Smart Places: A framework to develop proximity-based mobile applications

Samuel M. Coelho and Miguel L. Pardal
samuel.coelho,miguel.pardal@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa

Abstract. Proximity-based applications engage users while they are in the proximity of points of interest and these apps are becoming popular among the users of mobile devices. The apps are triggered when the user is at specific geographic coordinates or when tagged objects are detected to be nearby, using several technologies, like Bluetooth Low Energy (BLE).
In this paper, we present a solution to develop proximity-based applications, that we call Smart Places. Two examples were built: the Smart Restaurant and Smart Museum. The Smart Places apps allow anyone with a mobile device capable of detecting tags to access proximity-based servicescreated with the tool we developed. These apps were evaluated as viable in terms of energy consumption with results that show acceptable battery consumption.

Keywords: proximity-based; mobile apps; smart place; location based applications

1 Introduction

Nowadays, mobile devices are equipped with many sensors such as light sensor, Global Positioning System (GPS) and accelerometer. Also, these devices access multiple data sources such as the user's calendar and the activity on social networks. The applications (apps) that the user can install have access to this data provided by these sensors and data sources. Using this data, the apps can adjust settings and allow users to perform tasks according to it. The focus of this present work is location based-applications. These apps offer different possibilities, to the users, when they are in the proximity of a given point of interest.

Besides development of proximity-based services another question arises: *How can an owner of a given place offer such services, without having to develop everything him/herself?* We created a tool to develop proximity-based services, removing the need to build a back-end and to handle the location technology.

2 Related Work

We present some applications, where BLE beacons were used, to get good insights about the potential use cases of this technology.

BlueSentinel[1] is a occupancy detection system, for smart buildings, that uses BLE Beacons to detect the presence of people. It is focused on the power efficiency of the building. The idea is to optimize energy consumption according to people's presence.

The authors of ContextCapture[2] try to use context-based information to allow users to add more information to their status updates in the main social networks, such as Facebook[1] and Twitter[2]. Context information comes from the smart-phone itself, from its sensors and from the nearby devices through Bluetooth. Devices can be other smart-phones or BLE Beacons, which are used for indoor location.

There are already some frameworks to develop location-based applications.

In the work presented in Krevl et al.[3], a framework was constructed to allow developers to build location-based apps. Location information can come from any source, such as GPS receivers, Bluetooth receivers and WiFireceivers. The authors do not take into consideration constraints in terms of resources, such as lack of Internet connection and battery.

Dynamix[4] is a framework to develop mobile native and web apps that allow them to receive context information, for instance, position and device's orientation. This framework has plug-ins that get one or more sensor's raw data and turn that into event objects, that contain more high-level information. This framework supports many kinds of context information and it is possible to develop more plug-ins to allow the apps to generate additional events that are not already supported. The security policies of Dynamix state which information the app can have access to or which sensors it can use. However, the formulation of the policies meant a big development overhead that was avoided in our solution.

3 Solution

The main goal of our solution is to assist the development of proximity-based mobile applications. Before starting the description of our solution, we need to take a look at three kinds of users that will be part of it:

- End users: Anyone with a mobile device that installs an app to scan for nearby Smart Places;
- Owners: These users are the ones responsible for managing a given place that they want to turn into a Smart Place;
- Developers: The users that develop the code of the Smart Places.

[1] http://www.facebook.com
[2] http://twitter.com

Figure 1 shows the main components of our solution, which are the following:

– Beacons are the small devices that will act as tags, according to our definition of a Smart Place;
– Back-end is where all the data is stored, that is, the Smart Places that are available, the Smart Places that each owner has configured, information about each beacon, etc;
– End Users Mobile App is another Android mobile app that allows users, with a mobile device, BLE enabled, to have access to nearby Smart Places and to detect the beacons that belong to those Smart Places;
– Owners Mobile App is an Android mobile app that owners use to select which Smart Places they want to configure. It also allows to configure each individual beacon that belongs to a given Smart Place;
– Developers Application Programming Interface (API) provides the necessary methods that developers can use to create their proximity-based services, based on the concept of a Smart Place.

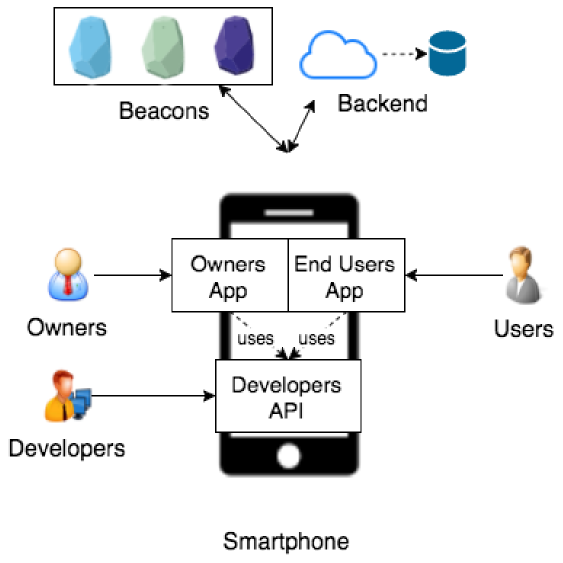

Fig. 1: Overview with the main components of our solution

Owners have a mobile app that allow them to turn the places they manage into Smart Places. There is another mobile app to allow anyone, with a mobile device, such as a smart-phone, to use the services provided by Smart Places nearby. To develop these services our solution offers an API that developers can integrate in their web applications to make them react to the presence of the user.

To create the back-end, we have used Parse[3] Backend as a Service (BaaS). A BaaS is a particular type of Software as a Service, that allows developers to specify which entities and their fields, that they need to store. An BaaS allows developers to not have to be concerned about aspects related to distributed systems, such as, scalability and security. Since our back-end only needs to be able to store and retrieve data, we chose this solution instead of doing an implementation of a Representational State Transfer (REST) API from the scratch.

In our solution, there is an Android app, that notifies the user when he is nearby any Smart Place. When the mobile device approaches any Smart Place, the app notifies the user. When the user touches these notifications, the app shows an embedded web browser that contains a web page that can react to nearby objects, that is, beacons with meaning to the application.

The Smart Places' owners manage one or more places where they want to provide some service to visitors in their mobile devices. In order to make owners be able to offer this kind of services an Android app designed for them is offered by this solution. This app offers features such as, get a list of all available Smart Places and configure an instance of a Smart Place. In order to configure a Smart Place first, owners need to tag physical objects. They need to deploy beacons in the right places. Then, they use the mobile app to create an instance of a Smart Place following a small set of steps. First, the app shows a list of all available Smart Places. Then, the owner selects one and he can see a text explaining what that Smart Place is about. Finally, the owner just needs to type a title and a message, that will appear in the users' mobile devices notifications when they are nearby

The library was turned into an open-source project, hosted on a GitHub repository[4] and it is available to install using bower[5], which is a tool to manage dependencies in web applications. Then, developers just need to include the library and use the available functions. The library is event-based, that is, the mobile apps, for owners and end users emit events to the library such as, a nearby beacon is detected to the web application running inside a embedded web browser. In this library there is a global object, which is "SmartPlaces" with several methods. All those methods need to receive a callback because, as already mentioned, the library follows an event-based approach.

We have created two examples of Smart Places, the Smart Restaurant and Smart Museum. The Smart Restaurant allows customers of a restaurant to place their orders without the need to wait. The Smart Museum allows museum's visitors to have access to more information, in their mobile devices, about a given object in an exhibition, when they are in the proximity of that object. We first built the Smart Restaurant. While building this example, we wrote JavaScript code to handle the events emitted by the mobile apps for owners and end users. From this code, it was possible to create a library that resulted in a complete independent project, from which, the examples depend on. After building the

[3] http://parse.com
[4] http://github.com/samfcmc/smartplaces-js
[5] http://bower.io

first example, we developed another one, which is the Smart Museum. We defined the JavaScript library as a dependency and observed that the same API that fits the Smart Restaurant example, also could be used in the other example.

4 Evaluation

The evaluation focused on two aspects. First, the reliability of the method to get the distance from a given beacon. Second, the battery consumption, since service is running in background, periodically scanning for beacons. In the evaluation process, a smart-phone and a set of three beacons from Estimote™ were used. The smartphone was a Motorola™ Moto G[6], with the following technical specifications:

– Central Processing Unit (CPU): Quad-core 1.2 GHz Cortex-A7[7]
– Graphics Processing Unit (GPU): Adreno 305
– Random-access Memory (RAM): 1 Gigabyte (GB)
– Internal storage: 16 GB
– Screen: 4.5 inches
– Battery: Non-removable Li-Ion 2070 Milliampere-hour (mAh) battery
– Operating System (OS): Android 5.0.2 (Lollipop[8])

4.1 Nearest Beacon Detection

Our solution relies on a library, which its API allows us to get the distance from a given beacon. However, this value is related to the signal's strength, that comes from the beacon. We performed a set of experiments to verify how reliable was this value and if we can use that value to compute which beacon is the nearest one.

The set of experiments, is summarized in Table 1. Figure 2 shows the layout that was used where value d is the distance between beacons. In these experiments, the Smart Musem example was used. In each experiment, it ran for 5 minutes using 10 seconds as the interval between each scan. 10 seconds was chosen because it is a reasonable value to walk in the museum to have enough time to perform any computation, that was needed, after each scan. Running the experiment for 5 minutes, with the mentioned interval between each scan, allowed us to have more than 20 scans. Then, in Android Studio log output, it was possible to check how many times each beacon was detected as the nearest one.

The mobile app for end users scans for beacons but only requests data for the nearest one. To get the nearest one, it has to rely on the signal strength to calculate the distance. We performed 4 experiments in order to try to get the

[6] http://www.gsmarena.com/motorola_moto_g-5831.php
[7] http://www.arm.com/products/processors/cortex-a/cortex-a7.php
[8] https://www.android.com/versions/lollipop-5-0

Variables	**Experiments**			
	1	**2**	**3**	**4**
Number of beacons			3	
Interval between each scan			10s	
Experiment duration			5m	
Distance between beacons (meters)	0.5	1	1.5	2

Table 1: Experiments to get the accuracy of the method to get the nearest beacon

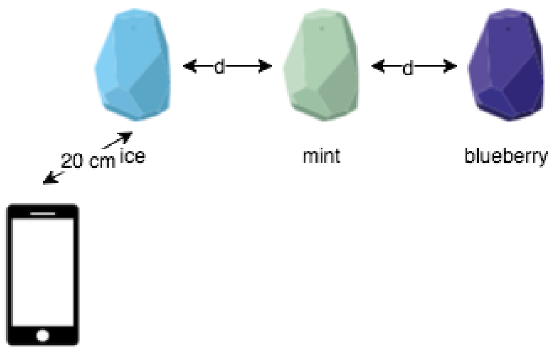

Fig. 2: Layout used for the experiments to get the accuracy of the distance value

accuracy of the mechanism that calculates the distance that the mobile device is from a given beacon.

For each experiment, we counted how many times each beacon was detected as the nearest one. Looking at the layout used in these experiments we can see that the app should detect the beacon named *ice* as the nearest one. Figure 3 shows the percentage of times that the beacon *ice* was detected as the nearest one, showing that, as we increase the distance between beacons, the accuracy to detect the nearest beacon also increases. From the results, we can conclude that it is recommended that the beacons are, at least, 1.5m or 2m distant from each other.

In an environment, where the beacons are close to each other, our solution might not work as expected. For instance, in the previously described Smart Restaurant example, the tables should not be close to each other. This is not always possible because, some restaurants try to optimize space and have tables as much close to each other as possible. In the Smart Museum example, two objects, in a given exhibition, should not be close to each other. Otherwise, the visitor would be notified about an object and he might be looking at another one.

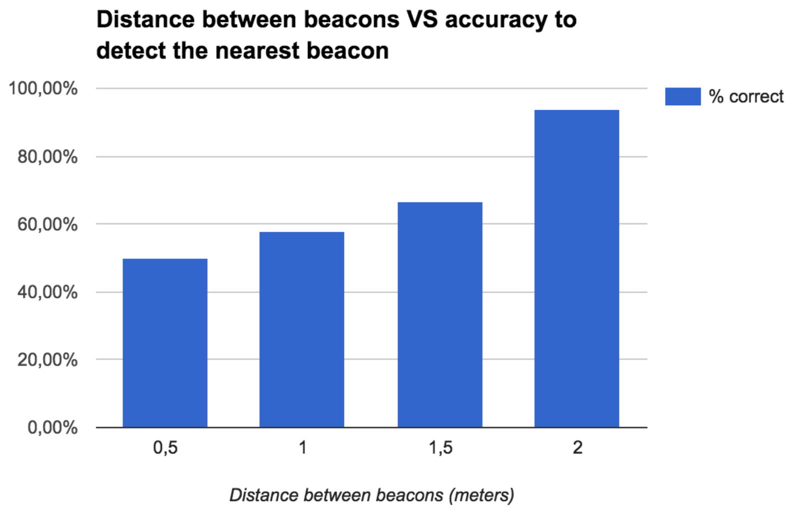

Fig. 3: Relation between beacon distance and detection accuracy

4.2 Energy Consumptions

Another important aspect of this solution is the battery consumption. Since our mobile app for end users runs on background to scan for nearby beacons, that can have a negative impact on the device's battery. If the user notices that the battery drains too fast, he will not use this solution.

Table 2 summarizes the experiments performed to evaluate the battery consumption. Figure 4 shows the layout used for this group of experiments. We have used the same beacons as in the experiments described in section 4.1. The beacons are equally distant 25 cm from each other. The smart-phone is at the same distance from the beacon in the middle, the green one named mint. We performed 4 experiments. In each one the app was turned on and ran for 1 hour in background mode scanning for beacons in order to discover nearby Smart Places. The first two used Wireless Fidelity, Wireless Internet (WiFi) data connection. The remaining used Third Generation (3G) mobile network. Different data connection means can lead to different energy consumptions. We need to understand which connection, WiFi or 3G, drains more power. If it is 3G, the user might only use our solution if he/she is connected to a WiFi Access Point (AP). We want our mobile app to be always turned on scanning for nearby Smart Places. Using it only when WiFi is available would make its usage very limited and the user would not take the full advantage of it because he/she needs be aware that a WiFi AP is available and turn the mobile app on again. We tested two values for the interval between each scan, 5 minutes because it is the default value that the beacons library use in background mode, and 2 and half minutes. The second value is half the first in order to see how much more power is drained when we set a smaller value for the interval between each scan. Trying to find a smaller interval is important because it will reduce the probability that the user was not able to discover a nearby Smart Place. The following scenarios were tested: When the user stays in the same Smart Place the entire experiment; When the user moves from one Smart Place to another. This is done removing existing data about Smart Places already detected to force the app to request this data again from the Back-end in each scan process.

Variables	**Experiments**			
	1	**2**	**3**	**4**
Data connection type	WiFi		3G	
Interval between each scan	2m30s	5m	2m30s	5m
Experiment duration	1h			

Table 2: Summary of experiments to get the battery consumption when the mobile app is scanning for beacons in the background

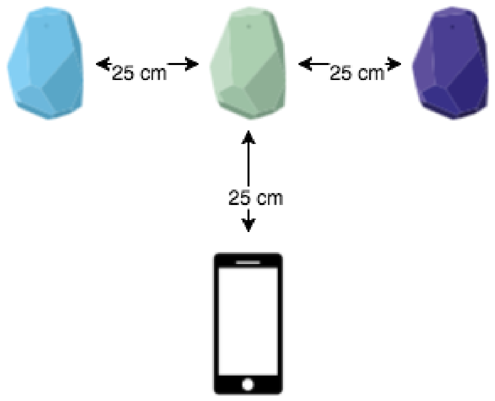

Fig. 4: Layout used for the experiments to get the battery consumption

Data communications, WiFi or 3G, are the major source of battery drain, as suggested in studies, such as [5]. We used Battery Historian[9] to measure the power drain and how much data was transferred (sent and received). Figure 5 shows the results for the first scenario, where the user does not move. WiFi is

[9] https://developer.android.com/tools/performance/batterystats-battery-historian/index.html

not represented in the Figure because no power use was reported for it. However, 3G drained 0.29% and 0,40% of the total battery available. From these results it is possible to conclude that our solution introduces the most overhead when using 3G as the mean to perform data communications.

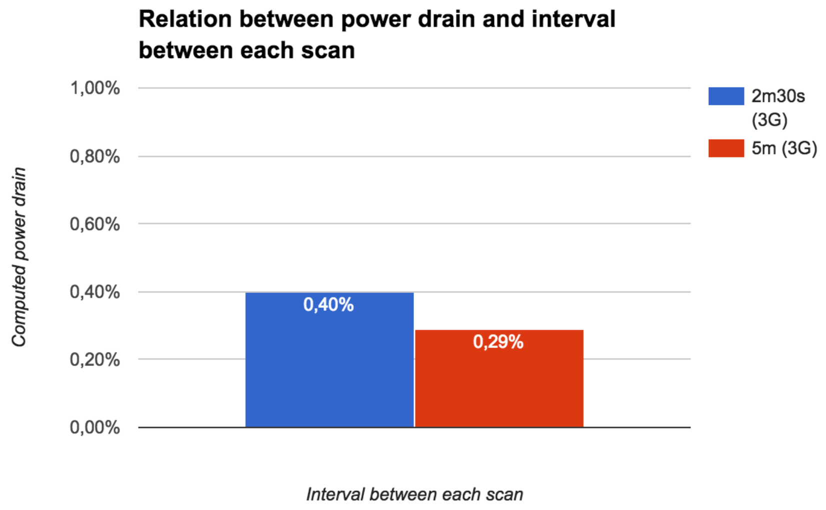

Fig. 5: Relation between power drain and interval between scans when the user stays in the same Smart Place

After the first set of experiments we tested Facebook[10] since it is one of the popular apps and also has services running in background. It is also known to be one of most power draining apps[11]. We let it ran for 1 hour as we did in our first experiments. While running Facebook, we used it to check our news feed in 5 minutes. The rest of the time the app was running in background. Table 3 summarizes the conditions and results of the test performed using Facebook app. We used these values as a reference to compare with the power drain in the second scenario, that is, at each scan the user moves from one place to another which implies more requests to the backend.

Then, we performed the second set of experiments, that is, the second scenario where the user moves from one Smart Place to another. Here, the app requests more data from the back-end. Figure 6 shows the results of these experiments. Here, we got more power drain than in the previous scenario. These results shows that, the more communication with the back-end is required, more power our solution drains. Once more, using WiFi we have got almost zero

[10] http://play.google.com/store/apps/details?id=com.facebook.katana
[11] http://www.forbes.com/sites/jaymcgregor/2014/11/06/facebook-and-instagram-are-killing-your-phones-battery-heres-a-simple-fix

Variables	Values
Data connection type	3G
Experiment duration	1h
Computed power drain (%)	3.77
Data transferred (KB sent and received)	4627.83

Table 3: Battery consumption of Facebook app

power drain. However, similar to the results in the previous scenario, there is more power drain using 3G. Using two minutes and half of interval between each scan, we got more 0.71% than using five minutes. As happened before, there is more power drain when we increase the interval between each scan.

In the worst case we obtained a power drain of 2.71% which is approximately 71% less than using Facebook in the same period of time. Using this app as a reference in terms of apps that run services in background, we can say that the battery consumption is acceptable for a daily basis usage.

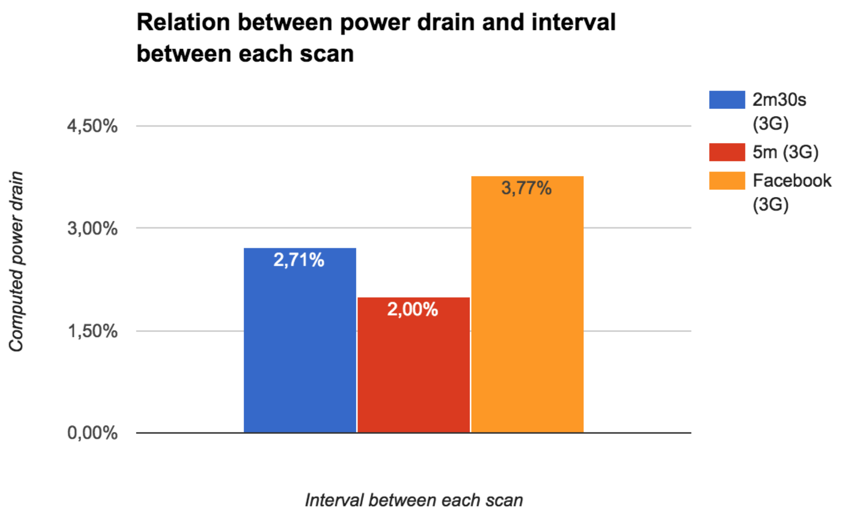

Fig. 6: Relation between power drain and interval between scans when the user moves along multiple places

5 Conclusion

We have developed the Smart Places solution which offers a tool to create proximity-based services and end-user and owner management interfaces. We introduced the concept of Smart Place which is a physical space with tags that mobile devices can detect and offer different possibilities to the users, according to each tag. We chose BLE beacons using iBeacon protocol because it is compatible with any smart-phone with Bluetooth version 4.0 or above. The owners of Smart Places are responsible to place those beacons in the right place. To use the Smart Place, users need only to download a single app and turn on the Bluetooth receivers of their mobile devices.

We have created a solution that allows developers, of Smart Places, that provide proximity-based services, using web technologies, such as, HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and Javascript. With this tool, any web application can react to the presence of tags placed in a given Smart Place. This solution makes the development of Smart Places easier due to the fact that, developers do not need to handle the technology details to handle tags and the back-end where the information about Smart Places and their tags is stored. Each Smart Place is a web application that runs inside an embedded web browser in the mobile app, allowing users to have access to any Smart Place without the need to install a new native app for each one.

Two examples of Smart Places were created, using our solution, a Smart Restaurant and a Smart Museum. The Smart Restaurant had the goal to allow customers, of a restaurant, to place their orders, after they take a sit using their mobile devices. Using this solution, customers do not need to specify which table the orders belong to. Using the tag system through BLE beacons, the app automatically get the table's number and add that information when the customer places the final order. In the Smart Museum we created the experience of a museum where visitors can have access to more information about an object that they are close to.

We evaluated our solution in terms of battery consumption. Users will not use our solution if they run out of battery faster than if they are not using our solution. For each experiment, we have tried two different types of data connection, using WiFi and 3G. Using WiFi the power drain is almost zero. However, using 3G, the power drain was above 2%, in just one hour. Comparing with the power drain of Facebook app running in the same period we can conclude that the power drain is acceptable but there is room for improvement. For instance, we could store geographical coordinates for each tag. When detecting a tag, the app could use these coordinates to fetch data about the other tags in the same area in just one request. This way, we need less requests to the back-end which can result in less power drain than the actual solution.

5.1 Future Work

There are aspects of our work that can be improved in the future. Owners of Smart Places need to deploy tags and use the mobile app to configure those tags.

However, there is not any interface, that they can use, to register themselves, as the owners of such tags. In the development of this work we introduced the needed data manually, that is, all the associations between tags and their owners. There is one more interface missing to allow developers to register the Smart Places they develop. When owners are configuring their Smart Places, they can choose from a list. The Smart Restaurant and Smart Museum examples were introduced in the back-end manually. There is no way for developers to add a Smart Place to this list. A future improvement could be these missing interfaces. One that would allow owners to register the ownership of tags and another for owners that would use it to register the Smart Places they developed. Another limitation is the fact that, using our solution, only web applications can offer proximity-based services. In one side, this is a good fit because this is what allows to have one app to access multiple Smart Places. On the other side, there could be developers that could use our solution to add proximity-based features in their existing native mobile apps. This could be solved having an Software Development Kit (SDK) available for the three most used mobile OSs, iOS, Android and Windows Phone, to allow to integrate our functionality in existing apps.

Proximity-based applications are an important type of context-aware applications since they provide services, to the user, that are relevant in the right place. However, developers need the right tools to create these applications and users need an easy way to have access to such applications. Our work targets not only developers but also users and owners. A complete solution is needed because developers would not create proximity-based applications if there are no users. Also, users would not be engaged if developers do not have the tools to create these applications. Our Smart Places solution and its evaluation can be considered a step forward to be able to use, develop and manage all these and many more possibilities.

References

1. G. Conte, M. De Marchi, A. A. Nacci, V. Rana, and D. Sciuto, “BlueSentinel: a first approach using iBeacon for an energy efficient occupancy detection system,” in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings.* ACM, Nov. 2014, pp. 11–19.
2. V. Antila and J. Polet, “ContextCapture,” in *Proceedings of the 13th international conference on Ubiquitous computing - UbiComp ’11.* New York, New York, USA: ACM Press, Sep. 2011, p. 585.
3. A. Krevl and M. Ciglaric, “A framework for developing distributed location based applications,” in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium.* IEEE, 2006, p. 6 pp.
4. D. Carlson and A. Schrader, “Dynamix: An open plug-and-play context framework for Android,” in *2012 3rd IEEE International Conference on the Internet of Things.* IEEE, Oct. 2012, pp. 151–158.
5. A. Pathak, Y. C. Hu, and M. Zhang, “Where is the energy spent inside my app?” in *Proceedings of the 7th ACM european conference on Computer Systems - EuroSys ’12.* New York, New York, USA: ACM Press, Apr. 2012, p. 29.