IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received April 4, 2019, accepted May 13, 2019, date of publication May 22, 2019, date of current version June 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2918239

Improving Resource Consumption in Context-Aware Mobile Applications Through Alternative Architectural Styles

GUADALUPE ORTIZ[1], ALFONSO GARCÍA-DE-PRADO[1], JAVIER BERROCAL[2], AND JUAN HERNÁNDEZ[2]
[1]Ucase Software Engineering Group, Escuela Superior de Ingeniería, University of Cádiz, 11500 Cádiz, Spain
[2]Quercus Software Engineering Group , University of Extremadura, Escuela Politécnica, 10003 Cáceres, Spain

Corresponding author: Guadalupe Ortiz (guadalupe.ortiz@uca.es)

This work was supported in part by the Spanish Ministry of Science and Innovation and the European Union FEDER Funds under Grant TIN2015-65845-C3-3-R, Grant TIN2016-81978-REDT, Grant RTI2018-093608-B-C33, and Grant RTI2018-094591-B-I00, in part by the Interreg V-A España-Portugal (POCTEP) 2014-2020 Program under 4IE Grant 0045-4IE-4-P, in part by the Department of Economy and Infrastructure of the Government of Extremadura under Grant GR18112 and Grant IB18030, and in part by the European Regional Development Fund.

ABSTRACT Over the last years, the Internet of Things has fostered a growing interest in context-aware mobile applications; this fact is mainly due to highly favoring information provision from multiple Internet-connected devices. To identify user context, these applications collect information from the user and his/her environment and typically filter app information, so that the user receives only the interesting and relevant information. However, such a task usually implies further resource consumption on user mobile devices, not only regarding battery usage but also in terms of network traffic. Accordingly, although context-aware applications can improve user experiences in their daily lives, they must ensure the maintenance of low-level resource consumption; otherwise, the applications are promptly replaced by less consuming ones, and therefore, removed from the mobile market. In this paper, we evaluate and discuss several architectural styles for context-aware mobile applications, as well as, providing a set of guidelines to decide on the right architecture for a particular app depending on its characteristics. The use of such guidelines when choosing the right architectural style can strongly influence the resource consumption of context-aware mobile applications. Following these guidelines, user satisfaction of a context-aware mobile application may be improved, thus guaranteeing the app success.

INDEX TERMS Context awareness, mobile computing, Internet computing, resource consumption.

I. INTRODUCTION

Nowadays, smartphones are undoubtedly omnipresent worldwide and their success relies on several factors that have evolved over the last decade: improvement of their hardware capabilities, weight reduction, a decrease in mobile communications costs and higher speed Internet connection, among others. However, unquestionably, the parallel evolution of mobile software applications (*apps*) has been key to their success: currently successful mobile applications consume low resources and provide key functionalities to mobile users. In this scenario, most people have replaced their laptops by mobile phones for daily operations related to Internet connection, such as browsing for information, using social networks or additional apps which would have previously been installed as desktop applications or accessed from their laptops through HTML browsers.

Concurrently, the impressive evolution of Internet of Things (IoT) over the last years has strongly favored the provision of information by multiple sensors and other devices connected to the Internet, as well as fostering interest in context-aware applications [1]. These applications gather users' contexts in order to adapt their behavior to their needs and circumstances. Whereas IoT systems usually require interaction with a large number of devices and the management of a lot of information from varying sources,

The associate editor coordinating the review of this manuscript and approving it for publication was Juan Liu.

context-aware applications can adapt and particularize the aforementioned information and interactions to user contexts, improving users' experiences [2].

To identify user context, these apps collect information from the user and their environment and the particular app information is filtered accordingly, to only provide the user with the relevant one to him/her; however, such a task might imply further resource consumption [3], not only concerning battery usage, but also network traffic. Nevertheless, in order to be accepted by users, apps have to be efficient and consume a reasonable amount of resources [4], [5]. The user will have to weigh the functionalities offered by the app against resource consumption, but if there are several options available, they will probably choose the one that consumes the least amount of resources.

Resource consumption depends on the functionalities and the architectural design [6] being used. Whereas functionalities are fixed in a particular app, the most appropriate architectural style for the app can be decided upon. Currently, for instance, context-aware apps may be developed following a server-centric, a mobile-centric or a hybrid architectural style. Server-centric architectures are those in which mobile devices act as simple clients and most of the information storage, processing, and communication tasks are relegated to one or more servers, usually located in the cloud. Mobile-centric ones are some emerging architectures inspired by distributed processing that harness the current processing and storage capabilities of mobile phones. Finally, hybrid architectures combine some information processing and storage both in the server and client side. In this scope, server-centric approaches [7], [8], mobile-centric [9]–[11] and hybrid architectural ones [12], have supporters and detractors, since they all present some advantages and drawbacks in terms of resource consumption.

The number of context-aware apps is increasing, but some of them do not succeed due to difficulties in selecting a suitable architecture to collect and filter the appropriate contextual information whilst keeping resource consumption reasonably low [2]. This is why this paper aims at providing the means to decide on the right architecture for a particular app and, therefore, to improve its resource consumption, guaranteeing user satisfaction and, therefore, the app success. For this purpose, we will conduct an early analysis and an experimental one for a real case study, whose results will be instrumental in helping developers choose the appropriate architecture for each specific context-aware app.

In particular, two of the authors of this paper proposed CARED-SOA (Context-AwaRe Event-Driven Service-Oriented Architecture) [13] in the past, a server-centric architecture which facilitates the incorporation of data coming from devices connected to IoT in order to provide real-time notifications to users, mainly through a mobile app. The architecture was particularized in the scope of air quality and a server-centric app called Air4People was provided. In this paper, we have considered the evolution of CARED-SOA and Air4People to mobile-centric and alternative hybrid architecture applications in order to assess how to improve the app resource consumption and, therefore, user satisfaction. To start with, we have made use of an early analysis framework [6]—proposed by the other two authors of this paper—to asses several architecture alternatives for social applications. For the early analysis we have considered three general alternatives: a server-centric approach, a mobile-centric approach and a hybrid architectural one, where the cost for primitive operations for dynamic and static contexts have been taken into account. After an early analysis of resource consumption, a mobile-centric and a hybrid architecture have been implemented and evaluated in comparison with the original server-centric one; besides, we have compared and discussed the theoretical results provided by the early analysis with those obtained from the implementations. Afterwards, we have argued which one is the optimal architecture depending on the particular case study and its context characteristics, validating the early analysis frameworks and providing more detailed guidelines to choose the right architectural style. Therefore, the main aim and contribution of this paper is to provide generic formulae, which may be used to follow an early analysis evaluation for different architectural implementations of context-aware mobile applications. In particular, such formulae allow developers to evaluate which architectural style —server-centric, mobile-centric or hybrid architecture— provides better performance with regards to data and battery consumption of context awareness features. Such context features have been classified, as later explained with more details, as static or dynamic: static context features do not need to be continuously monitored and dynamic context features might require continuous monitoring. That differentiation clearly influences the resource consumption of the application, for this reason, the formulae provided in this paper consider consumption levels for both types. The fact that our approach focuses on the resource consumption of context features as well as being able to discriminate consumption depending on the type of context before actually implementing the app is what differentiates our proposal from other existing ones. We also provided a case study and an empiric evaluation of more complex scenarios where several context features are considered, which were compared to the early analysis' theoretical evaluation so as to have a reference about how both results may differ. Consequently, as a second contribution, based on the results of the empirical and theoretical evaluations, we provide guidelines and suggestions for the most suitable architectural style according to the type of context feature.

The rest of the paper is organized as follows. Section II gives the motivation and case study that motivated this research paper and Section III describes related work. Then, Sections IV and V provide an early analysis of the different evaluated architectural styles in a generic way and particularized for the case study, respectively. Afterwards, Section VI explains how the selected architectural styles have been implemented for the case study, as well as showing

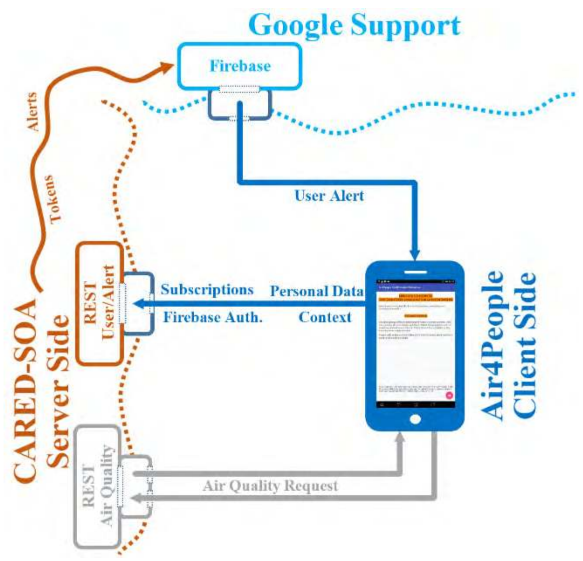


FIGURE 1. Air4People and CARED-SOA architecture.

the tests' configuration and results. Section VII compares the early analysis and the final evaluation results, and discusses the results applied to the Air4People app, as well as how these could be applied to other scenarios' architectures and apps. To end with, conclusions and future work are presented in Section VIII.

II. MOTIVATION AND CASE STUDY

As previously mentioned, in the past we proposed the context-aware server-centric architecture CARED-SOA. CARED-SOA is composed of a set of key elements such as an enterprise service bus, a set of REST services, a complex event processing engine, a context broker and a mobile application. Such elements and the full architecture functionality are explained in [13], whose details are out of the scope of this paper. In our present work, we are going to abstract from the main elements which detect the IoT data and provide real time notifications, to focus on the app context awareness and resource consumption.

In [13], CARED-SOA is particularized for a case study related to air quality; as a result, we obtained Air4People, a system which facilitates air quality monitoring and user notification. Its high-level architectural design, represented in Fig. 1, is composed of the server infrastructure, a context-aware client mobile application and Firebase support. Fig. 1 represents CARED-SOA and Air4People, where we have used gray for those elements which are not relevant to this evaluation.

Air quality monitoring involves measuring several pollutants, the most relevant ones being Particulate Matter ($PM_{2.5}$ and PM_{10}), Carbon Monoxide (CO), Ozone (O_3), Nitrogen Dioxide (NO_2) and Sulphur Dioxide (SO_2)). Currently, Air4People server side is monitoring air quality in the Spanish Andalusian region. The Andalusian Governmental Department of Environment establishes ranks for each air pollutant at four levels: *good*, *acceptable*, *unhealthy* and *very unhealthy*. With the support of a pulmonologist, we established a set of recommendations for citizens according to the air quality level and the citizen's context. For instance, an acceptable value of Ozone—with no recommendation for healthy people—would imply recommending people with lung disease not to engage in any physical activity outside. REST services in the architecture server-side allow the client (1) to set their personal details and notification subscription preferences, (2) to send contextual information and (3) to check air quality values for a particular date, upon request.

Depending on the scope of the app, the context may embrace different sets of characteristics [14]. The context for Air4People will be the particular *location*, *personal* characteristics and *physical* activity of the user in question, as explained in the following paragraphs.

A. LOCATION CONTEXT

When the user logs in to the system, he or she can register for a particular location or choose for his/her location to be monitored: using the GPS, the mobile device will know the user's location and the system will receive continuous updates on its location.

B. PHYSICAL CONTEXT

We have just mentioned that poor air quality affects people doing physical exercise outside more seriously. Therefore, it would be important to know if the user is, for instance, running or making some sort of effort outdoors. For such purposes we use Google Awareness: if the user is running or biking, lower level notifications should be submitted to such users.

C. PERSONAL CONTEXT

This type of context consists of the following personal characteristics, according to the official AQI technical assistance document [15]:

- Lung diseases.
- Heart diseases.
- Children (including teenagers): everybody younger than 19 years old [16].
- Older people: we will regard everybody older than 60 years old as an older person [17].
- Genetic variants: there are several papers which link genetic variants and poor air quality exposure to an increase in certain illnesses ([18]).
- Diets limited in Omega-3 and vitamins. According to several studies, it seems significant that Omega-3 and certain vitamins are key to preventing health risks derived from air pollution [19].

Such context characteristics can be classified, as shown in Fig. 2, as static (they do not need to be continuously monitored and, therefore, the user sends them to the system during registration or occasionally updates his/her data) or dynamic (they might require continuous monitoring). In particular:

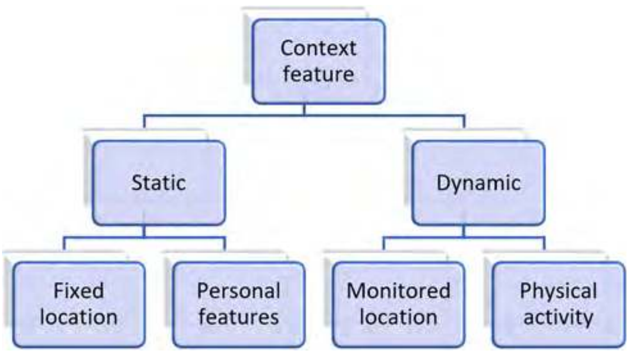


FIGURE 2. Context taxonomy for the case study.

- Location context can be obtained statically (the user would set a fixed location) or dynamically (the GPS is constantly monitoring user location).
- Physical context will mainly be obtained dynamically, thus the app will constantly be checking user physical activity.
- Personal context refers to illnesses or particular personal features or conditions the user might have, therefore these are static characteristics set once in the app, which should not vary over time.

The Android application implemented to facilitate context-aware air quality notifications to interested users provides two main functionalities: (1) checking current or past air quality values and (2) receiving notifications concerning current air quality. In this paper, we focus on (2), since notification reception is an activity which requires constant context monitoring. With the said app, the user sets his/her personal details when he or she registers in the app. Then, they can initially subscribe to alerts of interest (alerts for the four existing levels of air quality conditions for each relevant pollutant). All these data can be updated at any time. This information is sent to the server side through the User/Alerts REST service invocation.

Such an application obtains the user's context through the mobile device, submits it to the server side through the invocation of the User/Alert service, and receives context-aware notifications sent from the server side by means of Firebase [20]. Firebase is a Google platform which improves Android applications significantly. Among other utilities, it facilitates cloud storage and notifications to mobile devices through a cloud manager. Besides, the Firebase platform facilitates user secure login using REST services, and mobile notifications, under subscription.

To summarize, Air4People mobile notifications are particularized depending on the user's context; the said context is classified as static or dynamic, as previously explained. In particular, we monitored location and physical activity dynamically and we took into account static particular user features related to air quality issues, such as lung diseases. Currently, all the static information is stored in the server side and the mobile app is continuously sending the dynamically monitored information to server-side databases. When an air quality alert is detected in the server side, the databases are checked to see which users might be affected by such an alert depending on their current location, activity and personal features, and only those are accordingly notified in their mobile app.

Air4People was tested with real data coming from the sensor stations the Andalusian regional government has all over its territory, and we also performed load and stress tests with an emulator. The results showed that the architecture is suitable for context-aware IoT applications but two limitations were detected. On the one hand, continuously monitoring and sending user location to the server side might consume excessive battery and, on the other, when the number of users increases it will be costly to continuously send user contexts to the server, as well as searching which users have to be notified whenever an alert is detected. These limitations encourage us to examine alternative architectures in order to improve performance and reduce resource consumption.

Please note that, for the sake of readability of this paper and the comparison of the different architectural designs, we have selected 2 context characteristics —a dynamic and a static one—, since both types have different resource consumption patterns. Nevertheless, the evaluation at the end of the paper has been extended to additional context characteristics.

III. RELATED WORK

Mobile devices are constantly increasing their computing and storage capabilities; however, their resources (especially battery and data traffic) do not increase at the same pace and, therefore, consumption is key to the success of any mobile application [21]. Consequently, developers have to make a trade-off between the capabilities to utilize in their apps and the resulting resources that would be consumed in general, and for context-aware applications in particular.

A. MOBILE DEVICE RESOURCE CONSUMPTION

Currently, there are a number of studies focused on measuring the consumption of mobile devices or some of their components. In [22], the authors present a characterization of the different settings of a mobile device using crowdsourced battery discharge measurements. This characterization allows any user to fine-tune a mobile device in order to reduce battery consumption. Nevertheless, developers also need studies to help them to identify how to implement the app or decide on the architectural style to be selected in order to reduce resource consumption.

Studies such as [23] and [24] compare consumption information provided by the devices with measurements obtained using an external power monitor. Others, such as [25] and [26], go a step further, and obtain consumption information from the device battery. These studies present precise information on the device consumption. However, they are complex to reproduce when the mobile device and the app have mobility requirements. The methods presented in this work provide accurate information on the consumption with each architectural alternative and, in addition, can be used in different environments.

B. MOBILE APPLICATION RESOURCE CONSUMPTION

Other papers focus on the consumption of mobile applications. In [27], the authors propose a data-driven method to estimate the discharge rates for all hardware components. To that end, they analyze different reports from thousands of users with information about the components used by each application as well as battery discharge. They also reuse the obtained models by using them to predict the app battery use on specially instrumented devices. Nevertheless, these models focus on the consumption of mobile apps, not on their functionalities.

In [28], the authors analyze the interdependencies between devices' hardware and their applications in order to characterize the energy demand. This study allows the authors to propose an energy-aware operating system for mobile devices. Although this is an important step that should be considered by any operating system, it is still necessary to analyze and improve the resource consumption of mobile apps in order to obtain an even higher efficiency.

In this sense, the focus is on resource consumption within specific applications in [29]. To that end, the authors measure the energy spent by an application performing various tasks such as rendering images on the device screen, or building an internal database for the application. Even though this information is quite important for any application, there is still a lack of measurements related to the gathering and management of contextual information. On the other hand, other studies, such as [30] and [31], focused on identifying the development languages which support a reduction in resource consumption. In [32], the authors analyze currently available cross-platform frameworks to measure how their architecture impacts energy consumption. In addition, in [33], the authors present a systematic literature review of different methods, analyzing how consumption patterns impact on the effectiveness of mobile applications.

C. CONTEXT-AWARE APPLICATIONS RESOURCE CONSUMPTION

In the scope of context-aware systems and applications, relevant challenges, such as context acquisition and characterization, have been faced by multiple frameworks over the last years [14]. In particular, local versus centralized processing and energy consumption are still major challenges [36], especially when dealing with mobile applications [5]; however, most context-aware frameworks proposed in the past (such as [37]–[41]) do not even provide a performance analysis of resource consumption.

Paying special attention to the resource consumption of contextual information gathering, in [34], the authors analyze the consumption of the sensors deployed on mobile devices. In addition, they propose a framework for managing the sensing requirements of mobile applications. Nevertheless, the computation of that information and its interaction with other devices sharing it should also be taken into account when measuring the consumption of context-aware apps. Min et al. [35] indicate that continuous sensing apps introduce non-trivial persistent battery drain and, more significantly, some applications drain battery at different rates depending on the user's context. To face battery issues, they propose a mobility-aware information advisor to help users manage remaining battery and schedule recharging patterns. The mobility-aware battery model could also be used to reduce battery consumption.

Resource consumption of context-aware mobile applications has also been thoroughly analyzed in *Wireless Sensor Networks* (WSN). WSN consist of spatially distributed autonomous sensor-equipped devices to monitor physical or environmental conditions [42]. The sensing capabilities of these devices are usually energy-constrained. To reduce energy consumption in WSN, some works define new protocols and optimization techniques, such as coverage protocols for turning off some sensors [43]–[45]. Other works, such as [46], introduce an active inference of data using dynamic Gaussian Bayesian networks, able to identify when a specific sensor should be pulled for a reading and when it ought to be in power-saving mode, limiting battery consumption while maximizing data accuracy. Some of these techniques could be applied to identify when or how frequently specific mobile phone sensors should be activated. Nevertheless, some guidelines helping developers to identify the less consuming architectural designs are also needed.

Other approaches focus on *Mobile Crowd Sensing* (MCS) to get information from users' contexts and perform distributed sharing and computing of the gathered data. In MCS systems, it is important to minimize energy consumption on users' mobile devices, since high energy consumption severely reduces their willingness to participate. Therefore, these approaches analyze the architectural design of the applications to reduce battery consumption [47], as well as analyzing different energy-aware techniques (such as task assignment [48], reduction of data transferring [49] or data aggregation [50]).

Finally, in [51], the authors analyze the most important optimization techniques oriented to reducing battery consumption. Some of these techniques are focused on offloading resource-consuming tasks to cloud environments. In this scope, for instance, [36] and [37] the authors explore the integration of mobile sensing with a cloud-based system to store and obtain the sensed data. The gathered information is further reused in order to tailor the behavior of a mobile app. These approaches provide excellent results, being widely used by commercial mobile applications. However, as mentioned before, not all tasks consume the same amount of resources. A thorough analysis of the application's functionalities should be performed in order to identify under which circumstances each task should be migrated to the cloud environment or deployed on the mobile phone.

Even though context awareness has been an established term for a couple of decades in the field of computer science [54], it has been in the very last years when has taken unprecedented relevance and this is why there is a lack and clear need for guidelines and best-practice guidance for context-aware low consumption development in accordance with current scenarios. Currently, mobile edge computing is also taking great relevance, however the main reference architectures refer to network related issues [55], whereas we focus on end-user software implementation architectures. Therefore, this paper presents a study and provides several formulae and guidelines to facilitate the selection of the architectural style which permits reducing resource consumption of context-aware mobile apps in the early stages of development. In particular, this study analyzes resource consumption depending on the number of contextual characteristics to monitor and taking into account whether that contextual information is static or dynamic. Such guidelines may be complementary to guidelines for network-related issues, on which most related works focus.

IV. RESOURCE CONSUMPTION GENERIC EARLY ANALYSIS

Prior to the implementation and performance evaluation of alternative architectures, we are going to follow a resource consumption early analysis with the aim of discerning which might be the most appropriate architecture and discarding others.

In [6], a conceptual framework for the early analysis of social apps was proposed. This framework details a set of steps that should be followed to identify under which circumstances each architectural style is less resource consuming or, even, how application evolution (in terms of success, number of users, user interaction with the application etc.) could lead to a change in resource consumption, therefore demanding application redesign. The proposed steps are the following:

- Define the different architectural designs to be analyzed.
- Select the set of resources that should be estimated.
- Identify the set of primitive operations. These are the most common basic operations of mobile applications. Thus, an application's most important functionalities can be composed of these primitive operations.
- Identify which use cases are relevant to the system and especially affect consumption.
- Calculate resource consumption of each use case, as well as the entire application's consumption for each architectural design.

In general terms, we can consider three general alternatives: a server-centric approach, a mobile-centric approach and a hybrid architectural one (although, as later explained, two hybrid architectural approaches will be analyzed), where a dynamic context (DyC) and a static one (StC) have been taken into account. Let $A = \{SC, MC, HA1, HA2\}$ be the set of architectural styles evaluated, $C = \{DyC, StC, null\}$ be the set of contextual information evaluated and $R = \{Battery, Data\}$ be the set of resources analyzed.

Let Opt be the set of primitive operations that can be used to compose an application's functionalities. For each Opt_i, $i = 1, \ldots, nopt$, its resource consumption can be calculated using formula (1):

$$Copt_i : C\ x\ R \rightarrow \mathbb{R} \tag{1}$$

Finally, let UC be the set of functionalities or use cases of an application $\{uc_1, \ldots, uc_{nuc}\}$. For each uc_i, $i = 1, \ldots, nuc$, its resource consumption can be calculated using formula (2):

$$Cuc_i : A\ x\ C\ x\ R \rightarrow \mathbb{R} \tag{2}$$

In order to facilitate the readability of the formulae we will represent the consumption of each use case as follows (formula (3)):

$$Cuc_i^A : C\ x\ R \rightarrow \mathbb{R} \tag{3}$$

Let f be the frequency at which the *opt* primitive operation is executed. This frequency highly depends on the specific architectural design being applied and the contextual information to be sensed or used to limit this frequency. Therefore, this frequency can be calculated using formula (4):

$$f : A\ x\ Copt\ x\ C \rightarrow \mathbb{R} \tag{4}$$

In order to improve the readability of the formula we will represent the frequency as shown in formula (5):

$$f_{Copt}^A : C \rightarrow \mathbb{R} \tag{5}$$

In Table 1, we have included a set of definitions for primitive operations and frequencies to facilitate the reading of the following sub-sections. In spite of this, all frequencies are defined and explained in their first occurrence in the text.

Please note that battery consumption will be expressed in μAh and data consumption in *bytes*.

A. SERVER-CENTRIC APPROACH

For dynamic contexts, in the server-centric approach, we need to obtain the context in the mobile device at a particular frequency and submit it to the server-side. Then, let $f_{get}^{SC}(DyC)$ be the frequency at which we need to obtain DyC value and $f_{post}^{SC}(DyC)$ theDyC context submission frequency to the server. In order to obtain dynamic context DyC, battery consumption would be as shown in (6):

$$Cuc_{get}^{SC}(DyC, Battery) = Copt_{get}(DyC, Battery) * f_{get}^{SC}(DyC) \tag{6}$$

Posting context data to the server every time it is obtained would consume battery and data in (7) and (8), respectively:

$$\begin{aligned} Cuc_{post}^{SC}(DyC, Battery) &= Copt_{post}(DyC, Battery) \\ &\quad * f_{post}^{SC}(DyC) \quad (7) \\ Cuc_{post}^{SC}(DyC, Data) &= Copt_{post}(DyC, Data) \\ &\quad * f_{post}^{SC}(DyC) \quad (8) \end{aligned}$$

Receiving the relevant push notifications (filtered in the server side according to contexts DyC and StC) would also consume battery and data (see (9) and (10)),

TABLE 1. Primitive operations and frequencies.

Nomenclature	Definition
Primitive operations	
get	Obtaining (sensing) dynamic context information from the mobile device
post	Posting context information from the mobile device to the server
push	Receiving push notifications in the mobile device
read	Reading static context information stored in the mobile device
Frequencies	
$f_{get}^{SC}(DyC)$	Frequency at which DyC value is sensed in a SC approach
$f_{get}^{MC}(StC)$	Frequency at which DyC should be sensed when push notifications meet context StC (MC approach)
$f_{get}^{HA1}(DyC)$	Frequency at which DyC value is sensed in a HA1 approach
$f_{post}^{SC}(DyC)$	Frequency at which DyC value is posted in a SC approach
$f_{post}^{HA1}(DyC)$	Frequency at which DyC value is posted in a HA1 approach
$f_{push}^{SC}(DyC \wedge StC)$	Frequency at which push notifications are received in a SC approach (notifications which meet both DyC and StC)
$f_{push}^{MC}(null)$	Frequency at which push notifications are received in a MC approach (all notifications)
$f_{push}^{HA1}(DyC)$	Frequency at which push notifications are received in a HA1 approach (notifications which meet DyC)
$f_{push}^{HA2}(StC)$	Frequency at which push notifications are received in a HA2 approach (notifications which meet StC)

$f_{push}^{SC}(DyC \wedge StC)$ being the frequency at which notifications meet both contexts:

$$Cuc_{push}^{SC}(DyC \wedge StC, Battery) = Copt_{push}(null, Battery) * f_{push}^{SC}(DyC \wedge StC) \quad (9)$$

$$Cuc_{push}^{SC}(DyC \wedge StC, Data) = Copt_{push}(null, Data) * f_{push}^{SC}(DyC \wedge StC) \quad (10)$$

As regards the static context, since it is not expected to vary over time we have not considered resource consumption by submitting it once.

B. MOBILE-CENTRIC APPROACH

For a mobile-centric application, we have to take into account that the mobile phone receives all the push notifications ($f_{push}^{MC}(null)$) and discards those which the user is not subscribed to: first, notifications based on static context *StC* are discarded and afterwards, out of the remaining notifications, those based on dynamic context *DyC* are also eliminated. Gathering information from dynamic contexts would only be required for those notifications that have been filtered through static context. Therefore, this would consume battery as shown in (11), where $f_{get}^{MC}(StC)$ is the frequency at which context *DyC* should be sensed in order to filter the push notifications meeting context *StC*.

$$Cuc_{get}^{MC}(DyC \wedge StC, Battery) = Copt_{get}(DyC, Battery) * f_{get}^{MC}(StC) \quad (11)$$

Since there is no need to post context data to the server, resources are not consumed by this operation.

However, receiving frequent push notifications and reading static context information (*read*) to discern whether a certain notification needs to be shown to a particular user would consume battery and data (see (12) and (13)):

$$\begin{aligned} Cuc_{push}^{MC}&(DyC \wedge StC, Battery) \\ &= (Copt_{push}(DyC \wedge StC, Battery) \\ &\quad + Copt_{read}(StC, Battery)) * f_{push}^{MC}(null)) \end{aligned} \quad (12)$$

$$\begin{aligned} Cuc_{push}^{MC}&(DyC \wedge StC, Data) \\ &= (Copt_{push}(DyC \wedge StC, Data) \\ &\quad + Copt_{read}(StC, Data)) * f_{push}^{MC}(null) \end{aligned} \quad (13)$$

Please, note that following this architectural style, it is not necessary to post or store the obtained context *DyC*, since once a push notification reaches the mobile device and passes all the filters related to the static information, dynamic context *DyC* is obtained in order to filter the remaining push notifications. We need to take into account that we could have filtered by dynamic context first and then by the static one, both filters being on the mobile phone, but such a configuration would consume considerably more resources, since the dynamic context would have to be checked more frequently.

C. HYBRID ARCHITECTURAL APPROACHES

We considered two hybrid architectural approaches: *Hybrid Architecture 1 (HA1)* frequently submits a dynamic context *DyC* to the server-side, and it also filters received notifications according to a static context *StC*. In this scenario, obtaining dynamic context *DyC* would consume the battery in (14):

$$Cuc_{get}^{HA1}(DyC, Battery) = Copt_{get}(DyC, Battery) * f_{get}^{HA1}(DyC) \quad (14)$$

Posting context data to the server with a particular frequency would consume battery and data in (15) and (16), respectively:

$$Cuc_{post}^{HA1}(DyC, Battery) = Copt_{post}(DyC, Battery) * f_{post}^{HA1}(DyC) \quad (15)$$

$$Cuc_{post}^{HA1}(DyC, Data) = Copt_{post}(DyC, Data) * f_{post}^{HA1}(DyC) \quad (16)$$

However, only those notifications that passed the filters for DyC ($f_{push}^{HA1}(DyC)$) would be sent to the smartphone to check static context *StC* filters. This would also consume battery and data (see (17) and (18)):

$$\begin{aligned} Cuc_{push}^{HA1}&(DyC \wedge StC, Battery) \\ &= (Copt_{push}(StC, Battery) \\ &\quad + Copt_{read}(StC, Battery)) * f_{push}^{HA1}(DyC) \end{aligned} \quad (17)$$

$$\begin{aligned} Cuc_{push}^{HA1}&(DyC \wedge StC, Data) \\ &= (Copt_{push}(StC, Data) \\ &\quad + Copt_{read}(StC, Data)) * f_{push}^{HA1}(DyC) \end{aligned} \quad (18)$$

Hybrid Architecture 2 (HA2) would be the approach in which the static context is sent to the server so that the latter only submits notifications according to the user's subscription. Then, once a notification is received by the mobile device, dynamic context *DyC* is obtained and only those notifications related to the context in question are shown to the user.

Obtaining context *DyC* for each received push notification would consume the battery in (19):

$$Cuc_{get}^{HA2}(DyC \wedge StC, Battery) = Copt_{get}(DyC, Battery) * f_{push}^{HA2}(StC) \quad (19)$$

Since there is no need to post the context *DyC* data to the server, resources are not consumed by this operation.

However, receiving the subscribed push notifications, already filtered according to context *StC*, would consume battery and data (see (20) and (21)):

$$Cuc_{push}^{HA2}(DyC \wedge StC, Battery) = Copt_{push}(DyC, Battery) * f_{push}^{HA2}(StC) \quad (20)$$

$$Cuc_{push}^{HA2}(DyC \wedge StC, Data) = Copt_{push}(DyC, Data) * f_{push}^{HA2}(StC) \quad (21)$$

As we can see, the formulae results may vary largely depending on the particular frequency required for dynamic context update for each architectural style, the amount of push notifications received and the size of both the context information to be submitted and the push notifications to be received. This is why we proceed to particularize the early analysis for our case study in the following section.

V. RESOURCE CONSUMPTION EARLY ANALYSIS FOR THE CASE STUDY

As previously explained, we are going to evaluate four architectural designs: the already developed server-centric app, a mobile-centric app where all features are monitored and verified in the mobile side, and two alternative hybrid architecture apps where some of such features are monitored in the mobile-side and others are checked in the server side. These have been evaluated non-specifically in Section III and will be particularized according to the case study context types in the next subsections. A dynamic context —location— and a static one —personal features—have been selected in order to carry out the assessment. Therefore, the alternative architectures, also represented in Fig. 3, are the following:

1. For the server-centric style, (i) the mobile phone sends the personal features to the server once and (ii) the mobile has to constantly check the GPS location and immediately send it to the server. The server, once an air quality alert is detected, checks which users are subscribed to a certain type of alert and requests Firebase to send them a notification. The mobile phone only receives those push notifications which are relevant to the user in question.

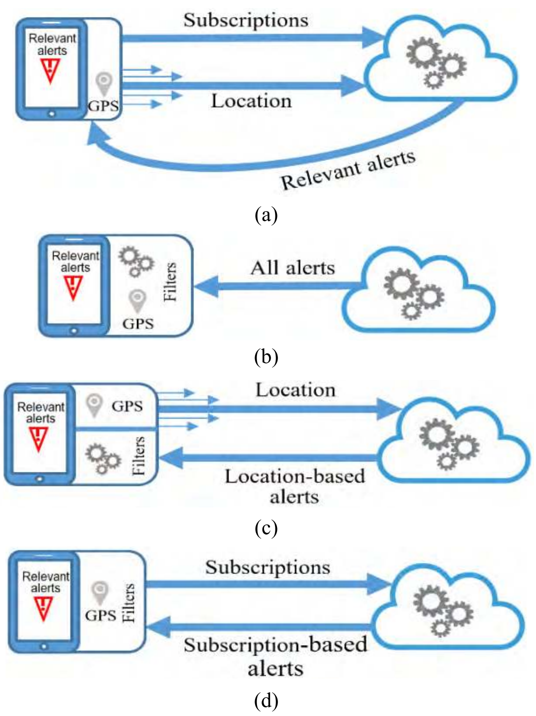


FIGURE 3. Evaluation of four alternative architectures in the early analysis stage. (a) Server-centric architecture. (b) Mobile-centric architecture. (c) Hybrid architecture 1. (d) Hybrid architecture 2.

2. For a mobile-centric application the mobile receives all the push notifications corresponding to Andalusian air quality alerts and (i) firstly, it has to filter the notifications according to the stored personal features, discarding those not relevant to the user in question; and (ii) secondly, it needs to obtain the GPS location and filter the remaining notifications, only selecting those which correspond to the user's current location. Note that personal features are stored in advance, usually only once when the app is installed and configured; because of that, they have not been taken into account when estimating resource consumption during the app daily use.
3. The hybrid architecture application could have two configurations:
   i) To send the GPS location constantly and filter the received notifications according to the personal features (PFs) stored in the device.
   ii) To send the static information to the server so that it only sends notifications according to the particular user's features; then, once a notification is received, the GPS location is obtained and only those related to the location in question are shown.

Secondly, in this analysis, our focus has been on battery and data traffic consumption since they are the two resources that affect user satisfaction the most and, particularly, to Air4People users.

Thirdly, according to [6], in mobile applications the most relevant and resource-consuming primitive operations

TABLE 2. Average resource consumption estimated for most relevant operations in Air4People.

Primitive Operation	Definition	Battery (µAh)	Data traffic (bytes)
get (GPS)	Obtaining GPS location information	7.2	0
post (GPS)	Posting GPS location information	16.83	63
push (null)	Receiving push notifications filtered by GPS location and personal features	18.36	136
push (PFs)	Receiving push notifications filtered by GPS location	18.36	232
push (GPS)	Receiving push notifications filtered by personal features	18.36	285
push (GPS∧PFs)	Receiving all push notifications	18.36	370
read (PFs)	Reading personal features information	0.089	0

Frequencies	Definition	Value (per hour)
$f_{get}^{SC}(GPS)$	Frequency at which GPS location is obtained in the mobile device (every 10 seconds)	360
$f_{post}^{SC}(GPS)$	Frequency at which GPS location is submitted from the mobile device to the server (every 10 seconds)	360
$f_{push}^{SC}(GPS \wedge PFs)$	Frequency at which notifications already filtered by GPS location and personal features are received.	$0 \leq f_{push}^{SC}(GPS \wedge PFs) \leq 18$
$f_{get}^{MC}(PFs)$	Frequency at which GPS location has to be obtained (notifications only filtered by personal features)	$0 \leq f_{get}^{MC}(PFs) \leq 1080$
$f_{push}^{MC}(null)$	Frequency at which notifications are received (all notifications, not filtered in the server side)	1080
$f_{get}^{HA1}(GPS)$	Frequency at which GPS location is obtained in the mobile device (every 10 seconds)	360
$f_{post}^{HA1}(GPS)$	Frequency at which GPS location is submitted from the mobile device to the server (every 10 seconds)	360
$f_{push}^{HA1}(GPS)$	Frequency at which notifications only filtered by GPS location are received	18
$f_{push}^{HA2}(PFs)$	Frequency at which notifications already filtered by personal features are received	$0 \leq f_{push}^{HA2}(PFs) \leq 1080$

are (a) storing data in the local memory, (b) posting data to a server, (c) getting data from a server, (d) receiving a push notification and (e) obtaining the GPS location. In our case study, storing and obtaining data from a server are not used so these primitives do not consume resources; we will therefore base our analysis on the other three primitives.

For the performed analysis, we took into account the values estimated in [6] for average resource consumption for most relevant operations with the following particular considerations in our app, as shown in Table 2: we took into account the values estimated in [6] for average resource consumption for most relevant operations; in our app, the following considerations were also made:

- In all cases, we will keep the battery consumption estimated in [6]. The reason is that, the length of the data posted and received in our case study being rather close to the one described in [6], it can be considered accurate enough for an early estimation.
- Since we are posting location through Firebase services, we will take into account the content length provided by Firebase for the procedure.
- Also, Firebase provides us with the content length for push notifications. We will have four types of push notifications: the server-centric option only includes the alert to be shown to the user; the hybrid architecture 1 option includes the alert to be shown to the user and additional details to discern which type of alert has taken place; the hybrid architecture 2 option includes the alert to be shown to the user and the location where the alert occurred; finally, the mobile-centric option includes the alert to be shown to the user and additional details to discern the location and type of alert which has occurred. Even though data traffic poses different values for each alternative, we theorized that battery consumption differences would be insignificant, and therefore used the value given in [6] for all of them.
- Finally, we have measured a new primitive operation not specified in the early analysis framework and required by context-aware domains. This operation focuses on reading information stored on the mobile device. In this case study, it is used to estimate the consumption of personal information reading in order to know whether an alert needs to be shown to a particular user.

Fourthly, for the analyzed system, we may consider three use cases that directly affect resource consumption and on which the success of the system largely depends. These use cases focus on getting and updating the dynamic information and sending user alerts:

- Getting the location. The mobile device activates the GPS sensor in order to get the device longitude and latitude.
- Posting the location. The obtained location is posted and stored on a server.
- Getting an alert. The mobile device receives a push notification warning the user about a possible air quality alert.

In the following subsections, the resource consumption of every use case for each architectural design is calculated. To do that, the specific behavior of the use case is also detailed. Besides, in order to be able to estimate resource consumption, we have taken into account that 72 sensor stations from the Andalusian region are currently active. Unfortunately, not all stations measure the 6 relevant air quality pollutants, but 60 of them do, thus providing measurements from 360 sensors of our 6 relevant pollutants (bear in mind that the 6 measurements are not necessarily taken at exactly the same time, but the frequency is approximately the same). Applying the required patterns to monitor air quality and taking into account that each station submits information every 20 minutes, we might have an average of 1080 notifications per hour. Alerts are triggered for *Good*, *Acceptable*, *Unhealthy*and *Very Unhealthy*air quality for every pollutant; we may not necessarily be interested in all of them. For testing purposes, we estimated that the average user would subscribe to one level of every pollutant; that is, to 6 different alerts. We made these assumptions based on the AQI guide to air quality and health [56] where all the referred pollutants are relevant to everybody to a greater or lesser degree; depending on the personal characteristic, he or she might be interested in a higher or lower level for each pollutant, yet always in at least one level per pollutant.

A. SERVER-CENTRIC APPROACH

As previously explained, the following aspects should be considered in a server-centric application:

- The app has to constantly check the GPS location and immediately send it to the server: to be as accurate as a mobile-centric app, we evaluate the location in which the GPS coordinates are obtained and sent every 10 seconds (i.e. frequencies $f^{SC}_{get}(GPS)$ and $f^{SC}_{post}(GPS)$ are 360 times in an hour).
- The mobile only receives those notifications which are relevant to the user in question. Therefore, since he or she is subscribed to 6 different pollutant alerts (an alert per pollutant) and alerts can be received every 20 minutes (i.e. three times per hour), he or she can receive from 0 to 18 alerts ($f^{SC}_{push}(GPS \wedge PFs)$) per hour, since the user is expected to be subscribed only to one station.

According to these assumptions, resource consumption per hour for the server-centric approach would be as we explain in the following paragraphs:

Obtaining GPS location every 10 seconds would consume battery in (22):

$$\begin{aligned} &Cuc^{SC}_{getLocation}(GPS, Battery) \\ &\quad = Copt_{get}(GPS, Battery) * f^{SC}_{get}(GPS) \\ &\quad = 7.2 * 360 = 2592\ \mu Ah \end{aligned} \tag{22}$$

Posting GPS data to the server every 10 seconds would consume battery and data in (23) and (24), respectively:

$$\begin{aligned} &Cuc^{SC}_{postLocation}(GPS, Battery) \\ &\quad = Copt_{post}(GPS, Battery) * f^{SC}_{post}(GPS) \\ &\quad = 16.83 * 360 = 6058.8\ \mu Ah \end{aligned} \tag{23}$$

$$\begin{aligned} &Cuc^{SC}_{postLocation}(GPS, Data) \\ &\quad = Copt_{post}(GPS, Data) * f^{SC}_{post}(GPS) \\ &\quad = 63 * 360 = 22680\ bytes \end{aligned} \tag{24}$$

Receiving the relevant push ($0 \leq f^{SC}_{push}(GPS \wedge PFs) \leq 18$) notifications would also consume battery and data (see (25) and (26)); with the server-centric approach, a small number of notifications is received since alerts are sever-side filtered according to location and subscription, so received notifications only contain alerts for the user in question:

$$\begin{aligned} &Cuc^{SC}_{getAlert}(GPS \wedge PFs, Battery) \\ &\quad = Copt_{push}(null, Battery) * f^{SC}_{push}(GPS \wedge PFs) \\ &\quad = 18.36 * f^{SC}_{push}(GPS \wedge PFs) = \{0, \ldots, 330.48\}\ \mu Ah \end{aligned} \tag{25}$$

$$\begin{aligned} &Cuc^{SC}_{getAlert}(GPS \wedge PFs, Data) \\ &\quad = Copt_{push}(null, Data) * f^{SC}_{push}(GPS \wedge PFs) \\ &\quad = 136 * f^{SC}_{push}(GPS \wedge PFs) \\ &\quad = \{0, \ldots, 2448\}\ bytes \end{aligned} \tag{26}$$

Therefore, total battery and data consumption per hour for the server-centric approach would be as shown in formulae (27) and (28):

$$\begin{aligned} &Capp^{SC}(GPS \wedge PFs, Battery) \\ &\quad = Cuc^{SC}_{getLocation}(GPS, Battery) \\ &\qquad + Cuc^{SC}_{postLocation}(GPS, Battery) \\ &\qquad + Cuc^{SC}_{getAlert}(GPS \wedge PFs, Battery) \\ &\quad = \{8650.8, \ldots, 8981.28\}\ \mu Ah \end{aligned} \tag{27}$$

$$\begin{aligned} &Capp^{SC}(GPS \wedge PFs, Data) \\ &\quad = Cuc^{SC}_{postLocation}(GPS, Data) \\ &\qquad + Cuc^{SC}_{getAlert}(GPS \wedge PFs, Data) \\ &\quad = \{22680, \ldots, 25128\}\ bytes \end{aligned} \tag{28}$$

In order to better illustrate the evolution of this consumption depending on the number of received push notifications, Fig. 4(a) and Fig. 4(b) show how battery and data traffic are respectively affected when increasing this number.

As can be seen, battery and data consumption increase per additional push notification is linear. Nevertheless, this increase is minimal compared with the total consumption. Particularly, every additional push notification entails a 0.21% increase in battery consumption and a 0.59% data traffic increase.

We could do the performance study with a wide variety of frequencies for notifications, but since our goal is to minimize consumption, our main concern is consumption when there are more notifications. Therefore, to facilitate comparison among approaches, we will always consider the worst case scenario, making calculations for the maximum amount of notifications (in this case, 18 per hour). Based on this assumption, the total battery and data consumption would be the one

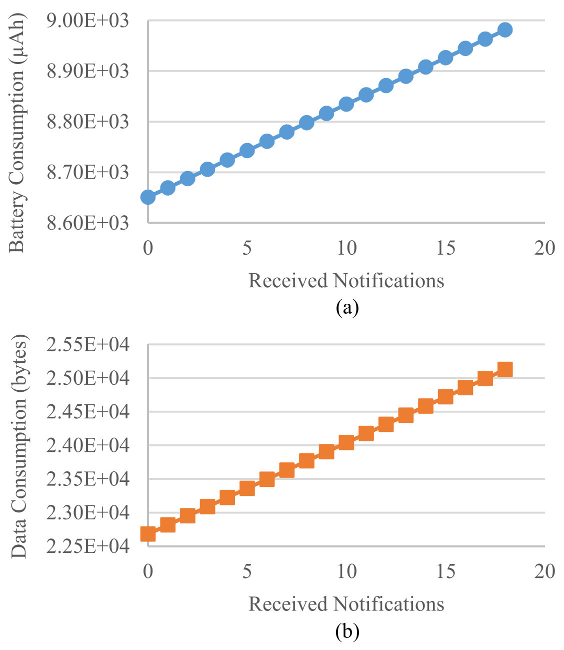


FIGURE 4. Estimated evolution of battery and data consumption depending on received notifications, in the server-centric approach. (a) Battery consumption estimated evolution. (b) Data consumption estimated evolution.

shown in (29) and (30):

$$Capp^{SC}_{AQI}\,(GPS \wedge PFs, Battery) = 8981.28\ \mu Ah \quad (29)$$

$$Capp^{SC}_{AQI}\,(GPS \wedge PFs, Data) = 25128\ bytes \quad (30)$$

It should be noted that the user can make some choices related to the frequency of notification; for instance, they may choose not to be notified if the air quality level for a given pollutant remains unchanged. We set our system to a large number of notifications for evaluation purposes.

B. MOBILE-CENTRIC APPROACH

For a mobile-centric application, we have to take into account that:

- The mobile phone receives all the push notifications corresponding to Andalusian air quality alerts: 1080 alerts per hour ($f^{MC}_{push}(null)$).
- Once an alert is received, the app has to discard those notifications which the user is not subscribed to, according to information stored in the mobile phone. Then, for the remaining alerts, the GPS location is obtained and those alerts which are irrelevant to the current location must be discarded.

Therefore, the GPS location would have to be obtained more or less frequently depending on the number of subscribed alert types ($0 \leq f^{MC}_{get}(PFs) \leq 1080$); 1080 would be our maximum number of notifications if subscribed to all pollutant alerts. This consumption highly depends on the number of subscribed alerts (see (31)).

$$\begin{aligned}&Cuc^{MC}_{getLocation}\,(GPS \wedge PFs, Battery)\\&\quad= Copt_{get}\,(GPS, Battery) * f^{MC}_{get}(PFs)\\&\quad= 7.2 * f^{MC}_{get}(PFs) = \{0, \ldots, 7776\}\ \mu Ah \qquad (31)\end{aligned}$$

As there is no need to post the GPS data to the server, resources are not consumed by this operation.

However, receiving 1080 push notifications would consume battery and data (see (32) and (33)). In this case, larger push notifications are received since alert location information as well as the type of alert have to be included in the message. This way, the mobile-centric app, reading the personal information (*read*), can then discern if a certain notification needs to be shown to this particular user:

$$\begin{aligned}&Cuc^{MC}_{getAlert}\,(GPS \wedge PFs, Battery)\\&\quad= Copt_{push}\,(GPS \wedge PFs, Battery)\\&\qquad+ Copt_{read}\,(PFs, Battery)) * f^{MC}_{push}(null))\\&\quad= (18.36 + 0.089) * 1080 = 19924.92\ \mu Ah \qquad (32)\\&Cuc^{MC}_{getAlert}\,(GPS \wedge PFs, Data)\\&\quad= (Copt_{push}\,(GPS \wedge PFs, Data)\\&\qquad+ Copt_{read}\,(PFs, Data)) * f^{MC}_{push}(null))\\&\quad= (370 + 0) * 1080 = 399600\ bytes \qquad (33)\end{aligned}$$

Please, note that following this architectural style, it is not necessary to post or store the obtained GPS location, since once an alert reaches the mobile device and has been filtered through all the static information (executing the *GetAlert* use case), device location is obtained from the sensor in order to also check that filter (executing the*GetLocation* use case).

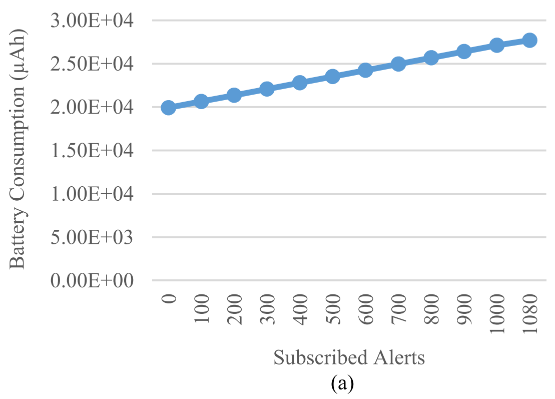
Let us bear in mind that we could have filtered first by location and then by subscription, both filters being in the mobile phone, but such a configuration would consume much more resources since checking the GPS location more frequently would be required.

Therefore, battery and data consumption per hour for this approach would be as shown in formulae (34) and (35), respectively:

$$\begin{aligned}&Capp^{MC}\,(GPS \wedge PFs, Battery)\\&\quad= Cuc^{MC}_{getLocation}\,(GPS \wedge PFs, Battery)\\&\qquad+ Cuc^{MC}_{getAlert}\,(GPS \wedge PFs, Battery)\\&\quad= \{19924.92, \ldots, 27700.92\}\ \mu Ah \qquad (34)\\&Capp^{MC}\,(GPS \wedge PFs, Data)\\&\quad= Cuc^{MC}_{getAlert}\,(GPS \wedge PFs, Data)\\&\quad= 399600\ bytes \qquad (35)\end{aligned}$$

Fig. 5(a) and Fig. 5(b) show the evolution of battery and data consumption depending on the number of alerts the user is subscribed to.

As Fig. 5(a) shows, the increase in battery consumption per subscribed alert is linear, but this increase only entails 28% of the total consumption in the worst case scenario (i.e., when



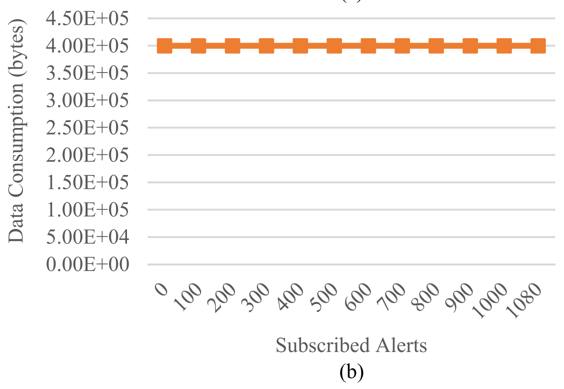


FIGURE 5. **Estimated evolution of battery and data consumption depending on subscribed alerts, in the mobile-centric approach. (a) Battery consumption estimated evolution. (b) Data consumption estimated evolution.**

the user is subscribed to all alerts). Most of the consumption is due to the reception of push notifications and not to obtaining the location. This makes sense, since getting a push notification consumes 255% more than obtaining location.

Moreover, data consumption is stable. Only the reception of push notifications consumes data, since the GPS location does not need to be uploaded to the server.

In this case, taking us back to the worst case scenario, we can assume that the user receives all possible notifications because the pollutant level is the one which the user is subscribed to, so the app would receive notifications from all 1080 sensors. Then, the particular battery and data consumption of the application for this design would be as shown in (36) and (37):

$$Capp_{AQI}^{MC}\left(GPS \wedge PFs, Battery\right) = 27700.92\ \mu Ah \quad (36)$$

$$Capp_{AQI}^{MC}\left(GPS \wedge PFs, Data\right) = 399600\ bytes \quad (37)$$

C. HYBRID ARCHITECTURAL APPROACHES

We consider the two hybrid architecture approaches; let us use Hybrid Architecture 1 for the system where GPS location is constantly submitted and received notifications are filtered according to the personal features stored in the device. Bearing in mind the previous assumptions, location would be submitted every 10 seconds (i.e. frequencies $f_{get}^{HA1}(GPS)$ and $f_{post}^{HA1}(GPS)$ are 360 times in an hour) and we would receive one alert per pollutant 3 times in an hour; that is, 18 alerts per hour ($f_{push}^{HA1}(GPS)$). Please, note that the user would receive all pollutant alerts regardless of whether he or she is subscribed to them or not. This time, the filter is performed on the mobile device. Obtaining GPS location every 10 seconds would consume the battery in (38):

$$\begin{aligned} Cuc_{getLocation}^{HA1}&\left(GPS, Battery\right)\\ &= Copt_{get}\left(GPS, Battery\right) * f_{get}^{HA1}(GPS)\\ &= 7.2 * 360 = 2592\ \mu Ah \end{aligned} \quad (38)$$

Posting GPS data to the server every 10 seconds would consume battery and data (see (39) and (40)):

$$\begin{aligned} Cuc_{postLocation}^{HA1}&\left(GPS, Battery\right)\\ &= Copt_{post}\left(GPS, Battery\right) * f_{post}^{HA1}(GPS)\\ &= 16.83 * 360 = 6058.8\ \mu Ah \end{aligned} \quad (39)$$

$$\begin{aligned} Cuc_{postLocation}^{HA1}&\left(GPS, Data\right)\\ &= Copt_{post}\left(GPS, Data\right) * f_{post}^{HA1}(GPS)\\ &= 63 * 360 = 22680\ bytes \end{aligned} \quad (40)$$

Receiving 18 push notifications and checking the personal information filters would also consume battery and data (see (41) and (42)):

$$\begin{aligned} Cuc_{getAlert}^{HA1}&\left(GPS \wedge PFs, Battery\right)\\ &= \left(Copt_{push}\left(PFs, Battery\right)\right.\\ &\quad \left. + Copt_{read}\left(PFs, Battery\right)\right) * f_{push}^{HA1}\left(GPS\right)\\ &= (18.36 + 0.089) * 18 = 332.082\ \mu Ah \end{aligned} \quad (41)$$

$$\begin{aligned} Cuc_{getAlert}^{HA1}&\left(GPS \wedge PFs, Data\right)\\ &= \left(Copt_{push}\left(PFs, Data\right)\right.\\ &\quad \left. + Copt_{read}\left(PFs, Data\right)\right) * f_{push}^{HA1}\left(GPS\right)\\ &= (232 + 0) * 18 = 4176\ bytes \end{aligned} \quad (42)$$

Therefore, total hourly battery and data consumption for the Hybrid Architecture 1 approach would be as shown in formulae (38) and (39):

$$\begin{aligned} Capp^{HA1}&\left(GPS \wedge PFs, Battery\right)\\ &= Cuc_{getLocation}^{HA1}\left(GPS, Battery\right)\\ &\quad + Cuc_{postLocation}^{HA1}\left(GPS, Battery\right)\\ &\quad + Cuc_{getAlert}^{HA1}\left(GPS \wedge PFs, Battery\right) = 8982.882\ \mu Ah \end{aligned} \quad (43)$$

$$\begin{aligned} Capp^{HA1}&\left(GPS \wedge PFs, Data\right)\\ &= Cuc_{postLocation}^{HA1}\left(GPS, Data\right)\\ &\quad + Cuc_{getAlert}^{HA1}\left(GPS \wedge PFs, Data\right) = 26856\ bytes \end{aligned} \quad (44)$$

Again, receiving the maximum number of notifications, the total battery and data consumption would be as shown in (45) and (46):

$$Capp_{AQI}^{HA1}\left(GPS \wedge PFs, Battery\right) = 8982.882\ \mu Ah \quad (45)$$

$$Capp_{AQI}^{HA1}\left(GPS \wedge PFs, Data\right) = 26856\ bytes \quad (46)$$

Since the battery and data consumption values for this architectural style remain constant in the case study regardless of the number of alerts the user is subscribed to, we have not represented them in a chart.

Hybrid Architecture 2 would be the approach in which personal features are sent to the server so that the latter only submits notifications according to the types of alert the user is subscribed to ($0 \leq f_{push}^{HA2}(PFs) \leq 1080$), where the maximum number of alerts per hour is 1080, as previously explained.

Then, once a notification is received by the mobile device, the GPS location is obtained and the user is only shown notifications related to his or her location.

Obtaining GPS location for subscribed alerts would consume the battery in (47):

$$\begin{aligned} & Cuc_{getLocation}^{HA2}\ (GPS \wedge PFs, Battery) \\ & \quad = Copt_{get}\ (GPS, Battery) * f_{push}^{HA2}(PFs) = 7.2 * f_{push}^{HA2}(PFs) \\ & \quad = \{0, \ldots, 7776\}\ \mu Ah \end{aligned} \tag{47}$$

Since there is no need to post the GPS data to the server, resources are not consumed by this operation.

However, receiving the push notifications would consume battery and data (see (48) and (49)):

$$\begin{aligned} & Co_{getAlert}^{HA2}\ (GPS \wedge PFs, Battery) \\ & \quad = Copt_{push}\ (GPS, Battery) * f_{push}^{HA2}(PFs) \\ & \quad = 8.36 * f_{push}^{HA2}(PFs) = \{0, \ldots, 19828.8\}\ \mu Ah \end{aligned} \tag{48}$$

$$\begin{aligned} & Co_{getAlert}^{HA2}\ (GPS \wedge PFs, Data) \\ & \quad = Copt_{push}\ (GPS, Data) * f_{push}^{HA2}(PFs) = 285 * f_{push}^{HA2}(PFs) \\ & \quad = 285 + f_{push}^{HA2}(StC) = \{0, \ldots, 307800\}\ bytes \end{aligned} \tag{49}$$

Consumption of battery and data per hour for this approach would be as shown in (50) and (51), respectively:

$$\begin{aligned} & Capp^{HA2}\ (GPS \wedge PFs, Battery) \\ & \quad = Cuc_{getLocation}^{HA2}\ (GPS \wedge PFs, Battery) \\ & \qquad + Cuc_{getAlert}^{HA2}\ (GPS \wedge PFs, Battery) \\ & \quad = \{0, \ldots, 27604.8\}\ \mu Ah \end{aligned} \tag{50}$$

$$\begin{aligned} & Capp^{HA2}\ (GPS \wedge PFs, Data) \\ & \quad = Cuc_{getAlert}^{HA2}\ (GPS \wedge PFs, Data) \\ & \quad = \{0, \ldots, 307800\}\ bytes \end{aligned} \tag{51}$$

In order to better illustrate the evolution of this consumption depending on the number of subscribed alerts, Fig. 6(a) and Fig. 6(b) show how battery and data traffic are respectively affected as the amount increases.

As Fig. 6(a) and Fig. 6(b) show, each subscribed alert entails a linear increase in battery and data consumption. Consumption only depends on the number of received alerts; therefore, users only subscribed to a few alerts would consume much fewer resources than those subscribed to many. Once more, we decided to take into account the maximum number of subscribed pollutants and received alerts in order to evaluate the architecture's resource consumption. Then, total battery and data consumption of the application using this architectural style would be as shown in (52) and (53), respectively:

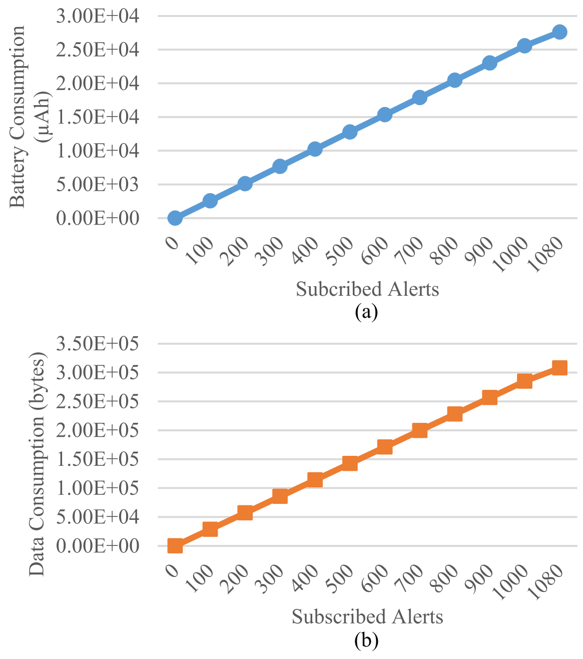


FIGURE 6. **Estimated evolution of battery and data consumption depending on subscribed alerts, in the hybrid architecture 2 approach. (a) Battery consumption estimated evolution. (b) Data consumption estimated evolution.**

$$Capp_{AQI}^{HA2}\ (GPS \wedge PFs, Battery) = 27604.8\ \mu Ah \tag{52}$$

$$Capp_{AQI}^{HA2}\ (GPS \wedge PFs, Data) = 307800\ bytes \tag{53}$$

D. DATA ANALYSIS

In Table 3, we can see the data obtained for the early analysis. If we represent the total battery and data consumption expected for each configuration (see Fig. 7) we can see that, in this scenario, the most appropriate architectures from a resource consumption point of view would be the server-centric and hybrid architecture 1 approaches. However, the mobile-centric architecture would be satisfactory in order not to overload the server side when we have a large number of users. Hybrid architecture 2 consumes as much as the mobile-centric system but does not provide any advantage regarding not overloading the server since (1) both for mobile-centric system and hybrid architecture 2 we need to receive all the notifications and filter according to dynamic context in the mobile and (2) for server-centric system and hybrid architecture 2 we need to filter according to static context in the server side; therefore, we will dismiss this option for the remainder of this paper.

We need to take into account that, in the server-centric and hybrid architecture 1 approaches, GPS consumption was pushed to the limit; therefore, these values will not increase in any case study implementation (they will probably decrease, not requiring 10 second accuracy for location), but data traffic due to push notifications can increase or decrease depending

TABLE 3. Summary of early analysis data.

Approach	Measured Resource	GetLocation	PostLocation	GetAlert	Total Resources Consumed per Hour
Server-Centric Architecture	Battery (µAh)	2 592	6 058.8	330.48	8 981.28
	Data (bytes)		22 680	2 448	25 128
Mobile-Centric Architecture	Battery (µAh)	7 776	0	19 924.92	27 700.92
	Data (bytes)		0	399 600	399 600
Hybrid Architecture 1	Battery (µAh)	2 592	6 058.8	332.082	8 982.882
	Data (bytes)		22 680	4 176	26 856
Hybrid Architecture 2	Battery (µAh)	7 776	0	19 828.8	27 604.8
	Data (bytes)		0	307 800	307 800

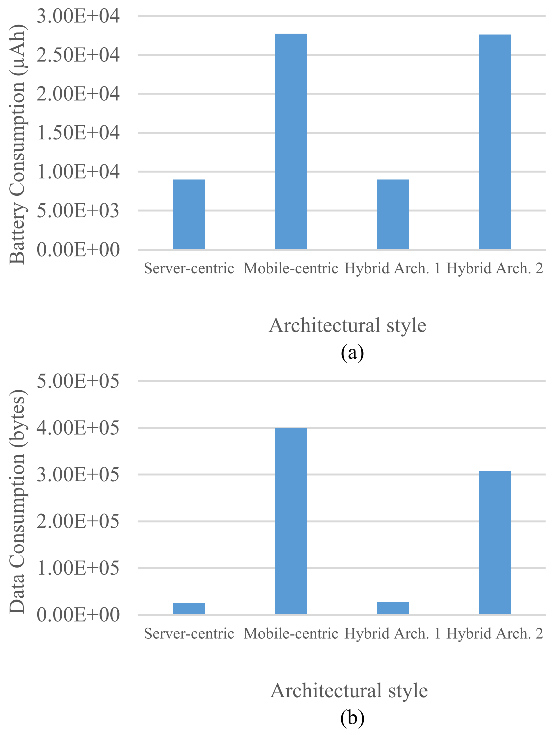


FIGURE 7. Total battery consumption and data traffic obtained in the early analysis for all alternative architectures. (a) Battery consumption in the early analysis. (b) Data consumption in the early analysis.

on the case study. Therefore, we are discarding the hybrid architecture 2 option and we will proceed to implement the server-centric, mobile-centric and hybrid architecture 1 configurations to verify the theoretical study and to measure additional features.

VI. EXPERIMENTAL SETUPS AND RESULTS

In the following subsections, we first of all explain how the three selected alternatives were implemented; secondly, we describe how the tests were performed and what the measured features are and, finally, obtained results are shown.

A. AIR4PEOPLE ALTERNATIVE ARCHITECTURE IMPLEMENTATION

For the experiment's conduction we have implemented the three previously mentioned configurations for CARED-SOA architecture and Air4People app, as represented in Fig. 8 and explained in the following paragraphs.

1) SERVER-CENTRIC IMPLEMENTATION

In this case, all user subscriptions' information and GPS location will be sent to the server through the REST API.

For testing purposes, we have used Firebase to store such data. Therefore, Firebase will store all the data in the cloud and the server will be in charge of filtering the messages based on user location and alert subscription according to personal features, only sending relevant notifications to the particular user. Thus, with such a configuration, the mobile is not involved in any filtering, all taking place on the server side.

2) MOBILE-CENTRIC IMPLEMENTATION

In this case, personal subscription data are stored in the device and the user's GPS location is also filtered in it. For this purpose, we have used Nimbees (http://nimbees.com). NimBees is a commercial mobile push notification platform supporting mobile-centric architectural styles. The platform is composed of an API for mobile applications, which stores users' profiles and allows applications to receive segmented push notifications, and a backend, in charge of managing sent notifications. Once the push notification is sent and has reached the mobile device (through the internal use of Firebase), the API based on the stored profile decides if the owner is an appropriate recipient for that message. Only when the owner is selected as a recipient is the push notification shown in the mobile device, otherwise all notifications remain transparent.

This way, all notifications are sent to the device where it checks whether the user is subscribed to the said alert type, and only then device location is obtained and, if it is within the area of the received notification air quality station, then the notification is shown in the mobile device.

3) HYBRID ARCHITECTURE IMPLEMENTATION

Finally, the hybrid architecture implementation is constantly sending the GPS location to the server and storing it through the use of Firebase; the server filters users by location and only sends notifications to those near the air quality station in question. Subscription data are stored in the mobile

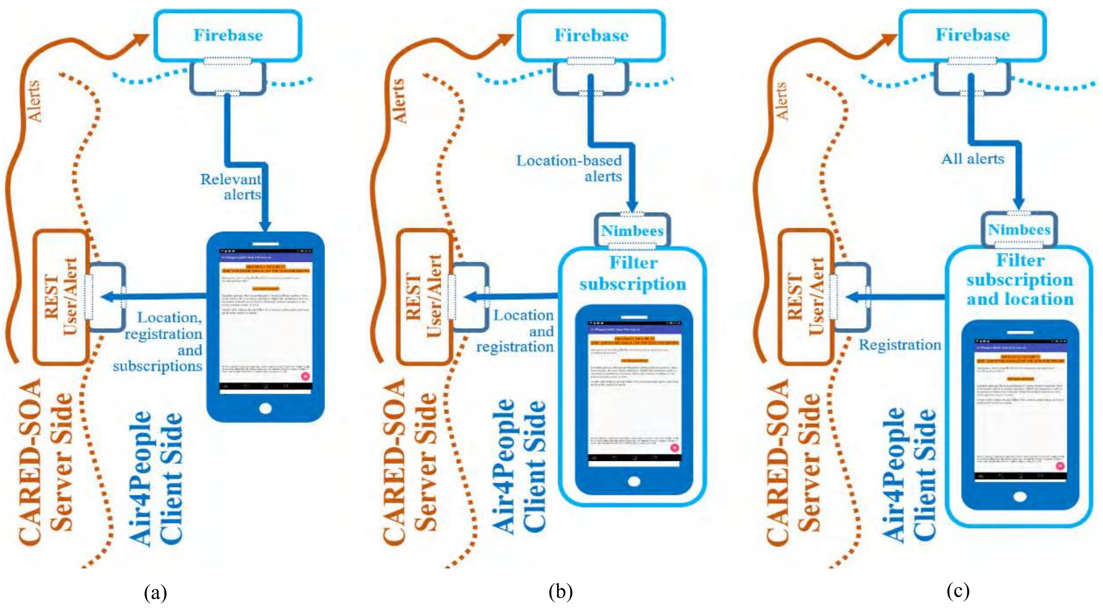


FIGURE 8. Three implemented Air4People app architectural styles. (a) Server-centric architecture. (b) Hybrid architecture. (c) Mobile-centric architecture.

device and Nimbees is used to filter the received notifications according to user subscriptions. From now on, the term hybrid architecture will be used to refer to the previously called hybrid architecture 1 approach.

B. TESTS CONFIGURATION

We have made performance tests with each configuration, location being activated all the time. The test consisted of monitoring the resources consumed by each architecture's application separately once they had been working for an hour. We also tested the system with the three applications at the same time for one hour, to verify relative battery consumption. All tests were carried out first of all in the following handset: a Huawei Honor 6x with a HiSilicon Kirin 655 Octa-core (4x2.1 GHz Cortex-A53 & 4x1.7 GHz Cortex-A53) processor, 3GB of RAM memory and 3340 mAh battery with Android 7.0 (Nougat) as operating system.

During the first five minutes, user personal features and subscriptions were actively changed. Then, the one-hour test was performed. The following data were collected:

- Battery consumption
- GPS usage
- Data posted to the server (after initial configuration)
- Data received through push notifications
- Background system activation

Such data were obtained thanks to several Android applications—GSam Battery Monitor [57], Ampere Meter Pro [58] and Data Usage [59]—, and using Firebase performance utilities [60].

We tested the system in the vicinity of the town of San Fernando air quality station in Spain (current information about San Fernando air quality can be checked at http://airservices.uca.es/Air4People/moteCurrentQAirEnglish/San_Fernando). Through the mobile application, we subscribed to Good CO, Very Unhealthy SO, Acceptable NO_2, Unhealthy O_3, and Very Unhealthy PM_{10} and $PM_{2.5}$ alerts.

TABLE 4. Battery consumption and GPS use of the compared implementations.

Approach	Battery consumption (μAh)	Battery consumption percentage (%)	GPS use (seconds)
Server-centric architecture	51 670	0.6	33
Mobile-centric architecture	235 280	1.1	6
Hybrid architecture	60 190	0.6	33

C. RESULTS

In this sub-section, we explain the results of the evaluation carried out on the three implementations. As we will see, we have been able to obtain additional information which had not been anticipated by the early analysis.

1) BATTERY CONSUMPTION AND GPS USE

Table 4 shows battery consumption in μ Ah and using percentages as well as GPS use in seconds. As we can see in the table, the mobile-centric implementation shows considerably higher battery consumption. In order to rule out higher battery consumption being due to any unexpected process being executed in the system, we verified this result by executing the three applications at the same time, and each application's battery consumption levels were checked and compared when executing them. The results showed again higher battery

TABLE 5. Data consumption of the compared implementations.

Approach	Measured operation	Unit Content length (bytes)	Occurrences in 1 hour (times)	Total data in 1 hour (bytes)
Server-centric architecture	PostLocation	63	204	12 852
	GetAlert	136	3	408
Mobile-centric architecture	PostLocation	0	0	0
	GetAlert	370	1 034	378 880
Hybrid architecture	PostLocation	63	201	12 663
	GetAlert	232	10	2 320

TABLE 6. System information of the compared implementations.

Approach	CPU Use (s)	Background CPU Use (s)	Time Awaken (s)	Awakening (times)
Server-centric architecture	26	0	20	485
Mobile-centric architecture	17	17	13	3 107
Hybrid architecture	26	0	20	520

consumption percentage for the mobile-centric implementation than for the other two.

Regarding GPS use, we can see that the mobile-centric implementation only uses it for 6 seconds, only requiring it when a notification for this particular user's profile is received; however, with the other two applications, the GPS is used for 33 seconds, as location is constantly being monitored and submitted to the server side. Thus, the GPS use is equal between the server-centric and the hybrid architecture, being in both cases much higher than in the mobile centric.

2) DATA TRAFFIC CONSUMPTION

We have two sources of data traffic consumption: location data posted by the application and push notifications received by it. Table 5 shows both types of data traffic for each implementation.

To understand these data, we need to bear in mind that there were 1 034 air quality alerts over the Andalusian territory during the hour the system was tested; out of these, only 10 were for the San Fernando town area and only 3 conformed to the user's personal subscriptions. In the table we can see that posting the location consumes a larger amount of data for server-centric and hybrid architectures, but in terms of getting alerts (receiving notifications), the mobile-centric implementation consumes much more than the other two and the server-centric is clearly defined with the lowest consumption.

3) SYSTEM INFORMATION

As complementary information, we have analyzed additional data obtained from the system: we measured (1) overall CPU usage and only in background (in seconds) and (2) the total time the mobile phone remained awaken (in seconds) and the number of times it awakened. The results are shown in Table 6.

TABLE 7. Additional information on the compared implementations' mobile awakenings.

Approach	Awakening (time length, occurrences and service waking up the system)
Server-centric architecture	14,9s(124)NlpWakeLock 3,9(204)LocationManagerService 2,3s(54)*alarm* 0,6s(97)GCoreFlp 0,01s(3)wake:com.example.air4people/com.google.firebase.messaging.FirebaseMessagingService
Mobile-centric architecture	6,8s(589)GOOGLE_C2DM 5,2s(369)NlpWakeLock 1,3s(56)GCoreFlp 0,9s(1034)wake:com.example.air4people2/com.nimbees.platform.gcm.NimbeesGcmListenerService 0,8s(1034)wake:com.example.air4people2/com.google.firebase.messaging.FirebaseMessagingService 0,2s(25)*alarm*
Hybrid architecture	18,4s(136)NlpWakeLock 4,2(201)LocationManagerService 3,1s(56)*alarm* 0,8s(97)GCoreFlp 0,7s(10)wake:com.example.air4people3/com.nimbees.platform.gcm.NimbeesGcmListenerService 0,1s(10)wake:com.example.air4people3/com.google.firebase.messaging.FirebaseMessagingService 0,05(10)GOOGLE_C2DM

As can be seen in Table 6, CPU use is quite similar in all of the approaches, but in the mobile-centric approach half of the time it is used in background. Consequently, in such an implementation the amount of time the mobile was awaken is lower, despite awakening times being higher; this is due to receiving all notifications without a previous filter from the server side.

We have shown additional data on the processes waking up our Android device in Table 7 to have a better understanding of mobile awakenings.

We can see that, in the mobile-centric implementation, the system wakes up every time a message is received, since all message filtering takes place in the device; awakening time is high compared to the other two configurations; more particularly, out of the 3 107 awakenings, 589 times it was due to GOOGLE_C2DM (used by Google in some notification reception-related tasks) and 1034 times it was caused by Firebase (message reception). Another 1034 times were due to Nimbees (message filtering, that is, checking which messages need to be shown to this particular user). Do bear in mind that even though COOGLE_C2DM has been deprecated and replaced by GCM, there are some internal processes in Nimbees which still use this terminology. The full awakening number decreases to 485 times in the sever-centric-based implementation, and to 520 in the hybrid architectural one. The main process waking up the device now is LocationManagerService, in charge of obtaining device location; messaging services are now used rather less.

Let us note that we repeated the tests with a different handset: a BQ Aquaris V with Qualcomm Snapdragon 435 (octa-core at 1.4 GHz) processor, 4G of RAM memory and

TABLE 8. Battery consumption with a different handset.

Approach	Battery consumption in 1 hour (mAh)	Battery consumption in 24 hours (mAh)
Server-centric architecture	13	186
Mobile-centric architecture	41	589
Hybrid architecture	16	248

3100 mAh battery with Android 7.1.2 (Nougat). The results for both handsets, as expected, were quite similar: the BQ device showed slightly better battery behavior, a completely negligible difference (see Table 8). Besides, we also followed longer period tests; in particular, each implementation was tested for 24 hours. Data and battery consumption were proportional; the amount of data increased proportionally 24 times and battery consumption remained low, as shown in Table 8. It should be noted that each test was repeated three times in order to verify the validity of the obtained results; Table 8 shows the average values for all tests, which display really similar figures. As a result, we might conclude that the selection of the correct architectural design is key to save battery and data consumption. There are two different requirements that are crucial for making this decision, how often the dynamic data varies and the frequency at which such data will be consumed. If an application does not require updated data, but it is frequently consumed, a server-centric style would be more efficient. Instead, if it requires updated data that is consumed occasionally, a mobile-centric style would have a lower resource consumption. This guideline could also be applied to the specific features of the application in order to design and evaluate a hybrid approach.

VII. DISCUSSION

In this section, first of all, we compare the theoretical results provided by our early analysis with those obtained from the implementations; secondly, the obtained results are discussed with regards to the three implemented architectures for Air4People; finally, we discuss how the results may be applied to other case studies.

A. COMPARISON OF EARLY ANALYSIS WITH EXPERIMENTAL RESULTS

Fig. 9 and Fig. 10 show how the experimental results are rather similar to those obtained in the early analysis; even though there are large differences for the obtained battery consumption (see Fig. 9), the important issue is that it has been verified that the estimated resource consumption for the three approaches is reliable. This difference may be due to the use of additional APIs and frameworks (such as Nimbees, and Firebase) that overload the mobile phone. We can say that, for this case study, battery consumption is similar in the server-centric and the hybrid architecture approaches, increasing considerably in the mobile-centric

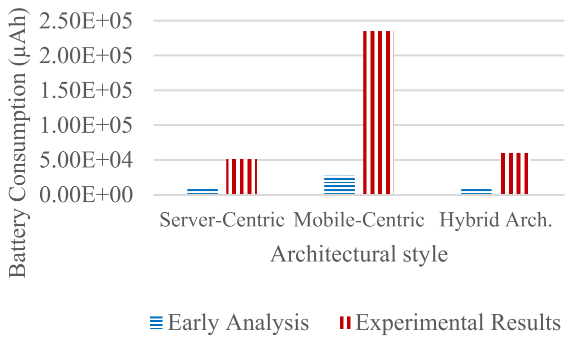


FIGURE 9. Battery consumption comparative study for early analysis and experimental results.

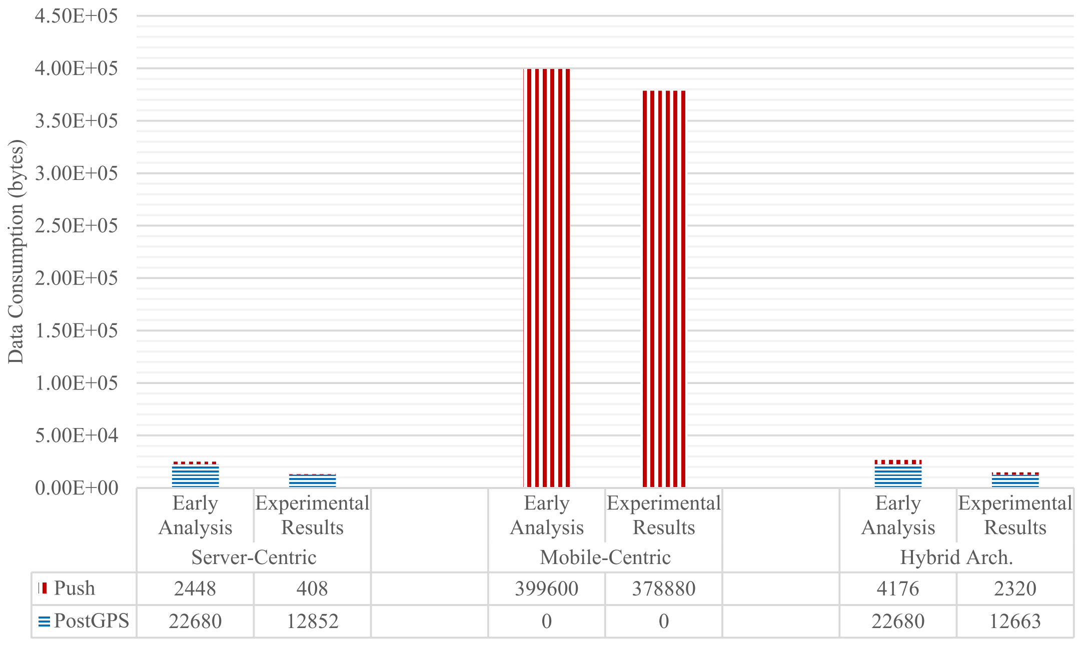
approach. However, it must be taken into account that the actual battery consumption depends on the final device; the relevant fact is to know which approach will consume more and whether the differences are considerable or trivial.

The same observation can be made regarding data traffic (see Fig. 10). We can see that the mobile-centric option consumes more data due to push notifications, whereas the other two approaches present lower (and similar) data consumption, that being mainly due to the GPS location submission to the server side. Still, what is relevant is that the tendency highlighted by the early analysis is confirmed by the experimental results. The similarity between the estimated and the experimental resource consumption shows us that the divergence in battery consumption may be due to the use of specific frameworks and a greater use of the screen (being this what most impacts the battery consumption). The consumption of the primitive operations for the conceptual framework was obtained with the device screen completely off. In contrast, many devices when they receive a push notification turn the screen on. This may be one of the reasons for the divergence, since the biggest difference was found in the architecture that receives the higher number of notifications.

Therefore, we can conclude that our early analysis is reliable and, therefore, additional results for further case studies and characteristics may be estimated.

B. DISCUSSING THE RESULTS FOR AIR4PEOPLE

As we have already mentioned, the mobile-centric approach shows high data traffic. We also mentioned that GPS location submission is not bound to increase in other case studies (we do not expect to require a higher accuracy than 10 seconds); however, it might be the case that push notifications increase or decrease. In order to evaluate which is the better architecture for the case study, we would need to roughly estimate the number of expected notifications. In any case, the above result confirms that, as stated in Section 4.1.2 in [6], whenever the application mainly focuses on getting content rather than posting it, the server-centric option might prove more efficient, always depending on the balance between sending and receiving information. We can therefore state that for the Air4People app, the mobile-centric option is not a solution, but what about the hybrid architecture one?



	Server-Centric		Mobile-Centric		Hybrid Arch.	
	Early Analysis	Experimental Results	Early Analysis	Experimental Results	Early Analysis	Experimental Results
▮ Push	2448	408	399600	378880	4176	2320
≡ PostGPS	22680	12852	0	0	22680	12663

FIGURE 10. Battery consumption comparative study for early analysis and experimental results.

In Fig. 7, Fig. 9 and Fig. 10, as well as Table 4 and Table 5, we could see how, although the results obtained in the server-centric and hybrid architecture solutions were similar, the server-centric solution still obtained better results. Nevertheless, since the difference is rather small, the developer could take into account other aspects that might influence his/her decision. For instance, if sensitive data are used for filtering notifications, we could decide to keep such data in the mobile phones, as might be the case of the respiratory conditions the user might have in relation to air quality notifications.

Furthermore, Air4People has additional features to be taken into account (such as the physical activity being carried out). If we had to monitor and post the data concerning the physical activity or other personal contexts acquired in the smartphone, data traffic would definitely increase. For the sake of simplicity, we will assume that posting such information will consume the same data traffic as posting location data. Sending such data every minute (we chose a one-minute frequency but, again, this will depend on the particular case study) would imply the extra battery and data consumption shown in Table 9 (for instance, when calculating battery consumption for posting the extra feature, since it is being posted 60 times in an hour, we multiply 60*16.83 —see Table 2—).

As we can see, the hybrid architecture implementation is still overtaking the server-centric one. However, the best option will still depend on the number of received notifications and the number of context features to be checked; not only that, also on the overhead we may avoid in the server side when checking a particular user's context features in the client side. This, together with the advantages of keeping sensitive data in the smartphone, makes us still lean towards supporting the hybrid architecture solution for Air4People. In the following section, we perform a further analysis for additional scenarios.

TABLE 9. Additional battery and data consumption per hour when posting one additional context feature every minute.

Approach	Posting Extra Context (battery in μAh)	Total battery (μAh)	Posting Extra Context (traffic data in bytes)	Total traffic data (bytes)
Server-centric architecture	1 009.8	9 991.08	3 780	28 908
Mobile-centric architecture	0	27 700.92	0	399 600
Hybrid architecture	1 009.8	9 992.682	3 780	30 645

C. EXTENDING THE RESULTS AND PROVIDING GUIDELINES TO OTHER SCENARIOS

If we try to extend the results to other scenarios, we must insist on the fact that every scenario and application have different requirements and an early analysis would be advisable in any case, in order to evaluate what the best option might be for them.

However, we need to bear in mind that context-aware apps, which are the main focus of this paper, have one relevant characteristic to consider, that the context is taken into account; such contexts can be dynamic or static. In the case of a *static context* (*personal features* in our case study), it will probably be more efficient to keep such a context in the server side since it is occasionally submitted from the mobile to the server. However, if we are dealing with sensitive data we might consider a hybrid architecture solution (depending on the expected number of notifications to be discarded). In the case of a *dynamic context* –obtained in the mobile device-, we have to bear in mind that posting it might consume more resources and we have to evaluate how many push notifications we can save if we post such a context to the server (depending on the expected number of notifications to be discarded).

In order to extend the results to scenarios with a higher number of context characteristics (for instance, if we want to include physical activity in the evaluated scenario), a variable regarding the number of additional context features has been considered. In particular, we have included $NDyC \epsilon \mathbb{N}$, the domain of natural numbers, dynamic contexts and $NStC \epsilon \mathbb{N}$, static ones. For the sake of simplicity, we are going to assume that all context features require the same frequency for dynamic context submission and the same number and length of received push notifications (varying notification frequencies and lengths would change the obtained result but the consumption trend would remain unchanged).

Let $NDyC$ be the set of dynamic contextual information of an application $\{DyC_1, \ldots, DyC_{ndyc}\}$ and $NStC$ the set of static contextual information of an application $\{StC_1, \ldots, StC_nstc\}$.

In a server-centric approach, if we extend the formulae in Section IV.A with additional features, final battery and data consumption for $NDyC$ dynamic contexts and $NStC$ static ones would be those shown in (54) and (55), respectively:

As a guideline, we can see how both battery and data consumption increase linearly with additional dynamic contexts, but are not affected by static ones. However, consumption highly depends on the required frequency for dynamic context update and the number of received push notifications.

$$\begin{aligned} &Capp^{SC}_{multipleContexts}(NDyC \wedge NStC, Battery) \\ &\quad = \sum_{i=1}^{i=ndyc} \Big[Cuc^{SC}_{get}(DyC_i, Battery) \\ &\quad\quad + Cuc^{SC}_{post}(DyC_i, Battery)\Big] \\ &\quad\quad + Cuc^{SC}_{push}(NDyC \wedge NStC, Battery) \end{aligned} \tag{54}$$

$$\begin{aligned} &Capp^{SC}_{multipleContexts}(NDyC \wedge NStC, Data) \\ &\quad = \sum_{i=1}^{i=ndyc} \Big[Cuc^{SC}_{post}(DyC_i, Data)\Big] \\ &\quad\quad + Cuc^{SC}_{push}(NDyC \wedge NStC, Data) \end{aligned} \tag{55}$$

In a mobile-centric approach, according to the formulae in Section IV.B, final battery and data consumption for $NDyC$ dynamic contexts and $NStC$ static ones would be as shown in formulae (56) and (57), respectively:

$$\begin{aligned} &Capp^{MC}_{multipleContexts}(NDyC \wedge NStC, Battery) \\ &\quad = \sum_{i=1}^{i=ndyc} \Big[Cuc^{MC}_{get}(DyC_i \wedge NStC, Battery)\Big] \\ &\quad\quad + (\sum_{j=1}^{j=nstc} \big[Copt_{read}(StC_j, Battery)\big] \\ &\quad\quad + Copt_{push}(NDyC \wedge NStC, Battery)) * f^{MC}_{push}(null) \end{aligned} \tag{56}$$

$$\begin{aligned} &Capp^{MC}_{multipleContexts}(NDyC \wedge NStC, Data) \\ &\quad = \sum_{j=1}^{j=nstc} \big[Copt_{read}(StC_j, Data)\big] \\ &\quad\quad + Copt_{push}(NDyC \wedge NStC, Data) * f^{MC}_{push}(null) \end{aligned} \tag{57}$$

In this approach, we should take into account as a guideline that consumption apparently does not increase that much when adding dynamic contexts. However, not filtering dynamic contexts in the server side might imply receiving a huge number of notifications. In this case, static context increase leads to further battery consumption for reading the static context from the local memory, however such a reading operation requires very low consumption.

In a hybrid architecture approach, according to the formulae in Section IV.C, final battery and data consumption for $NDyC$ dynamic contexts and $NStC$ static ones for hybrid architecture 1 would be as shown in (58) and (59):

$$\begin{aligned} &Capp^{HA1}_{multipleContexts}(NDyC \wedge NStC, Battery) \\ &\quad = \sum_{i=1}^{i=ndyc} [Cuc^{HA1}_{get}(DyC_i, Battery) \\ &\quad\quad + Cuc^{HA1}_{post}(DyC_i, Battery)] \\ &\quad\quad + (\sum_{j=1}^{j=nstc} \big[Copt_{read}(StC_j, Battery)\big] \\ &\quad\quad + Copt_{push}(NStC, Battery)) * f^{HA1}_{push}(NDyC) \end{aligned} \tag{58}$$

$$\begin{aligned} &Capp^{HA1}_{multipleContexts}(NDyC \wedge NStC, Data) \\ &\quad = \sum_{i=1}^{i=ndyc} [Cuc^{HA1}_{post}(DyC_i, Data)] \\ &\quad\quad + (\sum_{j=1}^{j=nstc} \big[Copt_{read}(StC_j, Data)\big] \\ &\quad\quad + Copt_{push}(NStC, Data)) * f^{HA1}_{push}(NDyC)) \end{aligned} \tag{59}$$

In this case, the hybrid architectural approach entails a linear increase of both battery and data consumptions, since the sensed information needs to be constantly gathered and posted to the server. Similarly, to the server-centric approach, this consumption highly depends on the frequency at which the data are sensed. In addition, every received push notification requires reading the static context, which, although minimal, is a further consumption effort.

As can be seen, each architecture provides advantages and disadvantages and may reduce consumption of the mobile application under specific circumstances. This is why we need to guide developers, through the analysis of resource consumption, to select the most appropriate architecture that

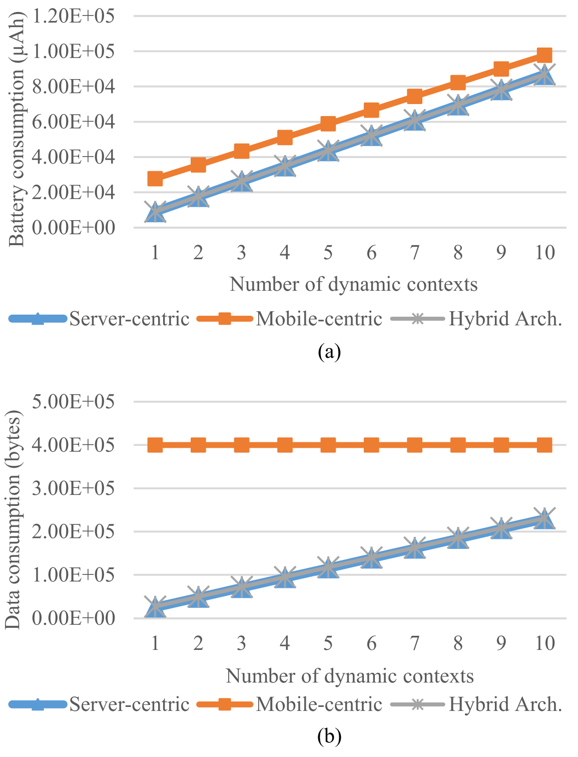
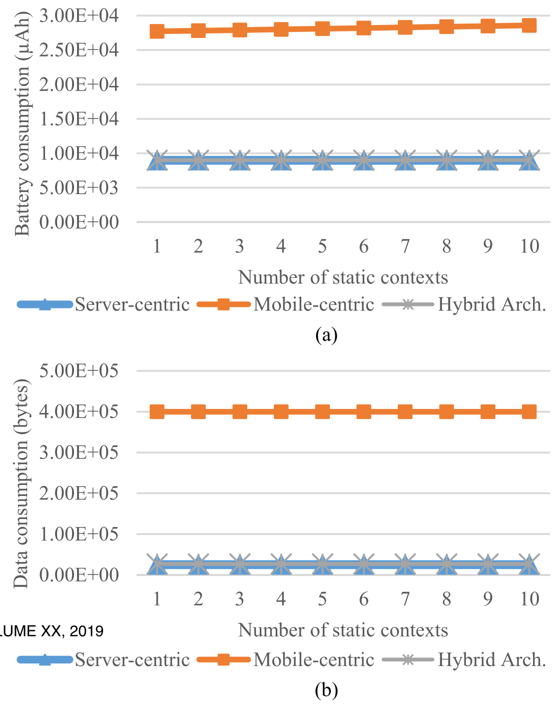


FIGURE 11. Estimated battery and data consumption evolution when increasing dynamic context features. (a) Battery consumption when increasing dynamic contexts. (b) Data consumption when increasing dynamic contexts.



VOLUME XX, 2019

FIGURE 12. Estimated battery and data consumption evolution when increasing static context features. (a) Battery consumption when increasing static contexts. (b) Data consumption when increasing static contexts.

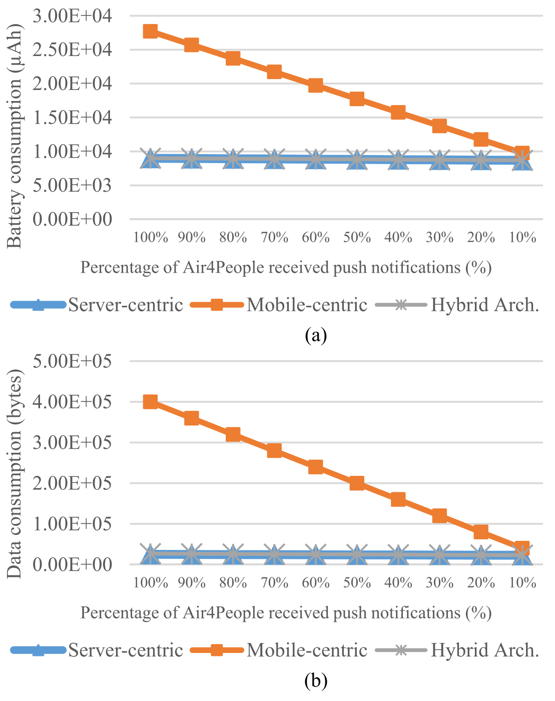


FIGURE 13. Estimated battery and data consumption evolution when decreasing push notifications. (a) Battery consumption when decreasing push notifications. (b) Data consumption when decreasing push notifications.

consumes fewer resources. For instance, in our case study, taking into account additional contexts does not vary the amount of received notifications significantly (for example, when you are running you might receive some additional alerts concerning air quality, but it might be 3-4 more every hour).

To illustrate how additional contexts may affect resource consumption, we estimated some additional values through the early analysis procedure. Fig. 11 and Fig. 12 show how battery and data consumption are affected when increasing the number of dynamic and static context features, respectively— from 1 to 10 additional features (with the same number of notifications).

The hybrid architecture approach seems to be the most efficient one in this case. Please, recall that increasing the number of features does not imply reducing the number of notifications (even though it might be the case for some case studies), but it might mean having different or more personalized notifications according to the context. Analogously, Fig. 13 shows how reducing the total amount of push notifications received by the app —from 100% to 10%— drastically reduces data and battery consumption in the mobile-centric architecture.

The analysis and estimations presented in this work confirm that resource consumption of context-aware applications highly depends on the monitored contextual information and on the selected architectural style. Therefore, before starting to implement the application, it is advisable to carefully analyze which architectural style would be more appropriate.

VIII. CONCLUSION

The state of the art has revealed that, even though context awareness has taken unprecedented relevance over recent years, there is an outstanding lack of guidelines and best-practice guidance for context-aware low consumption development in accordance with current scenarios.

After a preliminary generic resource consumption evaluation of several alternative architectures (server-centric, mobile-centric and hybrid architectures) together with a specific case study's early analysis and later implementation of the said architectural styles, we can conclude that the best architecture for context-aware mobile applications will mainly depend on two factors: the number of notifications or communications required by the app —a higher number of possible notifications to be filtered depending on the context will balance toward server-centric applications— and the type of features to be monitored in the context —a context obtained dynamically in the mobile calls for a mobile-centric application; hybrid architecture applications are also perfect choices when trying to avoid personal context data travelling outside the smartphone. Such statements have been confirmed through an extension and discussion of the generic formulae to a varying number of context features. We can also affirm that the early analysis according to [6] provides a reliable estimation of the app resource consumption; therefore, it is highly recommended to perform it before deciding which is the most appropriate architecture for a particular context-aware app. With regards to CARED-SOA, since we are taking more than one context feature into account, we settle for migrating the initial server-centric implementation to a hybrid architecture one.

One of our current lines of research focuses on the development of an Internet of Things RandOm GENerator (nITROGEN) data emulator to test the functionality and performance of applications for the IoT. In our future work, we hope to condition this emulator to give further support to developers for consumption estimations. In addition, another important part of context-aware applications is the infrastructure cost. Currently, we also work on estimating the economic cost of deploying a context aware application in a cloud environment. This cost could also be evaluated, together with the resource consumption, in order to identify which architecture has a correct balance between the mobile resource consumption and the operational cost.

ACKNOWLEDGEMENT

The authors thank Alberto Espina Granados, Pablo Caballero Torres and Rubén Rivas Troncoso for their support with the experimental setup.

REFERENCES

[1] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.

[2] O. Yurur, C. H. Liu, Z. Sheng, V. C. M. Leung, W. Moreno, and K. K. Leung, "Context-awareness for mobile sensing: A survey and future directions," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 68–93, 1st Quart., 2016.

[3] O. Yurur, C. H. Liu, and W. Moreno, "Modeling battery behavior on sensory operations for context-aware smartphone sensing," *Sensors*, vol. 15, no. 6, pp. 12323–12341, May 2015.

[4] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Softw.*, vol. 32, no. 3, pp. 70–77, May 2015.

[5] N. Capurso, B. Mei, T. Song, X. Cheng, and J. Yu, "A survey on key fields of context awareness for mobile devices," *J. Netw. Comput. Appl.*, vol. 118, pp. 44–60, Sep. 2018.

[6] J. Berrocal *et al.*, "Early analysis of resource consumption patterns in mobile applications," *Pervasive Mobile Comput.*, vol. 35, pp. 32–50, Feb. 2016.

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches: A survey of mobile cloud computing," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013.

[8] D. S. Yadav and K. Doke, "Mobile cloud computing issues and solution framework," *Int. Res. J. Eng. Technol.*, vol. 3, no. 11, pp. 1115–1118, 2016.

[9] J. Guillen, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Softw.*, vol. 31, no. 2, pp. 48–53, Mar. 2014.

[10] S. Tramp, P. Frischmuth, T. Ermilov, S. Shekarpour, and S. Auer, "An architecture of a distributed semantic social network," *Semantic Web*, vol. 5, no. 1, pp. 77–95, Jan. 2014.

[11] F. S. Tsai, W. Han, J. Xu, and H. C. Chua, "Design and development of a mobile peer-to-peer social networking application," *Expert Syst. Appl.*, vol. 36, no. 8, pp. 11077–11087, Oct. 2009.

[12] R. Morrow, "Hybrid application design: Balancing cloud-based and edge-based mobile data," Gigaom, Austin, TX, USA, Tech. Rep., 2013.

[13] A. G. De Prado, G. Ortiz, and J. Boubeta-Puig, "CARED-SOA: A context-aware event-driven service-oriented Architecture," *IEEE Access*, vol. 5, pp. 4646–4663, 2017.

[14] U. Alegre, J. C. Augusto, and T. Clark, "Engineering context-aware systems and applications: A survey," *J. Syst. Softw.*, vol. 117, pp. 55–83, Jul. 2016.

[15] U. S. Environmental Protection Agency. (2016). *Technical Assistance Document for the Reporting of Daily Air Quality—The Air Quality Index (AQI)*. Accessed: May 14, 2019. [Online]. Available: https://www3.epa.gov/airnow/aqi-technical-assistance-document-sept2018.pdf

[16] World Health Organization. (2013). *Definition of Key Terms*. Accessed: May 14, 2019. [Online]. Available: http://www.who.int/hiv/pub/guidelines/arv2013/intro/keyterms/en/

[17] World Health Organization. (2018). *Ageing and Health*. Accessed: May 14, 2019. [Online]. Available: http://www.who.int/news-room/fact-sheets/detail/ageing-and-health

[18] A. Levinsson, A.-C. Olin, L. Modig, S. Dahgam, L. Bjsörck, A. Rosengren, and F. Nyberg, "Interaction effects of long-term air pollution exposure and variants in the *GSTP1*, *GSTT1* and *GSTCD* genes on risk of acute myocardial infarction and hypertension: A case-control study," *PLoS ONE*, vol. 9, no. 6, p. e99043, Jun. 2014.

[19] S. Péter, F. Holguin, L. G. Wood, J. E. Clougherty, D. Raederstorff, M. Antal, P. Weber, and M. Eggersdorfer, "Nutritional solutions to reduce risks of negative health impacts of air pollution," *Nutrients*, vol. 7, no. 12, pp. 10398–10416, Dec. 2015.

[20] Google. (2018). *Firebase*. Accessed: May 14, 2019. [Online]. Available: https://firebase.google.com/

[21] S. S. K. Kasireddy and V. R. Bojja, "Measurements of energy consumption in mobile applications with respect to quality of experience," Ph.D. dissertation, School Comput., Blekinge Inst. Technol., Karlskrona, Sweden, 2012.

[22] E. Peltonen, E. Lagerspetz, P. Nurmi, and S. Tarkoma, "Where has my battery gone?: A novel crowdsourced solution for characterizing energy consumption," *IEEE Pervasive Comput.*, vol. 15, no. 1, pp. 6–9, Jan. 2016.

[23] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES/ISSS)*, New York, NY, USA, Oct. 2010, pp. 105–114.

[24] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "DevScope: A nonintrusive and online power analysis tool for smartphone hardware components," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, New York, NY, USA, Oct. 2012, pp. 353–362.

[25] M. Curti, A. Merlo, M. Migliardi, and S. Schiappacasse, "Towards energy-aware intrusion detection systems on mobile devices," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Helsinki, Finland, 2013, pp. 289–296.

[26] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. 9th Int. Conf. Mobile Syst., Appl. Services (MobiSys)*, New York, NY, USA, 2011, pp. 335–348.

[27] P. Eastham, A. M. Medina, A. Sharma, U. Syed, S. Vassilvitskii, and F. Yu, "Learning battery consumption of mobile devices," in *Proc. 33rd Int. Conf. Mach. Learn. (JMLR: W&CP)*, New York, NY, USA, vol. 48, 2016.

[28] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice, "Exhausting battery statistics: Understanding the energy demands on mobile handsets," in *Proc. 2nd ACM SIGCOMM Workshop Netw., Syst., Appl. Mobile Handhelds, (MobiHeld)*, New York, NY, USA, 2010, pp. 9–14.

[29] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof," in *Proc. 7th ACM Eur. Conf. Comput. Syst. (EuroSys)*, New York, NY, USA, 2012, pp. 29–42.

[30] W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of android app development approaches," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*, May 2017, pp. 42–52.

[31] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: How do energy, time, and memory relate?" in *Proc. 10th ACM SIGPLAN Int. Conf. Softw. Lang. Eng.*, New York, NY, USA, 2017, pp. 256–267.

[32] M. Ciman and O. Gaggi, "An empirical analysis of energy consumption of cross-platform frameworks for mobile development," *Pervasive Mobile Comput.*, vol. 39, pp. 214–230, Aug. 2017.

[33] H. S. A. Al Nidawi, K. T. Wei, K. A. Dawood, and A. Khaleel, "Energy consumption patterns of mobile applications in android platform: A systematic literature review," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 24, pp. 6776–6787, Dec. 2017.

[34] A. A. Moamen and N. Jamali, "Share sens: An approach to optimizing energy consumption of continuous mobile sensing workloads," in *Proc. IEEE Int. Conf. Mobile Services*, New York, NY, USA, Jun./Jul. 2015, pp. 89–96.

[35] C. Min, C. Yoo, I. Hwang, I. Hwang, S. Kang, Y. Lee, S. Lee, S. Park, C. Lee, S. Choi, and J. Song, "Sandra helps you learn: The more you walk, the more battery your phone drains," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, New York, NY, USA, 2015, pp. 421–432.

[36] K. P. Subbu and A. V. Vasilakos, "Big data for context aware computing—Perspectives and challenges," *Big Data Res.*, vol. 10, pp. 33–43, Dec. 2017.

[37] B. Djoudi, C. Bouanaka, and N. Zeghib, "A formal framework for context-aware systems specification and verification," *J. Syst. Softw.*, vol. 122, pp. 445–462, Dec. 2016.

[38] C. Ntanos, C. Botsikas, G. Rovis, P. Kakavas, and D. Askounis, "A context awareness framework for cross-platform distributed applications," *J. Syst. Softw.*, vol. 88, pp. 138–146, Feb. s2014.

[39] A. Badii, M. Crouch, and C. Lallah, "A context-awareness framework for intelligent networked embedded systems," in *Proc. 3rd Int. Conf. Adv. Hum.-Oriented Personalized Mech., Technol. Services*, Nice, France, 2010, pp. 105–110.

[40] E. Baralis, L. Cagliero, T. Cerquitelli, P. Garza, and M. Marchetti, "CAS-MINE: Providing personalized services in context-aware applications by means of generalized rules," *Knowl. Inf. Syst.*, vol. 28, no. 2, pp. 283–310, Nov. 2010.

[41] H. Truong, L. Juszczyk, A. Manzoor, and S. Dustdar, "Escape—An adaptive framework for managing and providing context information in emergency situations," in *Proc. 2nd Eur. Conf. (EuroSSC)*, Kendal, U.K., 2007, pp. 207–222.

[42] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Bus. Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[43] A. More and V. Raisinghani, "A survey on energy efficient coverage protocols in wireless sensor networks," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 29, no. 4, pp. 428–448, Oct. 2017.

[44] X. Gu, J. Yu, D. Yu, G. Wang, and Y. Lv, "ECDC: An energy and coverage-aware distributed clustering protocol for wireless sensor networks," *Comput. Elect. Eng.*, vol. 40, no. 2, pp. 384–398, Feb. 2014.

[45] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Comput. Netw.*, vol. 67, pp. 104–122, Jul. 2014.

[46] C. Komurlu and M. Bilgic, "Active inference and dynamic Gaussian Bayesian networks for battery optimization in wireless sensor networks," in *Proc. AAAI Workshop Artif. Intell. Smart Grids Smart Buildings*, 2016, pp. 241–247.

[47] P. Bellavista, J. Berrocal, A. Corradi, and L. Foschini, "Mobile crowd sensing as an enabler for people as a service mobile computing," in *Ad-Hoc, Mobile, and Wireless Networks*. Heidelberg, Germany: Springer, 2017, pp. 144–157.

[48] H. Xiong, D. Zhang, G. Chen, L. Wang, V. Gauthier, and L. E. Barnes, "iCrowd: Near-optimal task allocation for piggyback crowdsensing," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 2010–2022, Aug. 2016.

[49] L. Wang, D. Zhang, Z. Yan, H. Xiong, and B. Xie, "effSense: A novel mobile crowd-sensing framework for energy-efficient and cost-effective data uploading," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 12, pp. 1549–1563, Dec. 2015.

[50] L. Wang, D. Zhang, Y. Wang, C. Chen, X. Han, and A. M'hamed, "Sparse mobile crowdsensing: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 161–167, Jul. 2016.

[51] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 179–198, 1st Quart., 2013.

[52] J. Hansen, T. M. Gronli, and G. Ghinea, "Cloud to device push messaging on android: A case study," in *Proc. 26th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2012, pp. 1298–1303.

[53] T.-M. Grønli, G. Ghinea, and M. Younas, "Context-aware and automatic configuration of mobile devices in cloud-enabled ubiquitous computing," *Pers. Ubiquitous Comput.*, vol. 18, no. 4, pp. 883–894, Apr. 2014.

[54] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proc. 1st Int. Symp. Handheld Ubiquitous Comput.*, Karlsruhe, Germany, 1999, pp. 304–307.

[55] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of MEC in the Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 84–91, Oct. 2016.

[56] U. S. Environmental Protection Agency. (2014). *AQI Air Quality Index. A Guide to Air Quality and Your Health*. Accessed: May 14, 2019. [Online]. Available: https://www3.epa.gov/airnow/aqi_brochure_02_14.pdf

[57] G. Labs. (2017). *GSam Battery Monitor*. Accessed: May 14, 2019. [Online]. Available: http://www.gsamlabs.com/

[58] Phuongpn. 2018. *Ampere Meter Pro*. Accessed: May 14, 2019. [Online]. Available: https://play.google.com/store/apps/details?id=com.phuongpn.amperemeter

[59] oBytes LCC. (2018). *Data Usage*. Accessed: May 14, 2019. [Online]. Available: https://play.google.com/store/apps/details?id=com.sigterm

[60] Google. (2017). *Firebase Performance Monitoring*. Accessed: May 14, 2019. [Online]. Available: https://firebase.google.com/docs/perf-mon

GUADALUPE ORTIZ received the Ph.D. degree in computer science from the University of Extremadura, Spain, in 2007, where she was an Assistant Professor and a Research Engineer with the Computer Science Department, from 2001 to 2009. In 2009, she joined the Department of Computer Science and Engineering, University of Cádiz, as an Associate Professor. She has authored or coauthored numerous peer-reviewed papers in international journals, workshops, and conferences. Her research interests include aspect-oriented techniques as a way to improve web service development, with an emphasis on model-driven extra-functional properties and quality of service, service context awareness and their adaptation to mobile devices, and complex event processing integration in service-oriented architectures. She has been a member of various programs and organization committees of scientific workshops and conferences over the years, and is a Reviewer for several journals.

ALFONSO GARCÍA-DE-PRADO received the Ph.D. degree in computer science from the University of Cádiz, Spain, in 2017. For several years, he has been a Programmer, an Analyst, and a Consultant for various international industry partners, focusing on software development, evolution, and management for large sports events. Over the years, he has paid special attention to research within the scope of mobile devices and web services for which he has analyzed the advantages of using model-driven and aspect-oriented techniques for service context awareness and published his findings in several journal papers. His research focuses on trending topics, such as the complex event processing integration in service-oriented architectures and context awareness in the Internet of Things.

JAVIER BERROCAL received the Ph.D. degree in computer science from the University of Extremadura, Spain, in 2014. From 2010 to 2016, he was an Assistant Professor with the Department of Informatics and Telematics System Engineering, University of Extremadura, where he obtained an Associate position, in 2016. He has authored or coauthored numerous peer-reviewed papers in international journals, workshops, and conferences. His main research interests are mobile computing, context awareness, pervasive systems, crowd sensing, the Internet of Things, and fog computing. Finally, he is a cofounder of the company Global Process and Product Improvement S.L. (Gloin), which is a software-consulting company founded, in 2010.

JUAN HERNÁNDEZ received the B.Sc. degree in mathematics from the University of Extremadura, Spain, in 1984, and the Ph.D. degree in computer science from the Technical University of Madrid, Spain, in 1995. He is currently a Full Professor of languages and systems, and also the Head of the Quercus Software Engineering Group, Extremadura University, Spain. His research interests include service-oriented computing, ambient intelligence, aspect orientation, and model-driven development. He is involved in several research projects as responsible and senior researcher related to these subjects. He has published the results of his research in more than 100 papers in international journals, conference proceedings, and book chapters. He has participated in many workshops and conferences as a speaker and a member of the program committees, such as the Early-Aspects Workshop Series, the RE track at ACM-SAC series, the ECSA series, the CibSE series, and AOSD. He is currently a member of the Spanish Steering Committee on software engineering, and has organized several workshops and international conferences.