, IoT means introducing the possibility to interact directly with a physical entity to modify it or to retrieve information about it, on-demand and independently from the location. To achieve this scenario, sensors, actuators and elaboration nodes could be deployed and connected into three ways as reported in Fig. 1:

- Direct Wireless Area Network (WAN) from sensor to the Internet.
- Nodes connected through an ad-hoc gateway that provides the WAN connection (usually on a 3G/WiFi link).
- Nodes connected through general-purpose platforms, already deployed for different use, acting as gateways for the considered application. A typical example could be a modern Smartphone acting as gateway between its own embedded sensors or an external, complex, sensory system and the Cloud.

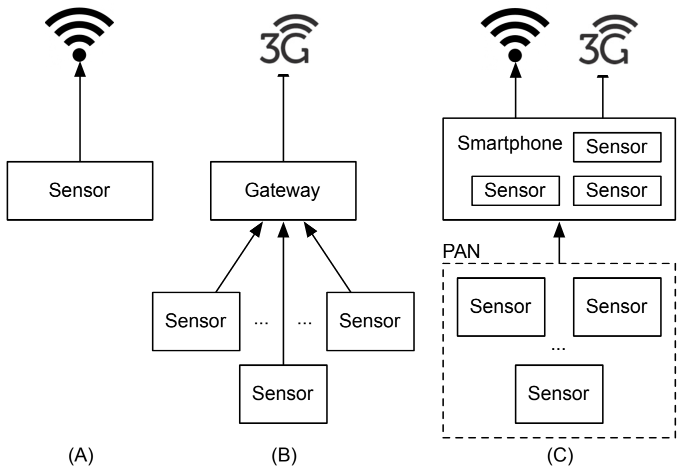

Fig. 1. Examples of architectures. PAN is the acronym for Personal Area Network.

Nowadays, smartphones are pervasive in our everyday lives; for instance, in the European Union the average number of such devices for each person was 1.25 in 2014 [2], [3]. Therefore, it is likely that the last of the aforementioned architectures would be the most widespread in smart living systems, whereas the others would probably be dominant in other fields (e.g., automotive and related infrastructures [4]).

Smartphones are naturally equipped with several embedded sensors and this number is constantly growing. Exploiting such sensors into an Application (APP) is rather trivial. Conversely, the use of external sensors/embedded systems is definitely more complex, and it requires a remarkable programming effort, especially with a high number and wide typology of sensors.

Indeed, this is the case of the "Brescia Smart Living" project, a smart city project funded by the Research and University Italian Ministry and involving the University of Brescia and several other local institutional subjects and companies. The aim of this project is to realize an infrastructure of services to increase the quality of life of fragile subjects (e.g., elderly), citizens' safety, security, and to achieve a general saving of resources. This is fulfilled by means of ambient assisted living, teleservices, and inexpensive helpline systems. To offer such services, a large number of

This work has been partially supported by Smart Cities and Communities and Social Innovation through the "Brescia Smart Living" under Grant MIUR SCN_00416, and in part by the University of Brescia Health&Wealth Grant through "Work, Wealth, Production, Productivity".

sensors is employed. Sensors are often redundant, to ensure continuity of service and reliability even in case of failure/battery exhausting. Users are equipped with a smartphone where a personalized APP interacts with the infrastructure and the sensors to provide personalized services. Due to the large number of deployed sensors, involved participants (about 3.000), and APP personalization, the use of traditional approaches for APP development could result in complex source code, thus not assuring proper efficiency and reliability.

Similar problems arise in "Work, Wealth, Production, Productivity", a multidisciplinary project financed by the University of Brescia and involving several research groups of the same University. One of the goal of the project is the development of a smartphone-based system to assist users and emergency staff during an emergency evacuation of workplace. The system architecture is based on indoor localization obtained with the deployment of a large number of (potentially different) beacon devices. The user APP should be easily configured to cope with modifications of the number and typology of employed beacons.

Smartphones are usually equipped with Operating Systems (OS); iOS and Android are the most popular in Europe [5]. Due to the open source nature of Android, the development of APPs for this OS is easier and can take advantage of a large online user community; for this reason, in this paper the attention is focused on Android devices. One of the advantages of using an OS is the hardware abstraction it provides for embedded sensors; actual data are read by the OS's driver, and sensors from different manufacturers are all accessed the same way by APPs. For this reason, it is quite straightforward to get data and information about measurements by means of functions named callbacks, provided in Android by the native Software Development Kit (SDK).

External sensors and embedded systems are usually connected to smartphones by wireless links [6], e.g., Bluetooth (BT). Android Things (AT) [7] is an OS for external devices that facilitates data communication with APPs running on Android Smartphones. Currently, AT is available only for few complex external devices (e.g., Raspberry Pi). In general, the integration of external devices could require the development of a purposely-designed firmware, in order to achieve the desired data exchange with the (high level and complex) Android APP. To facilitate this task, some device manufacturers provide BT Application Programming Interfaces (APIs). Others furnish evaluation APPs, which typically cannot be integrated with other device APPs. Thus, the development of APPs using only one external sensors is a relatively simple task. On the other hand, the complexity of the APP significantly increases with increasing number of external devices; APPs could be less reliable, because of the high Source Line Of Code (SLOC).

In this paper, a framework for the management of external devices and the support for easy realization of sensor management APPs with high level of personalization is employed in the context of the two aforementioned projects. This open source framework, called Sensors for Android Embedded (SAndroidE) [8], currently available on GitHub [9], is oriented to interface devices by means of low-power communication standards, such as BT Low Energy (BLE), since power consumption is typically one of the major constraints when dealing with wireless portable devices [10].

The paper is structured as follows: in Section II, the system architecture is described; Section III explains the realized framework, whereas in Section IV a case study, related to the aforementioned "Brescia Smart Living" project will be presented. Finally, conclusions are drawn.

II. System Architecture

A. Actual Scenario

Nowadays, the smartphone is a device widely used as general-purpose platform to achieve many different goals. For example, in [11] the high-resolution camera of the smartphone is used to implement an APP for skin self-examination and to improve the early diagnosis of Melanoma. In addition to the assistance in the self-examination, the APP allows the user to gather further information about the disease improving the user awareness as well as right behaviors. It is possible to detect the posture of the user [12], taking advantage of the smartphone's accelerometers and a suitable unsupervised (no a priori information needed) learning algorithm. Possibilities and performance could be improved if external hardware is connected through one of the many I/O channels of a smartphone. The market offers a large set of devices designed to add sensing capabilities to smartphones, as pressure, humidity, temperature and air quality sensors. Speaking about medical applications, examples are represented by sensors of blood pressure and oxygen saturation, heart rhythm and rate, body temperature, fatigue levels, respiration rate, blood and tissue glucose, walking distance, step count and many others, which can be monitored by means of dedicated devices [13].

Exploiting the BT channel for connecting an ECG sensor, for example, it is possible to use a smartphone as multi-lead ECG Monitoring System [14], [15], [16] as it can be turned into an electronic stethoscope, using the internal microphone, or linking an external one [17], [18]. In these projects, the smartphone provides the necessary computational power as well as the suitable communication channel to the Cloud. It is possible to combine multiple sensors to achieve more complex or accurate solutions, transforming the smartphone in an advance gateway which could completely avoid the human intervention, automatically delivering data to specific on-line services, as achieved for example in [19]. Besides digital health, many other application sectors are taking advantage of the smartphone platform as sensors gateway; an example of that are building automation systems [20] that can control various functions within a house, such as light control, heating, air conditioning, etc.

The implementation of any of the above solutions is not a trivial task, especially if more than one external device is used besides the smartphone. Fig. 2 shows the conceptual software blocks that a programmer has to handle to implement a two-sensor application in Android environment.

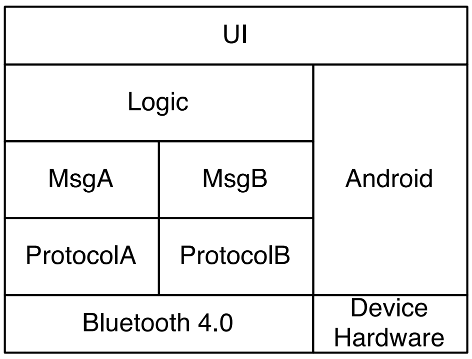

Fig. 2. Software conceptual blocks to be handled by a programmer of a two-sensors application in Android environment.

Starting from the bottom of the figure, as the Android OS can be installed on different hardware of different manufacturers, the programmer has to understand the hardware characteristics and how the physical layer of the communication link works. On the next level, he has to take care of the BT services (included in the BT protocol), needed to use the communication channel and to implement the transmission protocol as well as the messages structure and mechanics. After that, sensor-APP protocols, based on the underlying BT protocol, will be managed. Finally, the programmer has to cope with the Android OS for the development of a suitable graphic User Interface (UI). If more than one external sensor is used, the programmer should implement a suitable logic, to ensure the safe coexistence of all the hardware, as well as a user interface that could expose the new functionalities. This device-oriented structure, in multisensory scenarios, leads to APPs in which the sensors orchestration logic is moved towards the business logic (often on the Cloud). This generates heavy data flow that follows the behavior of the sensor (constant sampling, periodic forwarding or event forwarding) with transmission of duplicated data.

B. *SAndroidE-Powered Scenario*

The conceptual architecture of the same applications using the SAndroidE framework is shown in Fig. 3.

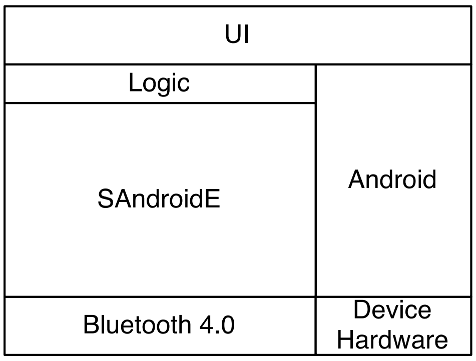

Fig. 3. Software conceptual blocks to be handled by a programmer of a two-sensors application in SAndroidE environment.

The scheme is much simpler, since the programmer does not have to deal with the device hardware (to understand its capabilities), and neither with the communication layer (to connect the external devices). He has to manage only the Android Environment (to write the logic of his APP). From the sensor management point of view, SAndroidE offers the huge enhancement to let the programmer deal only with the logic of the APP, exploiting a sensor interface that is equal to the standard one, provided by the Android Environment. Nothing more has to be learned as data structures, since callbacks and control logic have APIs in the same format of the native Android Sensor APIs.

To improve simplicity, in case of user programmable devices (described in Section III), SAndroidE natively provides an additional feature, allowing to easily set two configuration parameters for each remote sensor, in order to customize its behavior. It is possible to set:

- The **sampling interval**, which may be reduced or increased, depending on the type of sensor and overall system constraints. As an example, a temperature value is supposed to change very slowly, with respect to an accelerometer value. Therefore, it makes sense to read the temperature once every five minutes, or even once per hour, instead of once per second.
- A **threshold**, used to determine if the sensor value has changed. For example, in a temperature sensor, if the temperature value changes by only 0.05 °C, it probably makes no sense to transmit new data to the smartphone. Conversely, if the value changes of 1 °C, new data should be notified to the smartphone. The threshold is expressed as percentage of the full scale, as highlighted by the following pseudocode:

```
IsDataChanged=(CurrentVoltage-OldVoltage)/(VoltageMax-VoltageMin)>k
```

where *k* is the threshold (float value between 0 and 1).

These two parameters could be independently set by the developer. This allows the developer to reduce computational load both on the smartphone and on the device, and to reduce BT bandwidth as well, due to fewer transmissions. Obviously this also has an impact on reducing the overall consumption of Wi-Fi/3G bandwidth, if the Android APP has not an algorithm to discard data when they do not change. Therefore, data filtering is done directly by the remote device, reducing the total data amount, CPU cycles, as well as BT and wireless bandwidth. On the contrary, battery duration increases.

III. The Realized Framework

As already mentioned, SAndroidE allows data collection from external devices and sensors. In order to do that, it exploits a collection of component and features, shown in Fig. 4 and detailed hereinafter.

- An Android library written in Java, which needs to be included in any SAndroidE-powered Android APP. This library abstracts and virtualizes remote devices, regardless of their type and operating mode.
- Device Descriptor Files, written in eXtensible Markup language (XML) or JavaScript Object Notation (JSON), representing a description of the BT Services and

Characteristics provided by the devices, internally used by SAndroidE to know how to interact with each device.

- API Description Files, written in XML/JSON, representing the interface to connect and retrieve data from vendors Cloud servers, using HTTP requests.
- A SAndroidE Configuration Application (not shown in Fig. 4), used to detect nearby BT devices and give them a unique ID, used as reference to the device in the SAndroidE-powered app source code.

Fig. 4 clearly shows how SAndroidE completely abstracts communication and device-specific application protocols, giving access to remote resources, by means of simple Java callbacks.

Depending on the external device typology, SAndroidE can operate in four different modes, detailed hereinafter.

A. User programmable devices

These devices typically allow the developer (intended as the user) to write his own application firmware, with custom logic and algorithms, and to flash it on the device; for example Arduino, Raspberry Pi, RedBear BLE Nano and many others. Some of these devices (e.g. Raspberry Pi), thanks to high speed CPUs (on the order of GHz), RAM memories and persistent flash memories can provide high performance. On the contrary, devices like Arduino can be easily programmed even by users with limited programming skills. Other devices (e.g. ReadBear BLE Nano), due to their low power consumption, are suitable for remote 24/7 IoT monitoring with a simple 3.3V battery. The three aforementioned devices have already been integrated in SAndroidE. If the user wants to use just basic functions (like reading voltage level on an analog input pins, or setting a digital output pin value), it is sufficient to upload the compatible firmware onto the device; the SAndroidE App will be already able to exploit the device resources. If the user wants to perform custom operations with the external device, a customization of the device firmware needs to be carried out.

B. Non-programmable devices

These devices do not allow the user to perform firmware flashing, but BT primitives and data exchange protocols are known; such protocols do not use data encryption, or, if they do, encrypting keys are known. SAndroidE allows the communication with these devices by declaring these primitives and exchange protocol within Device Descriptors Files, which are the means for the device virtualization. This is the case for BLE beacons or other very simple devices, exposing non-sensitive data, like environment temperature, humidity or similar.

C. Vendor Locked device: data from vendor's APP

This mode applies to all those devices whose vendor, as part of its business strategy, has opted for a closed BT data exchange protocol and encryption, allowing access only by means of a dedicated proprietary APP. Flic buttons, as an example, are general purpose BT buttons; they send click events to the smartphone if pressed, held or double pressed. Such buttons need to be associated to the dedicated vendor's APP first. In this case, the SAndroidE APP works by receiving notifications by the vendor's APP.

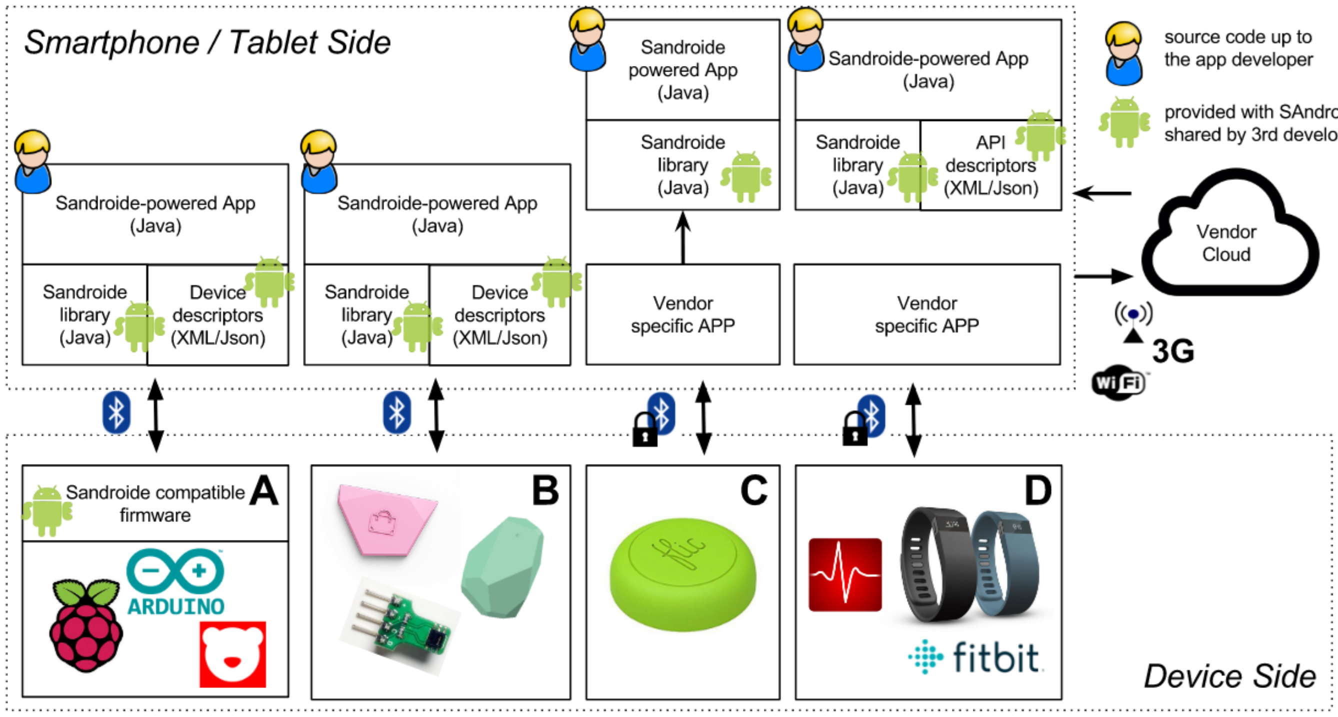
Smartphone / Tablet Side
Sandroide-powered App (Java)
Sandroide library (Java)
Device descriptors (XML/Json)
Sandroide-powered App (Java)
Sandroide library (Java)
Device descriptors (XML/Json)
Sandroide powered APP (Java)
Sandroide library (Java)
Vendor specific APP
Sandroide-powered App (Java)
Sandroide library (Java)
API descriptors (XML/Json)
Vendor specific APP
source code up to the app developer
provided with SAndroidE or shared by 3rd developers
Vendor Cloud
3G
Wi Fi
Sandroide compatible firmware
A
ARDUINO
B
C
D
fitbit
Device Side

Fig. 4. Software conceptual blocks to be handled by a programmer of a two-sensor application in SAndroidE environment.

D. Vendor Locked device: data from vendor's Cloud

This mode applies to all those devices where the vendor, as part of its business strategy, has opted to provide data only by accessing the proprietary Cloud (e.g. activity trackers, cardio-belts, etc…). Fitbit (https://www.fitbit.com) activity trackers, as an example, require the user to install the dedicated APP to the smartphone. Such APP contains the encrypting keys needed to correctly establish the BT connection with the device and to receive activity data. After receiving data, the APP uploads them to the vendor's Cloud, on the Internet. The user may access personal historical data by browsing the related website. Fitbit, as many other vendors that use a proprietary Cloud to store users' data, (in compliance with applicable law), also provides a suitable API, that is an Internet endpoint which guarantees access for users to their own personal and sensitive data in a programmatic way. SAndroidE is able to retrieve data from this kind of devices by means of these APIs, upon user explicit consent.

TABLE I shows the requirements to use SAndroidE with the aforementioned class of devices. It is important to highlight that the device interface can be achieved without the need of modifying the device firmware; only Android and Java programming skills are required, in order to develop the user APP. Moreover, SAndroidE is an open source project, thus, if needed, the customization of the compatible firmware of a device is still possible.

TABLE I. REQUIREMENTS TO USE SANDROIDE WITH DIFFERENT CLASS OF DEVICES

	HW Requirements	*Components Required*
A: User programmable devices	- BT connection	- BT Device Description (XML/JSON file) - Firmware flashing of the device
B: Non-programmable devices	- BT connection	- BT Device Description (XML/JSON file)
C: Vendor Locked devices: APP	- BT connection - Vendor's APP installed	- BT Device Description (XML/JSON file)
D: Vendor Locker device: Cloud	- BT connection - Vendor's APP installed - Internet connection	- Vendor REST API Description (XML/JSON file)

IV. CASE STUDIES IN A SMART CITY PROJECT

In order to test the effectiveness of the proposed architecture, the aforementioned project "Brescia Smart Living" has been adopted as a test case. As already mentioned, ambient assisted living, teleservices, and helpline applications are provided to fragile people by means of specific infrastructures and personal smart devices. In particular, for each user, a smartphone is employed to gather data from a wide range of embedded and external sensors and to communicate with the infrastructure, in order to provide the requested services. The user APP needs to be personalized for each case, in order to achieve a specific goal with a determined set of external connected sensors/devices. Due to the large number of involved users, this operation should be straightforward, still assuring a high level of reliability.

A brief description of a possible scenario with the employed external devices is provided hereinafter.

- Estimote beacons Nearables (http://estimote.com/#get-beacons): these small devices incorporate a tri-axial accelerometer and a BLE module. They can be attached to walls or objects, and by means of their internal sensors they can notice movement. In this specific case, they have been attached to a medicament box, to see if it has been moved and consequently to supervise drug intake.
- Flic buttons (https://flic.io/): they are plastic buttons embedding a BLE module, and they are designed to fulfill activation tasks. Here, they are used as lifesaver buttons, which can be place by the user in a handy spot (e.g., bedside table).
- RedBear BLE Nano (http://redbearlab.com/blenano): based on a Nordic chipset, this small device is typically used to quickly produce prototypes for IoT. In this specific case, it is physically connected to an analogic sensor that measures air quality, and it sends its data to the smartphone via a BT connection.
- Arduino (https://www.arduino.cc/): the well-known electronics platform is used to drive a LED that acts as alarm for medicine in-take notification. Since Arduino is not equipped with a BT module, it is necessary to use a BLE Shield (http://redbearlab.com/bleshield/) in order to exploit the BT connection with the smartphone.
- Raspberry Pi (https://www.raspberrypi.org/): the well-known single-board computer is here used to drive home lights by means of virtual buttons on the smartphone screen.

The implementation of the test case in a traditional approach is quite complex. In fact, various skills and development tools are needed: C programming and Arduino Integrated Development Environment (IDE) are needed for Arduino, JavaScript, (Node.js in particular) and Linux competence are needed for Raspberry Pi, the online device platform MBED is necessary for the RedBear BLE Nano, and specific APIs are needed for the Estimote and Flic device integration. Moreover, knowledge about the BLE protocol is probably inevitably. Thanks to the SAndroidE framework, all these requirements are no longer necessary; it is sufficient to use an Android APP development environment (such as Android Studio) and to have some Java code skill.

Moreover, due to the external device virtualization, SAndroidE APPs leads to a reduced SLOC compared to APPs developed with a traditional approach. In fact, it has been estimated that the lines of code (calculated by summing up Java code and all device specific codes) for the latter case are about ten thousand, whereas only five hundred lines of code are enough to manage all the devices in the SAndroidE APP.

Finally, it is important to highlight the multivendor nature of this approach; sensors from different vendors are managed by SAndroidE in the same way, thus assuring interoperability and the easiness of adoption of redundancy strategies to improve system reliability.

V. Conclusions

In this paper, the SAndroidE framework is presented as a solution for designing sensory systems; its use in the context of a smart living project, also concerning safety and security issues, is discussed. Such framework offers easiness of development for Android APPs communicating with external sensors and/or devices, regardless of the manufacturer. A sensible SLOC reduction is achieved, thus decreasing the possibilities of faults and increasing overall reliability. Moreover, thanks to the simple management of external resources, easiness of APP personalization and configuration is offered as well. The provided test case, with strict requirements in terms of large number of external sensor, APP personalization, and code reliability, has demonstrated the validity and the effectiveness of the proposed approach.

References

[1] "Towards a Definition of the Internet of Things (IoT)", *IEEE Internet Initiative*, Telecom Italia S.p.A., Milan, May 2015.

[2] https://www.cia.gov/library/publications/the-world-factbook/rankorder/2151rank.html

[3] http://ec.europa.eu/eurostat/tgm/table.do?tab=table&language=en&pcode=tps00001

[4] De Angelis G., De Angelis A., Pasku V., Moschitta A., Carbone P., "A simple magnetic signature vehicles detection and classification system for Smart Cities," *Proceeding of the 2016 IEEE International Symposium on Systems Engineering (ISSE)*, 3-5 Oct 2016.

[5] http://uk.kantar.com/media/876729/kantar_os_share_jan_2015_final.pdf

[6] Malatras A., Asgari A., Baugé T., "Web Enabled Wireless Sensor Networks for Facilities Management," *IEEE Systems Journal*, vol. 2, no. 4, Dec. 2008.

[7] https://developer.android.com/things/index.html

[8] Rinaldi S., Depari A., Flammini A., Vezzoli A., "Intergrating Remote Sensors in a Smartphone: the project "Sensors for ANDROID in Embedded systems"," *Proceedings of the 2016 IEEE Sensors Applications Symposium (SAS)*, 20-22 Apr 2016.

[9] https://github.com/SAndroidEOfficial

[10] Zhu C., Wang H., Liu X., Shu L., Yang L. T., Leung V. C. M., "A Novel Sensory Data Processing Framework to Integrate Sensor Networks With Mobile Cloud," *IEEE Systems Journal*, vol. 10, no. 3, Sep. 2016.

[11] S. Vañó-Galván, J. Paoli, L. Ríos-Buceta, and P. Jaén, "Skin self-examination using smartphone photography to improve the early diagnosis of melanoma.," *Actas Dermosifiliogr.*, vol. 106, no. 1, pp. 75–7, Jan. 2015.

[12] O. Yurur, C. H. Liu, and W. Moreno, "Light-Weight Online Unsupervised Posture Detection by Smartphone Accelerometer," *IEEE Internet Things J.*, vol. 2, no. 4, pp. 329–339, 2015.

[13] G. Appelboom *et al.*, "Smart wearable body sensors for patient self-assessment and monitoring.," *Arch. public Heal.*, vol. 72, no. 1, p. 28, 2014.

[14] Hongqiao Gao, Xiaohui Duan, Xiaoqiang Guo, Anpeng Huang, and Bingli Jiao, "Design and tests of a smartphones-based multi-lead ECG monitoring system," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2013, pp. 2267–2270.

[15] A. Depari, A. Flammini, E. Sisinni, and A. Vezzoli, "A wearable smartphone-based system for electrocardiogram acquisition," *IEEE MeMeA 2014 - IEEE Int. Symp. Med. Meas. Appl. Proc.*, pp. 1–6, 2014.

[16] C. Crema, A. Depari, A. Flammini, E. Sisinni, and P. Bellagente, "Virtual Respiratory rate sensors: an example of a smartphone based integrated and multiparametric mHealth gateway", *IEEE Transactions on Instrumentation and Measurements*, ISSN:0018-9456, vol. 66, issue 9, pp. 2456-2463, 2017.

[17] J. Y. Shin, S. L'Yi, D. H. Jo, J. H. Bae, and T. S. Lee, "Development of smartphone-based stethoscope system," *Int. Conf. Control. Autom. Syst.*, no. Iccas, pp. 1288–1291, 2013.

[18] A. Sinharay, D. Ghish, P. Deshpande, S. Alam, R. Banerjee, A. Pal, "Smartphone Based Digital Stethoscope for Connected Health – A direct Acoustic Coupling Technique," *2016 IEEE First Conference on Connected Health: Applications, Systems and Engineering Technologies*.

[19] P. Bellagente *et al.*, "The 'Smartstone': using smartphones as a telehealth gateway for senior citizens," *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 221–226, 2016.

[20] A. C. Olteanu, G. D. Oprina, N. Ţăpus, and S. Zeisberg, "Enabling mobile devices for home automation using ZigBee," *Proc. - 19th Int. Conf. Control Syst. Comput. Sci. CSCS 2013*, pp. 189–195, 2013.