Smart Places

A framework to develop proximity-based mobile applications

Samuel Filipe Capucho Mendes Coelho

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors:
Dr. Miguel Filipe Leitão Pardal
Dr. José Manuel da Costa Alves Marques

November 2015

Abstract

Proximity-based applications engage users while they are in the proximity of points of interest. These apps are becoming popular among the users of mobile devices. Proximity-based apps are triggered when the user is at specific geographic coordinates, but more interestingly, they can be triggered by tagged objects. A Smart Place is a physical place with tags that provide a service. Users with a mobile device capable of detecting such tags can use the provided service when they are in proximity of these tags. Several technologies can be used to create tags.

In this dissertation, we chose bluetooth low energy beacons following the iBeacon protocol to tag the points of interest. We have created a solution to develop proximity-based applications, based on this concept. Also, two examples of Smart Places were built: the Smart Restaurant and Smart Museum. The Smart Places apps allows anyone with a mobile device capable of detecting tags, to have access to any proximity-based service, based on our concept of Smart Place and created using the tool we developed. This app was evaluated in terms of energy consumption and the results show that the battery drain can be acceptable making it a viable technical approach.

Keywords: proximity-based; mobile apps; smart place; location based applications; bluetooth low energy beacons

Resumo

Aplicações baseadas em proximidade captam os utilizadores enquanto estes se encontram na proximidade de pontos de interesse. Estas *apps* têm vindo a tornar-se populares entre os utilizadores de dispositivos móveis. *Apps* baseadas em proximidade podem ser despoletadas em coordenadas geográficas específicas, porém, a possibilidade de poderem ser activadas com recurso a objectos com *tags*, torna este tipo de aplicações mais interessantes. Um *Smart Place* é um lugar fisico onde *tags* são colocadas para fornecer um serviço baseado em proximidade. Utilizadores com um dispositivo móvel capaz de detectar *tags* podem utilizar o serviço fornecido quando se encontram na proximidade das referidas *tags*. Diversas tecnologias podem user usadas para a criação de *tags*.

Nesta dissertação, utilizámos *beacons bluetooth low energy* com o protocolo *ibeacon* para colocar etiquetas nos pontos de interesse. Criámos uma solução para desenvolver aplicações baseadas em proximidade, seguindo esta definição. Também foram desenvolvidos dois exemplos de *Smart Places*: o *Smart Restaurant* e *Smart Museum*. As aplicações de *Smart Places* permitem a qualquer um com um dispositivo móvel capaz de detectar *tags* aceder a qualquer serviço baseado em proximidade, baseado no conceito de *Smart Place* e criado com a nossa solução. Foi avaliado o consumo energético desta aplicação e os resultados demonstram que a descarga de bateria pode ser aceitável tornando-a uma abordagem tecnicamente viável.

Palavras-Chave: proximidade; aplicações móveis; smart place; aplicações baseadas em localização; beacons bluetooth low energy

Contents

List of Tables

List of Figures

List of Listings

Acronyms

3G	Third Generation. 40–43, 46
4G	Fourth Generation. 46
AP	Access Point. 9, 40
API	Application Programming Interface. iv, 8, 17–20, 22, 26, 28–31, 33–37, 46
BaaS	Backend as a Service. 30, 32–34, 36
BLE	Bluetooth Low Energy. 9, 11–15, 17, 18, 23, 30, 31, 36, 45, 47
BP	Beacon Point. 13
CPU	Central Processing Unit. 1, 37
CSS	Cascading Style Sheets. 14, 28, 31, 35, 36, 45
GB	Gigabyte. 37
GPS	Global Positioning System. iii, 1, 7–9, 11, 12, 14, 15
GPU	Graphics Processing Unit. 1, 37
HTML	HyperText Markup Language. 14, 28, 31, 35, 36, 45
HTTP	Hypertext Transfer Protocol. 36
ID	Identifier. 10
IDE	Integrated Development Environment. 31
IFTTT	If This Then That. 1
IST	Instituto Superior Técnico. 35
JS	Javascript. 26
JSON	JavaScript Object Notation. 15, 23–25, 31, 32, 35
MAC	Media Access Control. 13
mAh	Milliampere-hour. 37
NFC	Near Field Communication. 9, 11, 12, 15, 47

OS	Operating System. 37, 47
PaaS	Platform as a Service. 36
PHP	Hypertext Preprocessor. 35
QR	Quick Response. 8, 9, 11, 12, 15, 47
RAM	Random-access Memory. 37
REST	Representational State Transfer. 15, 35, 36
SDK	Software Development Kit. 19, 29–34, 39, 47
SIG	Special Interest Group. 9
SOAP	Simple Object Access Protocol. 14, 15
UI	User Interface. 18, 19, 26, 30, 34
URL	Uniform Resource Locator. 8, 11, 19, 24, 30, 31
USB	Universal Serial Bus. 13
UUID	Universally Unique Identifier. 10, 23, 24, 31, 32
VCS	Version Control System. 29
WiFi	Wireless Fidelity, Wireless Internet. 9, 14, 40–43, 46
XML	Extensible Markup Language. 15

Chapter 1

Introduction

The use of mobile devices has increased in the last years. It is now possible to use devices such as smartphones and tablets to perform tasks that previously were only possible using a desktop computer. These devices are becoming more capable than ever before. They have powerful Central Processing Units (CPUs) and Graphics Processing Units (GPUs) and are also equipped with many sensors such as light sensor Global Positioning System (GPS) and accelerometer. Also, these devices have access to multiple data sources such as the user's activity on social networks and calendar. 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 mobile apps that use this information, which is called *context*, are named context-aware applications. For instance, an app that puts the phone in silent mode when the user is in a meeting is a context-aware app. The user's location is a particularly useful context information because it allows apps to offer the user special functionalities when they are in a specific place. The focus of this present work is location based applications.

1.1 Motivation

Proximity-based is a particular type of context-aware that takes into account users' location. These apps offer different possibilities, to the users when they are in the proximity of a given point of interest. We can find multiple examples of such apps. Swarm[1] allows users to check-in in a given place or point of interest according to user's location. For instance, if the user is in a restaurant he can use the app to perform a check-in in that restaurant and share his location on social networks. Skout[2] is another example app that allows finding nearby users and start talking to them. These apps are becoming popular. Besides these examples, it is possible to automate tasks based on location. For instance, send a message to someone when we arrive at a given location or turn on the lights when we arrive at home. Services such as "If This Then That" (IFTTT)[3] allow users to make this kind of automations. These are just examples of what is possible to do with context-aware and more specifically with location based applications.

To develop proximity-based apps we need to, somehow, get the device's location. Also, we need to associate data to points of interest. This data can be provided by the application itself or by a central server. Developers have to develop the mobile apps and integrate in those apps the technology needed to get the proximity of the user to a point of interest.

[1] http://www.swarmapp.com
[2] http://www.skout.com
[3] http://ifttt.com

Besides development of proximity-based services another question arrises. How can an owner of a given place offer such services without having to develop everything him/herself? For instance, a store owner wants to advertise some promotion in the customers mobile devices when they approach the store. How can he/she creates this promotion if there is no programming background?

Taking into account the previous store example, if there are multiple promotions in different stores, customers need to install one app for each store. They should be able, by installing one app, to discover these promotions or any other proximity-based service while they are walking instead of being aware of these services and know which apps they need to install.

1.2 Goals

In order to create proximity-based services available on mobile devices, developers need to build the mobile apps, choose the right technology for location and build the backend to maintain the data associated to each tag. Our main goal in this dissertation, is to create a tool to develop such services. With this tool, developers should be able to build the proximity-based services, without the need to develop their own backends and handle the location technology themselves.

In this dissertation, we want to introduce the concept of Smart Place, which is described in further detail in the next chapter. The idea is to have a physical place, which offers a proximity-based service due to existence of tags placed on objects that are relevant. For instance, the example of the store can be considered a Smart Place. The owner place some tags, inside the store, that will trigger a notification in the customers' mobile devices, such as smartphones.

Developers are not the only category of people we aim to target. Users should be able to access these services without having to install, on their mobile devices, one app for each service they want to use. In the store example, users should be able to have access to the advertised promotions without the need of one app for each store. One app should be enough, for anybody with a mobile device, such as, a smartphone, to use any proximity-based service, that is, a Smart Place where owners placed tags with useful meaning.

1.3 Contributions

We aimed to contribute in several ways. We created a tool to develop proximity-based applications. There are other tools that try to solve the same problem. We analyzed them in order to understand their advantages and limitations and built our own solution, which tries to make developers focus on their applications and not on the technology side, that is, the technology used to provide tags and make mobile apps being able to detect them. We evaluated our solution in order to conclude how much overhead, in terms of energy consumption, is introduced by a service running in background that reacts to the presence of tags.

1.4 Thesis Outline

The rest of this dissertation is organized as follows:

Chapter 2 (Background) provides background information that is needed to understand the problem this dissertation tried to solve and the solution that is proposed;

Chapter 3 (Solution) introduces the solution and describes its main components;

Chapter 4 (Implementation) presents the implementation process, the solution's architecture and all the tools used to create it;

Chapter 5 (Evaluation) shows the results from the evaluation process in order to get a good insight about the quality of the solution;

Chapter 6 (Conclusion) concludes this dissertation and states the what can be done in the future to improve this solution.

Chapter 2

Background

Before describing the solution we need to get a good insight about concepts, technologies available and related work. Proximity-based applications are a particular kind of context-aware applications. An application is context-aware when it takes into account the context such as, location, device's orientation, temperature, etc. Based on this definition, proximity-based applications are context-aware applications that take into account the user's location. They engage the user while they are on proximity of a given tag. A tag is a mark with meaning to a given application. Someone installs tags in a given space. Then, users can interact with those tags when they are nearby them. From the given definition of proximity-based application, the concept of Smart Place arises.

Our solution is based on the concept of Smart Place, which we define, in further detail, below, in section 2.1. We are going to look at a taxonomy, described in section 2.2, that allows us to classify the multiple kinds of location to justify our decisions in terms of solution and implementation. Then, we explore some technologies that could be used in our solution, in section 2.3. In section 2.4 we describe related work providing motivation for the development of our Smart Places solution. Section 2.5 summarizes the most important ideas in this chapter.

2.1 Smart Places

A Smart Place is a physical place with tags, which users can interact with using a mobile device such as, a smartphone. A place's owner installs tags and those tags offer some service to the users when they are nearby. For instance, a store owner could use these tags to advertise a promotion when the customers are nearby the store. However, Smart Places are not just for stores. This concept can be used to any kind of service that benefits from knowing the users to be nearby tags. For instance, it is possible to build a Smart Restaurant where tags have the information about the table's number and the customers are allowed to call a waiter using their mobile devices, without requiring to type the table's number.

Multiple kinds of people are involved in Smart Places. There are owners, developers and end users. Figure 2.1 shows how these three kinds of users interact with a Smart Place. Owners are responsible for managing and installing tags in their places where they want to offer a proximity-based service. Also, developers are needed to develop these services, for instance, the Smart Restaurant or the store promotions. Finally, the end users that interact with tags that are installed in Smart Places, using their mobile devices.

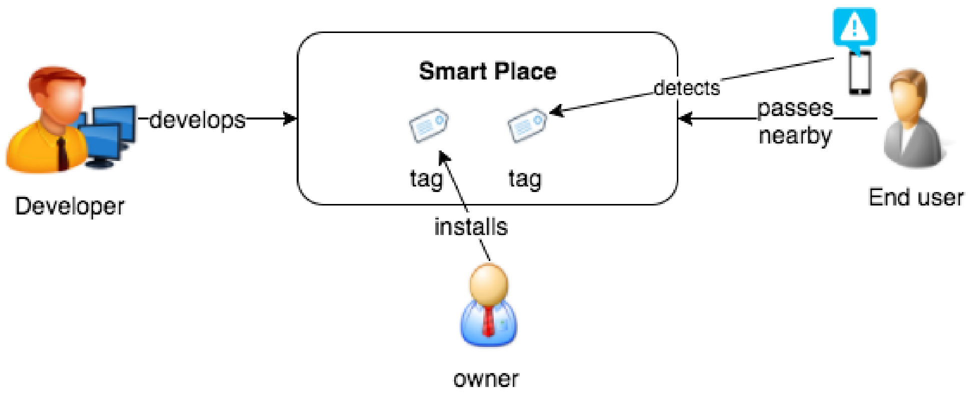


Figure 2.1: Smart Places Overview

2.2 Location

In order to better define what a Smart Place is, we need to have a good insight about concepts related to location. We have used a taxonomy[1] to classify some properties about location which will allow us to have a better understanding about the concept of Smart Place, in section 2.1. Also, these properties are needed to be able to classify the technologies, described in section 2.3 and justify if they can be used in the implementation of the concept of a Smart Place. Using the taxonomy previously mentioned, it is possible to classify location systems in terms of techniques, physical position or symbolic location, absolute or relative position, location computation, scale, cost and limitations.

2.2.1 Techniques

There are three techniques that can be used to get the device's location. These techniques are used to compute the location. A device can use one or combine two or all the three. The techniques are the following:

Triangulation It can be done via lateration or angulation. Lateration uses distance measurements between three well known points. Angulation uses measurements of angles relative to known points;

Proximity It takes into account the proximity of the user to a given point

Scene analysis It consists in examining a view from a given point.

2.2.2 Physical or Symbolic Position

The device's position can be classified in one of the two types:

Physical position This kind of position refers to a set of coordinates that identifies, unequivocally, a place on Earth where the device is. From this set of coordinates we know exactly where the object is;

Symbolic location Unlike physical location, using this kind of positioning it is not possible to identify where the object is. Symbolic can be, for instance, the object is in the kitchen. The meaning of its position depends on the application.

One location system can only be classified in one of the two kinds of positioning. However, physical position can be augmented to also have symbolic information. For instance, we can have a system that stores coordinates and for each set of coordinates we store symbolic information. A place on Earth could have symbolic information associated.

2.2.3 Computation

We have multiple ways of obtaining data about location and multiple kinds of location. However, this data needs to be computed in order to get meaningful information about the object we want to locate. This computation can be done in one of the two ways:

Located object computes its own location which means that the located object computes its location;

Location is computed by external infrastructure meaning that the located object delegates its location computation to an external infrastructure, for instance, a central server.

For instance, GPS receivers compute their own location based on signals that come from satellites. In location systems based on tags, the location's computation is delegated to another machine.

2.2.4 Scale

The scale of a location system refers to the number of objects that is possible to locate using a certain amount of infrastructure or over a given period of time. For instance, in systems that rely on a fixed number of sattelites, it is possible to serve an unlimited number of receivers. In systems based on tags, a reader can read a limited number of tags. In this case, adding more tags can compromise the performance of the entire system.

2.2.5 Cost

There are costs associated to any location system. It is possible to look at costs in three perspectives:

Time costs Any location system needs time to be spent in installation, administration and other tasks related with its maintenance;

Space costs There is always infrastructure associated with any location system. This infrastructure needs space. For instance, if servers are needed, we need to install them in a room;

Capital costs The processes and infrastructure associated to a location system requires capital. Each mobile unit or infrastructure element has its price. Also, there is people involved. Their salaries are also a capital cost that needs to be taken into consideration.

When comparing multiple location systems, the costs can be a decisive factor. We need not only to take into account the technology characteristics but also the costs.

2.3 Location Detection

Somehow, the mobile device needs to be able to detect the presence of tags that belong to a given Smart Place. Multiple technologies can be used. Some of them require the user interaction, such as the described in section 2.3.2. Others require the devices to be equipped with extra hardware, such as the one in section 2.3.3. We are going to look at these technologies and see which one best fits our purpose, using the taxonomy already introduced.

2.3.1 GPS

Global Positioning System (GPS)[2] is a location system that uses 24 sattelites plus 3 backups. Receivers send signals and sattelites answer back. Measurements are taken from this signals in order to receivers be able to calculate their own location. An object, that we want to locate, only needs a GPS receiver in order to to determine its location using this technology.

GPS uses triangulation as the technique to get the location. It computes physical location, that is, when an object is located we can look at a map and see where it is. The located objects have means to compute their own location based on measurements from the satellites' signals. It is a scalable system because the same number of satelites can handle an unlimited number of GPS receivers. The satelites are not able to recognize individual receivers. In this system, the major cost is in launching and maintaining the satellites It cost 12 billion dollars to put them in orbit. The annual operating cost is 750 million dollars[1]. One limitation of GPS is that, it does not work well indoors because the satellite's signal can be weak.

Objects to be located using GPS only require an adequate receiver. Most of the mobile devices nowadays are already equipped with these receivers. This can be a big advantage because it is a low cost solution for users. However, it is not adequate to locate objects indoors.

2.3.2 QR Codes

Quick Response (QR) codes is a type of two dimensional barcode. The user just needs a camera and software that read these codes. There are QR readers available, such as, QR Droid[2] for Android, QR Reader[3] for iOS and QR Code Reader[4] for Windows Phone. Whenever the user sees one of these codes, he/she can open the QR code reader app, scan the code and see the content provided by it. The content can be an Uniform Resource Locator (URL). Figure 2.2 shows an example of a QR code.

Figure 2.2: Example of a QR Code

There is no extra hardware involved. However, one big disadvantage of this technology is the fact that it requires user interaction. The user needs to be aware of this kind of codes and also needs to see them wherever they are. Providing proximity-based services using QR codes would be simply using the codes as tags and an app for users to scan these codes and have access to the particular service, that is, Smart Place.

There are several applications that use this technology to to get the user's indoor location, such as, the one described in [3]. It uses QR codes in combination with Google™ Maps API[5]. Others, try to use

[1]Source: http://nation.time.com/2012/05/21/how-much-does-gps-cost at 24, December, 2015
[2]http://play.google.com/store/apps/details?id=la.droid.qr
[3]http://itunes.apple.com/pt/app/qr-reader-for-iphone/id368494609?mt=8
[4]http://www.microsoft.com/en-us/store/apps/qr-code-reader/9wzdncrfj1s9
[5]http://developers.google.com/maps

it to solve real world problems such as hospital overcrowding. In the work presented in [4], the authors try to use QR Codes to identify patients in an hospital. This information is used in a mobile application that the hospital's staff use to register activities related to the patient.

2.3.3 NFC

Near Field Communication (NFC)[5] is a short distance radio communication technology. Its range is less than 10 cm. It can work in one of two modes

Active mode. In this mode, both devices generate their own electromagnetic field alternatively to exchange information;

Passive mode. Here, one device generates an electromagnetic field and the other device uses that same field for data transmission.

Simillary to QR Codes, this technology requires the user interaction and his/her awareness of the existence of these kinds of tags. Also, devices need to be equipped with Near Field Communication (NFC) readers. This technology is already being used for payments, such as Apple™ Pay[6].

2.3.4 WiFi Signal Mapping

Wireless Fidelity, Wireless Internet (WiFi) can be used as a location detection technology. WiFi fingerprinting[6] uses the Access Points (APs) signal's strength to locate the receiver inside a building. Google Maps Indoor is an example of an application that uses this technique to locate the user indoors.

Google Maps allows users to navigate all over the world and gives access to satellite images. There are mobile apps for Android and iOS. With more than 1×10^9 installs on Google Play Store and an average rating of 4.3[7], we can say it is a very popular and mainstream app. When installed on a smartphone, it uses multiple sensors such as, GPS and accelerometer, in order to get the user's location. However, since it relies on GPS it does not work well indoors. Google Maps Indoor[8] is an extension which allows the user to navigate inside a building. Figure 2.3 shows an example of Google Maps Indoor.

This service uses sensors already provided by the mobile device and it does not require extra sensors inside the building.

2.3.5 Bluetooth Low Energy

Bluetooth Low Energy (BLE)[7] is a short range wireless communication technology, developed by Bluetooth Special Interest Group (SIG)[9]. It is more focused on low power consumption than classic Bluetooth. To take advantage of this technology the mobile device needs to be equipped with Bluetooth version 4.0[8] or above. However, to be able to use it in order to get the user's indoor location, a protocol is needed. There are two protocols that can be used: iBeacon[10], developed by Apple™ and Eddystone[11], developed by Google™.

The iBeacon protocol works with small devices named beacons that broadcast a sequence of bytes, which acts as an identifier allowing to build proximity-based applications[9]. Figure 2.4 shows the structure of this sequence of bytes, where it is possible to see three parts:

[6]http://www.apple.com/apple-pay/
[7]Source: http://www.appannie.com/apps/google-play/app/com.google.android.apps.maps in 23 December 2015
[8]http://www.google.com/maps/about/partners/indoormaps
[9]http://www.bluetooth.org
[10]http://developer.apple.com/ibeacon
[11]http://github.com/google/eddystone

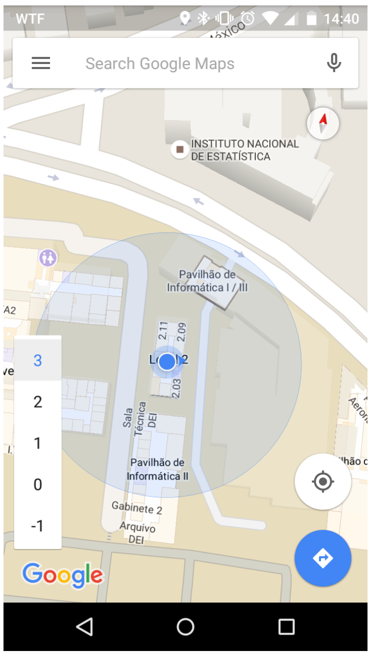

WTF
14:40
Search Google Maps
INSTITUTO NACIONAL DE ESTATÍSTICA
Pavilhão de Informática I / III
Sala Técnica DEI
Pavilhão de Informática II
Gabinete 2
Arquivo DEI
3
2
1
0
-1
Google

Figure 2.3: Screenshot of Google Maps with indoor functionality

Universally Unique Identifier (UUID) has 16 bytes (128 bits) and it identifies the organization that the beacon belongs to;

Major has two bytes (16 bits) and it identifies a group of beacons that belong to a given organization identified by the UUID;

Minor has two bytes (16 bits) and it identifies each individual beacon in the group identified bu the Major value.

In this protocol, location is reported to an application using one of the two operations:

Monitoring This operation is called when the beacon and the mobile device are in the same space;

Ranging It is related to a single beacon. The distance from a mobile device to a beacon is estimated using its transmissions.

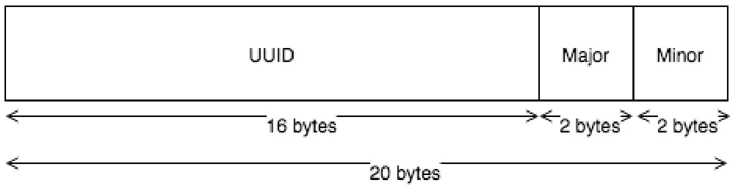


Figure 2.4: Structure of the sequence of bytes transmitted in iBeacon protocol

In Eddystone protocol the beacons also advertise a sequence of bytes. However, there are three types of messages that beacons can broadcast to mobile devices nearby[12]:

Eddystone-UID It broadcasts an unique 16 byte (128 bits) identifier. The first 10 bytes are for the namespace, which is used to distinguish groups of beacons and the remaining 6 are for the instance's Identifier (ID), which is used to identify individual beacons inside the same namespace;

[12]http://github.com/google/eddystone/blob/master/protocol-specification.md

Technology	**Techniques**	**Type of location**	**Computation**	**Scale**	**Costs**
GPS	Triangulation	Physical	In device	24 satellites serve unlimited receivers	Satellites
QR Codes	Proximity	Symbolic	External infrastructure	Depends on the external infrastructure	External infrastructure
NFC	Proximity	Symbolic	External infrastructure	Depends on the external infrastructure	External infrastructure + Tags
Google Maps Indoor	Triangulation	Physical	In device + External infrastructure	Depends on the external infrastructure	External infrastructure
BLE	Proximity	Symbolic	External infrastructure	Depends on the external infrastructure	External infrastructure + Tags

Table 2.1: Comparison of location technologies

Eddystone-URL As the name suggests, it broadcasts an URL;

Eddystone-TLM Here, telemetry information about the beacon is transmitted such as battery voltage and device's temperature.

The main advantage of this technology is that it only requires that the mobile device has Bluetooth 4.0. However, to develop proximity-based applications, we need to deploy some beacons that use the iBeacon or the Eddystone protocol. Fortunately, the extra hardware is the space owner's responsability. The user does not need anything else besides his/her mobile device.

Using this technology, if developers want to map beacons to more kinds of information, only the mentioned protocols are not enough. We need a backend to store the mapping between the beacons and that information. Besides developing the application itself, developers also need to deal with the backend and all the concerns around any distributed system, such as, scalability, performance, etc.

2.3.6 Comparison

There are no perfect location systems. Every system has its own limitations. There are ones that do not work well indoors, such as the GPS because it cannot get the sattelites' signal. If we want a location system that would work indoors, maybe a tag based system is a better fit. When choosing a system, this is important because one might not require extra hardware and it leads to a low cost system but, if the limitations compromise its performance it is better to pick one with higher costs. Multiple technologies were taken into consideration. Ones are tag based, such as QR codes, NFC and Bluetooth Low Energy (BLE) that can be used to provide symbolic location. Others, such as GPS and Google Maps Indoor, are used to provide physical location but can be augmented to provide symbolic location. However, GPS does not work properly indoors and Google Maps Indoor requires that buildings are mapped first. This mapping can be performed by a team from Google™ or by users. Each technology was classified in terms of techniques, type of location (physical or symbolic), computation, scale and costs. This classification was made using the taxonomy introduced in section 2.2 and is summarized in Table 2.1.

The concept of Smart Place is based on proximity and symbolic location. Also, it is supposed to work indoors. GPS is a location system that uses triangulation to provide physical location. It is possible to augment physical location in order to provide symbolic. However, it only works properly where it has a good receiption of the satellites' signal. Since the concept of Smart Place is a concept connected to an indoor space, the GPS is not a good option even if we augment it to provide symbolic location, because it does not work properly indoors.

QR Codes and NFC are based on the proximity technique. They can provide symbolic location if enough data is stored in an external infrastructure. The scale is dictated by this infrastructure and not not by the device that is being located. The costs depend on the amount of infrastructure. The main limitation of both is that it is only possible to read a tag at a time. QR Codes require that we spread codes which can be shown in a piece of paper. NFC tags are more expensive. However, both would take us to a solution where the user interaction is required, that is, the user needs to be aware of existing tags and start the interaction instead of just being notified that a given proximity-based service is available at that location. Smart Places could be implemented using Google Maps Indoor. This technology is based on triangulation technique and provides physical location. Similar to GPS it can be augmented to provide symbolic location requiring a given amount of external structure, which is what scale depends on and the major cost here. This would lead to a solution where no extra hardware is needed. However, it requires a previous mapping of the building before users are able to navigate in it using Google Maps Indoor. BLE is a good fit for our purpose because, it is based on proximity and can be used to provide symbolic location. It does not have any costs for the users besides the acquisition of the mobile device. We can place the beacons wherever we want and where it makes sense for the service we want to provide. Also, using this technology, the mobile app that will handle the nearby beacons can be running in background. This way, the user does not to be aware of any tags and he/she can be notified if a proximity-based service is found.

The final decision was to pick BLE because it has the best tradeoff between cost and kind of location it provides. Using this technology we have symbolic location. With the adequated infrastructure it is possible to associate any kind of information to each tag. It requires extra hardware but this is the responsability of who manages the place where the tags will be deployed. The user does not need anything else but a mobile device with Bluetooth, version 4.0 or later. Three beacons, from Estimote™ were used in the implementation of our solution, described in chapter 4.

2.4 Related Work

Here we discuss related work in order to have a good insight about the possibilities and existing solutions for the problem we are trying to solve.

2.4.1 Proximity-based Apps

BLE Beacons were used to develop our solution, based on the concept of Smart Places. Some applications where this technology is used are presented here to get good insights about the potential use cases of this technology and the apps developed using it.

BlueSentinel[10] is a occupancy detection system for smart buildings that uses BLE Beacons to detect the presence of people. The concept of a smart building is similar to Smart Place due to the existence of sensors and actuators. It is focused on the power efficiency of the building. The idea is to optimize energy consumption according to people's presence. For instance, if there are no people in a given room the heating system can be turned off. In this solution the users have to install an app that will get the beacons' signal and send data to a server, which will process it and send requests to actuators in order to perform actions to optimize the building's power efficiency. Unfortunately, there is a limitation of iBeacon protocol implementation in iOS. Beacons can be received by the apps only when these are active. When the apps are in background they are waken up only to handle enter/exit region events. To circumvent this limitation the authors developed custom beacons which advertise more than one region in a cyclic sequence. These custom beacons were created using an Arduino[13] and a Bluetooth Universal Serial Bus (USB) dongle. Since this solution is a native app users have to install it in order to make the smart building work to optimize power efficiency. Once the user starts the app, he/she does not need to interact with it anymore since it will run in background.

BlueView[11] is a system to help visually impaired people to perceive some points of interest. This solution has two main components: The viewer device and the Beacon Points (BPs). The first one is a mobile phone, carried by the user, which is bluetooth enabled. The BPs are just bluetooth tags instead of BLE Beacons. The name of a point of interest is associated with the Media Access Control (MAC) address of the tag which it is associated to. The steps involved in using the system are the following: first, the viewer device will scan for nearby BPs; then, a list of the names of BPs is created. This list is refreshed anytime a new BP is detected and the user is informed through auditory feedback. The second step consists of the user using the viewer's device establishing a connection with a BP attached to an object. Finally, using audio prompt, the BP will assist the user in locating the object. Despite of this solution being a mobile app, installed in the viewer's device the authors do not have in consideration the typical concerns of any mobile app, such as the energy consumption. The authors tested the application, in 2013, using Nokia N70 as the viewer device. This solution could be implemented using BLE Beacons and the viewer device could be any Android or iOS smartphone.

ContextCapture[12] uses context-based information to allow users to add more information to their status updates in the main social networks, such as Facebook[14] and Twitter[15]. This work had two main goals: first, demonstrate technical aspects of collaborative context such as, how to get contextual information from surrounding devices and how they can be used as a source of contextual information; second, test and analyze the user experience of context-aware systems. The user can decide the abstraction level (coordinates, address or semantic label). The authors implemented a mobile app and a server integrated with Facebook and Twitter. Context information comes from the smartphone itself, from its sensors and from the nearby devices through Bluetooth. Devices can be other smartphones or BLE Beacons which are used for indoor location. Similar to [13], devices communicate with each other as a network. Using this solution, the user can create status updates in the mentioned social networks in the format shown in Listing 2.1:

```
[User-defined message]
Sent from [Location] while [Activity]
[Description] [Topic] and [Applications Activity] with [Friends].
```

Listing 2.1: Format of status updates in ContextCapture

[13]http://www.arduino.cc/
[14]http://www.facebook.com
[15]http://twitter.com

2.4.2 Context-aware applications

In this section we describe related work about the development of mobile native and web context-aware applications. Since we have created a framework to develop proximity-based services, the state of the art of existing frameworks that deliver context information to the apps will be presented.

Frameworks for developing distributed location-based applications: There are frameworks to develop location-based applications. In the work presented in Krevl et al.[14], a framework was developed to allow developers to build location-based apps. Location information can come from any source, such as GPS, Bluetooth and WiFi receivers. The authors discuss some benefits and limitations of several technologies for getting the user's location. In terms of architecture, the main components are:

- Devices that are used to get location data, such as the ones already mentioned;
- The users' mobile devices;
- The Database Server, which is where the mapping between geographical coordinates and location information is stored;
- And, the Application Server, which provides web services for mobile clients. This server also communicates with the Database Server.

The mobile device gets geographical coordinates from any source and send that data in a Simple Object Access Protocol (SOAP)[15] message, to the appropriate web service in the Application Server. This server communicates with the Database Server to query the database, which sends back a response with location information, if there is any, for that particular group of geographical coordinates. The authors did not evaluate the system.

Dynamix[16] is a framework to develop mobile native and web apps that allows them to receive context information for instance, position and device's orientation. This framework has plugins 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 plugins to allow the apps to generate additional events that are not already supported.

2.4.3 Discussion

In order to get a good insight about the potential of proximity-based services, we analyzed three applications. The BlueSentinel is a system that uses BLE beacons to detect the presence of people in a building. It tries to optimize energy consumption based on people's presence. Another one, BlueView, is a system to help visually impaired people to be aware of some points of interest. The third is the ContextCapture that allows users to create status updates on social networks using context information.

We introduced a framework to develop distributed location-based applications and a more generic one that targets the development of context-aware applications. For each one, we described the idea, its main components, in order to get an overview about their limitations, which we try to circumvent in our solution. However, most of them only allow developers write native apps, that is, apps written for a specific platform that will run only on it. This leads to a situation where the user needs to install one app, in his/her mobile device, for each proximity-based service he/she wants to use. In order to circumvent this limitation, our Smart Places solution allows developers to use the same technologies that are used for any web application such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and Javascript to build Smart Places allowing them run on a web browser that can be embedded in a mobile app. This way, users only need to install one app to discover and use proximity-based services following the Smart Places approach.

The framework described in Krevl et al.[14] offers abstractions for the location information sources. Geographical coordinates can come from any source. It is a good approach for mobile native apps but it does not support web apps. The authors do not take into consideration constraints in terms of resources, such as Internet connection and battery. Since most users have limited data plans for their smartphones and SOAP messages can grow in size due to its Extensible Markup Language (XML) format, a more efficient message encoding could be used instead, for instance Representational State Transfer (REST) using JavaScript Object Notation (JSON).

To achieve our goal, our framework could be just a plugin for Dynamix. The plugin would need to get the beacon's raw data and turn that into a more high-level information using a backend. In this framework, the user needs to install an app that manages the service that runs in background and needs to define some security policies such as which information the app can have access to or which sensors it can use. This could mean a big overhead since we are more focused on developing proximity-based applications that do not require such complex security policies because in this kind of applications, there is only need to access the device's sensors that could provide positioning data to the applications

2.5 Summary

Smart Places are based on the proximity technique and symbolic location. Also, we need to take into account the cost and limitations because we want a solution as low cost as possible, in terms of time and capital, for end users, owners and developers.

In this chapter we introduced a taxonomy to classify location systems in terms of techniques, physical or symbolic location, computation, scale, cost and limitations.

The concept of Smart Place was defined as a place with tags, that provides a service to users when they approach those tags. Using the taxonomy with property definitions we need a location system that is based on the proximity technique and provides symbolic location.

In our solution, we needed to choose a location technology. We analyzed GPS, QR Codes, NFC, Google Maps Indoor and BLE, according to the taxonomy introduced before. We concluded that the BLE using ibeacon protocol is the best option because it works well indoors, unlike GPS. Also, it does not require user interaction, as it is in QR Codes or NFC. Using this technology, owners can place tags anywhere without the need to first map the entire area, as it is needed in Google Maps Indoor.

We explored related work about existing proximity-based applications and tools to develop this kind of applications. In most solutions users have to install one app for each proximity-based service they want to use. There is not a solution to allow users to discover new services and use them as soon as they found them. Our solution described in the next chapter tries to solve these limitations and turn the development and usage of services based on Smart Places in easy processes.

Chapter 3

Solution

The main goal of our solution is to assist the development process 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 are anyone with a mobile device that installs an app to scan for nearby Smart Places and interact with tags;

Owners are responsible for managing a given place that they want to turn into a Smart Place;

Developers develop the code for the Smart Places.

The Smart Places solution has a component that targets each one of the presented type of user. There is a mobile app to allow anyone with a mobile device to use the services provided by nearby Smart Places. Owners have a mobile app that allows them to turn the places they manage into Smart Places. 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.

In the next section, we introduce the main components of the solution and how they relate with each other. The solution's architecture is described in more detail in section 3.2. The Android apps are described in sections 3.3 and 3.4, respectively. Then, we present the API for developers in section 3.5, from its installation to its usage. In section 3.6, we describe two examples of Smart Places, the Smart Restaurant and the Smart Museum, including their main features and how they contributed to the development of our Developers API. Finally, in section 3.7 there is a summary of the important aspects of our solution.

3.1 Solution Overview

The Smart Places solution developed in this work is composed by several components, as shown in Figure 3.1:

Beacons are the small devices that will act as tags, according to our definition of a Smart Place;

Backend is where all the data about Smart Places and tags is stored.

End Users Mobile App is an 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 another 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 API provides the necessary methods that developers can use to create their proximity-based services, based on the concept of a Smart Place.

Each mentioned component is described, in further detail, in the next sections.

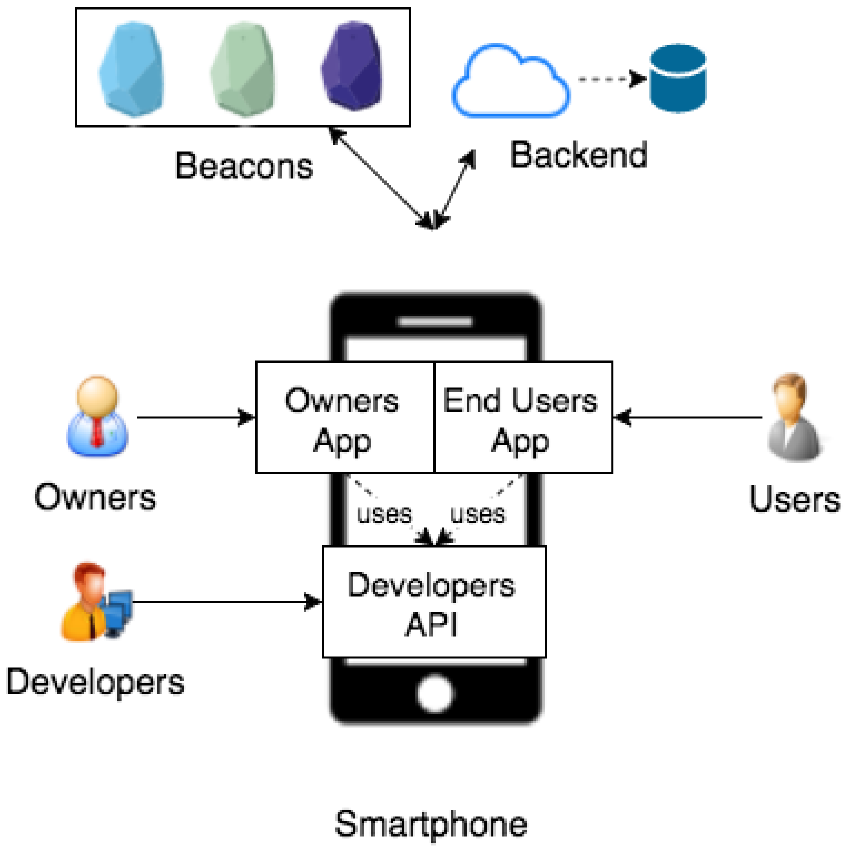


Figure 3.1: Overview with the main components of Smart Places solution

3.2 Architecture

In the previous section, we described important decisions for the development of our Smart Places solution. Those decisions were taken into account while designing its architecture. As already mentioned, we have two mobile apps, each one for a different kind of user. One allows end users to have access to services provided by Smart Places and the other offers an interface for owners to manage their Smart Places. Both apps need to be able to detect tags and send and receive data to and from the backend. Between these two apps the difference is on the User Interface (UI) that each one offers. Smart Places solution is composed by the following components, as shown in Figure 3.2:

Beacons are the hardware used to act as tags in Smart Places;

Backend is where all the data is stored, including data about each Smart Place and each tag;

Beacons Manager is an abstraction of the BLE layer. It offers a method to scan for nearby beacons. Once this component was created we did not need to handle the BLE communication each time a scan was necessary;

Data Store is the Backend client for the mobile apps. It has all the needed methods to get the information about Smart Places and tags from the beacons that were detected by the Beacons Manager;

User Interface is the mean for the user to interact with the mobile app and calls the other components in order to provide the necessary features;

Smart Places Web View is an instance of a Web View, available in the Android Software Development Kit (SDK). This is an extension of the default Web View because it has methods to call Javascript functions of the web application that is running here;

Smart Places API is the Javascript library that allows the web applications to react to the presence of tags. It is described in further detail in section 4.6.

Smart Place Instance is the instance of the proximity-based service developed according to the concept of Smart Place that is running inside the Web View and uses the Smart Places API to be able to execute code when a tag is detected.

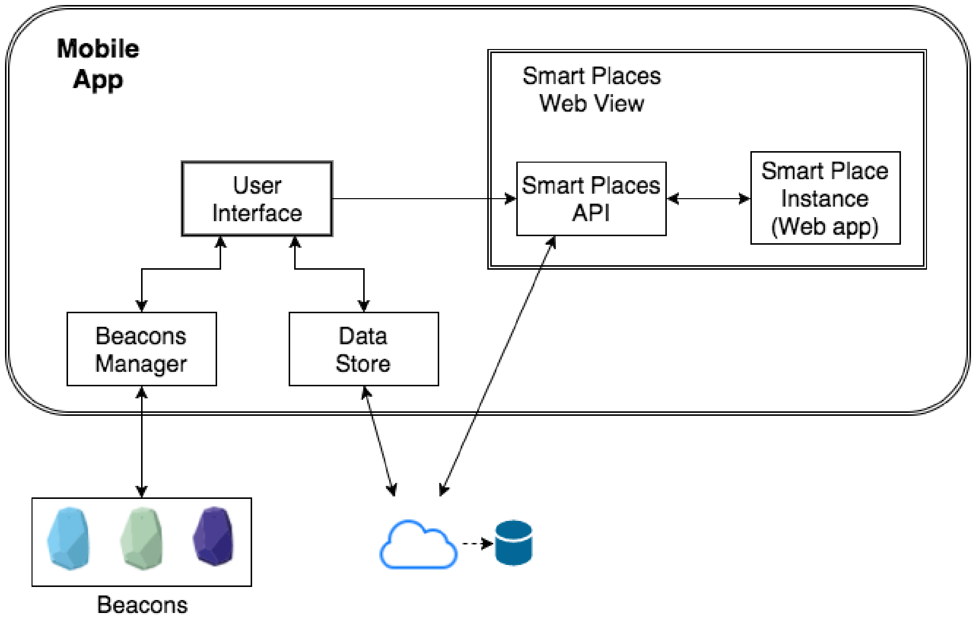


Figure 3.2: Architecture of Smart Places Solution showing its main components

The different components interact with each other in order to provide the features for end users or owners depending on the app. Figure 3.3 shows how the components of our architecture communicate with each other in order to provide a proximity-based service for end users. First, the Beacons Manager is called in order to scan for nearby beacons and it returns the nearest one. The beacon itself only broadcasts an identifier. It does not have enough data to know which Smart Places it belongs to. Next, the Data Store component is called in order to get the information about the Smart Places that use that beacon as a tag. Data Store makes a request to the Backend in order to get this information. When the Backend returns a list of Smart Places that list is used to notify the user about the Smart Places that were found. Then, if the user touches on the notifications a UI with a Web View is shown. Since each Smart Place is a web application the URL of the Smart Place that the user previously selected, when he touched the notification, is loaded to the Web View. After the page finishes loading, the Beacons Manager is called again to perform another scan for nearby beacons and it returns the nearest one. The

Data Store is called again but this time, to get the tag associated to that beacon that belongs to the selected Smart Place. The data of this tag is sent to the Web View which calls the Smart Places API allowing the web application to react to the presence of this tag.

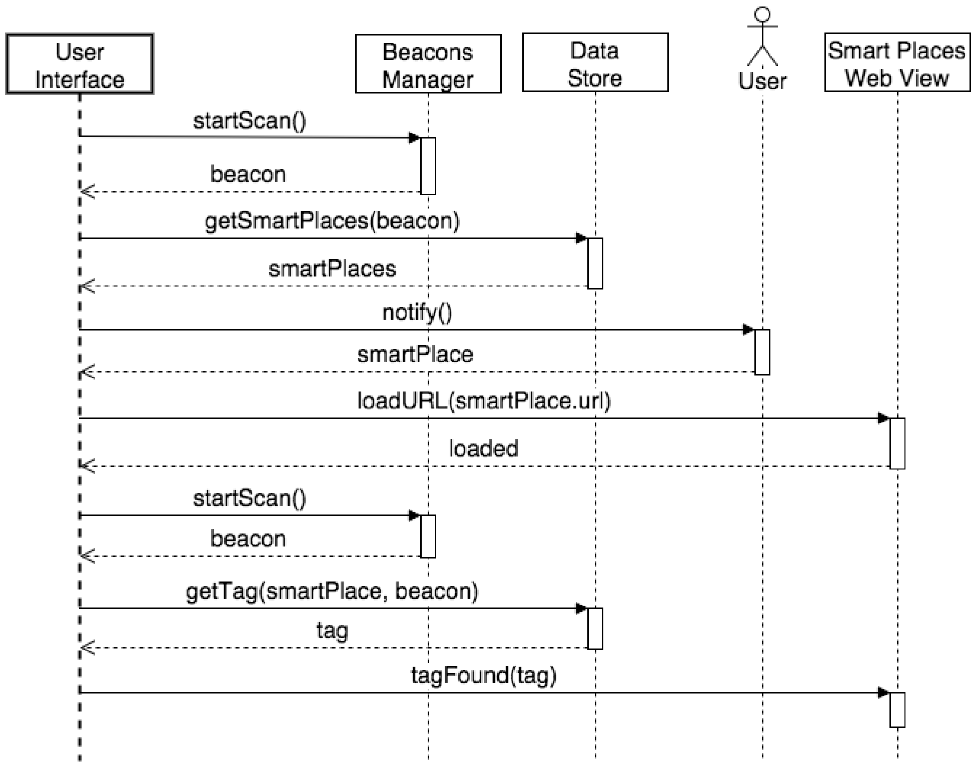


Figure 3.3: Sequence diagram showing the interaction between the Smart Places components

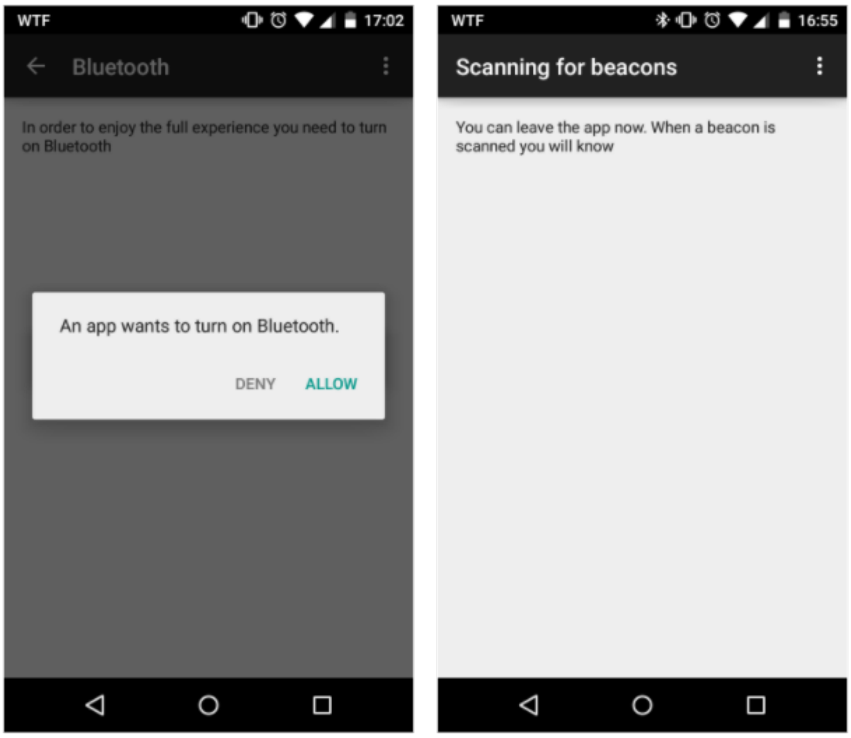
3.3 Mobile App for End Users

Anyone that has a mobile device, such as a smartphone, can use the services provided by any Smart Place. In our solution, there is an Android app distinct from the one described in section 3.4, that notifies the user when he is nearby any Smart Place. When the mobile device approaches any Smart Place the app notifies the user that he is near a Smart Place. 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.

After installing the app, the first time the user opens it, the app ask the user to turn on the device's bluetooth, as shown in Figure 3.4. This app will scan for nearby beacons periodically. Each time a beacon is scanned, the app shows a notification associated to a Smart Place. If the scanning period is small, the app can constantly notify the user. Otherwise, if this period is big the user will receive less notifications. This period also has an impact on battery consumption. The more times the app scans for nearby beacons the more battery is drained. According to his/her preference, our app allows the user to change the scan periods in background and foreground modes.

3.4 Mobile App for Owners

The Smart Place owners manage one or more places where they want to provide a service to visitors in their mobile devices. In order to make owners be able to offer this kind of services, an Android app



In order to enjoy the full experience you need to turn on Bluetooth

An app wants to turn on Bluetooth.

You can leave the app now. When a beacon is scanned you will know

Figure 3.4: First interaction with the user

designed for them is offered by this solution. This app offers the folllowing features:

- Get a list of all available Smart Places;
- Configure an instance of a given Smart Place;
- Delete an existing configuration of a given Smart Place;
- Update an instance of a given 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 the steps shown in Figure 3.5:

- First, the app asks owners to log in. The app only allows users to log in using their Facebook account.
- The app shows a list of all available Smart Places that the owner can configure. For instance, the Smart Restaurant and Smart Museum examples described in section 3.6 are examples of Smart Places that can appear in this list;
- 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

After creating an instance of a Smart Place the owner needs to configure tags, that is, associate information to the previously deployed beacons. This information will depend on the Smart Place. Figure 3.6 shows the needed steps to configure tags in an instance of a given Smart Place. First, the owner selects the instance from the list of ones that he has created. Then, he has access to an interface where he can manage the existing tags of that Smart Place instance. Each Smart Place has its own management interface. Developers are responsible for creating these interfaces, using the developers API, introduced in section 3.5.

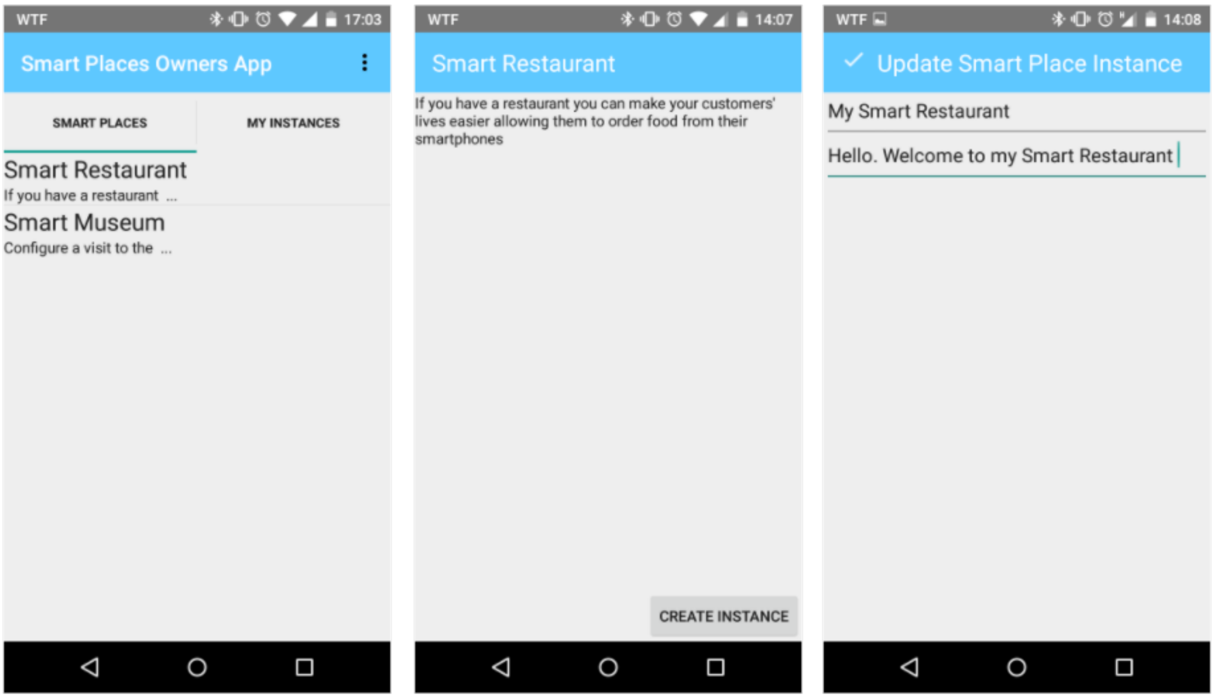

WTF
17:03
Smart Places Owners App
SMART PLACES
MY INSTANCES
Smart Restaurant
If you have a restaurant ...
Smart Museum
Configure a visit to the ...
WTF
14:07
Smart Restaurant
If you have a restaurant you can make your customers' lives easier allowing them to order food from their smartphones
CREATE INSTANCE
WTF
14:08
Update Smart Place Instance
My Smart Restaurant
Hello. Welcome to my Smart Restaurant

Figure 3.5: From left to right, the steps to create an instance of a given Smart Place

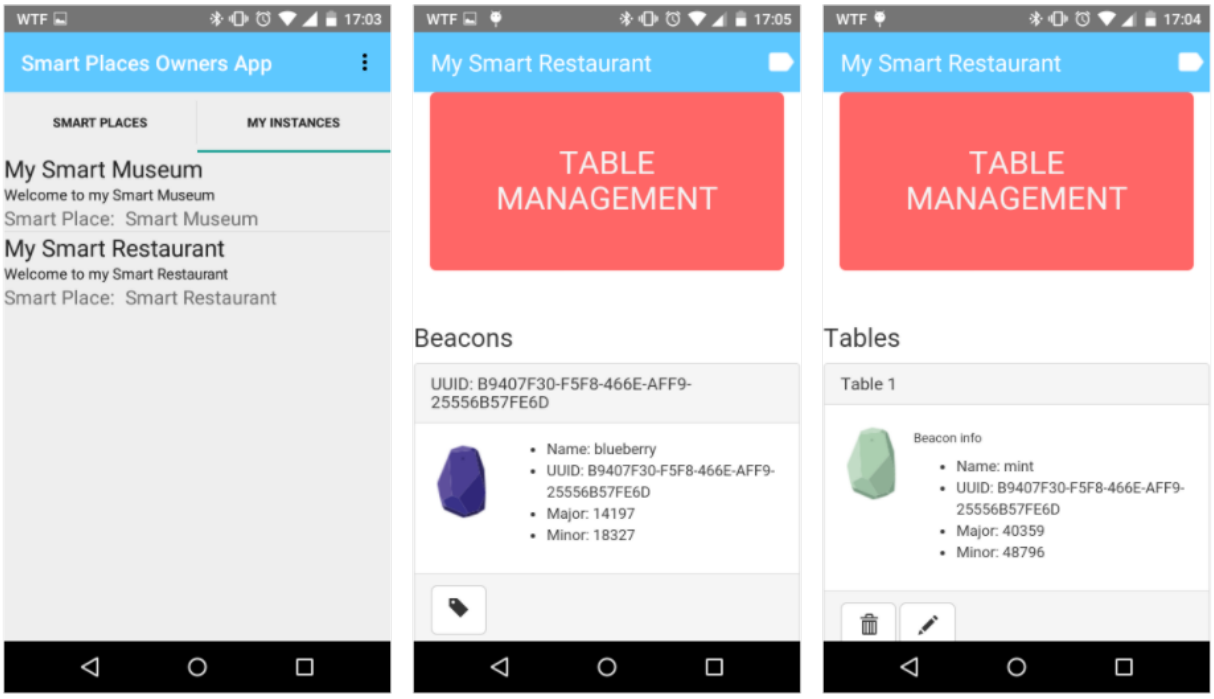


Figure 3.6: From left to right, the steps to configure an instance of a given Smart Place

3.5 Developers API

Owners configure the data of a Smart Place and end users can interact with objects nearby. But, who will add behavior to these Smart Places? Our solution offers a way for developers to create their Smart Places. Also, we want to avoid the user having to install one mobile app for each Smart Place. The app for end users has an embedded web browser, so they can use any Smart Place as they would use any web application without the need to install one more mobile app. That is why a Javascript library is part of our solution. This way, an existing web application can use this library and make it react to nearby objects tagged with BLE beacons.

The library was turned into an open-source project, hosted on a github repository[1] and it is available to install using bower[2], which is a tool to manage dependencies in web applications. If a developer wants to install this library, he/she just needs to run a command, as shown in Listing 3.1

```
bower install smartplaces-js --save
```

Listing 3.1: Command to install smartplaces-js library using bower

Then, he/she just needs to include the library and use the available functions. The library is event-based, that is, the mobile apps described in sections 3.4 and 3.3 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 because as already mentioned the library follows an event-based approach. The reason why it is not synchronous is explained in chapter 4, section 4.1. However, developers are also responsible for creating the interface to configure the Smart Place, that is, the steps that owners have to follow, as described in section 3.4 after they select the Smart Place they want. In the part of the web app that will be accessed by owners, developers first need to initialize the library, as shown in Listing 3.2.

```
SmartPlaces.onInit(function(smartPlaceInstance) {
  // Code after initialization of this Smart Place Instance
});
```

Listing 3.2: Javascript library initialization

When the owner is using the app, described in section 3.4, there is a button that when is touched the app scans for nearby beacons and sends an event to the javascript library. Developers have to define the behaviour when this event is emitted, as shown in Listing 3.3.

```
SmartPlaces.onBeaconsScanned(function(beacons) {
  // Code to handle beacons that were scanned
});
```

Listing 3.3: Defining a callback function when beacons are scanned by the mobile app for owners

The argument named “beacons” in the callback function in 3.3 is an array of JSON objects, where each one has the following keys:

- uuid: The UUID of the beacon that was scanned;

[1]http://github.com/samfcmc/smartplaces-js
[2]http://bower.io

- major: The major value, according to the ibeacon protocol;

- minor: The minor value, according to the ibeacon protocol.

However, there is more information about each beacon for instance, its name and its icon URL. To get this extra information, from a beacon JSON object, there is a function, which usage is shown in Listing 3.4, that makes a request to the backend in order to get all the information about the given beacon. This information includes a name and an URL for an image that represents the beacon, besides the data already provided such as the UUID, major and minor values. Since this function makes a request to the backend, we need to pass as an argument an object with two keys:

- success: Callback function when the request was successfully made and we got a response with an object that, besides the keys mentioned before, UUID, major and minor, also got the name and icon which is an URL that we can use to get the image of that particular beacon;

- error: Callback function, when the request returns an error.

```
SmartPlaces.getBeacon(beaconScanned, {
  success: function(beacon) {
    /*
    Code to handle when a beacon object was retrieved
     successfully from the backend
    */
  },
  error: function(error) {
    /*
    Code to handle when an error occurrs when trying
     to get a beacon object from the backend
    */
  }
});
```

Listing 3.4: Get beacon information from the backend

After we got the beacon object with all the information, it is possible to associate a JSON with any structure. To do that, there is an “associateTag” function which usage is illustrated in Listing 3.5. We need to pass the beacon object, the object with the data that we want to associate with the beacon and another object with success and error keys, similar to what is shown in Listing 3.4.

```
SmartPlaces.associateTag(beacon, data, {
  success: function(tag) {
    /*
    Code to handle when a tag is successfully
    associated to a beacon
    */
  },
  error: function(error) {
    /* Code to handle when an error occurrs trying
    to associate a tag to a beacon
    */
  }
});
```

Listing 3.5: Associate a tag to a given beacon and provide custom data

It is also possible to update an existing tag. For that, developers can use the “updateTag” function. Its usage is shown in Listing 3.6. This function requires the existing tag object and an object with success and error keys similar to the other functions that make requests to the backend. This function is similar to the previous one, shown in Listing 3.5 but instead of passing a beacon as an argument, we pass a tag object to update it with the data in the object provided as the second argument.

```
SmartPlaces.updateTag(tag, data, {
  success: function(updatedTag) {
    /*
    Code to handle when the given tag is successfully updated
    */
  },
  error: function(error) {
    /*
    Code to handle when an error occurrs when
    trying to update the given tag
    */
  }
});
```

Listing 3.6: Update data of a given tag

The previously mentioned functions, are available in order to make developers able to create the interfaces for owners. For the end users, the mobile app detects nearby tags and emit an event to the web application running inside the embedded web browser. Developers need to define a callback function for this event. To do that, the “onTagFound” can be used, as shown in Listing 3.7. The tag object, which is the argument of this callback function, is the JSON object created previously in the code illustrated in Listing 3.5.

```
SmartPlaces.onTagFound(function(tag) {
  // Code to handle when the mobile app detects a tag
});
```

Listing 3.7: Callback for when a nearby tag is found

3.6 Examples

We have created two examples of Smart Places to see if our solution would work in practice. First, we developed the Smart Restaurant. While building this example, we wrote Javascript code to handle the events emitted by the mobile apps, described in sections 3.4 and 3.3. 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 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 was used in the other example. Also, we avoided to develop the examples completely from the scratch. Instead, we tried to integrate the Javascript (JS) API, described in section 3.5 with existing applications or APIs.

As already mentioned, two examples were created. The first, described in section 3.6.1, is a Smart Place for restaurants to allow customers to place their orders without the need to wait. The other one, is a proximity-based service for museums. It allows users to have access to more information about an object that they are close to in a museum exhibition. Its features are described in section 3.6.2.

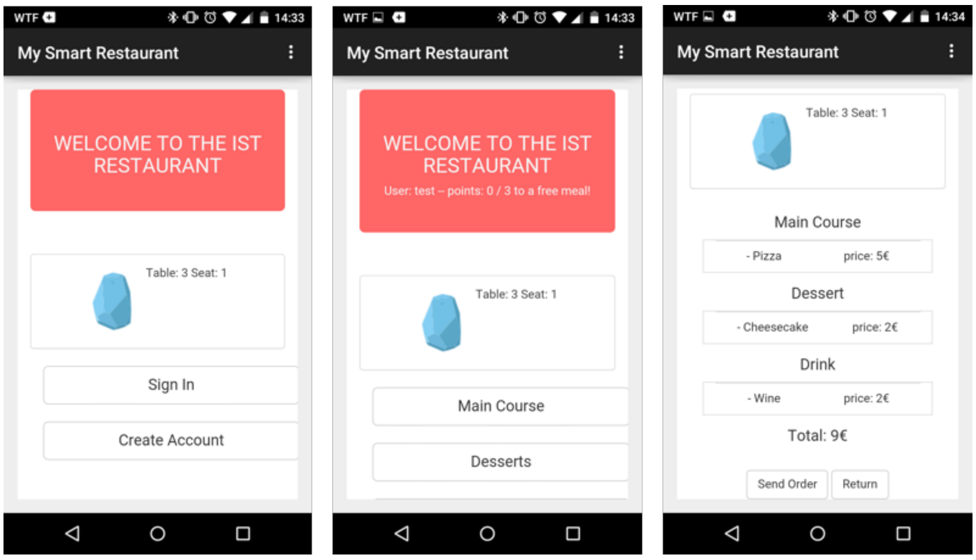
3.6.1 Smart Restaurant

The Smart Restaurant was the first app created to show the usage of the entire solution. The main goal here is to allow restaurants' customers to place their orders, using their mobile devices, such as smartphones, without having to wait for an employee coming to them. When customers arrive at this Smart Restaurant, a notification shows up in their devices with a message saying that they can place their orders using the mobile app. Then, they touch the notification and a new UI appears. Now, they have access to the restaurant's menu where they can pick what they want and in the end, place their orders. Figure 3.7, shows the steps that the customer follows to place an order using the mobile app. We can see from left to right that the app detects the table's number and after the sign in process several buttons appear, each one representing a family of products and there is a button in the bottom to see the complete order. When the customer touches this button, there is a list of the complete order and a button to send the order.

Also, there is a backoffice, where employees and managers, have access to an UI to manage the orders and the menu. This backoffice was implemented in another master thesis[17]. Our work here was to integrate our solution in this backoffice. There was already an interface to place orders. We changed this interface to use our Developers API, described in section 3.5 to get the table's number. We also added an option to the backoffice's UI to manage the mapping between tables and beacons. This integration shows the hability to use existing code, that is, our Smart Places solution can be integrated in existing applications.

3.6.2 Smart Museum

The Smart Museum is another example of a Smart Place, that is, a proximity-based service developed using our solution. The idea is to allow visitors of a museum to have access to information about a


My Smart Restaurant

WELCOME TO THE IST RESTAURANT

Table: 3 Seat: 1

Sign In

Create Account

My Smart Restaurant

WELCOME TO THE IST RESTAURANT

User: test – points: 0 / 3 to a free meal!

Table: 3 Seat: 1

Main Course

Desserts

My Smart Restaurant

Table: 3 Seat: 1

Main Course

- Pizza price: 5€

Dessert

- Cheesecake price: 2€

Drink

- Wine price: 2€

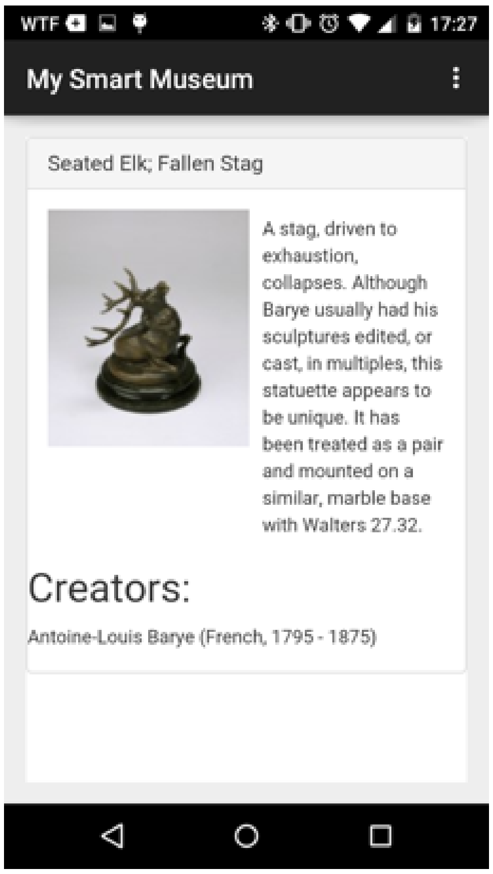
Total: 9€

Send Order

Return

Figure 3.7: Steps to place an order in the Smart Restaurant app

given object in a given exhibition using their mobile devices. Visitors go to the museum and as they look at a given object they are notified, through the mobile app that they can get more information. Then, when they touch in the notification, a screen appears with the information about that object that they are looking at. Figure 3.8 shows a screen, in the mobile device, after the museum's visitor has touched the notification.


My Smart Museum

Seated Elk; Fallen Stag

A stag, driven to exhaustion, collapses. Although Barye usually had his sculptures edited, or cast, in multiples, this statuette appears to be unique. It has been treated as a pair and mounted on a similar, marble base with Walters 27.32.

Creators:

Antoine-Louis Barye (French, 1795 - 1875)

Figure 3.8: Smart Museum app showing information

Instead of creating a fake museum with mock data, we used real data from a real data source.

The idea was to try to emulate the real experience. For that, we have used data from the Walters Art Museum[3], which is a public art museum, located in Mount Vernon-Belvedere, Baltimore, Maryland. It was founded in 1934 and its collection includes more than 30 000 objects. The data was obtained using the Walters Art Museum API[4] which provides methods to get collections and objects of each collection.

3.7 Summary

In this chapter we presented our Smart Places solution and its main components. There are the beacons which act as tags in a Smart Place. The Backend stores the data about each Smart Place and the tags. Since our solution targets owners and end users, there is a mobile app for each one that offers features according to their needs. Also, for developers, we introduced an API that they can be used to create proximity-based services according to the definition of a Smart Place.

We introduced our architecture and its main components. Inside the mobile apps we have the Beacons Manager which interacts with the beacons and the Data Store which is essentially a client for the Backend. One important component is the Web View which is an embedded web browser that gives access to Smart Places that are web applications. This allows developers to built Smart Places using web technologies such as HTML, CSS and Javascript. Also, this allows users to use any Smart Place just by installing one app in their mobile devices. The decision to support web applications is explained in further detail in the next chapter.

The app for end users, anyone with a mobile device that can detect tags in a given Smart Place and then forward to specific functionalities. The user needs to install the app and turn on the device's Bluetooth receiver. Then, the app will scan for nearby beacons, in background. When a Smart Place is found, the app notifies the user. When the user touches a notification the app will perform a new scan but this time, to find tags that belong to that Smart Place.

The mobile app for owners allows them to see a list of available smart places, configure an instance of a given Smart Place, delete an existing configuration or update an instance of a Smart Place they already have configured. Owners deploy beacons in the right places and use the app to create an instance of a Smart Place and configure the tags according to what is requested by the Smart Place that he is configuring.

Developers implement the custom behaviors of proximity-based services. We created an API for them to use to make the development of a Smart Place an easier process that without this API. Due to its asynchronous nature, each method of the API receives a callback function.

We created also two examples of Smart Places, the Smart Restaurant and the Smart Museum. The first allows customers of a restaurant to place their orders using their mobile devices when they arrive at the restaurant. Using the tags deployed in the restaurant, the table's number is automatically found allowing employees to know where the order comes from. The Smart Museum allows visitors of a given museum to get information about a given object when they are in the proximity of that object. These examples were created to demonstrate the solution in a pratical way.

[3]http://thewalters.org
[4]http://api.thewalters.org/

Chapter 4

Implementation

In chapter 3 we have presented the solution, that is, the mobile apps that end users and owners need and the API that developers use to develop proximity-based mobile apps for this two kinds of users. This chapter explains in detail the implementation of each part of the solution (tools, programming languages, etc). Since there are two examples that were created to test the concept, we will dive into their implementation also. The code is hosted on github[1], which is a hosting service for projects that use git[2] as their Version Control System (VCS).

In the next section we describe the important decisions behind the implementation of our Smart Places solution. Next, we summarize the tools that were used in the development of this solution. Then, in section 4.3, it is explained the technology that was used to support the existence of tags according to our definition of a Smart Place. Section 4.4 presents the backend and the data model that it is stored and managed. The implementation of the mobile apps for end users and owners is presented in section 4.5. Then, we describe the implementation of the API for developers. Next, we explain how the Smart Restaurant and Smart Museum examples were developed. Last section concludes this chapter.

4.1 Smart Places

A Smart Place has tags that provide a proximity-based service. The Smart Restaurant and Smart Museum presented in chapter 3 section 3.6, are examples of Smart Places. They were developed according to our definition of Smart Place. As also presented in chapter 3 there is a mobile app that allows users to have access to these proximity-based services. However, these services could be mobile apps themselves. We could create an SDK, for each platform to allow developers to introduce proximity-based developers in their apps. Following this approach we would have a situation where the users would need to install one app for each Smart Place they want to use. In order to understand why we have an app to access any Smart Place we need first, to compare the possible approaches.

We could allow Smart Places to be native mobile apps or web apps that run inside an embedded web browser. A native mobile app is an app, developed with the native tools already provided by the platform. This kind of apps only run in the platform that they were created for. A web app is an application that runs inside a web browser. It can run in any platform that has a web browser available. Table 4.1 compares native and web apps, showing the advantages and limitations of each category of apps. If Smart Places were developed as native apps users would need to install one app for each one. This approach would not allow them to discover new proximity-based services while they were walking. To circumvent this

[1] http://github.com/
[2] http://git-scm.com/

Features	Native	Web
Access to device's features (Camera, accelerometer,etc)	All	Limited
Installation needs	Need to install the app	Only a web browser is needed
Method for finding the app	App stores	URL
Updates	Users can choose to not update the app	All users have access to the same version
Where it can run	Only on the platform for which it was developed for	On any platform

Table 4.1: A comparison of some characteristics of Native and Web mobile apps

limitation, we decided that Smart Places would exist as web applications. This way, any web application can react to the existence of tags in a Smart Place. Also, it leads to a situation where the users only need one app to have access to any Smart Place, because, each one is a web application that will run inside an embedded web browser in the users mobile app.

Having Smart Places as web applications accessed by the users mobile app, introduced in chapter 3 section 3.3, is achieved by using the WebView[3] which is a widget provided by the Android SDK that allows to have web pages embedded inside any UI in an Android application. Smart Places are web applications that run in a WebView in the users mobile app. For instance, the Smart Restaurant and Smart Museum examples run in this WebView. Users do not need to install two different apps. Using our mobile app they can access any of these examples.

Since we chose web apps as the mean to support Smart Places somehow, they need to be able to detect nearby tags. As it is possible to see from Table 4.1, web apps have limited access to the device's resources. As described in section 4.3, BLE beacons were used to support the existence of these tags. How can a web application, running in an embedded web browser inside a native mobile app detect tags in a Smart Place? Our approach is to handle the tags, that is, the BLE beacons in the native app and pass the relevant data to the web application running inside the embedded web browser. To do that, the native app invokes Javascript functions that exist in the embedded web browser according to the API, described in chapter 3 section 3.5. To invoke Javascript code from the native app a special URL is loaded to the WebView. The URL is in the format: "javascript:functionToInvoke(arguments)", where "functionToInvoke" is the function we want to invoke inside the WebView and "arguments" is a list of arguments to pass to that function.

4.2 Tools

During the development of our solution, several tools were used. The components such as, the back-end, mobile apps, the Developers API and the examples, were developed using different tools, from development environments to build and dependency management tools.

Since this solution needs to store information about each beacon, we needed a backend to be able to change this information without having to change the mobile apps. However, this backend just has to be able to store data and retrieve that data when client applications request it. Instead of implementing this component from the scratch, we used Parse[4] which is a Backend as a Service (BaaS). It provides a

[3]http://developer.android.com/guide/webapps/webview.html
[4]https://parse.com

dashboard, where developers can create classes and define their fields and the type of each field which can be a string, a number, a boolean, a pointer to another object, a JSON object, etc.

4.2.1 Android Apps

For the Android apps, we have used the following tools:

Java was the programming language that was used to write the code of the mobile apps;

Gradle [5] is a build and dependency management tool for Java projects. Using Android Studio, any Android project already includes this tool and a configuration file where all the dependencies are specified;

Android Studio [6] is an Integrated Development Environment (IDE), based on IntelliJ[7], to develop Android applications.

4.2.2 Developers API and Examples

The Javascript API and the Smart Restaurant and Smart Museum examples were created using web technologies, such as HTML, CSS and Javascript. However, besides these languages, we have used tools to manage the build process and also the necessary dependencies:

Bower [8] is a tool used in web applications, to manage dependencies. It was used here to manage dependencies of our Smart Restaurant and Smart Museum examples. Also, it was used in our Javascript API to allow this API to be released as a library in order to make it avaible, to install, using this tool;

Grunt [9] is another tool used in web applications, to manage the build process of frontend assets. It is based on tasks that are added depending on the needs of the project. There are tasks available to concatenate all Javascript files in just one, minify Javascript files, copy files, etc.

4.3 Tags

According to our definition of Smart Place, we need to tag objects to provide proximity-based services. Several location technologies were presented in chapter 2. To support the existence of tags we chose BLE Beacons using the ibeacon protocol because most smartphones are BLE enabled and it does not require the user to acquire any extra hardware. We have used three beacons from Estimote[10] (Figure 4.1), that come with a development kit. In this kit, each beacon come with a predefined name. The blue, green and purple are named ice, mint and blueberry, respectively. Besides the hardware, this product offers a SDK for Android and iOS. It also has a cloud service where developers can set and get some information about each beacon, such as, the UUID, name, etc. However, instead of integrating this cloud service with our backend, we copied the name and the image URL of each beacon to our backend to accelerate the development process.

[5]http://gradle.org
[6]http://developer.android.com/sdk/index.html
[7]http://www.jetbrains.com/idea
[8]http://bower.io
[9]http://gruntjs.com
[10]http://estimote.com

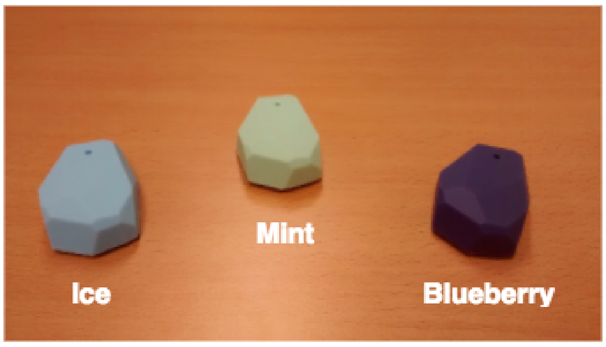


Figure 4.1: Beacons from Estimote, that were used during the implementation

4.4 Backend

As already mentioned, there was no implementation of the backend itself. Using Parse Backend as a Service (BaaS) in order to support the needs of this project the following classes were created, according to the data model in figure 4.2:

Beacon : Stores the information according to ibeacon protocol, which is UUID, major and minor values. It also has a pointer to an instance of class User, which is a class provided by Parse BaaS and represents its users. Fields such as, username, password, name and email are already provided by this class. Due to the existence of this class, the authentication mechanisms such as, using username and password or using accounts from social networks, such as Facebook or Twitter, are already provided; A pointer to an instance of class User represents the user that owns the beacon. Besides the existence of an owner, the Backend do not perform any verification when it receives a request to associate a tag to a given beacon introducing a security problem which is, any user can associate tags to any beacon without being its owner. This was not taken into account because it was out of the scope of this work;

Smart Place : Stores information about all smart places available, such as, name and description. It has a pointer to an instance of User, which is the user that created it;

Smart Place Instance : When an owner wants to install a given Smart Place, he creates an instance of a Smart Place. Owners can customize the title and the message that appears in the users' mobile devices notifications;

Tag : A Tag has a JSON object and has pointers to instances of Smart Place Instance and Beacon classes. With this pointers, when the end users app detects a beacon, it is possible to send a query to the backend in order to get the tag that is associated to that beacon.

Parse BaaS has SDKs available to many platforms and programming languages, such as, Android, iOS, Javascript, etc. In order to be able to make requests to the backend from the mobile apps and from the Javascript API the Parse Android and Javascript SDKs were used.

4.5 Mobile Applications

As already mentioned in section 4.2, we have used Android Studio to develop the mobile apps. This development environment allows developers to create multiple apps and modules inside the same project. Taking this into account, one project was created, with 2 apps and one library, as shown in figure 4.3. Since both apps use the Bluetooth receiver of the mobile device and make requests to the backend,

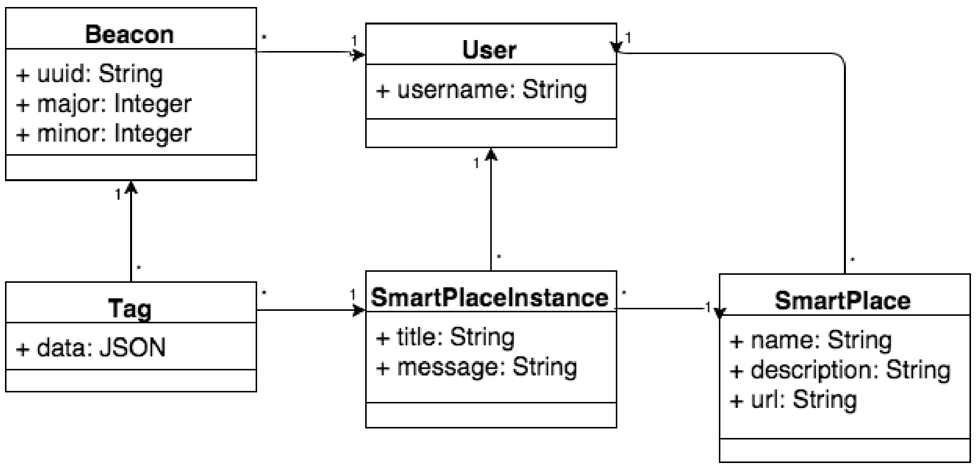


Figure 4.2: Data model stored in Parse BaaS

there is a library, that both apps depend on that offers some abstractions around the ibeacon protocol and the communication with Parse. Its implementation is detailed in section 4.5.1. This Android project depends on the following librarys:

AltBeacon [11] is a library that provides APIs to interact with beacons. We used this in our *smartplaceslib* to provide operations such as scanning for nearby beacons;

Parse Android SDK [12] was used to interact with our Backend;

Facebook SDK [13] was used to provide authentication using user's Facebook account in the owners mobile app.

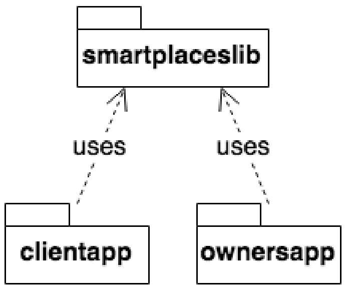


Figure 4.3: Android project's structure

4.5.1 Library

As already mentioned, a library *smartplaceslib* as shown in figure 4.3 was created in order to provide APIs that mobile apps can use. Because both apps use the Bluetooth receiver and interact with the backend, this library avoids having similar code in both apps. There are two important classes that this library offers:

IBeaconsManager : Offers methods to start scanning for beacons and change some settings such as, interval between each scan;

[11] http://github.com/AltBeacon/android-beacon-library
[12] http://parse.com/docs/android/guide
[13] http://developers.facebook.com/docs/android

AbstractParseDataStore : This class offers an abstraction of the Parse Android SDK. It is extended by two classes, *ClientParseDataStore* used in the end users mobile app and *OwnerParseDataStore* used in the owners mobile app. There are two different classes for each app because each app has different needs when interacting with the backend. Their common needs were encapsulated in methods in the *AbstractParseDataStore* class.

When IBeaconsManager class is used, it needs a BeaconScanCallback object which overrides a method that is executed when beacons are scanned because, the scanning is an asynchronous process. For instance, the code snippet in Listing 4.1 illustrates how to use IBeaconsManager class in a given Activity[14], to scan for nearby beacons.

```java
IBeaconsManager manager = new IBeaconsManager(this);
manager.startScan(this, new BeaconScanCallback() {
  @Override
  public void beaconsFound(Collection<BeaconInfo> beacons) {
    // Code to be executed after beacons are found
  }
})
```

Listing 4.1: Java code in an Android Activity to scan for nearby beacons

4.5.2 Mobile App for Owners

As stated in chapter 3, owners are the people who install beacons and configure Smart Places. This mobile Android app allows them to create an instance of a Smart Place. In terms of backend, they are creating an instance of Smart Place Instance class, explained in 4.4. In the android project it is called *ownersapp* and it uses the library *smartplaceslib* as shown in Figure 4.3. From the library, it uses the class IBeaconsManager to scan for nearby beacons and OwnersParseDataStore to allow owners to create Smart Place Instances and configure them. The user interface tries to follow the conventions of Material Design[15] which is a design language developed by Google™. This language states some conventions about how to design an UI. Each UI is implemented by a class that extends the Activity class provided by the Android API. An Activity represents a single task, that the user can do in an Android app.

In our Backend, we have the User class and it has a relation with the class Beacon which refers that the user owns beacons. Parse BaaS already provides authentication mechanisms using username and password or using social networks accounts such as Facebook and Twitter. To use this app owners have to log in using their Facebook accounts. To allow this authentication mechanis we just needed to include the Facebook SDK[16]and generate a secret key for our app using Facebook Developers website.

4.5.3 Mobile App for End Users

End users are the ones who actually use the Smart Places, that is, the users that have in their mobile devices an app that scans for nearby beacons and offer services provided by the Smart Places that were found. It is part of the project, as *clientapp*, as shown in figure 4.3. Similar to the owners app, described in 4.5.2, it uses the *smartplaceslib* to scan for nearby beacons and map them to the corresponding Smart Place Instances. When Smart Place Instances are found, the app shows a notification for each one.

[14]http://developer.android.com/guide/components/activities.html
[15]https://www.google.com/design/spec/material-design/introduction.html
[16]https://developers.facebook.com/docs/android

When the user click on the notification the app calls an Activity that contains a WebView. This Activity calls javascript functions on the WebView. To ease that process there is a class, SmartPlacesWebView that extends the original WebView class from the Android API. In that Activity there is an instance of the IBeaconsManager class that will scan for nearby beacons in order to try to find tags. As mentioned in section 4.4, each instance of Tag has a JSON object. That object must be sent to a Javascript function, for which developers have defined a callback that will handle it. The SmartPlacesWebView has methods to invoke javascript code running inside the webview. This way, the smart places app effectively combines the advantages of native apps with web apps, as compared in Section 4.1.

4.6 Developers API

In this solution, developers create their Smart Places using web technologies, such as HTML, CSS and javascript, because they run inside a WebView in the clientapp, as mentioned in section 4.5.3. This is separated from the Android project, because it does not have any dependency from it.

This API offer a global object, which is *SmartPlaces*, with several methods that developers use to define callbacks for some events. This library was developed in a modular way, that is, there are multiple javascript files that in the end, in the build process, are compiled in just one file. To manage this build process we used Grunt. There is a task that compiles all javascript files in just one, performs minification of that file and creates a new version, that is, creates a new tag in the repository and pushes those changes to it.

4.7 Implementation of Examples

In order to make sure that the Javascript API works as expected and that it can be applied to multiple kinds of smart places, two examples were implemented. These examples try to be a proof of concept, that is, demonstrate that using the Javascript API a web application can be aware of nearby tags, in the real world, and execute useful computation using those tags.

4.7.1 Smart Restaurant

The idea of this Smart Place is to offer an easy way for customers, in a restaurant, to place their orders using their mobile devices. We have used an existing solution created in the scope of a Master Thesis [17]. It has a REST API written in Hypertext Preprocessor (PHP) language. Our work here was to create a frontend that uses the Smart Places Javascript API and the provided API. This frontend was written in HTML and Javascript. It was deployed in the web server of Instituto Superior Técnico (IST). Similar to Smart Places Javascript API, the build process is managed by Grunt and there is a task to deploy this web application in the mentioned web server.

4.7.2 Smart Museum

The Smart Museum is another example that was created to use the Javascript API and verify that we can apply it to multiple applications. The main goal was to show some information about some objects in a museum.

To work with real data instead of creating our own museum we have used a real data source. The Walters Art Museum provides a REST API that any developer can use to get data about collections and objects. To use this API, we created an account in the API's website[17] and requested a token. This token is used in any request to allow the REST API to identify each individual request.

This web application was created using NodeJS[18] with Express[19] framework that allows to create an Hypertext Transfer Protocol (HTTP) server using javascript. It is currently deployed on Heroku[20] which is a cloud Platform as a Service (PaaS) that supports many programming languages and development stacks. Simillary to the other example, in section 4.7.1, this one also uses Grunt to manage the build process. Before the deployment, all javascript files that will run on the browser are concatenated in one file. Then, that file is minified.

4.8 Summary

In the previous chapter, we introduced our solution and the main components that are part of it. Here, we looked at each component and described its implementation.

First we introduced important decisions about the implementation of our Smart Places solution. Then, we described the architecture and its main components. We summarized the tools used in the development of this solution.

Another important component of our solution are the tags that are deployed in a given Smart Place. We decided to use BLE beacons using ibeacon protocol because nowadays, most of the smartphones are BLE enabled. Our solution does not require the user to acquire any extra hardware besides his mobile device.

Our solution stores information in a backend using Parse BaaS allowing us to only be concerned about the data model and not with the typical concerns of a distributed system such as, scalability, performance and deployment. In our data model, there is the concept of Smart Place. When owners configure a given Smart Place they are creating a Smart Place Instance. Smart Place Instances have tags which are represented by beacons.

Smart Places provide proximity-based services that could be supported through native or web apps. We discussed the advantages and limitations of native and web apps. The final decision was to have an embedded web browser, inside the mobile app for end users, that would allow to have access to any Smart Place without requiring users to install a native app for each one.

We have developed a Javascript library that offers an API to handle the events sent by the mobile apps, for instance, when a tag is found.

The Smart Restaurant and Smart Museum were implemented using web technologies, such as, HTML, CSS and Javascript. These examples contributed to the development of Developers API. While developing the Smart Restaurant, we wrote Javascript code to handle the events sent by the native mobile apps. From this code, it was possible to define an API that was released as a separated library. This library was used successfully in the Smart Museum example.

[17]http://api.thewalters.org
[18]http://nodejs.org
[19]http://expressjs.com
[20]http://www.heroku.com

Chapter 5

Evaluation

The evaluation focused on two aspects. First, we need to verify if the method to get the distance from a given beacon is reliable. Second, since we have a service running in background scanning for beacons from time to time we evaluated our solution in terms of battery consumption.

In section 5.1, we describe the setup used, such as the mobile device and the set of beacons. Then, we present two sets of experiments. The first one, presented in section 5.2, was performed in order to get a good insight about the reliability of the distance value we get from the beacons library API, to see if Smart Places that rely on the nearest beacon would work as expected. The other set of experiments, introduced in section 5.3, measured data transferred (sent and received) and power drain, to take conclusions about if the energy consumption overhead introduced by our solution would be acceptable in a daily basis usage and the relation between the data transferred, due to requests to the backend, and the battery consumption. For each set of experiments, we present the methodology such as the conditions and measures taken and the results. Finnaly, section 5.4 summarizes this chapter.

5.1 Setup

In the evaluation process, a smartphone and a set of three beacons were used. The smartphone was a Motorola™ Moto G[1]. This device has the following specifications:

CPU: Quad-core 1.2 GHz Cortex-A7[2]

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[3])

[1]http://www.gsmarena.com/motorola_moto_g-5831.php
[2]http://www.arm.com/products/processors/cortex-a/cortex-a7.php
[3]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 5.1: Experiments to get the accuracy of the method to get the nearest beacon

5.2 Nearest Beacon Detection

Our solution uses a library which allows 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. Next, related to this set of experiments, we describe the methodology used and summarize the results.

5.2.1 Methodology

The set of experiments, summarized in Table 5.1, try to test if the mobile app can detect the nearest beacon. 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.

In each scan, the app executes code that checks if the scanned beacons were already detected in a previous scan. If they were not, the data associated to those beacons is requested from the backend which, depending on the internet connection, can take some time. This time interval between each scan seemed reasonable to make the request and compute the result until a new scan occurs. With a much lower value, we could have a situation where another scan occurs while the previous one was not computed yet. 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 smartphone and the three beacons were disposed in a layout, where each beacon was equally distant from each other and the smartphone was close to one of them, as shown in Figure 5.1 where value *d* is the distance between beacons. The names below each beacon (ice, blueberry and mint), were defined by Estimote™ in the developer pack. In this first set of experiments, the value d starts at 50 centimeters and is increased by 50 centimeters in each experiment until the 4th one where it is 2 meters.

5.2.2 Results

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 accuracy of the mechanism that calculates the distance that the mobile device is from a given beacon. The results of these experiments are summarized in Table 5.2 where it is possible to see for each beacon how many times it was detected as the nearest one.

Taking into account the layout that was used (see Figure 5.1), the nearest beacon was the one with name "ice" (the blue one). From Table 5.2 it was possible to create the graphic shown in Figure 5.2 which shows that, as we increase the distance between beacons, the accuracy to detect the nearest

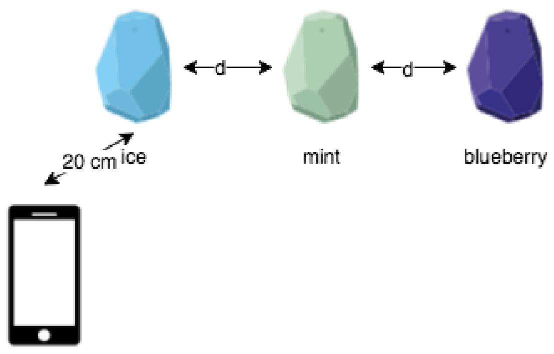


Figure 5.1: Layout used for the experiments to get the accuracy of the distance value

Experiments	Distance between Beacons (meters)	Beacons			Total scans	% Correct
		Ice	Mint	Blueberry		
1	0.5	11	7	4	22	50.00%
2	1.0	11	8	0	19	57.89%
3	1.5	14	7	0	21	66.67%
4	2	15	1	0	16	93.75%

Table 5.2: Results of measuring the accuracy to get the nearest beacon

beacon also increases. In all experiments, this beacon was detected as the nearest one at least 50% off all scans performed. The only difference between all experiments is the distance between beacons. We can conclude that, it is recommended that the beacons are at least, 1.5m or 2m distant from each other.

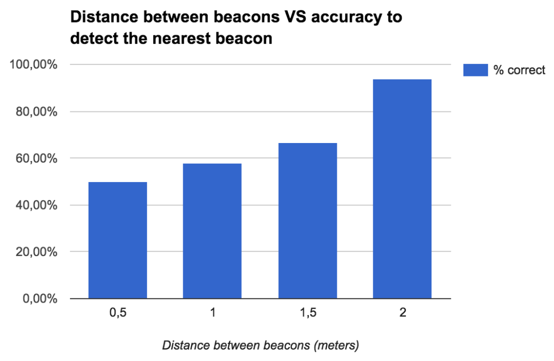


Figure 5.2: Relation between distance between beacons and accuracy to detect the nearest beacon

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. This error can be avoided using solutions such as Estimote Indoor SDK[4] that can be used to obtain the physical location of the user inside a building or a room. However, using it requires the user to place beacons around the room. Then, he needs to walk through the room in order for the app that is using the SDK to get the needed data to do the mapping. Only after this mapping it is possible to get the user's location. It is only available for iOS.

[4]http://github.com/Estimote/iOS-Indoor-SDK

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

Table 5.3: Summary of experiments to get the battery consumption when the mobile app is scanning for beacons in the background

5.3 Energy Consumption

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. We perfomed experiments to measure how much energy is drained only by our mobile app for end users. Similar to section 5.2 in the next sub sections we describe the metholodogy used and explain the results obtained in these experiments.

5.3.1 Methodology

Table 5.3 outlines the experiments performed to evaluate the battery consumption. Figure 5.3 shows the layout used for this group of experiments. We have used the same beacons as in the experiments described in section 5.2. The beacons are equally distant, 25 cm, from each other. The smartphone is at the same distance from the beacon in the middle, the green one named mint. We performed six 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 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 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, at each two and half minutes.

The mobile app for end users has a cache that stores the beacons that were already scanned. This way, repeated communications with the backend are avoided. Data communications, WiFi or 3G, are the major source of battery drain as suggested in studies such as [18]. To test both scenarios we would need more than three beacons spread along a big space. Instead, we simulated the user walking passing by multiple Smart Places just by cleaning the cache, at each scan process. This forced the app to make requests to the backend each time the three beacons were detected, allowing us to understand the

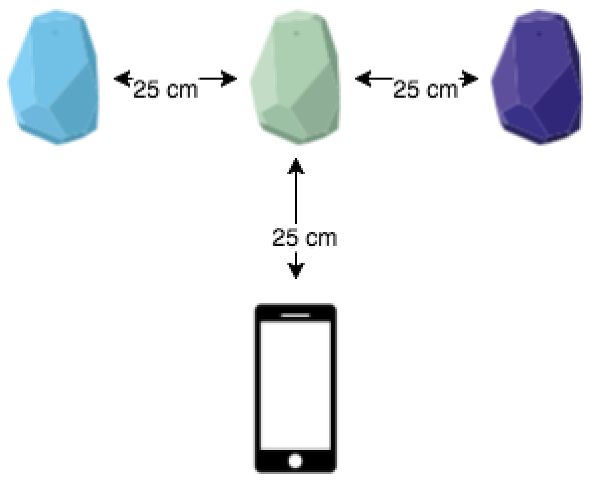


Figure 5.3: Layout used for the experiments to get the battery consumption

Variables	**Results**			
Data connection type	WiFi		3G	
Interval between each scan	2m30s	5m	2m30s	5m
Computed power drain (%)	0.00	0.00	0.40	0.29
Data transferred (KB sent and received)	34.96	31.10	21.67	35.28

Table 5.4: Results of the experiments to get the battery consumption in the scenario where the user stays in the same Smart Place

impact of these requests on the power drain. We used Battery Historian[5] to measure the power drain and how much data was transferred (sent and received). This tool allowed us to get the percentage of power drain.

5.3.2 Results

After we tested the first scenario, when the user stays in the same Smart Place we got the results shown in Table 5.4, where we can see, for each variation of data connection type (WiFi and 3G) and interval between each scan, the values obtained for power drain and data transferred. From these results we obtained the graphic in Figure 5.4. It is possible to see that using WiFi we got no power drain. However, using 3G it drained 0.29% and 0,40% of the total battery available. After these first results we concluded that the biggest overhead comes from using 3G connection.

After the first set of experiments we tested Facebook[6] 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[7]. 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 5.5 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.

[5]https://developer.android.com/tools/performance/batterystats-battery-historian/index.html
[6]http://play.google.com/store/apps/details?id=com.facebook.katana
[7]http://www.forbes.com/sites/jaymcgregor/2014/11/06/facebook-and-instagram-are-killing-your-phones-battery-heres-a-simple-fix

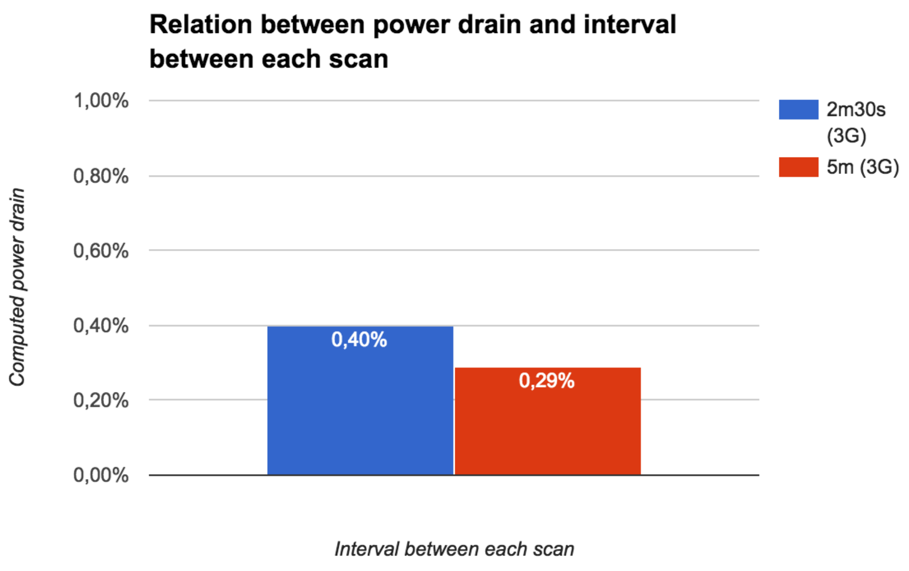


Figure 5.4: Relation between power drain and interval between each scan in the scenario where the user stays in the same Smart Place

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

Table 5.5: Battery consumption of Facebook app

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 backend. Table 5.6, shows the results for this scenario. Here, we got more power drain than in the previous scenario. These results shows that, the more communication with the backend is required, more power our solution drains. Once more, using WiFi, we have got almost zero power drain. However, similar to the results in the previous scenario, there is more power drain using 3G. Graphic shown in Figure 5.5 shows that, 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 decrease 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.

Variables	Results			
Data connection type	WiFi		3G	
Interval between each scan	2m30s	5m	2m30s	5m
Computed power drain (%)	0.01	0.01	2.71	2.00
Data transferred (KB sent and received)	71.44	67.86	229.96	138.26

Table 5.6: Results of the experiments to get the battery consumption in the scenario where the user is moving along multiple Smart Places

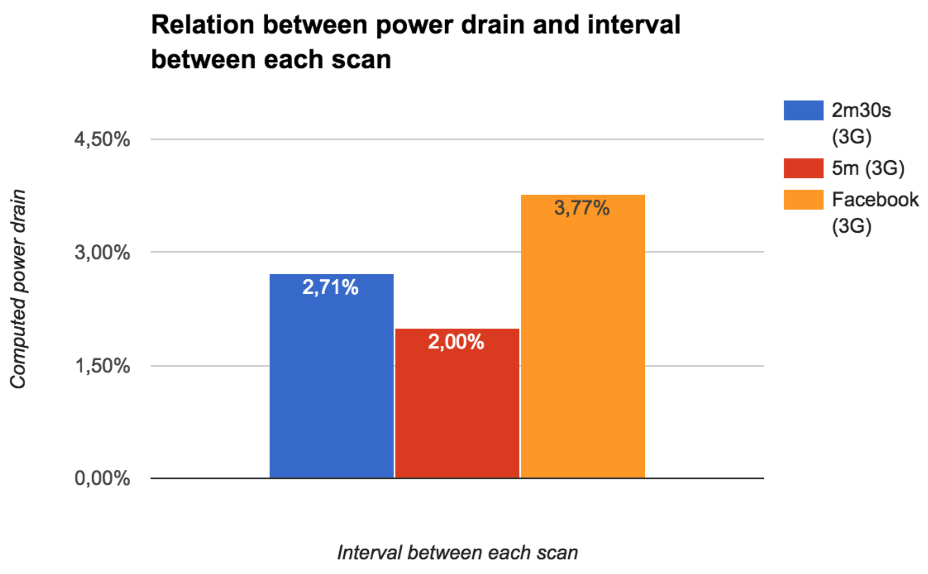


Figure 5.5: Relation between power drain and interval between each scan in the scenario where the user moves along multiple Smart Places

5.4 Summary

After developing the Smart Places solution, we performed an evaluation. We have used a Motorola™ Moto G smartphone and three beacons from Estimote™ and we performed two sets of experiments.

The mobile app for end users scans for nearby beacons and requests information from the backend. Our experiments showed that, to build applications that rely on detecting the nearest beacon, the beacons need to be, at least, 1.5m distant from each other. As we raise the distance between beacons, the nearest beacon is successfully detected.

The mobile app for end users scans for nearby beacons in background. As any service running in background, it can introduce an overhead in terms of energy consumption. We performed a set of experiments to get the power drain and data transferred (sent and received) by the mobile app. We have tested two different scenarios. A scenario where the user stays in the same Smart Place and another one where he is always moving from one Smart Place to another. Each experiment runned for one hour with a scanning period of 5 minutes or 2 and 30 seconds. For each scenario, we tested WiFi and 3G as data connection means. The results showed that the power drain is bigger when using 3G. When using WiFi, the power drain is almost zero. In the scenario where the user does not move, the power drain is less than 1%. However, in the scenario where the user is always moving, which implies more requests to the backend, because new Smart Places are detected in each new scan, the power drain can be more than 2%. We have measured the power drain of Facebook app, because it is one of the popular apps and also have services running in background. The power drain of this app was used as a reference in order to compare with our solution in the worst case, when it needs to make requests to the backend in each scan. Comparing the worst power drain in our solution with the Facebook app, we can say that the battery consumption of our solution is acceptable but needs to be improved.

Chapter 6

Conclusion

We have developed the Smart Places solution which offers a tool for developers to allow them to create proximity-based services and an interface for end users and owners. Using our solution end users have access to any proximity-based service based on the concept of Smart Place, which we introduced in this work and owners can manage their Smart Places. However, our solution is not perfect and there are limitations, which can be improved in the future. In this chapter, we conclude this dissertation, summarizing the important conclusions of our work in section 6.1 and discussing some of the limitations and future work in section 6.2.

6.1 Contributions

A Smart Place is an area with tags that mobile devices can detect and offer different possibilites to the users according to each tag. In this dissertation, we introduced this concept, which we consider a simpler way to look at proximity-based applications. However, these tags need to be provided by a location technology. That technology should be compatible with most used mobile devices. We analyzed some location technologies, using a taxonomy[1], in order to try to find a technology that would be a best fit for our concept of Smart Place. We chose BLE beacons using iBeacon protocol because it is compatible with any smartphone with Bluetooth version 4.0 or above. The owners of Smart Places are responsible to place those beacons in the right place. Users do not need any extra hardware. They only need 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 to provide proximity-based services, using web technologies, such as, HTML, 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 to handle tags neither the backend where the information about Smart Places and their tags is stored. There is a mobile app for end users, anyone with a mobile device, capable of reacting to tags in a Smart Place. Since each Smart Place is a web application they can run inside an embedded web browser in this mobile app, allowing users to have access to any Smart Place withouth 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.

After the implementation phase, we evaluated the system. We performed two sets of experiments. The first set of experiments was performed in order to check if this distance value was reliable. We have reached the conclusion that, if we want to rely on getting the nearest tag they need to be at least 1.5m distant from each other. The other set of experiments was about battery consumption. Since our mobile app has a service running in background scanning for nearby tags from time to time we need to get a good insight about the impact of that service in terms of energy. 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 more than 2% of total battery, in just one hour and it consumed 3.77%. We have compared the power drain values in the worst scenario with one popular app, which is Facebook that also have services running in background. We can conclude that the battery consumption is acceptable in a daily basis but there is room for more improvement.

6.2 Future Work

Our solution tries to target developers, owners and users of Smart Places. However, our solution has some limitations that can be improved in the future. Also, there are some features that were not implemented.

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. When owners are configuring their Smart Places, they can choose from a list. The Smart Restaurant and Smart Museum examples were introduced in the backend manually using Parse dashboard. There is no interface for developers to add a Smart Place to this list. A future improvement could be to develop 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.

One issue with our solution is the energy consumption. As the evaluation shows, using 3G, our mobile app for end users drains more than 2% of power in just one hour. Comparing with the power drain of Facebook which is a popular 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 backend which can result in less power drain than the actual solution. Also, we need to test other values smaller than 2 minutes and 30 seconds for the scanning period because it might be possible to find a smaller value resulting in an acceptable power drain. Most recent mobile devices are Fourth Generation (4G) enabled. This data connection mean was not tested due to limitations in terms of resources available during the development of this dissertation.

As previously mentioned we performed experiments in order to test our Smart Places solution in terms of detecting the nearest beacon and battery consumption. However, since we target end users and developers we need to evaluate the system in terms of usability. In the future we could perform usability tests to check if the mobile apps are easy to use. Besides end users there are experiments that can be done with developers to verify if we accomplish one of our goals which is making the development of Smart Places an easy process. For instance, we could write exercises that developers would do using our API.

There was no concern for security requirements. For exampple, when the Backend receives a request to associate a tag to a given beacon, it does not perform any verification, that is, it is not taken into account if the user trying to make this association is the owner of the beacon. Any user using the mobile app for owners can associate tags to beacons belonging to any Smart Place. In the future, the Backend needs to be improved in order to be more secure to not allow anyone to modify existing tags.

We picked BLE as the technology to support tags in a given Smart Place. However, developers might want to build Smart Places that make use of other location technologies such as, QR codes or NFC. Our solution only supports BLE. This might not be the right technology for some applications. One future improvement would be to add support for more technologies and let developers choose the right one for the application they are building

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 a SDK available for the three most used mobile OSs, iOS, Android and Windows Phone, to allow to integrate our functionality in existing apps.

Mobile devices are evolving and they are being able to get more context information and perform other tasks such as payments. 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. Proximity-based applications can offer several possibilities. For instance, companies that have online and physical stores could achieve more integration of offers such as, customers have access to exclusive discounts when they approach a given tag or a set of tags. Another possibility is to have tags in a supermarket, which could trigger reminders in an app that customers use to store their shopping list. For instance, the customer approaches the section where there is milk and this is on his shopping list. The app would remind him that he needs milk. 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.

Bibliography

[1] J. Hightower and G. Borriello, “Location systems for ubiquitous computing,” *Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[2] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global positioning system: theory and practice.* Springer Science & Business Media, 2013.

[3] E. Costa-Montenegro, F. J. Gonzalez-Castano, D. Conde-Lagoa, A. B. Barragans-Martinez, P. S. Rodriguez-Hernandez, and F. Gil-Castineira, “QR-Maps: An efficient tool for indoor user location based on QR-Codes and Google maps,” *2011 IEEE Consumer Communications and Networking Conference, CCNC’2011*, pp. 928–932, 2011.

[4] C. Charoensiriwath, N. Surasvadi, S. Pongnumkul, and T. Pholprasit, “Applying QR code and mobile application to improve service process in Thai hospital,” in *2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 114–119, IEEE, July 2015.

[5] V. SHARMA, P. GUSAIN, and P. KUMAR, “Near field communication,” in *Proceedings of the Conference on Advances in Communication and Control Systems-2013*, Atlantis Press, 2013.

[6] A. Farshad, J. Li, M. K. Marina, and F. J. Garcia, “A microscopic look at wifi fingerprinting for indoor mobile phone localization in diverse environments,” in *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, pp. 1–10, IEEE, 2013.

[7] C. Gomez, J. Oller, and J. Paradells, “Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology,” *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012.

[8] S. Bluetooth, “Bluetooth core specification version 4.0,” *Specification of the Bluetooth System*, 2010.

[9] M. S. Gast, *Building Applications with IBeacon: Proximity and Location Services with Bluetooth Low Energy.* ” O’Reilly Media, Inc.”, 2014.

[10] 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*, pp. 11–19, ACM, Nov. 2014.

[11] L. Chen, I. Hussain, R. Chen, W. Huang, and G. Chen, “BlueView,” in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp ’13 Adjunct*, (New York, New York, USA), pp. 143–146, ACM Press, Sept. 2013.

[12] V. Antila and J. Polet, “ContextCapture,” in *Proceedings of the 13th international conference on Ubiquitous computing - UbiComp ’11*, (New York, New York, USA), p. 585, ACM Press, Sept. 2011.

[13] F. Ben Abdesslem and A. Lindgren, “Demo: mobile opportunistic system for experience sharing (MOSES) in indoor exhibitions,” in *Proceedings of the 20th annual international conference on*

Mobile computing and networking - MobiCom '14, (New York, New York, USA), pp. 267–270, ACM Press, Sept. 2014.

[14] A. Krevl and M. Ciglaric, "A framework for developing distributed location based applications," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, p. 6 pp., IEEE, 2006.

[15] S. Seely, *SOAP: Cross Platform Web Service Development Using XML*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[16] 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*, pp. 151–158, IEEE, Oct. 2012.

[17] L. R. da Costa Carvalho, "SLOC: Sistema para Serviços Baseados na Localização," Master's thesis, Instituto Superior Técnico, Lisbon, Portugal, 2015.

[18] 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), p. 29, ACM Press, Apr. 2012.