Towards a Framework for Proximity-based Hybrid Mobile Applications

Valerio Panzica La Manna, Frank Pasveer
IMEC, The Netherlands
Eindhoven, The Netherlands
{valerio.panzicalamanna|frank.pasveer}@imec-nl.nl

ABSTRACT

Despite the increasing availability of IoT devices and technology, the current user interaction requires users to download dedicated ad-hoc mobile apps, each of these remotely monitoring and controlling a limited set of IoT devices. In the near future, with billions of deployed IoT devices, this approach will not scale since users cannot be expected to download different apps for every place they visit or every device they interact with.

To overcome such limitation, this paper presents the vision, the risks, and opportunities, of the development and deployment of future hybrid proximity services, as a paradigm shift for an intuitive interaction between users and the surrounding IoT environment. Hybrid proximity services are user-facing mobile applications developed and configured for specific locations and automatically installed on user mobile devices when the user is in their proximity and without the need of explicit download. A proximity service enabled environment can directly provide the user with the right service for the right place, and automatically instructs the user device on how to interact with the surrounding IoT devices.

KEYWORDS

proximity services, location-based services, hybrid mobile applications, physical web, IoT

ACM Reference Format:
Valerio Panzica La Manna, Frank Pasveer. 2018. Towards a Framework for Proximity-based Hybrid Mobile Applications. In *MOBILESoft '18: MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3197231.3197261

1 INTRODUCTION

During the last ten years, the introduction of smartphones and mobile apps has radically changed the way end users interact with software. Mobile apps are small pieces of software providing a specific functionality or service (e.g. business, gaming, social networks, messaging) that can be easily downloaded from global app stores, installed, and immediately used. Mobile apps became soon also a way to provide services for specific locations, such as touristic attractions, restaurants, events or retail stores.

Nowadays, we are also experiencing a rise of a new generation of smart connected systems. Machines, buildings, vehicles, personal appliances will all be equipped with small computing devices, sensors and actuators, that will be interconnected, creating together the Internet of Things (IoT). Following the vision of Ubiquitous Computing [13], in the near future the IoT will disruptively reshape the environment around us, and the user can benefit from that by having more useful insights and new services.

This paper addresses the problem of the mobile interaction between end users and the surrounding pervasive IoT-enabled environment. The current paradigm requires the user to search, download, install, and use an app for a specific location or a specific IoT device to interact with. If this approach can work with a limited set of known locations (e.g. home and offices) and IoT devices (e.g. smart watches and TVs), it will not scale in the near future where more and more locations, shops, and buildings publish their own apps, and when billions of IoT devices will be deployed in our environment [4].

At large scale, the current app paradigm suffers from two main limitations: (i) *discoverability*: users need to be aware in advance of the presence of a service in a specific location and need to know the type and brand of an IoT device before downloading the corresponding specific app; (ii) *usability*: most of these services are transient in nature, usable only at specific location, and do not require the user to store (and then immediately remove) an app if it is used only once and possibly for just a few minutes.

The problems above, and the need of *just-in-time-interaction* has been previously described by Jenson in [8, 9] and lays the foundation of the Google Physical Web project [6]. The project proposes the use of Bluetooth Low Energy (BLE) beacons to advertise public URLs. The Physical Web feature, integrated in the Chrome browser, allows the user to discover nearby Physical Web beacons and open the corresponding web page. However, web pages do not have full access to the underlying features offered by the smartphones (i.e. sensors, and network interfaces), and this can limit the types of supported user interaction and IoT control they can offer.

To address this problem, in this paper we present a vision for a next generation of mobile applications, called *hybrid proximity services (HPSs)*, to support the scale of the future user interaction with IoT-enabled environments. The term hybrid indicates both the underlying technology adopted (a hybrid mobile application framework), and the type of proposed solution, which combines and extends the concept of the physical web with native functionality. With the introduction of HPSs: (i) users will be able to directly access the desired services provided by the surrounding IoT-enabled

MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5712-8/18/05…$15.00
https://doi.org/10.1145/3197231.3197261

environment without downloading a specific app. The service is transient and is automatically removed when the user leaves the specific location; (ii) authorized users can discover IoT devices in their proximity and directly control them; (iii) the immediate interaction between IoT devices and users creates a richer context for customized services.

2 VISION

In this section we first introduce the vision of HPSs with a running example. We then describe the main elements of the overall supporting system.

2.1 Running Example

As an example, consider the design of a mobile application supporting technical operators in the installation, commissioning and maintenance of different air quality sensors in smart buildings. Air quality sensors differ in the type, producer, and capabilities. The building owner selects the types and number of sensors and purchases them from different vendors. He/she then contacts a technical operator to install the purchased infrastructure.

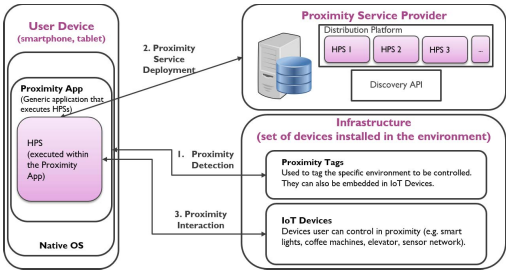
Following the current app paradigm, the technical operator, who does not have a-priori knowledge of the types of devices to install, once on-site, needs to search and download vendor-specific apps for each type of sensors, and use it singularly to correctly install them. When the number of different sensors is large, this process may require a considerable amount of time, especially if we consider that this process needs to be continuously repeated for each on-site installation the operator needs to perform, and for each other different operator who needs to provide maintenance. A more efficient installation of the sensors can be achieved when applying the paradigm of hybrid proximity services.

HPSs are user-facing hybrid mobile applications developed for a specific place, and configured for the available IoT devices. HPSs are automatically discovered and installed on user mobile devices when the user is at a specific location and without the need of explicit download. A HPS enabled environment can directly provide all the functionality a specific place offers and automatically instruct the user device on how to interact with the IoT devices in proximity.

Let's now consider the application of the new paradigm in the smart building example. With this paradigm, each sensor vendor develops an installation and maintenance HPS for each produced sensor type. The HPS of a sensor is a hybrid mobile application which specifies the interaction the technical operator is expected to use to properly configure it, install it, and maintain it. It includes the desired user interface the technical operator is expected to use, developed in HTML5, CSS3, and JavaScript, and the business logic for the installation and maintenance of the sensor, in JavaScript, which invokes standard hybrid native plugins to interact with the underlying native layer (e.g. the business logic can specify the Bluetooth characteristics to read and write to properly configure the device). Note that the development effort for the sensor vendor is a subset of the one required to develop a full native app and it is performed only once during the production of the sensor.

With the new paradigm, the technical operator is now able to more quickly complete the installation procedure. By using a generic installation-maintenance *proximity app*, he/she is able to automatically discover all the available HPSs in the surrounding environment. Each HPS is then automatically deployed in the proximity app and being able to configure the sensor as specified by the vendor. The operator can even speed up the future maintenance tasks for the future operator. He/she can create an additional HPS for each room the sensors are installed by combining the single sensor HPS into a mashup, which can be discovered and automatically executed when the maintenance operator visits the room.

Figure 1: System Overview



2.2 System Overview

Figure 1 shows an overview of the overall system, which consists of three main components: the infrastructure, the proximity app, and the proximity service provider.

The *infrastructure* is the set of devices characterizing the IoT-enabled environment to be controlled by HPSs. It includes the *IoT devices* the users can control in proximity (e.g. sensors, smart lights, coffee machines, elevators, parking systems, train ticket system), and the *proximity tags* which is the set of different enabling technologies (e.g. BLE beacons, ultrasound, telecom link for wide range proximity and localization, NFC...) used to tag identifiable specific location in the environment. Proximity tags are functional components which can be separately deployed in the environment (e.g. in the room of a smart building) or they can be directly embedded in the IoT devices (as in the sensors of the example).

The *proximity app* is a generic application, previously downloaded and installed on the user device acting as a container of the currently deployed HPS. The deployment of HPS into the proximity app is triggered when the user device is in the proximity of proximity tags placed and distributed in the physical environment. As a result, the proximity app will be able to adapt its behavior on the basis of the specific user context by executing the right proximity service at a specific location.

The *proximity service provider* offers additional support for the discovery, storage, and deployment of HPSs. It contains a repository of the available HPSs with metadata supporting their distribution, and a discovery API mapping proximity tags to corresponding HPSs. It can be deployed locally on a gateway and/or in the Cloud. Its presence may be optional for pure P2P interactions.

We can identify two main phases in the deployment of a HPS-based system: the configuration phase, and the operation phase.

The configuration phase requires the definition of the infrastructure, the presence/absence and deployment strategy of the proximity service provider, and the implementation of the HPSs for the devices and/or environments to be controlled. This phase is

likely to be distributed in time and responsibilities. In the example, the implementation of the sensors HPSs is provided by the individual vendors, it is a pre-requisite for the initial installation, and is fully embedded in each sensor to be retrieved in P2P. On the other hand, the definition of the additional room-level mashup HPSs, is introduced by another party (the technical operator), and installed on a gateway while the system is already in operation. This dynamic scenario, where new experiences can be created, removed, and modified without any changes required on the user side, is the key differentiating element in adopting this new paradigm. The current paradigm would instead require off-line software updates and explicit download of new apps.

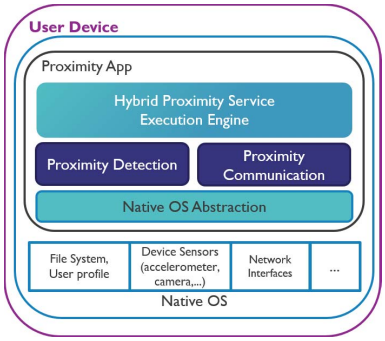
During the operation phase, the end user can benefit from the new services offered by the HPS-based system. In this phase, the main role is played by the proximity app, installed on the user device. The proximity app is an instance of a generic *proximity-based hybrid mobile application framework*. An architectural view of the framework is shown in Figure 2. The arrows in Figure 1 indicates the three main sequential steps of this phase.

1. Proximity Detection: the proximity app initiates the discovery of nearby proximity tags. This is performed by a background processing of the proximity detection layer of the framework. When no HPS are currently in execution, the default behavior of the framework is to continuously scan for nearby proximity tags. The proximity detection layer offers an API to the running HPS to pause and start the proximity scan. This is useful when the currently running application (or the user) is asking to perform a task that cannot be stopped by the detection of another proximity tag, and therefore by an eventual deployment of another HPS. If this is not the case, the proximity detection layer continuously monitors whether the current HPS is still in proximity and eventually triggers an event to remove it or replace it. Finally, the proximity detection layer abstracts away from the OS-specific technology used to perform the proximity detection (BLE, NFC, ...). This enables different scenarios spanning from large to small proximity range detection.

2. Proximity Service Deployment: The detection of a proximity tag triggers the deployment of a new HPS. The typical deployment approach relies on the presence of the proximity service provider, which can be discovered in the same local network in which the proximity app is connected, or available remotely in the Cloud. In both cases, a HPS identifier is resolved from the discovered proximity tag, and automatically downloaded by the framework, that is also responsible to eventually replace the currently executing service. In the case of P2P interaction, no service provider is available, the proximity tag itself identifies the HPS to deploy. A P2P connection is then established between the IoT device and the proximity app to receive and deploy the corresponding HPS.

3. Proximity Interaction: Once the HPS is deployed, the proximity interaction can be enabled. How this interaction is established, which interface to use, and which information to exchange is fully specified by the business logic of the HPS, which, on the basis of common shared native plugins, instructs the proximity app how to interact with the device (and how this interaction is visualized on the user screen). A proximity communication layer is then responsible for initiating the communication between the two devices independently of the available network interface.

Figure 2: Proximity-based Hybrid Mobile Application Framework



3 WHY IS IT NEW?

The idea of delivering services on the basis of user proximity (proximity services) or location (location-based services) is not new and it has been previously introduced in different domains, such as the 3GPP standardization [11], social interaction [3, 10], and media content delivery [12]. Differently from previous approaches, the notion of hybrid proximity services introduced in this paper focuses on automatically deployable full-featured mobile applications.

The vision of a framework that enables the development and deployment of hybrid proximity-based services builds on top of existing solutions and introduces a novel approach both from a technical perspective, and for the way end-users interact with IoT.

From a technical perspective, the proposed framework relies on the ideas and technical solutions introduced by the JavaScript-based hybrid mobile applications framework such as Apache Cordova [2], Ionic [7], and Adobe PhoneGap [1]. These frameworks separate the application-specific logic and look-and-feel of an app, developed as a web application using HTML5, CSS3 and JavaScript/TypeScript, from the application-independent interfaces that provide access to the underlying native layer of the different mobile operating systems. The basic architecture proposes to encapsulate the application-specific part into a web-view container, which can access a JavaScript abstraction layer, called *native plugins*, to benefit from the native features of the platform.

The framework we are proposing, extends the above architecture by adding two additional intermediate layers to support the proximity discovery and communication. It also adapts the web-view container to automatically download, deploy, and execute the HPSs, and is fully compatible with the Cordova native plugins. In addition to the technical differences, the novelty introduced by the framework is in the way it is used. The existing hybrid frameworks has the final goal of building specific apps (including the application logic) that are ready to be distributed in the app stores. In the vision we propose, the proximity app is instead a generic container which can dynamically load, interpret, and unload the specific web-based applications at run time, on the basis of the detected proximity and required metadata. This feature enables the possibility of just-in-time interaction with the surrounding environment.

As described in the Introduction, another approach is the one offered by the Physical Web where web pages can be discovered

in proximity and visualized in the browser. However, the current limitations of the browser limit the full interaction with the IoT devices. The hybrid solution we propose combines the advantage of the native close environment with the openness of the web.

The Android Instant Apps [5], is another promising project the allows users to share a link redirecting to a fragment of Android apps without the need of downloading it. It can be seen as a preview of an app informing the user what to expect before downloading the full version. If combined with the Physical Web, it may provide a similar functionality to the one offered by our framework. However, our framework supports an efficient remote deployment of full HPSs (and not fragments), which is achieved by incrementally downloading the service on the basis of the user interaction. Moreover, by relying on hybrid framework, the HPSs support multiple native platforms.

4 RISKS

The vision we proposed aims at drastically changing the current way we develop, install, and interact with mobile apps. This vision comes with associated risks and research challenges that need to be addressed for its full realization.

First, the full development of hybrid proximity services requires the creation of a *new ecosystem* covering the entire value chain of industries adopting the new paradigm. IoT manufacturers should develop HPSs and embed them in their devices to enable a proximity-based interaction. This is technically feasible given the possibility of including a micro web-server in embedded devices. For battery-powered devices, it would be interesting to further investigate the impact on energy consumption introduced by the developed services. A key role in the value chain is played by the proximity service providers, who have the role to distribute the HPSs. We envision the creation of HPS app stores which complement the existing ones. A similar business model to the one currently in place for traditional app stores could then be applied.

From a user perspective it is important to investigate the security and privacy risks the adoption of the framework can introduce.

The key feature of the proximity app is the capability to host and execute arbitrary HPSs based on proximity and providing access to the underlying native layer. This feature opens potential security threats that need to be addressed. When the system includes a trusted proximity service provider, similarly as it happens for the current app stores, it can require each HPS to pass a security review before being distributed. Its role can also be used to perform mediated proximity discovery that filter out all the services that are not registered or approved. Different security risks need to be addressed in the context of P2P IoT interaction where the role of the proximity service provider is lacking. In this case, the proximity app needs to be able to detect malicious proximity tags and services.

Hybrid proximity services, as all applications that use proximity detection technology, are open to privacy risks that need to be addressed. The proximity detection technology may enable tracking of the user location. Moreover, privacy-sensitive data could be transferred to external parties through HPSs. Also in this case, a proximity service provider can help in enforcing clear privacy protection rules and user configuration. On the other hand, the constant presence of a third party provider, which continuously monitors all the proximity interaction of the user, can also result into user tracking.

Finally, the introduction of the framework changes the paradigm also in terms of *user control*. The traditional approach requires the user to actively select the desired app to download. In the paradigm we propose, this role is left to the space owner who configures the IoT space with the services he/she expects the end user would need or want. However, this switch of control may enable the discovery of services the user does not want. Also in this case, the presence of the service provider can help in tagging and filter out unwanted services, and in ranking the preferred ones.

5 NEXT STEPS

In this paper, we present a vision of a future generation of services that aims at optimizing, simplifying, and enhancing the user interaction with the surrounding environment. We are currently working on a reference implementation of the framework where we can test our assumptions and extend it to address the security, privacy, and user control challenges presented in Section 4.

Moreover, we are interested in investigating the introduction of new languages and models for the design of complex proximity interaction. As described in the example of Section 2, HPSs can be created by the mashup of existing services. Complex interaction can also be created using flow-based models and languages, which can model and analyze when and how to perform the automatic switching between proximity services.

Finally, it is important to investigate new testing and debugging techniques that consider the proximity detection, proximity service deployment, and proximity interaction as first class objects.

REFERENCES

[1] Adobe. 2018. PhoneGap. (2018). Retrieved January 21, 2018 from https://phonegap.com/

[2] Apache. 2018. Apache Cordova. (2018). Retrieved January 21, 2018 from https://cordova.apache.org/

[3] L. Baresi, L. W. Goix, S. Guinea, V. Panzica La Manna, J. Aliprandi, and D. Archetti. 2015. SPF: A Middleware for Social Interaction in Mobile Proximity Environments. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 79–88. https://doi.org/10.1109/ICSE.2015.137

[4] Gartner. 2017. Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016. (Feb. 2017). Retrieved January 21, 2018 from https://www.gartner.com/newsroom/id/3598917

[5] Google. 2017. Android Instant Apps, Native Android apps, without the installation. (2017). Retrieved January 21, 2018 from https://developer.android.com/topic/instant-apps/index.html

[6] Google. 2017. The Physical Web. (2017). Retrieved January 21, 2018 from https://google.github.io/physical-web/

[7] Ionic. 2018. Ionic Framework. (2018). Retrieved January 21, 2018 from https://ionicframework.com/

[8] Scott Jenson. 2011. Mobile App Must Die. (Sept. 2011). Retrieved January 21, 2018 from https://jenson.org/mobile-apps-must-die/

[9] Scott Jenson. 2012. Triumph of the Mundane. (April 2012). Retrieved January 21, 2018 from https://jenson.org/triumph-of-the-mundane/

[10] Chieh-Jan Mike Liang, Haozhun Jin, Yang Yang, Li Zhang, and Feng Zhao. 2013. Crossroads: A framework for developing proximity-based social interactions. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 168–180.

[11] X. Lin, J. G. Andrews, A. Ghosh, and R. Ratasuk. 2014. An overview of 3GPP device-to-device proximity services. *IEEE Communications Magazine* 52, 4 (April 2014), 40–48. https://doi.org/10.1109/MCOM.2014.6807945

[12] A. Miyamoto, V. Panzica La Manna, and V. M. Bove. 2016. Live objects: A system for infrastructure-less location-based services. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. 1–3. https://doi.org/10.1109/PERCOMW.2016.7457068

[13] M. Weiser. 1993. Hot topics-ubiquitous computing. *Computer* 26, 10 (Oct 1993), 71–72. https://doi.org/10.1109/2.237456