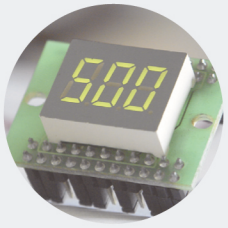
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






DESENVOLVIMENTO DE SOLUÇÃO PARA DETEÇÃO CONTÍNUA DE BEACONS EM DISPOSITIVOS MÓVEIS

JOSÉ PEDRO E SILVA
Outubro de 2017

POLITÉCNICO
DO PORTO

DESENVOLVIMENTO DE SOLUÇÃO PARA DETEÇÃO CONTÍNUA DE BEACONS EM DISPOSITIVOS MÓVEIS

José Pedro e Silva

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2017

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: José Pedro e Silva, Nº 1100445, 1100445@isep.ipp.pt

Orientação científica: Doutor, José Ricardo Teixeira Puga, jtp@isep.ipp.pt

Co-orientação científica: Doutor, Maria Judite Madureira da Silva Ferreira, mju@isep.ipp.pt

Empresa: Xhockware.SA

Supervisão: César Teixeira, cesar.teixeira@youbeep.com

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

7 de outubro de 2017

Agradecimentos

Aproveito esta oportunidade para agradecer a todos os que contribuíram para o meu crescimento enquanto pessoa e enquanto estudante. Àqueles que me apoiaram ao longo do desenvolvimento do projeto, em especial ao César Teixeira pela sua orientação. Ao Eng. José Puga e à Eng. Judite pelo acompanhamento prestado ao longo deste processo. À Xhockware e à sua equipa. Aos meus amigos e colegas de faculdade, porque acompanhado sempre foi mais fácil. À minha família que sempre esteve comigo e que se sente orgulhosa por mais esta etapa. A Deus.

A todos, um grande OBRIGADO.

Resumo

Numa era digital, quase todas as indústrias tiveram que se reinventar e seguir as tendências tecnológicas. A tecnologia Beacon aparece neste contexto como uma das ferramentas para personalizar a comunicação com os clientes, mas também como fonte de dados para perceber o seu comportamento.

Esta tese destina-se a descrever o estudo deste tipo de tecnologia e a desenvolver uma solução capaz de executar o rastreamento contínuo de dispositivos *beacon* em dispositivos móveis. Várias outras tecnologias relacionadas com o desenvolvimento de aplicações móveis também foram exploradas, com foco nas *frameworks* de desenvolvimento de aplicações híbridas, como o Cordova. Outro assunto de relevo desta tese foi a pesquisa e implementação de um mecanismo que permita que um aplicativo Android mantenha os serviços que estão a ser executados em segundo plano, independentemente do estado da aplicação principal.

O uso de uma arquitetura baseada em *middleware*, suportada por um *broker* RabbitMQ, origina uma solução escalável e robusta, juntamente com um sistema dinâmico de encaminhamento de mensagens. Esse tipo de arquitetura permite a fácil interoperabilidade entre outros componentes da arquitetura, como os microserviços responsáveis pelo processamento de dados fornecidos pelo serviço Android implementado.

Este projeto fornece uma solução capaz de ultrapassar as limitações dos sistemas operativos móveis, com foco no Android, em relação à execução de tarefas em segundo plano. É ainda realizada uma comparação entre diferentes abordagens e são apresentadas algumas funcionalidades da solução implementada

Palavras-Chave
Beacons, BLE, Wifi, RabbitMQ, Cordova, Scanner

Abstract

In the current digital age, almost every industry had to reinvent itself and follow technological trends. The Beacon technology appears in this context as one of the tools to customize the communication with the customers, but also as a data source for perceiving their behavior.

This thesis is intended to describe the study of this type of technology and to develop a solution that is able to continuously tracking of beacons on mobile devices. Several other technologies, related to the development of mobile applications were also explored, with a focus on the hybrid application development frameworks, such as the Cordova Framework. Another critical part of this thesis was the research and implementation of a background service, which allows an Android application to keep services that are running in background, independently of the state of the main Android application.

The usage of a middleware based architecture, supported by a RabbitMQ broker, gives the project a scalable and robust solution, along with a dynamic message routing system. This kind of architecture allows the easy interoperability among other components of the architecture, such as the microservices responsible for the processing of data provided by the implemented Android service.

This project provides a solution to overcome the limitations of mobile operating systems, focusing on the Android operating system, in terms of executing tasks in background. A comparison of different approaches is also given, presenting some features of the chosen solution.

Keywords
Beacons, BLE, Wifi, RabbitMQ, Cordova, Scanner

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AMQP - *Advanced Message Queuing Protocol*

AP - *Access Points*

API - *Application Programming Interface*

BLE - *Bluetooth Low Energy*

CRC - *Cyclic Redundancy Check*

FCM - *Firebase Cloud Messaging*

FIFO - *First In First Out*

GAP - *Generic Access Profile*

GPS - *Global Position System*

HTTP - *Hypertext Transfer Protocol*

IDE - *Integrated Development Environment*

NFC - *Near Field* Communication

PDU - *Protocol Data Unit*

RFID - *Radio-frequency Identification*

RSSI - *Received Signal Strength Indicator*

TLS - *Transport Layer Security*

UI - *User Interface*

UUID - *Universally Unique Identifier*

1. INTRODUÇÃO

O progresso tecnológico está a transformar todas as indústrias que conhecemos e o retalho não é exceção, sendo um dos sectores que tem sofrido maiores alterações, bem como um dos que mais potencial tem para servir os clientes de formas inovadoras e de proximidade. Dada a necessidade de se reinventar têm desenvolvido sido coordenados esforços para novas experiencias com a finalidade de reforçar a fidelização do cliente.

Uma destas tendências tecnológicas são os *beacons*, tendo um papel importante como ferramenta de personalização da comunicação com o cliente, mas também como fonte de dados para perceber os seus comportamentos.

Para este cenário contribui a emancipação dos *smartphones*, sendo que os telemóveis deixaram de ter unicamente a função de efetuar comunicações e passaram a ser utilizados para finalidades lúdicas, somando também cada vez mais funcionalidades que nos permitem efetuar tarefas do dia a dia.

Neste contexto conseguir automatizar o processo de rastreamento de dispositivos *beacon* nos dispositivos móveis fazendo com que o processo não esteja dependente do utilizador usar uma aplicação, torna-se relevante para o sucesso da implementação desta tenologia nestes espaços.

1.1. CONTEXTUALIZAÇÃO

Este projeto surge no seio da Xhockware SA, uma *start-up* portuense, cujo o seu produto, o Youbeep, é uma solução patenteada de *mobile shopping & checkout*, com o objetivo de melhorar a experiência de compra, fazendo poupar tempo e dinheiro e que já se encontra em funcionamento em várias lojas em território nacional.

O Youbeep centra-se numa aplicação para *smartphone* cuja a sua principal funcionalidade é permitir ao utilizador efetuar a leitura dos códigos de barras dos produtos que pretende comprar, fazendo com que a efetivação da compra seja mais célere. A aplicação tem outras funcionalidades como consultar o histórico de compras, gerir listas de compras ou consultar promoções personalizadas, entre outras. Esta solução permite ainda a integração com POS antigos, dando-lhe novas funcionalidades e o controlo do comportamento das vendas.

Neste contexto nasce a ideia de desenvolver uma solução que permita interagir com o utilizador quando este se encontra dentro de loja tendo em conta a sua localização, aproveitando uma tecnologia já existente, os *beacons*.

Para isso tornou-se necessário estudar como construir uma solução que funcionasse sem intervenção do utilizador e fosse adaptada à realidade da empresa.

1.2. OBJETIVOS

O objetivo principal do presente projeto é conseguir desenvolver o suporte para aquisição de dados sobre dispositivos *beacons* tornando-o compatível com as plataformas utilizadas na Xhockware tornando possível o desenvolvimento de soluções de alto nível assentes sobre lógica de negócio. Assim os objetivos dividem-se pelos seguintes pontos:

- Estudo dos dispositivos *beacons* e do seu protocolo;
- Desenvolvimento de solução para a deteção de *beacons* em dispositivos móveis em *background* e independe do estado da aplicação móvel Youbeep;
- Integração da solução com a *framework* Cordova através de um Cordova *Plugin*;
- Envio dos resultados para a rede para possível tratamento e armazenamento em base de dados

1.3. ORGANIZAÇÃO DO RELATÓRIO

No Capítulo 1, designado de "Introdução", é feito o enquadramento do projeto, introduzindo o tema, definindo os objetivos que se pretendem atingir e apresentar cada um dos capítulos que o constituem.

No Capítulo 2, chamado de "Beacons", é apresentada esta tecnologia, iniciando-se por uma perspetiva funcional onde é explicado a utilidade dos dispositivos, onde podem ser aplicados e como podem ser adquiridos. De seguida, numa perspetiva técnica, é apresentado o seu funcionamento e são apresentados os vários protocolos, são identificadas as suas debilidades e é feito o enquadramento desta tecnologia com outras.

No Capítulo 3, nomeado de "Tecnologias", são apresentadas as restantes tecnologias que tomam parte no projeto, de forma a corresponder com os requisitos deste. O capítulo inicia-se com análise do desenvolvimento de aplicações móveis, passando pela comparação entre as aplicações nativas e híbridas e de algumas *frameworks* para o desenvolvimento de aplicações híbridas. Depois é estudado o comportamento do Android na execução de aplicações/serviços em segundo plano. O capítulo termina com uma apresentação do RabbitMQ e do AMQP (Advanced Message Queuing Protocol), sendo antes feita uma introdução sobre *middleware*.

No Capítulo 4, intitulado de "Especificação Funcional", é feita uma síntese da estrutura da solução pretendida, assim como do seu funcionamento, dando uma perspetiva básica sobre os processos.

No Capítulo 5, denominado de "Desenho e Implementação, é apresentado o projeto tal como está implementado. O capítulo inicia-se por uma abordagem geral da sua arquitetura, sendo de seguida apresentado cada um dos módulos que o constituem onde é abordada a sua estrutura, a sua implementação e o resultado final.

No Capítulo 6, denominado de "Testes e Resultados", a abordagem escolhida para o desenvolvimento do *plugin* de *scanning*, que cumpre os requisitos do projeto, é comparada com outras abordagens, pretendendo-se comprovar esta escolha.

No último capítulo, designado de "Conclusão", são reunidas as principais conclusões e perspetivados possíveis desenvolvimentos futuros.

2. BEACONS

Neste capítulo pretende-se apresentar a tecnologia *beacon* na sua componente funcional e técnica. Na componente funcional pretende-se explicar a sua utilidade e enquadramento nos espaços onde pode ser aplicado. Na componente técnica pretende-se apresentar o funcionamento destes dispositivos, entender os protocolos, identificar as suas debilidades e perceber o seu enquadramento com outras tecnologias já existentes.

2.1. INTRODUÇÃO

Os *beacons* são pequenos dispositivos, alimentados a bateria, que assinalam a sua presença com a emissão repetida de informação, utilizando o protocolo *Bluetooth Low Energy* (BLE) que é detetável pelos dispositivos móveis (*smartphones, tablets*) ou compatíveis [1]. Na Figura 1 é apresentada uma representação de um *smartphone* a detetar um *beacon.*

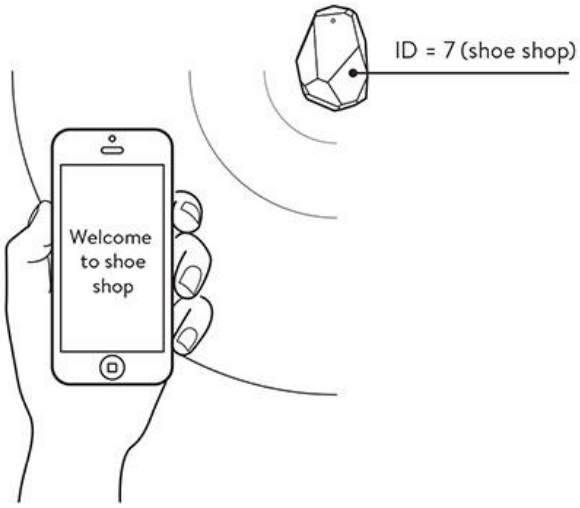


Figura 1. Ilustração representativa de um *smartphone* a detetar um *beacon* [2]

Normalmente estes dipositivos emitem informação estática, como a sua identificação, contudo em alguns casos esta informação pode ser dinâmica emitindo dados sobre o seu ambiente (temperatura, humidade, nível das baterias, etc) [1].

Como exemplificado na Figura 2, o *beacon* emite informação, identificada como "ID", que ao ser detetada pelo *smartphone* faz com que este perceba a sua posição relativa em relação ao *beacon*. Ao passar esta informação para o servidor este emite uma ordem para efetuar uma ação, neste caso uma promoção relativa a sapatos.

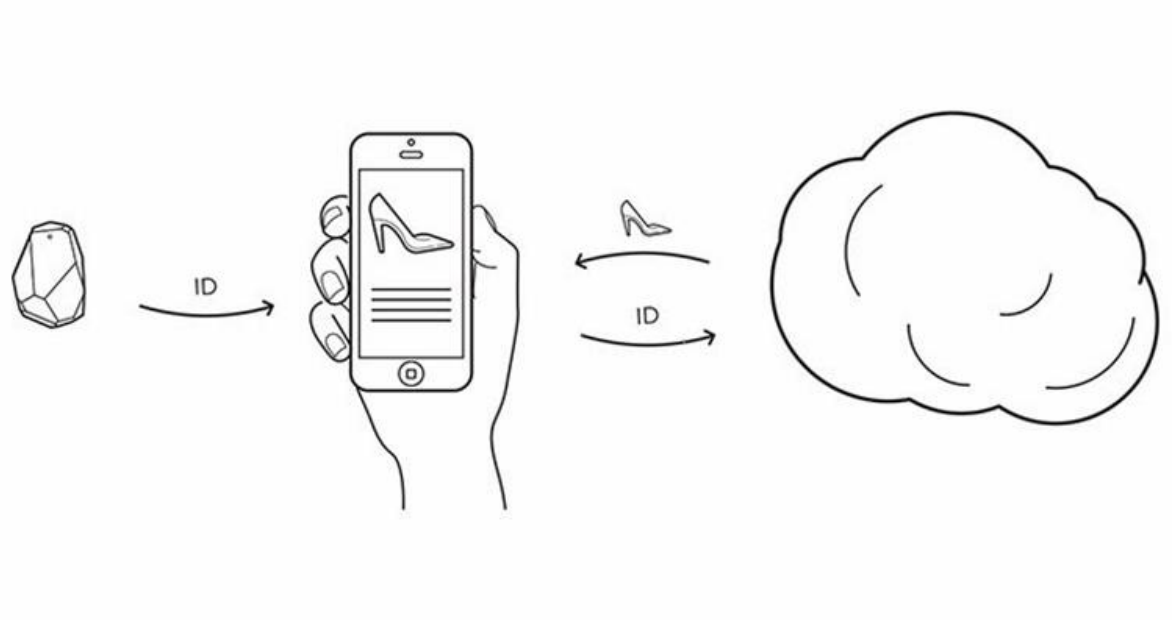


Figura 2. Esquema demonstrativo do funcionamento da interação entre um *Beacon* e um *smartphone* [2]

Esta tecnologia foi apresentada pela Apple em 2013, assim como o seu protocolo, o iBeacon. O seu objetivo é oferecer aos consumidores, de um espaço comercial, campanhas personalizadas através da sua localização nesse mesmo espaço, utilizando dispositivos baratos, pequenos e alimentados a baterias. Desde então a tecnologia foi-se desenvolvendo assim como as soluções e os fabricantes [1][3].

2.2. UTILIZAÇÃO DA TECNOLOGIA *BEACON*

A principal utilização da tecnologia *beacon* é no *marketing* dentro de loja, ou seja, na ativação de campanhas numa loja, mas o seu âmbito está para além desse uso, sendo também utilizado em aeroportos, túneis, museus, entre outros [1].

2.2.1. UTILIZAÇÃO DE *BEACONS* PARA *MARKETING* DENTRO DE LOJA

A utilização de *beacons* dentro de uma loja possibilita integrar vários elementos de *marketing* direcionados num só, tendo como recetor o *smartphone* do cliente. As possibilidades de utilização desta tecnologia dentro de uma loja são várias, podendo abranger várias áreas do retalho e usufruindo de outros métodos já enraizados de recolha de dados e de *marketing*, como os cartões de fidelização. O uso desta tecnologia permite direcionar e focar as ações de *marketing* nos hábitos e gostos do cliente [4].

O tipo de utilização destes dispositivos possibilita, por exemplo [4], [5] :

- Notificar o cliente das promoções existentes nessa loja;
- Direcionar o cliente para os setores da loja onde se situam os produtos que pretende;
- Fomentar o consumo de produtos novos ou em fim de *stock;*
- Notificar produtos em promoção num determinado setor;
- Com o emparelhamento com o cartão de cliente possibilita efetuar promoções direcionadas conforme os produtos que mais consome;
- Chamar um assistente;
- Conhecer os hábitos do cliente dentro do espaço e assim conhecer a distribuição temporal de um cliente nas compras.

Na Figura 3 encontram-se representados alguns dos exemplos acima descritos.

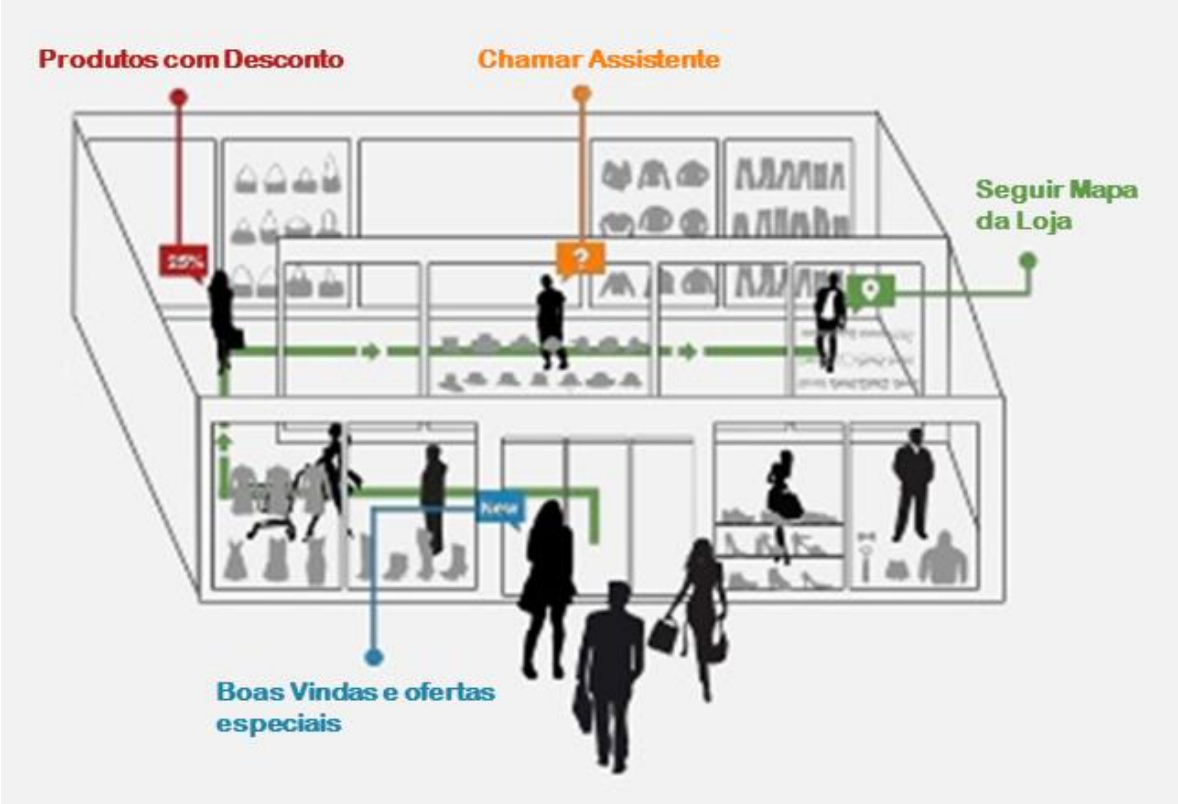


Figura 3. Representação de algumas implementações executadas em *beacons* dentro de loja [4]

O facto de o *smartphone* ser um elemento cada vez mais presente no dia-a-dia das pessoas, também permite que o seu uso, na execução das tarefas diárias, como as compras, não seja um entrave para o sucesso na utilização desta tecnologia nesta área.

2.2.2. UTILIZAÇÃO DE BEACONS EM EXPOSIÇÕES E MUSEUS

A utilização de *beacons* em exposições e museus permite uma maior interação do visitante com a exposição. Com a utilização desta tecnologia, facilmente se consegue mostrar informação adicional, ou até apresentações interativas sobre o espaço ou obra que está a ser visitada ou vista. Outra das possibilidades da utilização desta tecnologia passa por localizar o visitante no espaço e indicar-lhe o percurso por entre os vários pontos da exposição [6][7].

Na Figura 4, é apresentado um ecrã da aplicação do National Slate Museum, de Gales, a mostrar os artefactos da exposição que estão perto deste dispositivo. Este museu, em 2014, implementou 25 dispositivos *beacon* espalhados pelas suas instalações, oferecendo vários recursos de interação com os visitantes, como a explicação pormenorizada dos elementos

que integram a exposição, vídeos a demonstrar o seu funcionamento assim como fotos apresentando o antes e o depois de determinados espaços do museu. Tudo isto com recurso a um iPad e aos *beacons* instalados [6] [7].

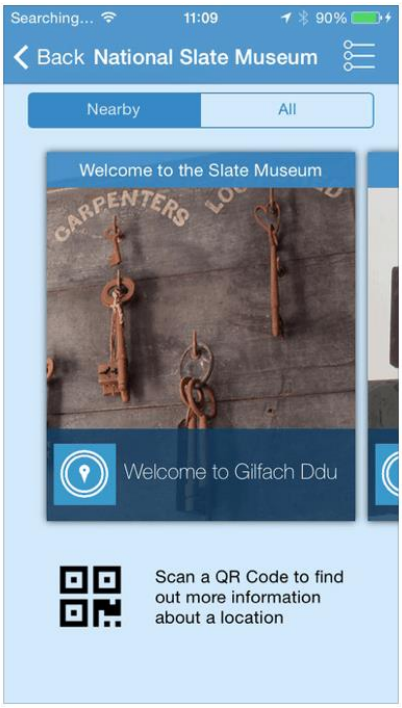


Figura 4. Aplicação do Museu Nacional de Gales [6]

2.2.3. UTILIZAÇÃO DE *BEACONS* EM AEROPORTOS

O aumento do tráfego aéreo, e consequente aumento do número de passageiros, fez com que os aeroportos se tornassem espaços em que por vezes existe um grande aglomerado de pessoas, tornando-se difícil a comunicação com os passageiros, por isso, já são vários os aeroportos que utilizam os *beacons* para comunicar e guiar os passageiros entre os vários pontos do aeroporto, levando a um embarque/desembarque mais eficaz e cómodo [5].

2.2.4. UTILIZAÇÃO DE *BEACONS* PARA LOCALIZAÇÃO EM TÚNEIS

Uma das debilidades da navegação GPS é a perda de sinal dentro de túneis. Para isso a *Waze*, uma aplicação gratuita de navegação para smartphone que funciona com base nos dados reportados pelos seus utilizadores [8], criou um plano para a instalação de *beacons* dentro de túneis para assim garantir que os seus serviços de localização consigam monitorizar ininterruptamente a posição do veículo e permitir guiar o condutor mesmo dentro do túnel. Estes dispositivos podem ser adquiridos por municípios e concessionários, tendo um custo

de US 28,50$ por *beacon* e sendo necessário aproximadamente 42 dispositivos por milha (apx. 1,609 km). Esta tecnologia já se encontra instalada em dois túneis na Califórnia e outro em Israel [5].

Na Figura 5 é apresentado o ecrã para ativação desta funcionalidade da aplicação *Waze*.

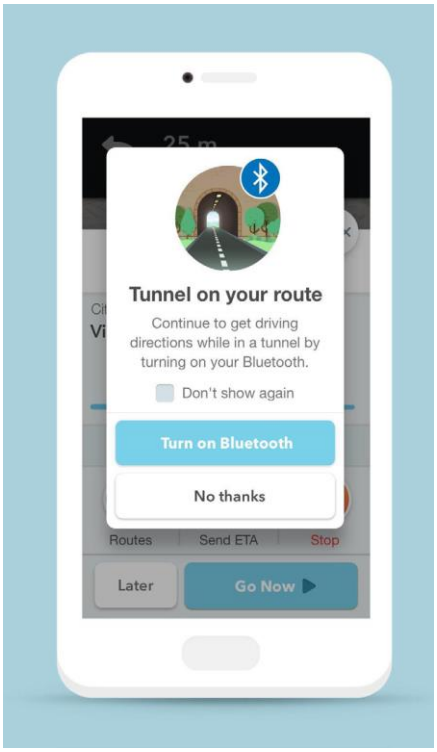


Figura 5. Ecrã da aplicação *Waze* apresentando o pop-up para ativação do rastreamento de *beacons* [5]

2.3. DISPOSITIVOS *BEACON* NO MERCADO

O mercado dos dispositivos *beacon* é variado e constituído por variados fabricantes de dispositivos. O seu modo de funcionamento é muito idêntico entre os vários fabricantes dependendo apenas dos *standards* que assumem (as características técnicas dos standards serão abordadas no ponto 2.5), destes os mais conhecidos são o iBeacon, o Eddystone e o Altbeacon [1].

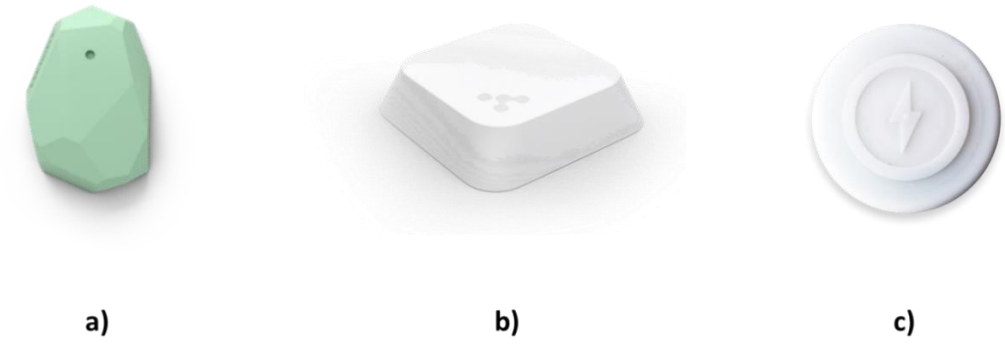
Quanto aos vendedores e fabricantes, esses são muitos e variados. Existem variadas lojas online que, com a concorrência do mercado asiático, se dedicam à venda destes dispositivos, destacam-se a Estimote, a Kontatkt.io e a Radius Networks. Alguns destes, além de disponibilizarem os dipositivos, também disponibilizam uma plataforma para gerir os dipositivos e desenvolver o serviço prestado [9][10][11].

Na Tabela 1 é apresentado um quadro comparativo entre os principais fornecedores de *beacons*, com dados obtidos a partir dos sites destes.

Tabela 1. Quadro Comparativo entre os principais fornecedores de *beacons* [9][10][11]

FORNECEDOR	STANDARD	PLATAFORMA	CUSTO (DISPOSITIVO BÁSICO)
estimote	Eddystone	Sim	US $20
kontakt.io	iBeacon/Eddystone	Sim	US $20
Radius Networks	iBeacon/Eddyston/ AltBeacon	Não	US $10

Na Figura 6 são apresentados alguns dos *beacons* disponíveis nas lojas online dos fornecedores acima mencionados.



a) b) c)

Figura 6. Exemplos de *beacons* de diferentes fornecedores disponíveis nas suas lojas a) Estimote [9] b) Kontakt.io [10] c) Radius Networks [11]

Após esta abordagem funcional sobre a tecnologia e dipositivos *beacon*, será feita uma abordagem técnica para se conhecer melhor as suas especificidades.

2.4. BLUETOOTH LOW ENERGY (BLE) – BROADCAST ADVERTISE

O *Bluetooth Low Energy* (BLE) é uma evolução do protocolo *Bluetooth* convencional, sendo a 4ª versão, embora não existam muitas diferenças ao nível das especificações em relação às versões anteriores. Os grandes objetivos desta versão do protocolo passam por dar capacidade de comunicação a dispositivos que normalmente utilizam comunicações por cabo, ou que não têm comunicação, focada nos dispositivos portáteis, mas com necessidade de ter baixo custo. Isto levou ao aumento da simplicidade do protocolo e a desenvolver dispositivos mais simples e com baixo consumo energético, descorando no alcance e na velocidade de transmissão [12].

Nos mecanismos de baixo consumo, em grande parte, a poupança energética é obtida através de ciclos *sleep* que apenas são interrompidos quando o controlador necessita de efetuar qualquer ação. Como o BLE é um protocolo síncrono, para diminuir o consumo, estes dispositivos ajustam o *duty cycle*, até 0,1%. Outro dos mecanismos é o ajuste do tamanho das mensagens. Estes mecanismos permitiram que um simples dispositivo de BLE conseguisse, durante meses, ser alimentado com uma bateria igual à utilizada nos relógios [12].

As aplicações deste protocolo passam desde os telemóveis, até ao ramo automóvel, eletrodomésticos ou *wearables* [12].

O *advertise* é a principal tarefa efetuada por um *beacon*, por isso mesmo torna-se importante conhecer o funcionamento do seu processo, assim como, estes dispositivos conseguem, partir desta técnica, cumprir os seus objetivos e comunicar com os outros dispositivos.

2.4.1. CONTROLO DE LIGAÇÕES E ADVERTISE

No BLE o controlo de ligações e de *advertise* dos dispositivos é feita pelo GAP (*Generic Access Profile*), mecanismo que permite os dispositivos estarem visíveis para o exterior e que determina como dois dispositivos podem interagir mutuamente [13].

O GAP define dois conceitos chave na gestão das ligações, dividindo os dispositivos em dois grupos, os *Central Devices* e os *Peripherical Devices*. Os *Peripherical Devices* são dispositivos pequenos, com baixa capacidade de processamento, como por exemplo os

leitores de frequência cardíaca. Os *Central Devices* são dispositivos com maior capacidade de processamento e de memória, vulgarmente *smartphones* ou tablets [13].

O processo de *advertise* é efetuado pelos *peripherical devices*, consiste no envio repetido de mensagens para todos os dispositivos que estão no alcance (*broadcast advertise)*, como demonstrado na Figura 7. Com este processo os *peripherical devices* tornam-se visíveis para os dispositivos à escuta no seu alcance. Na trama de *advertise* existe um campo de dados variáveis, o que possibilita os *peripherical devices* comunicarem com vários *central devices* simultaneamente [13]. A partir da utilização deste mecanismo os *beacons,* conseguem assim, ser detetados pelos dispositivos móveis.

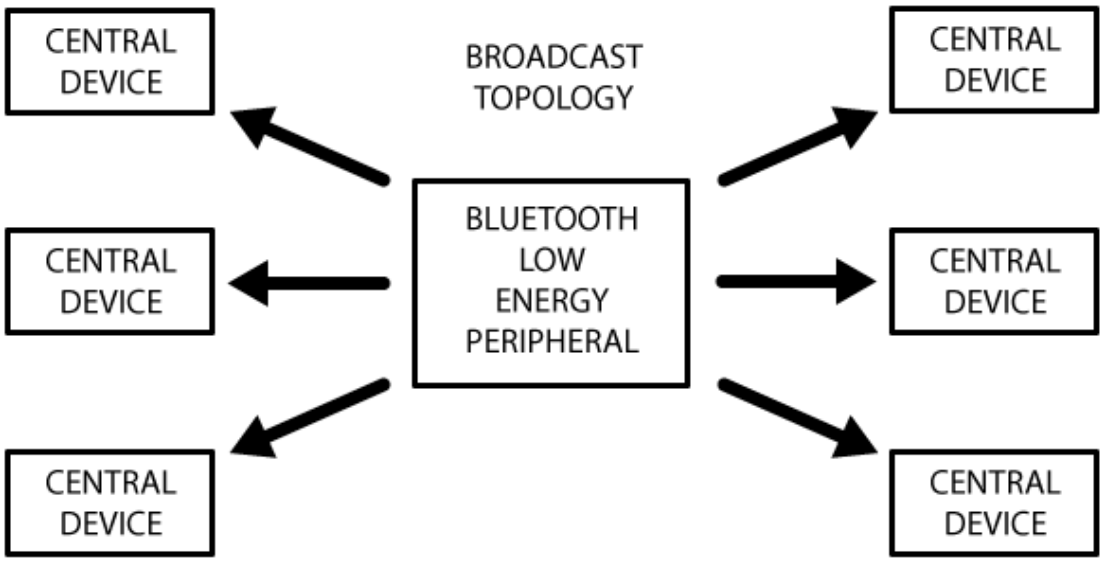


Figura 7. Demonstração do *broadcast* das tramas de *advertise* de um *peripherical device* [13]

A trama de *advertise* é repetidamente enviada em intervalos de tempo específicos, Figura 8, quanto maiores estes intervalos, maior a poupança energética, contudo menor é a eficiência operacional do dispositivo pois significa um maior período de silêncio. Caso um *central device* escute um *peripherical device*, pode responder à trama de *advertise* para obter mais informação sobre o *peripherical*. A partir daí é iniciado o processo de ligação entre os dois dispositivos [13].

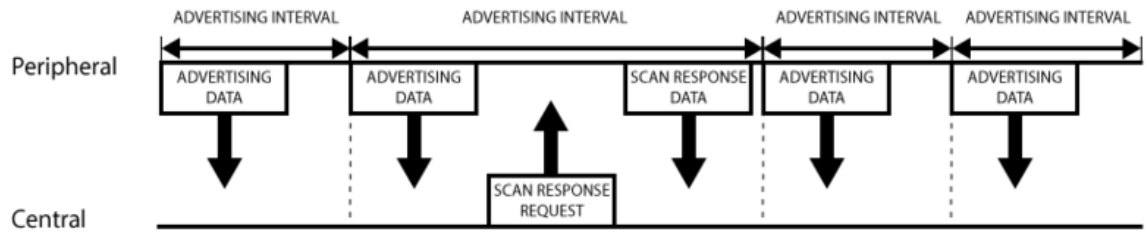


Figura 8. Esquema exemplificativo do processo de *advertise* [13]

Como os *beacons* se tratam de dispositivos não conectáveis, a informação é enviada aquando o *advertise*, não sendo possível efetuar a comunicação no sentido do *central device* para *o peripheticaI device* [13].

2.4.2. ESTRUTURA DA TRAMA DE *ADVERTISE*

A trama de *advertise* do BLE é igual às restantes tramas de dados, sendo o campo de *payload* utilizado para o transporte dos dados [14]. Este campo tem a estrutura apresentada na Figura 9.

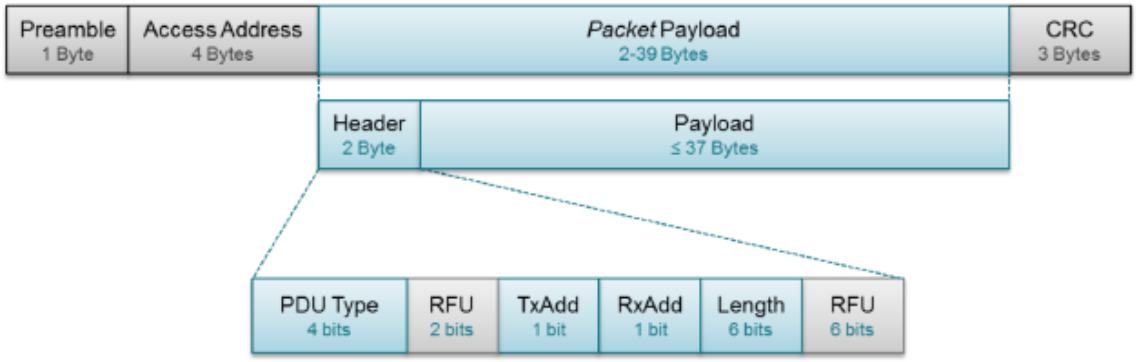


Figura 9. Trama de dados do protocolo BLE [14]

O primeiro campo da trama é o *Preamble*, com tamanho de um 1 byte, é usado para a sincronização com o recetor. Nos pacotes de *advertise* este campo tem sempre o valor 0xAA. O *Access Address* também tem um valor fixo em pacotes de *advertise*, neste caso 0x8E89BED6. O *Packet Payload*, também conhecido *por Protocol Data Unit* (PDU), é o conjunto do *header* e do *payload*. O *header* está dividido em quatro campos. Os primeiros 4bits são o *PDU type*, onde está identificado o tipo de BLE, no caso de uma trama de *advertise* o valor deste campo indica se o dispositivo é conectável ou não conectável, e se existe informação adicional disponível a partir da resposta ao *scanner*. O bit *TxAdd* indica

se o endereço do *advertise* é público ou privado e aleatório. Quanto ao *RxAdd*, não é utilizado no caso dos *beacons* [14].

No final da trama existe um *Cyclic Redundancy Check* (CRC). O CRC é um código de deteção de erro, tendo o objetivo de validar e detetar os pacotes, garantindo a confiabilidade e integridade da informação trocada [14].

O *Payload* assume a estrutura indicada na Figura 10. É dividido em dois campos, um deles reservado para o endereço do dispositivo, sendo o outro para os dados, com o tamanho de 31 bytes [14].

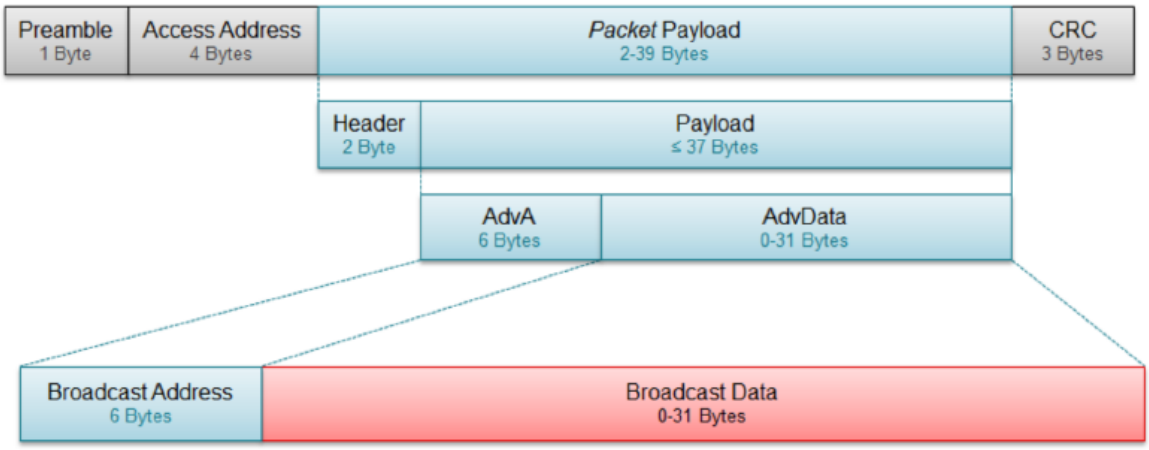


Figura 10. Trama de *advertise* do BLE [14]

2.4.3. PROBLEMÁTICA DA COEXISTÊNCIA DO BLE COM O *WIFI*

Na sua operação o BLE utiliza a frequência de 2,4GHz, frequência utlizada por outro tipo de protocolos como o *Wifi* ou o Zigbee, podendo esta partilha de espectro causar perdas de sinal. Outros focos de interferência são os eletrodomésticos, como por exemplo o micro-ondas. De forma a suprimir os efeitos de eventuais interferências o processo de *advertise* dos dispositivos é efetuado em três canais diferentes sequencialmente [14], tal como demonstrado na Figura 11.

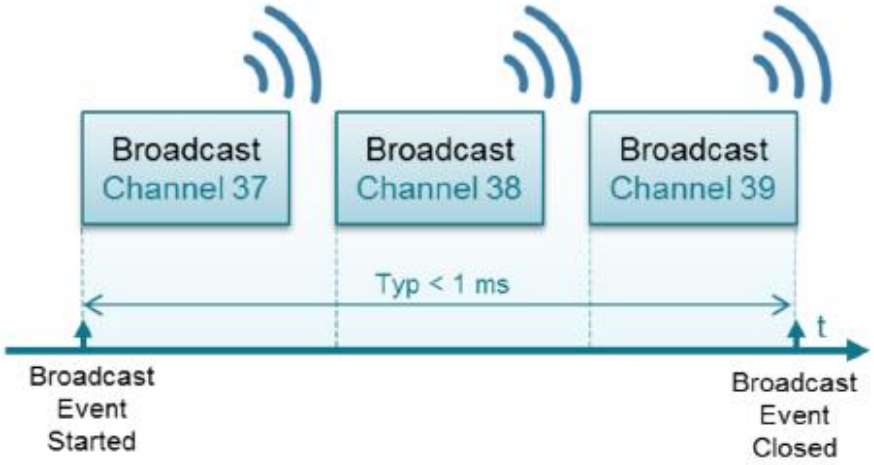


Figura 11. Demonstração da alteração de canal durante o processo de *broadcast advertise* [14]

No processo de *advertise* o BLE utiliza os canais 37,38 e 39, escolhidos por não colidirem com os canais mais frequentemente utilizados no *Wifi*, os canais 1, 6 e 11 [14], tal como se consegue observar na Figura 12.

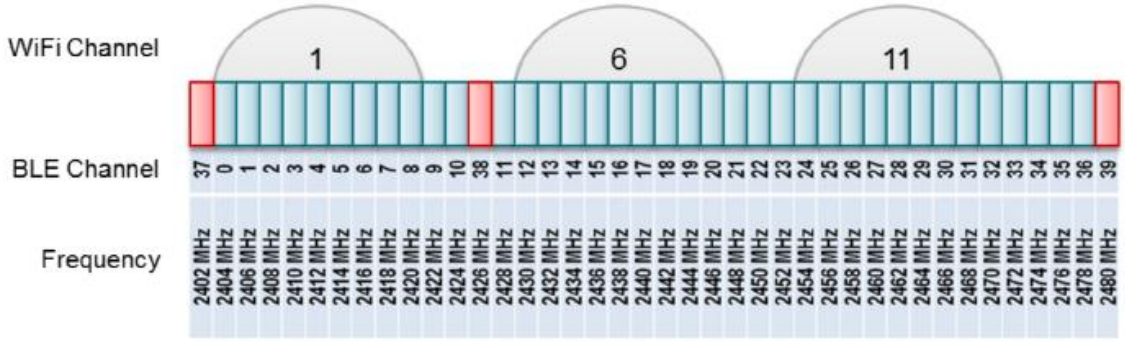


Figura 12. Enquadramento dos canais de *broadcast advertise* do BLE nas frequências em comparação com os canais *Wifi* mais utilizado [14]

Mesmo com a implementação desta técnica o *Wifi* consegue distorcer o sinal do BLE, principalmente se estes dispositivos estiverem muito perto. Pois a potência média de um sinal de Wifi ronda os 23dBm enquanto que a potência máxima de um BLE é de 10 dBm [14].

Na Figura 13, é apresentada uma comparação entre o espectro de Wifi, a funcionar no canal 1, e do processo de *advertise* de um BLE. Neste espectro consegue-se perceber que os canais de *advertise* foram estrategicamente colocados de forma a que a interferência provocada pelo Wifi seja diminuída, podendo ambos coexistir no mesmo espaço. Contudo se no *Wifi*

forem utilizados outros canais que coincidam com os canais do BLE é provável que existam interferências pois, como se consegue visualizar na figura, a potência do *Wifi* é muito superior à do BLE, assim como a sua ocupação de banda é bem superior.

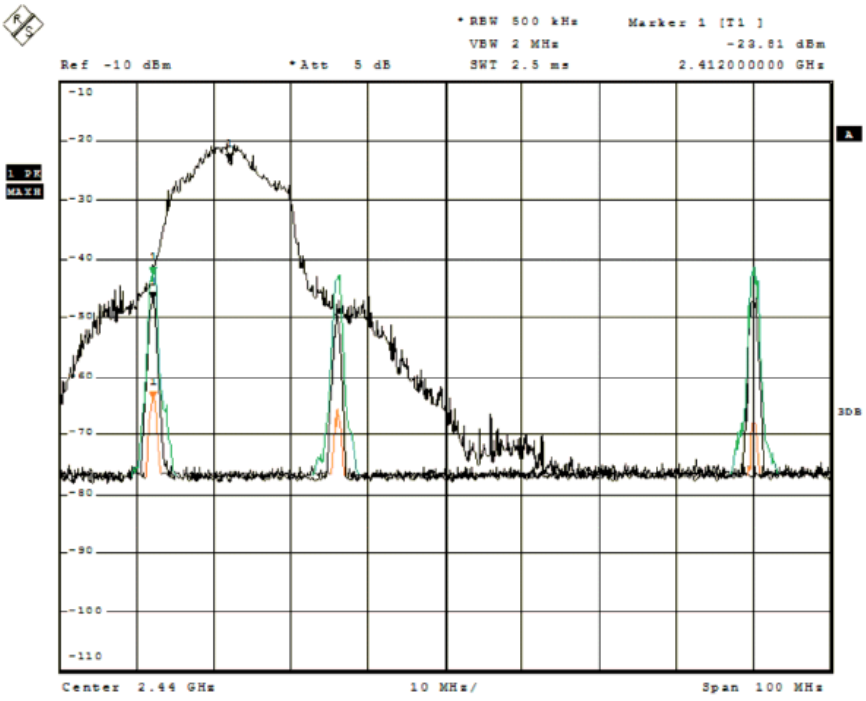


Figura 13. Espectro *Wifi* e do BLE em modo *advertise* [14]

2.5. PSEUDO – STANDARDS

Como não existe uma entidade que regulamente a tecnologia *beacon*, cada fabricante cria o seu *standard*, utilizando como base os recursos que o BLE oferece. Ao longo dos próximos pontos, irão ser apresentados os *standards* mais utilizados, nomeadamente o iBeacon, o Eddystone e o AltBeacon.

2.5.1. IBEACON

O iBeacon foi desenvolvido pela Apple, sendo o primeiro *standard* a ser apresentado e comercializado. Embora não seja *open-source*, a Apple oferece licenciamento grátis dos produtos que cumpram as suas especificações. Apesar de serem desenvolvidos pela Apple são compatíveis com outros sistemas operativos [1].

Estes dispositivos identificam-se através de três diferentes campos: o UUID (*Universally Unique Identifier*), o Major e o Minor. O UUID, com 16 bytes, identifica o dispositivo

conforme o seu caso de uso. O Major, com 2 bytes, especifica um caso de uso daquele dispositivo. O Minor, com 2 bytes, pode ser utilizado para criar uma subdivisão da especificidade criada pelo Major [15]. Os valores destes campos, após a sua configuração, são fixos e podem ser definidos ou editados por quem faz o desenvolvimento dos dispositivos contudo, em muitas das soluções que se encontram no mercado, essa possibilidade está bloqueada pelo fabricante [2][15].

Um exemplo de utilização destes identificadores é através de um retalhista que definiu um certo UUID para identificar os iBeacons dentro das suas lojas, definindo um determinado Major para cada loja, e para cada departamento de cada loja um determinado Minor. Com a utilização desta técnica de identificação um dispositivo consegue mais facilmente filtrar as informações obtidas e localizar-se [15]. Na Tabela 2 é aplicado o exemplo até agora descrito, como forma de melhor se perceber a aplicação do mesmo.

Tabela 2. Exemplo de aplicação dos identificadores (UUID, Major e Minor) dos iBeacons [15]

Localização da Loja		São Francisco	Paris	Londres
UUID		D9B9EC1F-3925-43D0-80A9-1E39D4CEA95C		
Major		1	2	3
Minor	Roupa	10	10	10
	Utensílios Domésticos	20	20	20
	Automóvel	30	30	30

O pacote de *advertise*, de um iBeacon além de incluir os campos UUID, Major e Minor, ainda é composto por outros campos não editáveis e de valor fixo. O conjunto destes campos compõem o PDU da trama de *advertise* do BLE (ver Figura 10). Na Figura 14 é apresentada a estrutura de uma trama de *advertise* de um iBeacon, assim como todos os seus campos.

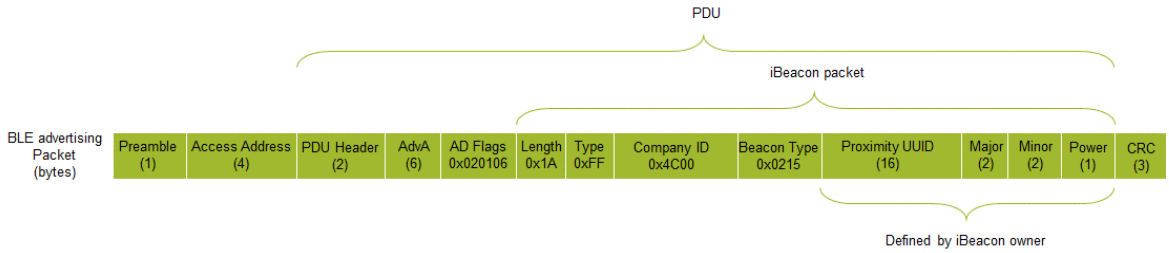


Figura 14. Trama de *advertise* de um iBeacon [1]

A partir da Figura 14, além do pacote de dados do iBeacon é de salientar o campo ADFlags. Com 3 bytes, está dividido por tamanho, tipo e valor, sendo o seu valor estático. O campo correspondente ao tamanho assume o valor 0x02 indicando que este tem 2 bytes de tamanho. O campo tipo assume o valor 0x01, indicando que se trata de um campo de *flags*. O campo "valor" assume o valor de 0x06 indicando o modo de funcionamento do BLE, neste caso não é conectável, com *advertise* indireto e funciona no modo *stand-alone* (sem ninguém ligado) [1].

Depois do campo AD*Flags* segue-se o pacote de dados do iBeacon, com os seguintes campos [1]:

- *Lenght*: Indica o tamanho do pacote de dados do iBeacon. Para estes dispositivos assume sempre o valor 0x1A, ou seja, 26 bytes.

- *Type*: Indica o contexto dos dados indicados nos campos que se seguem. Para estes dispositivos assume sempre o valor 0xFF, indicando que os campos que se seguem são informações específicas do fabricante.

- *Company ID*: Campo com 2 bytes que identificam o fabricante. Para estes dispositivos assume sempre o valor de 0x4C00, indicando que o fabricante é a Apple.

- *Beacon Type*: Campo com 2 bytes que indica o tipo de *beacon*, assume sempre o valor 0x0215.

- *UUID*: Campo com 16 bytes que indica o ID do dispositivo

- *Major*: Campo com 2 bytes, que especifica um caso de uso daquele dispositivo.

- *Minor*: Campo com 2 bytes, pode ser utilizado para criar uma subdivisão da especificidade criada pelo Major.

- *Measure Power*: Campo com um byte que retorna ao valor do RSSI (Received Signal Strength Indicator), ou seja, a potência do sinal do iBeacon em dBm.

A Apple oferece a *framework* Core Location para possibilitar mais facilmente o desenvolvimento de *apps* para implementar funcionalidades pela utilização de iBeacons.

2.5.2. EDDYSTONE

O Eddystone é um *standard open-source* desenvolvido pela Google. Este *standard* é composto por diferentes tipos de pacotes, com diferentes estruturas e objetivos, tais como os apresentados abaixo [1] [16]:

- Eddystone-UID: Emite uma trama de *advertise* com um identificador único do *beacon*. Funciona de forma idêntica ao iBeacon, ou seja, envia de forma cíclica uma trama com um identificador estático.

- Eddystone-URL: Emite um *advertise* com *URL* (*Uniform Resource Locator*), usado para direcionar o utilizador diretamente para uma página *web*.

- Eddystone-TLM: Emite um *advertise* com informações sobre o próprio dispositivo, como o nível de bateria, o número de tramas transmitidas, o tempo de atividade ou a temperatura do mesmo. A sua utilização é combinada com tramas do tipo Eddystone-UID ou Eddystone-EID com o envio alternado de informação.

- Eddystone-EID: Emite uma trama de *advertise* com um identificador do dispositivo, contudo a trama envidada é encriptada e variável ao longo do tempo.

Para facilitar o desenvolvimento de *apps* para a utilização destes dispositivos a Google desenvolveu a Proximity Beacon API.

Para cada s*tandard*, existe uma estrutura que difere do campo de dados da trama de BLE, contudo todos partilham a mesma estrutura de base, idêntica à apresentada na Figura 15. Dentro da sua estrutura básica salientam-se os seguintes campos [1][17]:

- *Eddystone Identifier*: campo que identifica os campos seguintes como sendo de um *beacon* Eddystone. Tem o tamanho de 8 bytes e assume um valor estático, com exceção no campo "tamanho", identificado com "—" na Figura 15, que é preenchido com o valor da soma do campo "*Frame Type*" e do campo de dados.

- *Frame Type*: campo que indica o tipo de trama Eddystone, podendo conforme a seguinte lista:

    - Eddystone-UID: 0x00
    - Eddystone-URL: 0x10
    - Eddystone-TLM: 0x20
    - Eddystone-EID: 0x30

- *Frame-specific Data*: Campo de dados da trama Eddystone. Assume uma estrutura variável consoante o tipo de trama Eddystone.

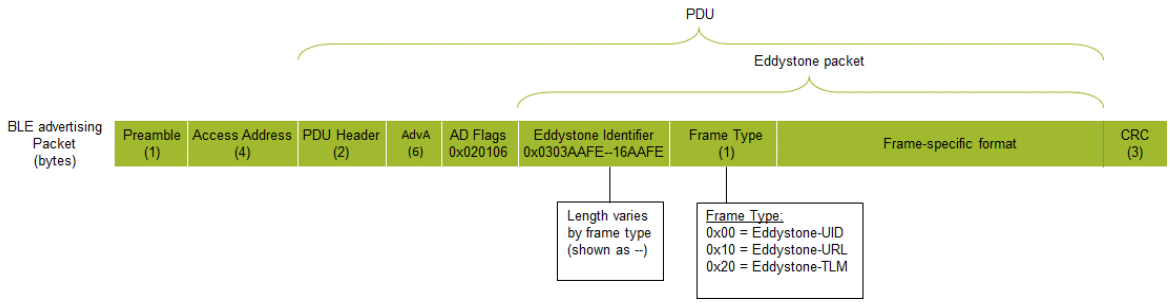


Figura 15. Estrutura básica de uma trama de advertise de um *beacon* Eddystone [1]

Caso se trate de um *Eddystone-ID* o campo de dados é composto pelo valor do RSSI, pelo identificador do dispositivo e por mais 8 bytes auxiliares. Caso se trate de um *Eddystone-URL* o campo de dados divide-se entre o valor do RSSI e o url numa versão condensada assumindo a estrutura da Figura 16. Caso se trate de um *Eddystone-TLM* o campo de dados está dividido entre a versão do *beacon*, o valor da bateria (1 byte), o valor da temperatura (2 bytes), o valor do número de tramas de *advertise* enviadas desde o último *reset* (4 bytes) e o valor do número de segundos que o dispositivo se encontra ativo (4 bytes) [1].

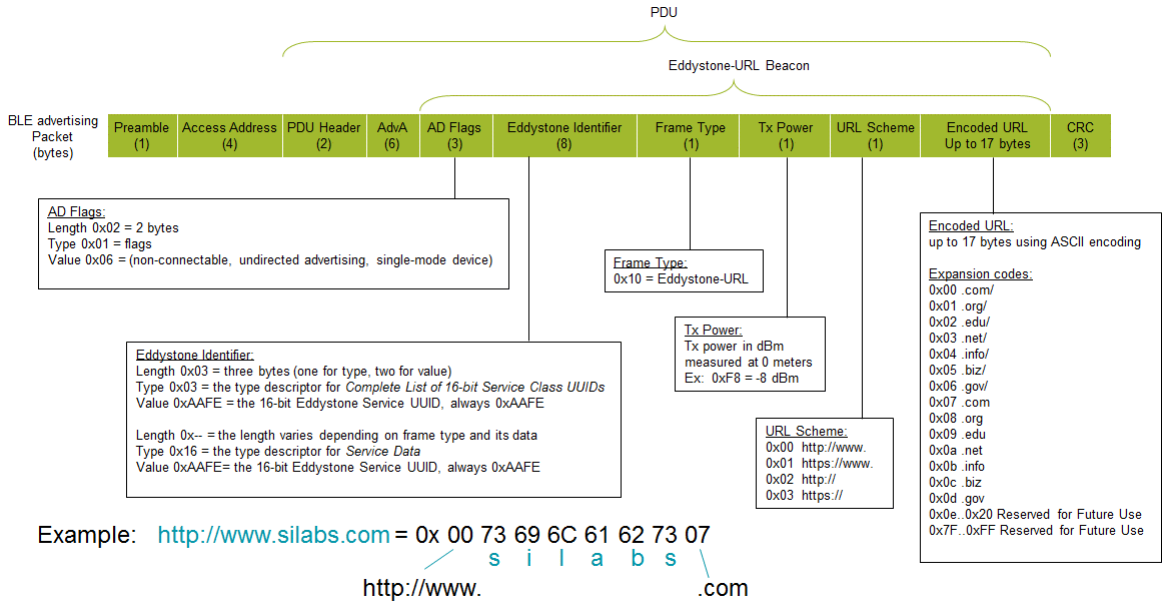


Figura 16. Estrutura de uma trama de *advertise* de um *beacon* do tipo Eddystone-*URL* [1]

2.5.3. ALTBEACON

O AltBeacon é uma especificação *open-source* para dipositivos *beacon*, desenvolvida pela Radius Networks, que tem como objetivo criar um *standard* sem qualquer ligação a fabricantes de dipositivos móveis. O funcionamento dos dispositivos, que seguem estas especificações, têm um funcionamento igual aos até agora apresentados [1][17].

A sua trama de *advertise* segue uma estrutura fixa onde além de campos de identificação do dispositivo ainda existem campos de identificação do seu fabricante. Na Figura 17 é apresentada a estrutura de uma trama de *advertise* de um *beacon* deste tipo, dos quais se salientam os seguintes campos [1][17]:

- *Lenght*: campo que indica o tamanho do pacote de dados do AltBeacon. Tem 1 byte de tamanho e assume sempre o valor 0x1B, ou seja, o volume dos dados da trama é de 27 bytes.

- *Type*: campo que indica o tipo de dados que se seguem a este campo. Tem 1 byte de tamanho e assim sempre o valor 0xFF, indicando que se trata de uma trama que inclui dados do fabricante

- *Mfg ID*: campo que identifica o fabricante do *beacon*, com 2 bytes obtidos a partir da lista de fabricantes Bluetooth SIG.

- *Beacon Code*: campo com 2 bytes que assume sempre o valor 0xBEAC
- *Beacon ID*: campo do identificador único do *beacon* com 20 bytes de tamanho
- *Reference RSSI*: campo com o valor médio da força do sinal do RSSI, com 1 byte de tamanho e valor variável entre 0 e 127.
- *Mfg Reserved*: campo com 1 Byte, reservado para uso livre por parte do fabricante do Beacon

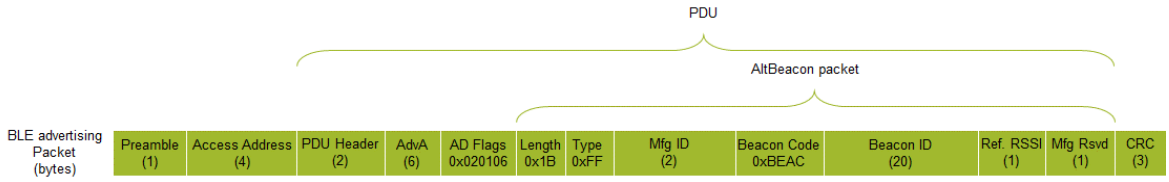


Figura 17. Estrutura da trama de *advertise* de um beacon AltBeacon [1]

Tal como os *standards* referidos anteriormente, também é disponibilizada uma API (*Application Programming Interface*) para Android para apoio no desenvolvimento de aplicações.

2.6. COMPORTAMENTO DA POTÊNCIA DO SINAL

Aquando da apresentação da estrutura das tramas de *advertise* dos *beacons*, percebeu-se que estas além de identificarem o dispositivo também indicam a potência do sinal emitida, também conhecido como RSSI (*Received Signal Strength Indicator*). Quanto maior for este valor maior é a distância entre o *beacon* e o dispositivo móvel. Contudo, a relação entre o valor do RSSI e a distância não é linear, mas sim logarítmica, por ser representado em decibéis (dBm).

Sendo esta uma tecnologia rádio e sem fios, as condições do ambiente tem uma influência notória nos valores da força de sinal, assim como as características dos dispositivos que intervêm na comunicação, levando a muitas deficiências na estimativa da distância a partir da força de sinal. Como forma de demonstrar e testar como as várias variações do valor de RSSI influenciam os *beacons* de seguida irão ser analisados exemplos do livro "*Beacon Technologies: The Hitchhiker's Guide to the Beacosystem*" [19] e das suas fontes.

2.6.1. INFLUÊNCIA DE CORPOS NO COMPORTAMENTO DO RSSI

No primeiro exemplo apresentado no livro "*Beacon Technologies: The Hitchhiker's Guide to the Beacosystem*", é estudada como a interferência de corpos pode influenciar o valor do RSSI. Neste exemplo, são apresentados três testes diferentes efetuados utilizando um iBeacon fabricado pela Estimote e diferentes *smartphones*, estando os resultados apresentados no gráfico da Figura 18. No primeiro teste, representado a laranja, são efetuadas várias leituras, em diversos pontos e a distâncias diferentes, sem qualquer interferência, no segundo teste, representado a amarelo, são efetuadas medições nos mesmos pontos, do teste anterior, mas colocando uma pessoa a movimentar-se entre os dois dispositivos e num terceiro teste, representado a azul, foi colocada uma placa de metal entre os dois dispositivos [19].

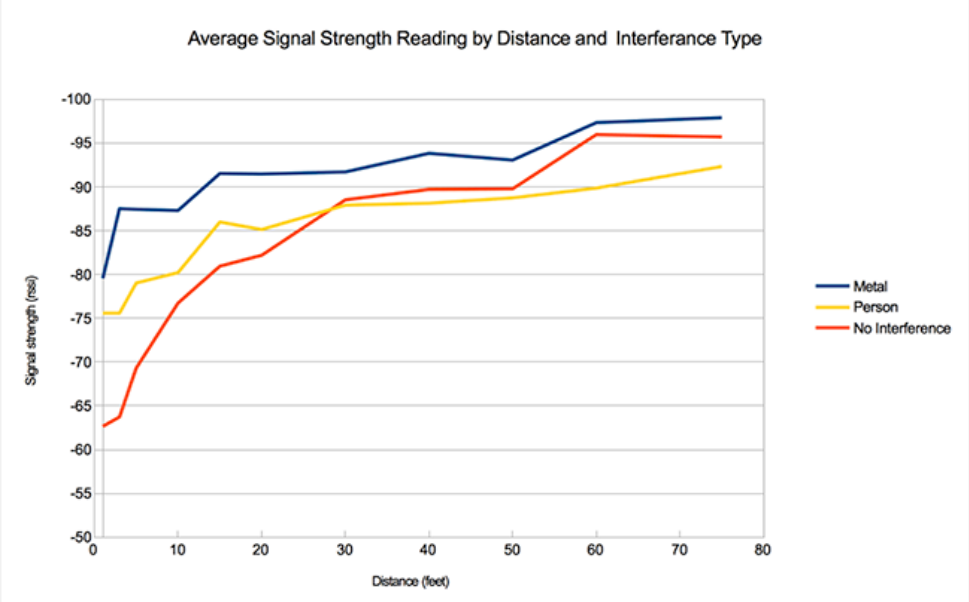


Figura 18. Influência da interferência de corpos entre um *beacon* e um dipositivo recetor [19]

Analisando os resultados, denota-se uma atenuação provocada pela presença de corpos entre o *beacon* e o *smartphone*, no caso do Humano essa atenuação é diluída ao longo do espaço, muito pela forma como as ondas rádio se propagam nesse mesmo espaço. Com a presença do metal a atenuação verificada é mais acentuada, provocada pela maior densidade do material.

Tendo em conta que os *beacons* são dispositivos concebidos para o retalho e colocados no interior de lojas, espaços que são constituídos por prateleiras normalmente metálicas e onde a carga de pessoas é variável é de denotar que a estabilidade do sinal é de certeza afetada.

2.6.2. INFLUÊNCIA DOS DISPOSITIVOS NO VALOR DO RSSI

Neste segundo exemplo, é estudada a influência do dispositivo recetor no valor do RSSI. Para tal, é feita a leitura de valores do RSSI a diferentes distâncias do *beacon* utilizando diferentes smartphones, obtendo-se os resultados apresentados na Figura 19. Neste teste foram utilizados um Nexus 4, representado a laranja, um Samsung S3, representado a amarelo, e um iPhone 5, representado a azul [19].

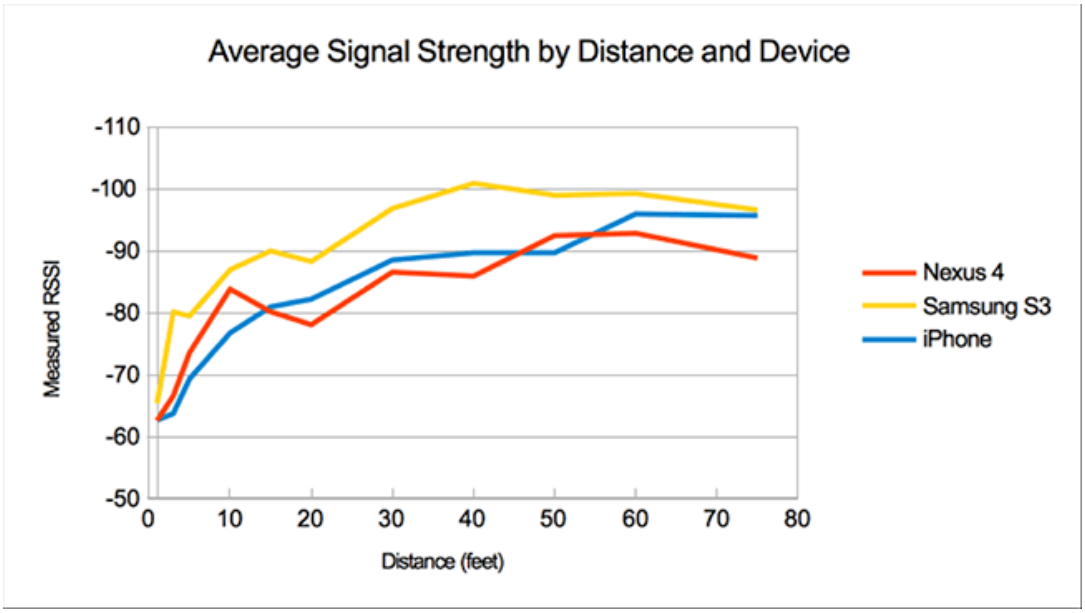


Figura 19. Influência dos dispositivos no valor das leituras do RSSI [19]

Analisando os resultados deste teste consegue-se perceber que o iPhone tem um comportamento mais estável, mas comparando os resultados dos três dispositivos verifica-se, em alguns pontos, um diferencial superior a 10dBm entre as amplitudes dos resultados obtidos.

Dado o grande número de dispositivos com características diferentes o diferencial de resultados de medições pode ainda ser mais díspar.

2.6.3. ESTABILIDADE DO SINAL DE RSSI

Neste terceiro exemplo é estudado o comportamento do valor de RSSI ao longo do tempo num dispositivo estacionado. Assim se obteve o gráfico da Figura 20, no qual são apresentados os resultados das leituras com o mesmo dispositivo a 10 pés (3,048 metros) do *beacon*, representado a azul, e a 15 pés (4,572 metros) do *beacon*, representado a laranja, ao longo de um intervalo de tempo [19].

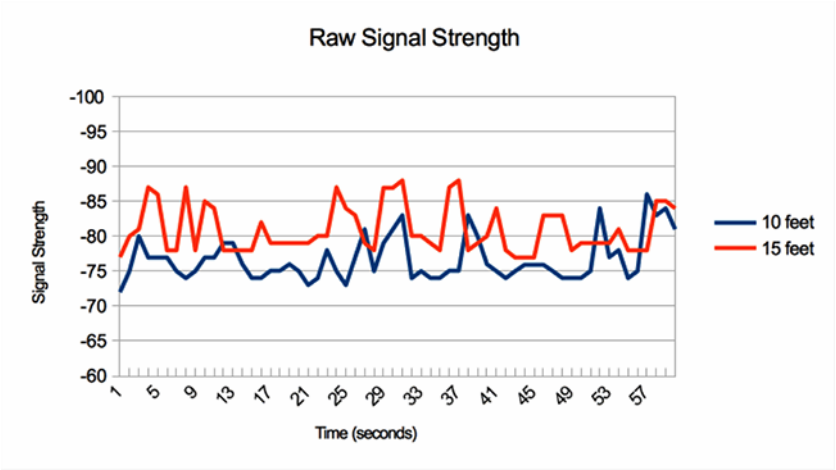


Figura 20. Resultados obtidos a partir da análise do valor de RSSI num dispositivo estacionado à distância de 10 pés (3,048 metros) e de 15 pés (4,572 metros) [19]

Neste teste consegue-se verificar a pouca estabilidade do valor de RSSI, sendo que por vezes, à distância de 10 pés, se obtém valores de RSSI superiores aos valores obtidos a 15 pés. Como forma de conseguir aumentar a estabilidade dos resultados obtidos ainda se tentou efetuar a média dos valores de 3 segundos, obtendo-se os resultados do gráfico da Figura 21, levando a uma estabilização de 50% do nível de sinal [19].

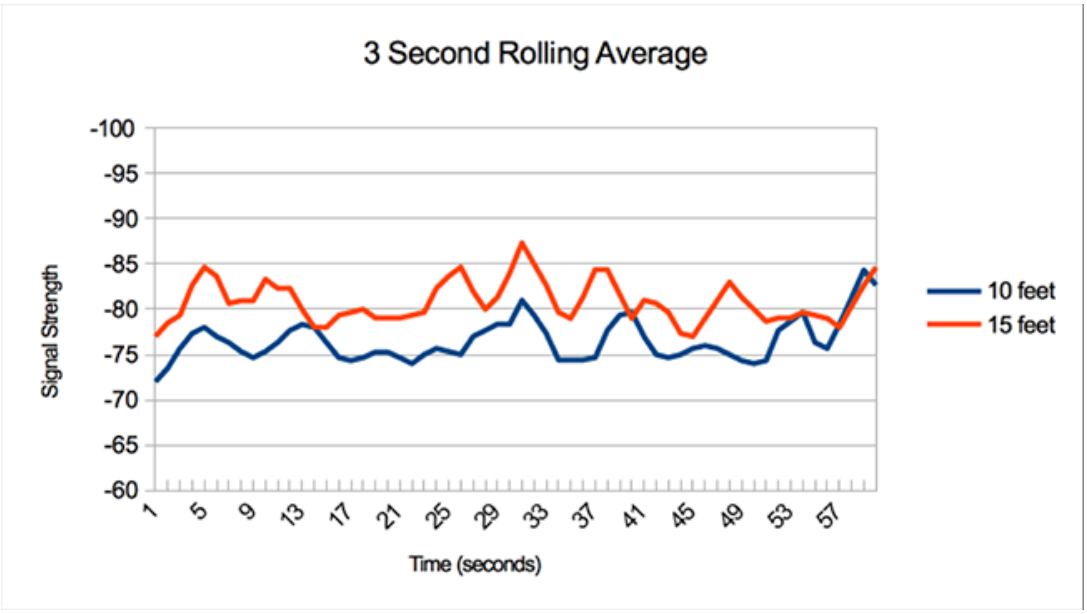


Figura 21. Resultados obtidos a partir da análise da média do valor de RSSI ao longo de 3 segundos num dispositivo estacionado à distância de 10 pés (3,048 metros) e de 15 pés (4,572 metros) [19]

A partir dos resultados obtidos neste exemplo consegue-se comprovar a pouca estabilidade do valor de RSSI destes dispositivos.

2.6.4. FORMAS DE MINORAR A INSTABILIDADE DO RSSI

Para minorar os efeitos da instabilidade do RSSI, as especificações de alguns dispositivos aconselham a efetuar uma calibração através da utilização de recursos da própria API do fabricante, após colocar o dispositivo a um metro do *beacon* e durante a calibração efetuar movimentos retos repetidos com amplitude de 30 cm, tal como exemplificado na Figura 22 [15], [18]. Embora esta calibração consiga ter algum efeito, o mesmo apenas é notado em ambiente de desenvolvimento, sendo que em ambiente de produção não é comportável efetuar a calibração de todos os dispositivos.

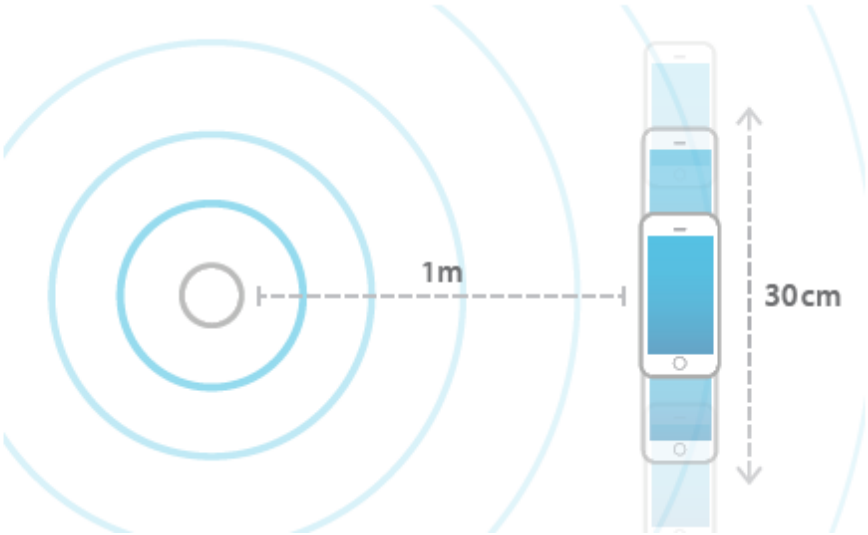


Figura 22. Esquema exemplificativo da calibração de um dispositivo [15]

Para evitar efetuar calibrações contínuas dos dispositivos e diminuir os efeitos da instabilidade do valor do RSSI é utilizada uma técnica em que em detrimento da precisão se opta por dividir o alcance do dispositivo por zonas, resultando numa aproximação da posição real. Assim, seguindo a especificação da Apple para os iBeacons, também seguida por outros fabricantes, pode-se dividir o alcance em quatro zonas, podendo ser utilizadas nas situações indicadas [15]:

- Imediato (*Immediate*): O dispositivo móvel encontra-se muito próximo do *beacon;*
- Perto (*Near*): O dispositivo encontra-se aproximadamente entre 1 a 3 metros do *beacon.* A precisão pode ser afetada por quaisquer obstáculos que existam entre os dois pontos;

- Longe (*Far*): O dispositivo encontra-se longe de tal forma que não pode ser considerado que este está perto do *beacon*, contudo a sua presença está a ser captada;

- Desconhecida (*Unknown*): O *beacon* não está a ser captado ou são necessários mais resultados para considerar quaisquer dos estados anteriores.

A utilização de *beacons*, mais propriamente do valor do RSSI para a obtenção da localização *indoor*, tem de ser bem ponderada ou melhor, bem trabalhada, pois é necessário criar métodos de forma a diminuir as suas alterações ou, em alternativa, diminuir a precisão da localização e utilizar métodos aproximados.

2.7. COMPARAÇÃO COM TECNOLOGIAS ALTERNATIVAS

Existem algumas tecnologias que, idênticas aos *beacons*, conseguem ladear com estes em algumas funções, entre as quais, importa referir o GPS (*Global Position System*), o NFC (*Near Field* Communication) e o *Wifi* [3][20][21] .

O GPS, uma tecnologia de localização com uma taxa de utilização elevada, tem como grandes desvantagens, o fraco alcance dentro de edifícios e uma fraca precisão. Isto revela uma deficiência enorme quando se pretende implementar uma tecnologia baseada em localização dentro de um edifício ou de uma superfície comercial. Contudo, esta tecnologia pode-se tornar interessante para este tipo de implementação quando usada em conjunto com tecnologia *beacon*, conseguindo fornecer, informações de localização *outdoor* e *indoor* [3] [19].

O NFC é uma tecnologia de comunicação de curto alcance baseada em RFID (*Radio-frequency Identification*). Um sistema NFC é composto por um *reader/writer* e um *transponder* (*Tag*) sendo a comunicação entre estes efetuada quando se aproximam numa tipologia ponto a ponto, em que a distância entre os dois dipositivos não pode ser superior a 10 cm. Remetendo para um caso prático, o alcance desta tecnologia obriga a uma ação do utilizador para que o seu dispositivo móvel consiga detetar o *tag* [20][3][21].

A utilização de serviços de localização baseados em *Wifi* já é uma prática comum quando o serviço de GPS é inadequado. Comparativamente com os *beacons*, o *Wifi* tem maior alcance e maior área de cobertura. Isto permite que, a partir do mapeamento dos APs (*Access Points)*, seja possível efetuar uma localização *indoor*, contudo, estes APs muitas vezes não são

confiáveis e estão mal posicionados, levando a uma precisão deficiente. Embora os *beacons* tenham um alcance inferior, isto também ajuda a que a sua precisão seja superior à do *Wifi*. Os *beacons*, por seu lado, são normalmente dipositivos pequenos, alimentados por pilhas e com baixo consumo energético, o que leva a que a sua instalação seja mais fácil, vantagem relevante quando se pretende implementar uma rede de *spots* [3][20][21].

Analisando as tecnologias alternativas aos *beacons* torna-se visível a complementaridade que estas proporcionam, principalmente o GPS e o *Wifi* revelam características que quando aplicadas oferecem um leque diversificado de soluções de localização.

3. TECNOLOGIAS

O presente capítulo pretende apresentar as restantes tecnologias utilizadas no projeto, as suas características, as suas vantagens e desvantagens, respondendo aos requisitos inicias do projeto.

O capítulo inicia-se com análise do desenvolvimento de aplicações móveis, passando pela comparação entre as aplicações nativas e híbridas e de algumas *frameworks* para o desenvolvimento de aplicações híbridas. Depois é estudado o comportamento do Android na execução de aplicações/serviços em segundo plano. O capítulo termina com uma apresentação do *broker* RabbitMQ e do protocolo AMQP (*Advanced Message Queuing Protocol*), sendo ainda feita uma breve introdução sobre *middleware*.

3.1. DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS

Com a emancipação dos *smartphones,* os telemóveis deixaram de ter unicamente a função de efetuar comunicações e passaram a ser utilizados para finalidades lúdicas, somando também cada vez mais funcionalidades que nos permitem efetuar tarefas do dia a dia. Neste contexto, as organizações sentiram necessidade de disponibilizar os seus serviços numa aplicação móvel.

O processo de lançamento de uma aplicação é normalmente complexo, existindo necessidade de ter em conta não só fatores técnicos, como também funcionais que irão afetar a aplicação não só na sua conceção, mas também em todo o seu ciclo de vida. Neste confronto de ideias uma das questões abordadas é a opção pelo desenvolvimento de uma aplicação nativa, ou de uma aplicação híbrida.

3.1.1. APLICAÇÃO NATIVA VS APLICAÇÃO HÍBRIDA

As aplicações nativas são desenvolvidas para um sistema específico (Android, iOS, Windows Phone) as suas linguagens de programação são também elas nativas e específicas, como é o caso do Objetive-C ou do Swift que são utilizadas no iOS; do Java que é utilizado no Android e do C# utilizado no caso do Windows Phone [22].

As aplicações híbridas utilizam tecnologias *web*, em vez das nativas. Mesmo assim estas conseguem ser executadas dentro dos sistemas nativos e comunicarem com os componentes do equipamento, como por exemplo, o GPS ou o acelerómetro [23], através de *plugins*.

A grande vantagem do desenvolvimento de aplicações híbridas é a sua portabilidade. Isto é, o mesmo código consegue ser compatível com as várias plataformas e reagir da mesma maneira nos diferentes sistemas operativos. No caso do desenvolvimento de aplicações nativas cada plataforma tem a sua linguagem, fazendo com que para cada plataforma exista um código diferente originando assim uma aplicação diferente para cada dispositivo [22], [23], [24].

Comparando cada uma das abordagens denota-se, no imediato, que o desenvolvimento de uma aplicação híbrida tem um desenvolvimento mais rápido, mais barato e a manutenção, do seu código, mais simples. Não é necessária uma equipa de desenvolvimento extensa e multidisciplinar porque não é necessário conhecer as linguagens de cada um dos sistemas operativos, bastando dominar a linguagem utilizada, nomeadamente o *Javascript*, *HTML* e *CSS* [22], [23], [24].

É evidente que por utilizarem tecnologias diferentes das nativas a compatibilidade das aplicações híbridas com todos os dipositivos é uma preocupação, porém atualmente estas já são compatíveis com quase todos os dipositivos móveis, dependendo apenas da *framework* utilizada. Outra preocupação é devida às aplicações híbridas implementarem várias camadas de tecnologia, sendo que por consequência a performance da aplicação acaba por ser

diminuída quando comparada com as aplicações nativas, contudo a experiência do utilizador é cada vez menos afetada [22]–[24], tendo em conta a grande adoção deste tipo de aplicações.

Dadas estas evidências, consegue-se perceber que as aplicações híbridas conseguem, atualmente, ter algumas vantagens sobre as aplicações nativas contudo, deve-se sempre ter em conta os recursos necessários no desenvolvimento da aplicação, dado que em alguns casos, sobretudo nos de maior exigência, o desenvolvimento de aplicações nativas é uma exigência [24].

3.1.2. FRAMEWORKS DE APLICAÇÕES HÍBRIDAS

Existem várias *frameworks* dedicadas ao desenvolvimento de aplicações híbridas. Cada uma delas tem as suas próprias características conseguindo interrelacionar um grande número de tecnologias possibilitando que uma comunidade possa mais facilmente desenvolver aplicações para dispositivos móveis.

Nos próximos pontos irão ser apresentadas quatro *frameworks* utilizadas neste âmbito: o Apache Cordova, o Adobe PhoneGap, o Ionic e o Xamarim.

3.1.2.1. APACHE CORDOVA

O Cordova é uma *framework open-source*, detida pela Apache, que permite utilizar linguagens web, como o HTML5, CSS3 e javascript, para o desenvolvimento de aplicações móveis com suporte para vários sistemas operativos [26].

No Cordova a aplicação é implementada como se fosse uma página web que é executada numa WebView [26]. A WebView é um recurso do sistema operativo que permite visualizar páginas web dentro de uma aplicação sem apresentar o aspeto de um *browser* [27]. O Cordova oferece ainda recursos que permitem efetuar o interface com os componentes nativos do sistema operativo, denominados de *plugins*. Existe uma listagem longa de *plugins*, sendo também possível o desenvolvimento de *plugins* à medida das necessidades do negócio. Estes como implementam o interface de comunicação com API's do sistema operativo nativo, têm de ser desenvolvidos na linguagem nativa sendo que, utilizando os recursos da API do Cordova, é possível fazer a integração da componente nativa com a componente Web da aplicação [26].

A arquitetura de uma aplicação desenvolvida em Cordova, Figura 23, divide-se em dois grandes grupos, o sistema operativo móvel e a aplicação Cordova. A aplicação é executada numa WebView que faz a execução das linguagens web e das APIs da *framework*. Aqui reside a interface com o utilizador e a lógica da aplicação. No Sistema Operativo são recebidos todos os *inputs* requisitados ao nível do Cordova, ou seja, toda a comunicação com o *hardware* ou com os recursos do sistema operativo nativo, sendo esta ponte feita a partir dos *plugins* [26].

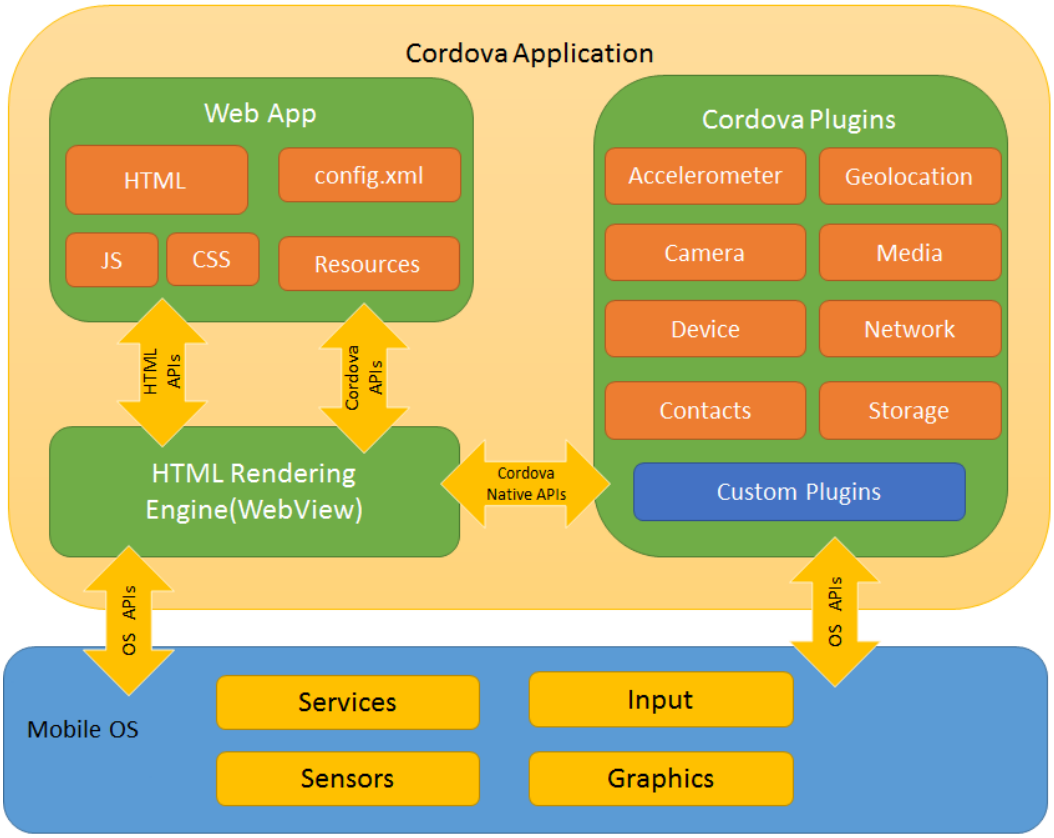


Figura 23. Arquitetura de alto nível de uma aplicação Cordova [26]

3.1.2.2. ADOBE PHONEGAP

O Adobe PhoneGap é uma *framework*, para o desenvolvimento de aplicações móveis a partir de linguagens web. Esta *framework*, inicialmente *open-source* e detida pela Apache, é que deu origem ao Cordova, quando em 2011 a Apache doou todo o código fonte à Adobe fazendo de seguida surgir o Cordova [28].

Toda a sua estrutura é igual à do Cordova, sendo que atualmente todas as atualizações do Cordova são replicadas no Phonegap. A esta *framework* a Adobe apenas acrescentou

serviços que em conjunto com esta permite apoiar e simplificar o processo de desenvolvimento de uma aplicação. [28]

Entre as funcionalidades adicionadas à *framework* tem-se a salientar a possibilidade de conseguir remotamente efetuar atualizações na aplicação a desenvolver e a possibilidade de efetuar a compilação da aplicação desenvolvida com o recurso a um serviço de *cloud* em que a partir dos ficheiros HTML, CSS, *javascript* este gera os executáveis da aplicação para o sistema operativo pretendido (Android, iOS, Windows Phone). [28]

3.1.2.3. IONIC

O Ionic é uma *framework, open source,* utilizada para apoiar a construção de aplicações híbridas baseadas em linguagens *web*. Esta *framework* é baseada em HTML5, CSS, *Javascript* e AngularJS e oferece recursos para a construção de aplicações híbridas com o aspeto e fluxo de uma aplicação nativa [29] [30].

Fazendo a comparação com plataformas já existentes para a construção de páginas *web,* esta *framework* tem um funcionamento idêntico ao Bootstrap mas para aplicações móveis, ou seja, fornece vários componentes que podem ser inseridos na interface com a UI (*User Interface*) de uma aplicação. A abrangência desta *framework* vai desde o aspeto de um ecrã, passando pelo aspeto dos elementos do mesmo (como botões, texto, etc.), até ao fluxo entre os ecrãs e as suas transições. Embora o Ionic englobe aspetos variados sobre a construção da UI este também permite a personalização destes, conforme as necessidades [29] [30].

Com a utilização do Ionic pretende-se facilitar a construção de aplicações híbridas baseadas em linguagens web mas também normalizar os aspetos ligados ao seu aspeto, oferecendo também recursos que permitem maior fluidez na aplicação e melhor desempenho [29] [30].

Como esta *framework* apenas abrange aspetos ligados à UI da aplicação a compilação da mesma deverá ser feita com a utilização do Cordova ou do Phonegap [29] [30].

Juntamente com o Ionic são fornecidos uma gama de outros serviços para o auxílio na construção de aplicações, onde se insere um IDE (*Integrated Development Environment*) próprio para apoiar na construção do aspeto da aplicação, também é possível efetuar a compilação da aplicação na plataforma na *cloud* por eles oferecida. Essa *cloud* permite ainda a atualização, a distribuição, o rastreamento e a monitorização remota das aplicações

desenvolvidas. Estas funcionalidades estão apenas disponíveis através de subscrição dos planos disponibilizados [31].

3.1.2.4. XAMARIM

O Xamarim é uma *framework* que permite desenvolver aplicações a partir de .NET utilizando a linguagem C#. Ao compilar é gerado um código nativo, conseguindo-se um desempenho idêntico ao obtido com aplicações nativas [32].

No entanto o código não é totalmente compatível com as diferentes plataformas. Em média 75% do código é partilhado entre sistemas operativos, dependendo das aplicações. O código partilhado é referente ao *core* da aplicação, sendo constituído por toda a lógica da aplicação, incluindo a lógica de negócio e o acesso à rede. Para cada plataforma é necessário desenvolver o acesso aos seus periféricos e a interface com o utilizador, tal como demonstrado na Figura 24. Isto faz com que o desenvolvimento da aplicação seja mais lento e obrigue a que o desenvolvedor tenha conhecimentos sobre cada uma das plataformas [32].

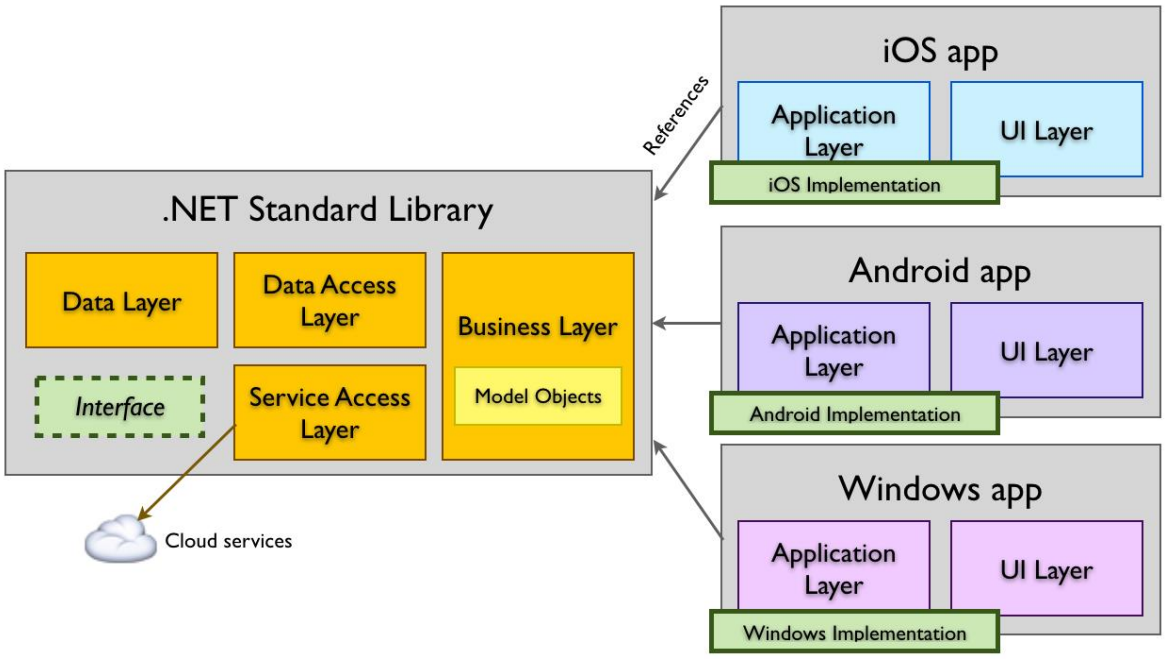


Figura 24. Arquitetura de código do Xamarim [32]

A grande vantagem do Xamarim é pelo facto de que a partir deste se consegue obter uma aplicação nativa com uma só lógica que é partilhada por todas as plataformas, sendo mais fácil fazer a gestão deste código [32].

3.1.2.5. CONSIDERAÇÕES FINAIS

Efetivamente qualquer uma das opções é válida, conseguindo-se a partir destas opções construir uma aplicação que cumpra com os objetivos definidos inicialmente.

Quanto à performance, Xamarim apresenta melhores resultados em comparação com as restantes opções. O Cordova, o PhoneGap e o Ionic originam um código mais portável levando a que a gestão de versões tenha um impacto inferior no desenvolvimento de aplicações. Por serem baseadas em linguagens *web,* estas *frameworks*, têm uma grande comunidade com múltiplas ferramentas de apoio no desenvolvimento não só ao nível do *front-end* como do *back-end.*

Pelas vantagens acima descritas, mas também por ser uma opção completamente *open-source* e por possibilitar o desenvolvimento de uma aplicação sem necessitar utilizar mais ferramentas, escolha recai sobre o Cordova.

3.2. EXECUÇÃO DE APLICAÇÕES EM SEGUNDO PLANO

As aplicações móveis são sempre constituídas por um ou vários componentes. Nesses componentes existe sempre um que implementa uma interface gráfica, contudo, existem componentes que funcionam em segundo plano sem interface para o utilizador.

As *frameworks* baseadas em linguagens web, como o Cordova, são executadas numa WebView. As WebView são componentes que executam em primeiro plano, ou seja, com interface para o utilizador, não conseguindo efetuar atividades em segundo plano.

Isso resulta num problema quando se pretende efetuar tarefas quando a aplicação não esteja em execução, com é o caso da deteção contínua de *beacons*, objetivo deste projeto. Para isso torna-se necessário estudar as formas de se conseguir efetuar tarefas em segundo plano nos sistemas nativos, para depois integrar o seu controlo num *plugin* compatível com a *framework* utilizada.

Neste ponto, o objeto de estudo irá ser o sistema operativo Android, apresentando alguns fundamentos sobre os serviços em segundo plano, o seu funcionamento, as suas limitações e restrições. Também se apresentam as boas práticas e o comportamento do Android quanto à execução deste tipo de serviços.

3.2.1. SERVIÇOS ANDROID

Um Serviço Android é um componente da aplicação que pode realizar operações de longa duração sem necessitar de interface gráfica. A sua execução é iniciada por outro componente da aplicação, com ou sem interface gráfica. Os serviços que são executados em segundo plano podem comunicar entre si, podendo não estar dependentes das aplicações que os instanciaram estarem ou não em execução [33].

Um serviço pode ser iniciado ou vinculado, sendo que o seu comportamento é afetado por estas duas opções. Quando um serviço é iniciado, este pode ficar em execução em segundo plano por tempo indefinido, mesmo que o componente que o iniciou seja destruído. Deve-se, por isso, interromper a execução deste tipo de serviços quando a sua tarefa é terminada. Um serviço vinculado oferece uma interface do tipo cliente-servidor que permite que, os componentes que interagem com a interface enviem solicitações e obtenham resultados. Um serviço vinculado mantem-se em execução enquanto outro componente da aplicação estiver vinculado ao serviço. Vários componentes podem-se vincular ao mesmo serviço, mas quando todos desfizerem o vínculo, o mesmo será destruído. O mesmo serviço pode ser iniciado, vinculado, ou ambos, podendo ser até utilizado por qualquer componente de outra atividade, no entanto, esse acesso pode ser restringido [33].

3.2.1.1. CICLO DE VIDA

O ciclo de vida de um Serviço apresenta apenas um tronco unidirecional mostrando-se bastante simples, principalmente, se o compararmos com o ciclo de vida de uma atividade normal. O seu encadeamento apenas está dependente do facto do serviço ser iniciado ou vinculado.

Na Figura 25, é apresentado o ciclo de vida de um serviço Android. Embora nesse diagrama os serviços sejam apresentados de forma separada, conforme os seus tipos, estes podem-se cruzar, como já apresentado acima, ou seja, um serviço iniciado pode ser vinculado por outros componentes.

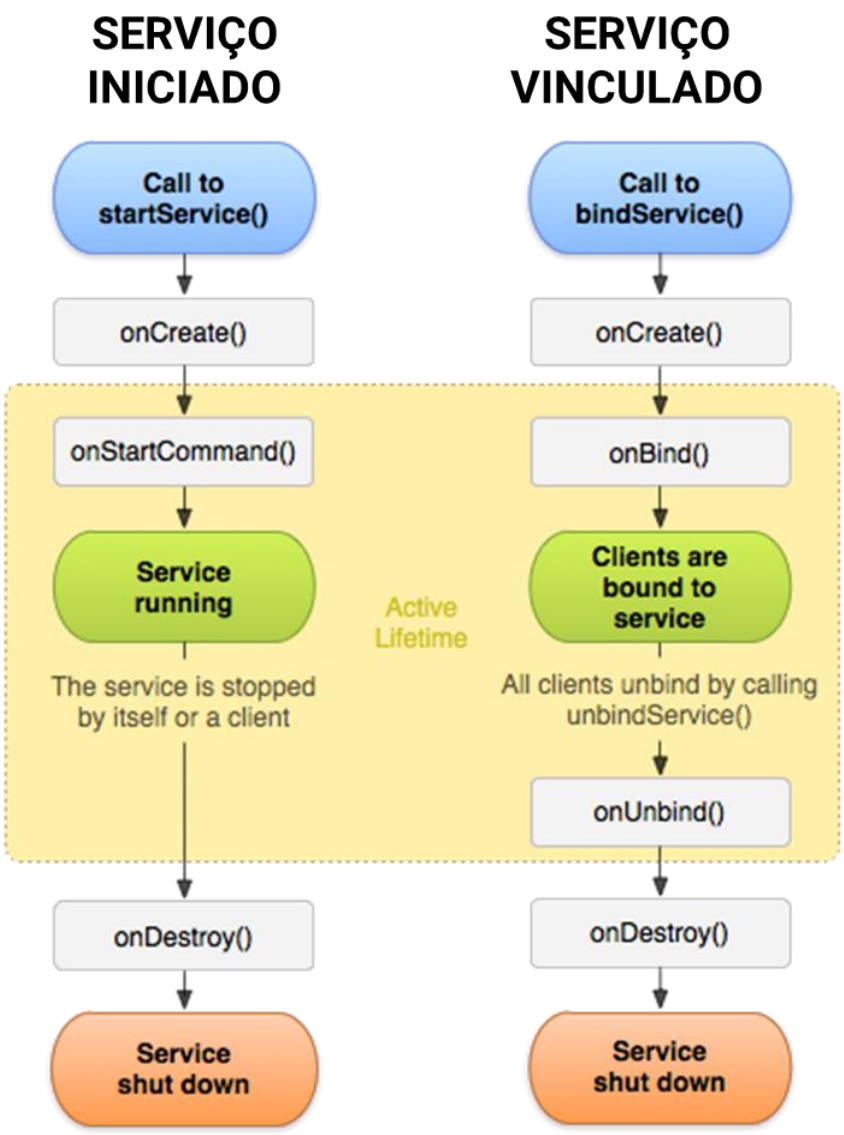


Figura 25. Ciclo de Vida de um Serviço Android [33]

3.2.2. LIMITES DE EXECUÇÃO EM SEGUNDO PLANO

Sempre que uma aplicação é executada em segundo plano acaba por consumir recursos do dispositivo, como a RAM, que são limitados. Isto pode resultar numa degradação de desempenho do próprio sistema, principalmente quando este usa aplicações que necessitam de mais recursos. Por isso, o Android restringe o que as aplicações podem realizar em segundo plano [34].

O sistema operativo distingue as aplicações entre segundo plano e primeiro plano. Contudo, no caso de restrição de execução de serviços a sua definição é diferente do que no caso da gestão de memória. Assim, uma aplicação está em primeiro plano sempre que [34]:

- Existir uma atividade visível, independente de essa atividade estar em execução ou pausada;
- Existir um serviço em primeiro plano;
- Existir outra aplicação vinculada ao serviço.

Se nenhuma das condições acima se registar a aplicação está em segundo plano.

Enquanto uma aplicação está em primeiro plano, esta pode criar e executar serviços em segundo plano livremente. Quando uma aplicação passa para segundo plano, esta tem uma janela de vários minutos onde ainda tem permissão para executar serviços. No final dessa janela é considerada que a aplicação está em *idle*. Nesse momento a aplicação termina com todos os serviços em segundo plano da aplicação [34].

3.2.3. COMPORTAMENTO DO ANDROID EM MODO DE REPOUSO

Para prolongar a vida da bateria dos dispositivos o Android implementa mecanismos para reduzir o consumo de bateria quando o dispositivo não é utilizado durante longos períodos de tempo. Quando isto se verifica, enquanto se encontra desligado de uma fonte de energia, este entra num estado de "repouso", denominado de *Doze*. Neste modo, o sistema tenta restringir o consumo de energia, adiando o acesso das aplicações à rede e ao uso do CPU [35].

Periodicamente, o sistema sai do modo de *Doze* por um breve período de tempo, permitindo a realização das atividades adiadas. Durante esta janela de manutenção, o sistema admite que as aplicações acedam à rede e que realizem e executem as suas tarefas [35].

Na conclusão de cada janela de manutenção o sistema entra novamente em modo de *Doze*, suspendendo também o acesso à rede e adiando as suas tarefas. Ao longo do tempo a frequência destas janelas de manutenção vai diminuindo[35][34]. Assim que o utilizador ativa o dispositivo, movendo-o, ligando o ecrã, ou ligando a uma fonte de energia, este sai do modo de *Doze* e todas as aplicações voltam à atividade normal [35].

Na Figura 26, é apresentado o comportamento do dispositivo quando está em repouso durante um longo período de tempo, sem estar ligado a uma fonte de energia. Vê-se que na fase inicial, mesmo após o ecrã se desligar o dispositivo contínua a sua atividade,

normalmente, até entrar no modo *Doze*. Aí diminui drasticamente o seu processamento, sendo que nas janelas de manutenção o volume de processamento volta ao normal. Verifica-se ainda que o espaçamento entre janelas vai aumentando ao longo do tempo [35].

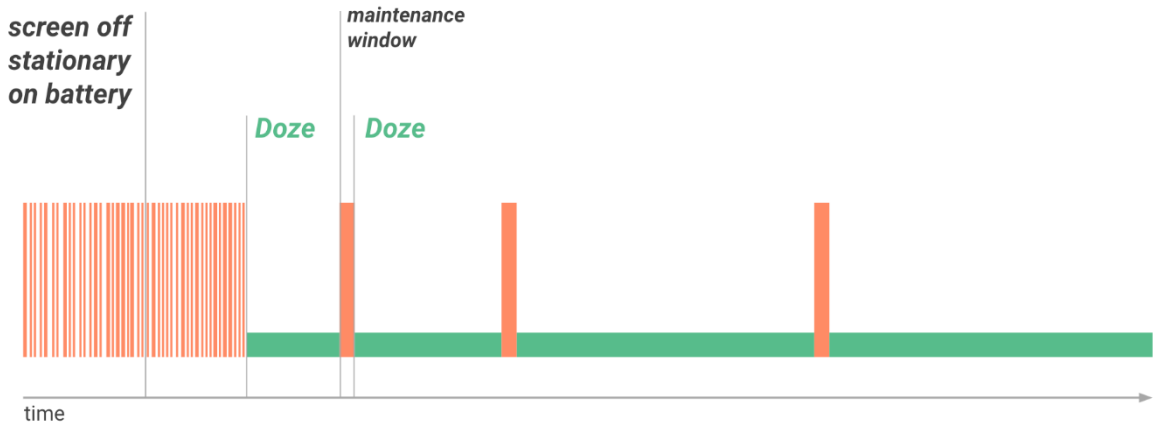


Figura 26. Comportamento de um dispositivo Android em modo de *sleep* [35]

3.2.4. Mecanismos de Contorno de Restrições de Execução

Apesar das restrições impostas pelo Android, quando as aplicações estão em segundo plano ou quando o dispositivo entra em modo de repouso, existem alguns mecanismos que permitem, de certa forma, contornar estas restrições.

Um dos mecanismos é o *JobScheduler*. Este mecanismo permite executar tarefas quando a aplicação não está a ser usada ativamente, e ainda dá a possibilidade de o sistema executar essas tarefas em janelas em que o processamento é inferior para que este não afete a experiência do utilizador. Este mecanismo pode ser executado periodicamente, embora existam algumas limitações quando é executado com o dispositivo em modo *Doze* [34].

Outro dos mecanismos que se pode utilizar para contornar as limitações provocadas pelo modo de *Doze*, é o *Firebase Cloud Messaging* (FCM). O FCM é um serviço da *cloud* para dispositivos móveis que permite transmitir mensagens com origem num serviço de *backend* para serem entregues numa aplicação. Com a utilização de mensagens de FCM de alta prioridade é possível despertar uma aplicação mesmo que o dipositivo esteja no modo *Doze.* Além de Android, o FCM fornece suporte para iOS.

3.3. MIDLEWARE

Umas das tarefas mais importantes a realizar neste projeto é o envio dos resultados para os serviços *Web* que os processam e armazenam. Estes serviços deverão estar disponíveis a partir de um endereço público, utilizando a pilha protocolar TCP/IP. Contudo, ao nível da camada de aplicação é necessário estudar qual o melhor serviço a escolher. É de salientar que os mecanismos de comunicação devem estar preparados para uma utilização exaustiva por um número elevado de dispositivos em simultâneo.

Dados os requisitos, crê-se que a melhor opção seja utilizar um protocolo de comunicação ao nível do *Middleware*. Estes protocolos estão preparados para interligarem um número variado de dispositivos de forma heterogénea, criando uma camada de abstração quanto à gestão das ligações [36].

Existem variadas soluções para integração de *Middleware*, no entanto o mais compatível com esta solução é um sistema de *Middleware* orientado às mensagens (MOM). Num sistema do tipo MOM, cada cliente pode enviar ou receber mensagens de outros clientes a partir de um servidor que funciona como intermediário. Este tipo de sistema tem várias funcionalidades na gestão e reencaminhamento de mensagens, filtragem, assim como também oferece mecanismos para comunicações síncronas e assíncronas. No seu fluxo, estes sistemas podem assumir um dos três paradigmas diferentes, *Message Passing*, *Message Queue* ou *Publish*-Subscribe [24].

Foram analisadas e avaliadas várias tecnologias, nomeadamente RabbitMQ e eJabberd para servirem de camada de transporte. Optou-se por utilizar o RabbitMQ [35], já utilizado pela empresa em conjunto com o eJabberd, no entanto adequa-se mais ao contexto do projeto.

Comparando o protocolo XMPP do eJaberd e o protocolo AMQP do RabbitMQ, pode salientar-se a funcionalidade de assinalar a presença de máquinas ligadas ao servidor implementada pelo XMPP, contudo, como este necessita de uma ligação persistente para efetuar a comunicação, e tendo o cliente de estar sempre à escuta para conseguir receber as mensagens direcionadas a si [35], conclui-se que esta não é a melhor opção quando se pretende implementar ligações com um dispositivo onde não se consegue controlar o estado da comunicação com a rede. Além disso o AMQP é um protocolo mais robusto, com um débito superior e que consegue garantir a entrega das mensagens, mesmo que a ligação seja instável [35].

Por isso nos próximos subcapítulos, são compostos por uma breve explicação da estrutura e modo de funcionamento do protocolo AMQP, assim como do RabbitMQ, servidor que o implementa.

3.3.1. AMQP - Advanced Message Queueing Protocol

O AMQP é um protocolo da camada de aplicação, que permite a troca de mensagens entre aplicações. Este protocolo de *Middleware* é orientado às mensagens, assíncrono e implementa um padrão de *publish-subscribe* [37].

Este protocolo pode ser dividido em duas camadas, a camada funcional e a camada de transporte, como apresentado na Figura 27. A camada funcional define um conjunto de comandos e de funcionalidades utilizadas ao nível da aplicação, a camada de transporte tem como função traduzir os métodos da aplicação para o servidor, gerir as ligações, a codificação do conteúdo das mensagens, a representação dos dados e o tratamento de erros [37].

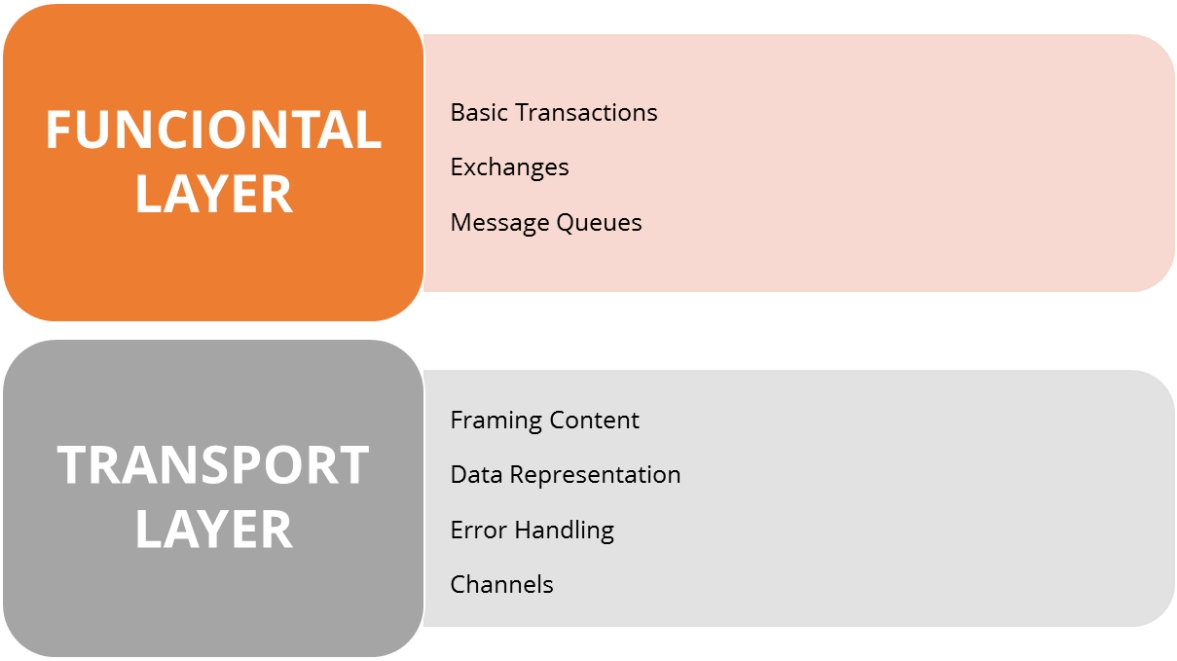


Figura 27. Camadas do protocolo AMQP [37]

Para o encaminhamento das mensagens são utilizados servidores de *Middleware*, também chamados de *brokers*, nomeadamente o RabbitMQ. Os *brokers* utilizam ferramentas que permitem efetuar a gestão dos mesmos.

Num paradigma de *publish-subscribe*, um cliente que publica as mensagens (*publisher*), não tem conhecimento dos clientes que receberão as mesmas (*consumers*) [38]. Isto cria uma camada de abstração ao cliente em relação ao destinatário da mensagem. Deixa de ser necessário saber quem é o consumidor ou o seu endereço, consegue garantir a entrega de mensagens mesmo que o(s) consumidor(es) não esteja(m) disponível(eis) naquele momento.

3.3.2. RABBITMQ

O RabbitMQ atua como intermediário na troca de mensagens AMQP. De uma forma simplista, uma aplicação, o *producer*, envia uma mensagem que em vez de ir diretamente para o seu destinatário é direcionada para um *broker* RabbitMQ, uma terceira máquina. Esse *broker* ao receber a mensagem faz o encaminhamento para uma fila de mensagens. Quando uma aplicação, o *consumer*, se liga a esse *broker* e efetua a subscrição dessa fila de mensagens, recebe a informação depositada nessa fila de mensagens [39]. Na Figura 28, é apresentado este fluxo.

Figura 28. Fluxo básico das mensagens de RabbitMQ [39]

3.3.2.1. CONCEITOS BÁSICOS

Antes de se conhecer mais especificamente o funcionamento do RabbitMQ é necessário conhecer alguns conceitos utilizados, pois são importantes no entendimento do fluxo das mensagens, sendo os mais relevantes apresentados de seguida [39]:

- *Producer*: Sistema que envia mensagens.
- *Consumer*: Sistema que recebe mensagens.
- *Queue*: Fila onde as mensagens estão armazenadas.
- *Exchange*: Recetor, por parte do *broker* RabbitMQ, das mensagens enviadas pelo *producer*, reencaminhando logo as mensagens para as respetivas *queues*,

dependendo das regras definidas e do tipo de *exchange*. A *queue* tem que estar vinculada à *e*xchange para existir toca de mensagens.

- *Binding*: Tem como função efetuar a relação entre a *exchange* e a *queue*. A *binding* específica os argumentos para efetuar o filtro das mensagens direcionada para *exchange*. Estas dependem do tipo de exchange ao qual a *queue* está vinculada. Quando uma *queue* é destruída, as suas *bindings* também são.

- *Routing Key*: É como um endereço da mensagem, utilizado para fazer o encaminhamento da mensagem para a *queue.*

3.3.2.2. FUNCIONAMENTO

Percebendo os termos descritos acima torna-se fácil perceber o fluxo das mensagens dentro do RabbitMQ. A partir da Figura 29 consegue-se dividir esse fluxo em 5 passos. No primeiro passo a mensagem é publicada pelo *producer*; no segundo, a mensagem é recebida pela *exchange*, aqui a mensagem é encaminhada conforme os seus atributos, tal como a *routing key*, dependendo do tipo de *exchange*. Seguidamente são criados os *bindings* entre a *exchange* e a *queue*, em que neste caso existem dois *bindings*, um para cada *queue*. No quarto passo as mensagens que foram recebidas na *queue* são lá armazenadas até existir um *consumer* que se ligue a essa *queue*. Por último, é representado o *consumer* ligado a uma das *queues*.

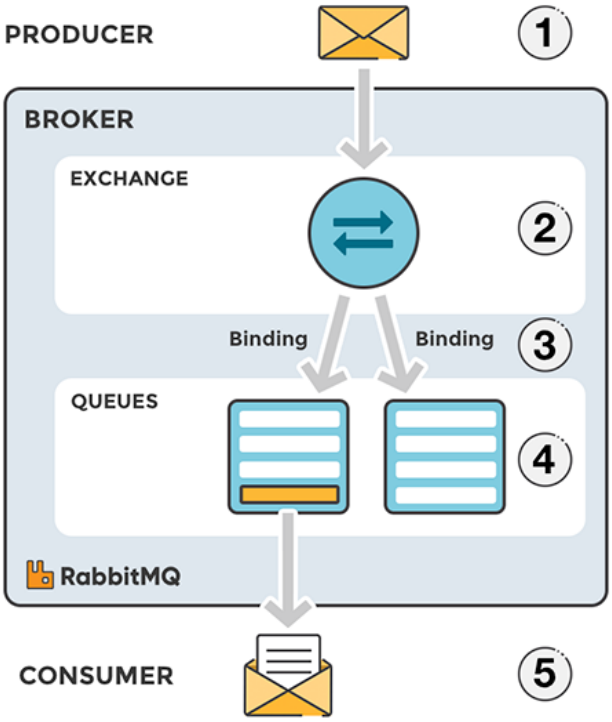


Figura 29. Demonstração do fluxo de mensagens no RabbitMQ em 5 passos [39]

3.3.2.3. LIGAÇÕES E *CHANNELS*

O RabbitMQ implementa o protocolo AMQP que funciona sobre ligações TCP/IP que asseguram a fiabilidade das mensagens recebidas. Todas as ligações estão dependentes de autenticação e podem ser encriptadas em TLS (*Transport Layer Security*). As mensagens do AMQP são transportadas a partir dos *channels* que têm também como função evitar ligações múltiplas ao mesmo *broker*, pois uma ligação AMQP pode ter vários *channels*, sendo cada um identificado com o ID único. Assim é evitado um número redundante de ligações entre as mesmas máquinas, facilitando a lógica de ligações. Por isso, em AMQP, é recomendado efetuar apenas uma ligação entre duas máquinas, sendo depois favorável criar múltiplos *channels* para efetuar a troca de mensagens entre eles [40]. Na Figura 30, é apresentada uma representação gráfica de uma ligação AMQP com múltiplos *channels*.

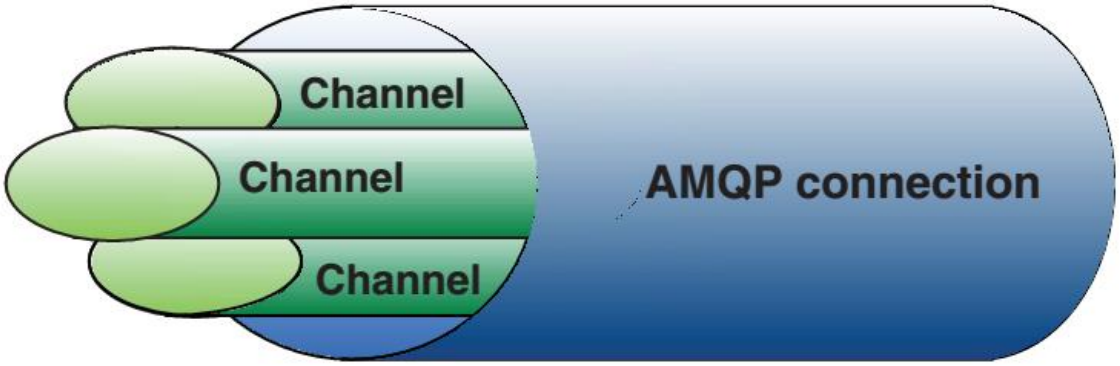


Figura 30. Representação de uma ligação AMQP com múltiplos *channels* [40]

3.3.2.4. EXCHANGE

As exchanges do RabbitMQ podem assumir quatro tipos diferentes: *direct, fanout, topic* ou *headers*. Cada um destes tipos tem uma forma diferente de efetuar o reencaminhamento de mensagens para as *queue*. De seguida serão apresentados cada um destes tipos de *exchange*.

Direct Exchange

Numa *direct exchange* o encaminhamento das mensagens pela *queue* é baseado na *routing key*, sendo que a mensagem é apenas entregue na *queue* que tiver exatamente a mesma *binding key* [39].

No exemplo da Figura 31, existe uma *exchange* do tipo dire*ct* com o nome "*pdf_events*". Quando uma mensagem é entregue nessa e*xchange,* a mesma é reencaminhada para a *Queue* A, se a *routing key* for "*pdf_create*", ou reencaminhada para *Queue B* se a *routing key* for "*pdf_log*". Nos casos em que a *routing key* tenha um valor diferente dos apresentados anteriormente, a mensagem não é entregue a qualquer *queue*. Já uma mensagem nunca pode ser entregue a duas *queues* em simultâneo.

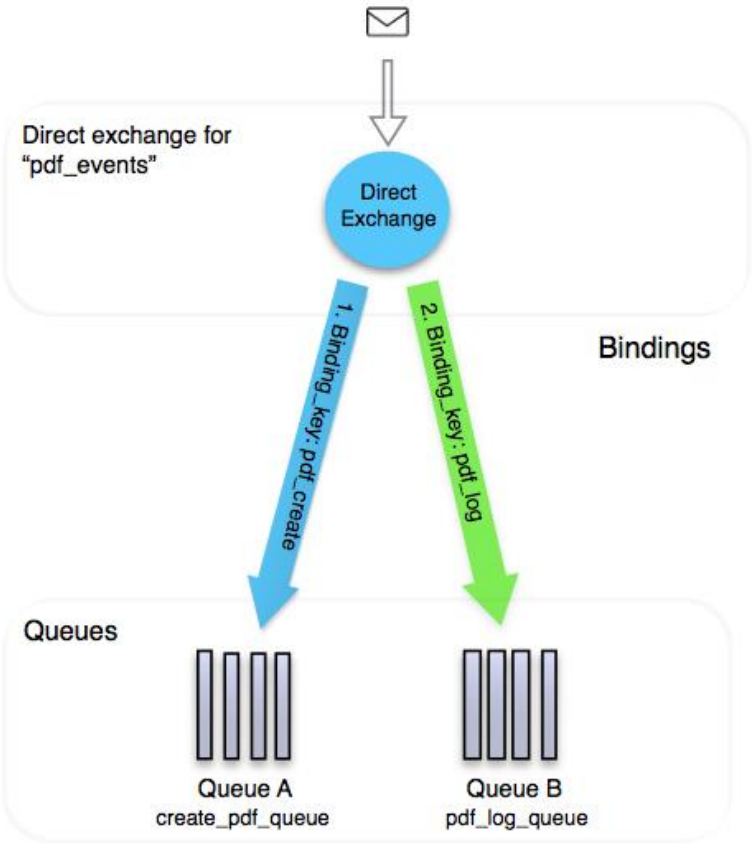


Figura 31. Exemplo de uma *Direct Exchange* [39]

Topic Exchange

Numa *exchange* do tipo *topic* as mensagens são encaminhadas através das *routing keys* das mensagens. As *routing keys* podem ser constituídas por uma lista de palavras separadas por um ponto ".". Neste caso uma mensagem pode ser entregue a mais do que uma *queue* simultaneamente dependendo das *routing keys*, neste caso chamadas *binding keys* [39].

Se uma *binding key* tiver o valor de cardinal (#), esta recebe todas as mensagens publicadas nessa Exchange. Se, por exemplo, uma *binding key* de uma *queue* tiver um valor igual a "*valor*.#" significa que essa *queue* vai receber todas as mensagens encaminhadas para essa Exchange cuja primeira palavra da *routing key* seja "*valor*". Se, por exemplo, uma *binding key* de uma *queue*, tiver um valor igual a "*valor.***.final*", significa que essa *queue* vai receber todas as mensagens cuja *routing key* tenha um valor inicial igual a "valor" e que o último valor seja igual a "*final*". Assim são permitidas várias conjunções de forma a encaminhar a mesma mensagem para múltiplas filas de mensagens

Na Figura 32, é apresentado um exemplo de uma *Topic exchange* com o nome "*agreements*". Caso seja recebida uma mensagem com um *routing key* com o valor igual a "agreements.eu.*berlin*.headstore", esta vai ser encaminhada para a três *queues*, mas se o valor dessa *routing queue* for "*agreements*.eu.lisbon.headstore" esta só vai ser recebida na *queue* B e na *queue* C, que podem ser apresentadas em múltiplas conjunções.

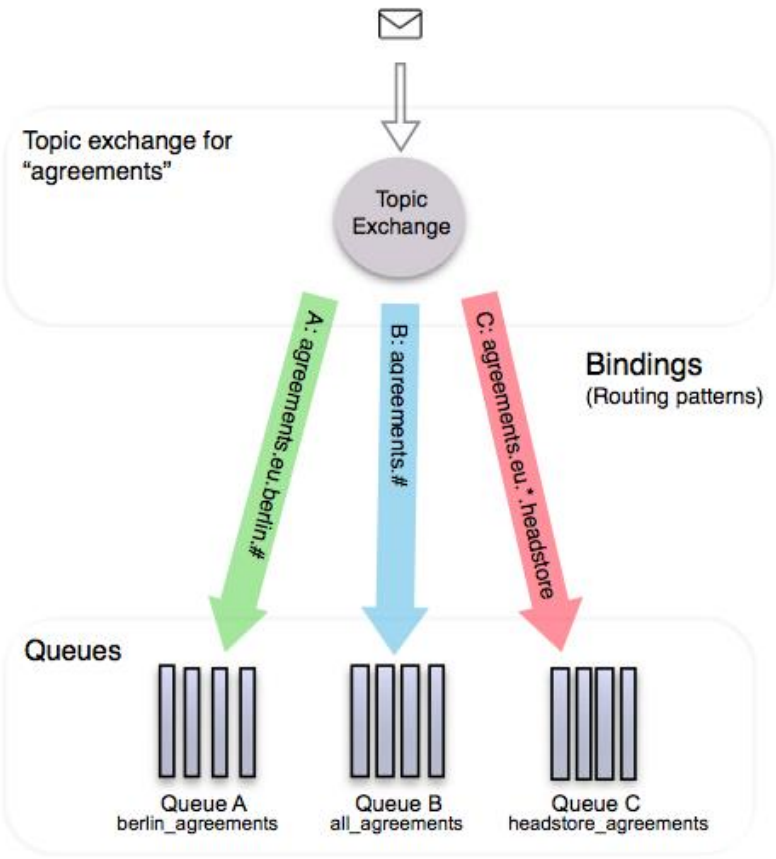


Figura 32. Exemplo de uma *Topic Exchange* [39]

Fanout Exchange

Uma *exchange* do tipo *fanout* encaminha as mensagens para todas as *queues* que a ela tiverem ligadas. Na Figura 33, é apresentada uma *exchange* com o nome "*sport news*", todas as mensagens recebidas nessa *exchange* são encaminhadas para a Queue A, Queue B e Queue C em simultâneo.

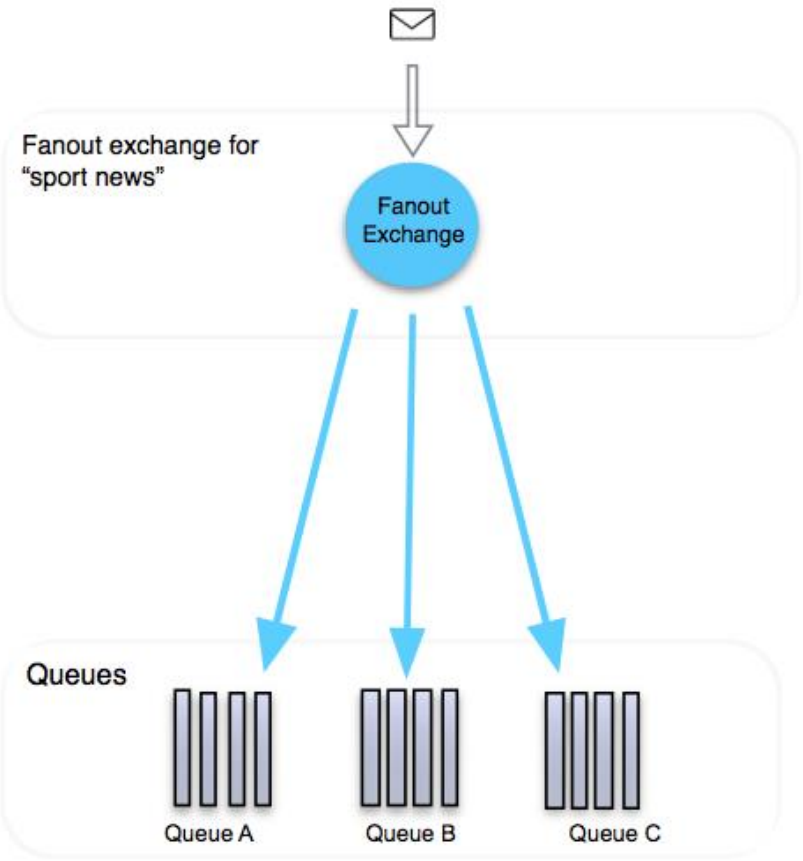


Figura 33. Exemplo de uma *Fanout Exchange* [39]

Headers Exchange

Uma *exchange* do tipo *Header* encaminha as mensagens conforme os seus cabeçalhos. O funcionamento assemelha-se bastante ao das *exchanges* do tipo *topic*, contudo, em detrimento das *routing keys*, são utilizados os argumentos do cabeçalho das mensagens para a execução do seu encaminhamento [39].

As mensagens deste tipo de *exchange* têm um argumento especial, de nome “x-*match*”, que define o grau de comparação entre os argumentos do cabeçalho das mensagens e as *binding keys* das *queues* vinculadas a essa *exchange*. O argumento “x-match” pode assumir dois valore “*any*” ou “*all*”. Se assumir o valor “*any*” a mensagem é encaminha para a *queue*, se algum dos argumentos presentes na *binding key* for igual a algum dos argumentos da mensagem. Caso assuma o valor “*all*” a mensagem é encaminhada para a *queue* apenas se todos os argumentos presentes na *binding key* forem iguais aos argumentos da mensagem [39].

No exemplo da Figura 34, é apresentada uma *exchange* do tipo *header* com o nome de "agreements". Se nesta *exchange* for recebida uma mensagem em que no cabeçalho tem os argumentos "*format = zip*","*type = report*" e "*x-match = all*", a mesma vai ser entregue na Queue C, porque todos os argumentos da *binding key* estão mencionados nos argumentos da mensagem. A mensagem não é reencaminhada para a Queue A, mesmo existindo um dos argumentos que é condizente porque o argumento x-match tem o valor de *all*. Se a esta *exchange* chegar uma mensagem com o cabeçalho contendo os argumentos "*format = pdf*","*type = report*" e "*x-match = any*", a mensagem é entregue às três *queues* vinculadas a esta *exchange*. Como o valor do argumento x-match é "*any"* basta um dos argumentos ser igual aos argumentos da *binding key*. Assim na Queue A existe concordância em relação a todos os argumentos, na Queue B existe concordância em relação ao argumento *format* e na Queue C em relação ao argumento *type*.

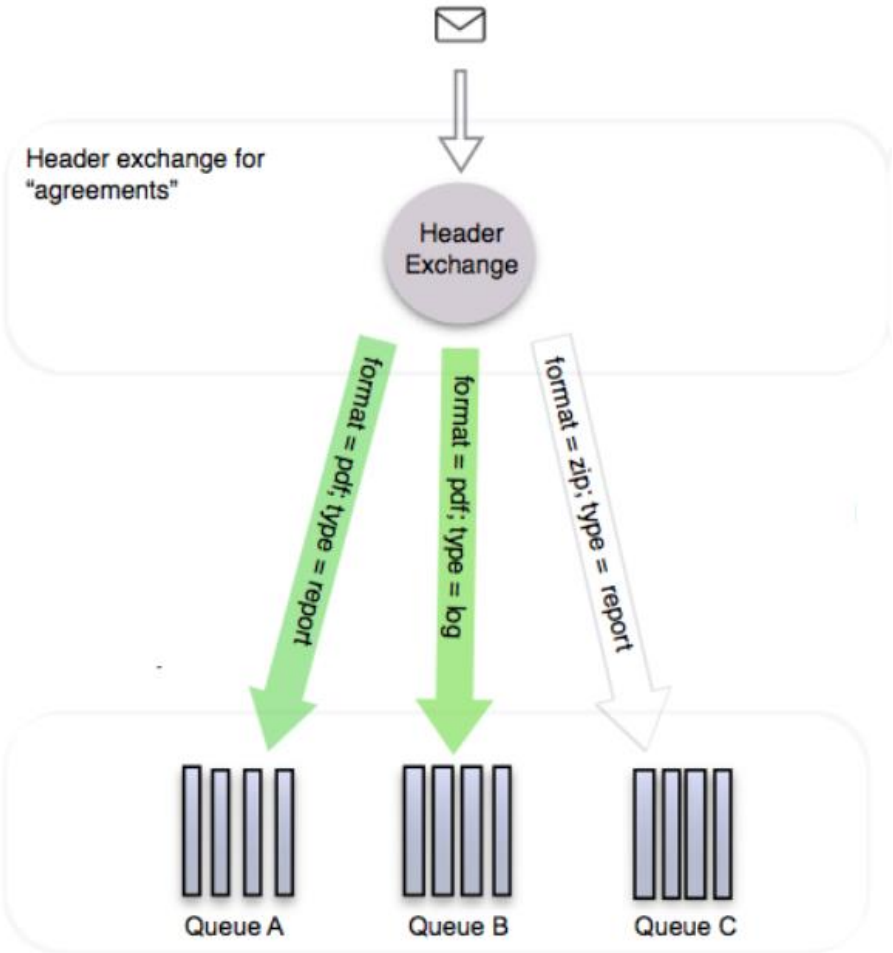


Figura 34. Exemplo de uma Header Exchange [39]

3.3.2.5. QUEUE

Para prevenir a sua perda as mensagens são armazenadas em *queues* (ou filas de mensagens), o RabbitMQ é responsável por facilitar a criação, o acesso e a gestão destas.

O *broker* é responsável por receber as mensagens dos clientes, o ato de *publish*, e enviar mensagens para que todos os clientes façam subscribe, O despacho das mensagens é feito a partir de uma abordagem de FIFO (*First In First Out*), ou seja, a primeira mensagem a ser recebida é a primeira mensagem a ser enviada [33]. O encaminhamento de mensagens para as respetivas *queues* é feito em *round-robin*, fornecendo uma garantia de que os consumidores não recebem mensagens em excesso. O encaminhamento das mensagens para a *queue* é feito conforme o tipo de *exchange* ao qual esta está vinculada e conforme a *routing key* da mesma [33].

3.4. NOTAS FINAIS

Ao longo deste capítulo foram apresentadas as tecnologias utilizadas ao longo do desenvolvimento deste projeto. Deste capítulo é importante reter a arquitetura do Cordova e as restrições impostas pelo Android na execução de tarefas em segundo plano, tal como os mecanismos existentes para contornar estes dispositivos. Para melhor se arquitetar todo o fluxo de comunicações torna-se também importante conhecer o funcionamento do RabbitMQ.

No próximo capítulo, será apresentada a especificação funcional do projeto e a arquitetura do mesmo. Nele pretende-se expor as bases para o desenvolvimento do projeto.

4. ESPECIFICAÇÃO FUNCIONAL

Ao longo deste capítulo apresenta-se em síntese a estrutura da solução pretendida assim como o seu funcionamento, dando ao utilizador uma perspetiva básica sobre os processos.

Numa ótica funcional toda a solução pode ser dividida em três grupos: Beacon, Smartphone e Servidor. Tal como apresentado na Figura 35. No início da cadeia apresentam-se os Beacons, que comunicam unilateralmente com o Smartphone. O Smartphone é o elemento central desta cadeia que comunica diretamente com o Servidor, enviando para este as informações obtidas nos Beacons e executando as ações indicadas por este. O Servidor recebe as informações obtidas pelo Smartphone e envia ordens para o mesmo.

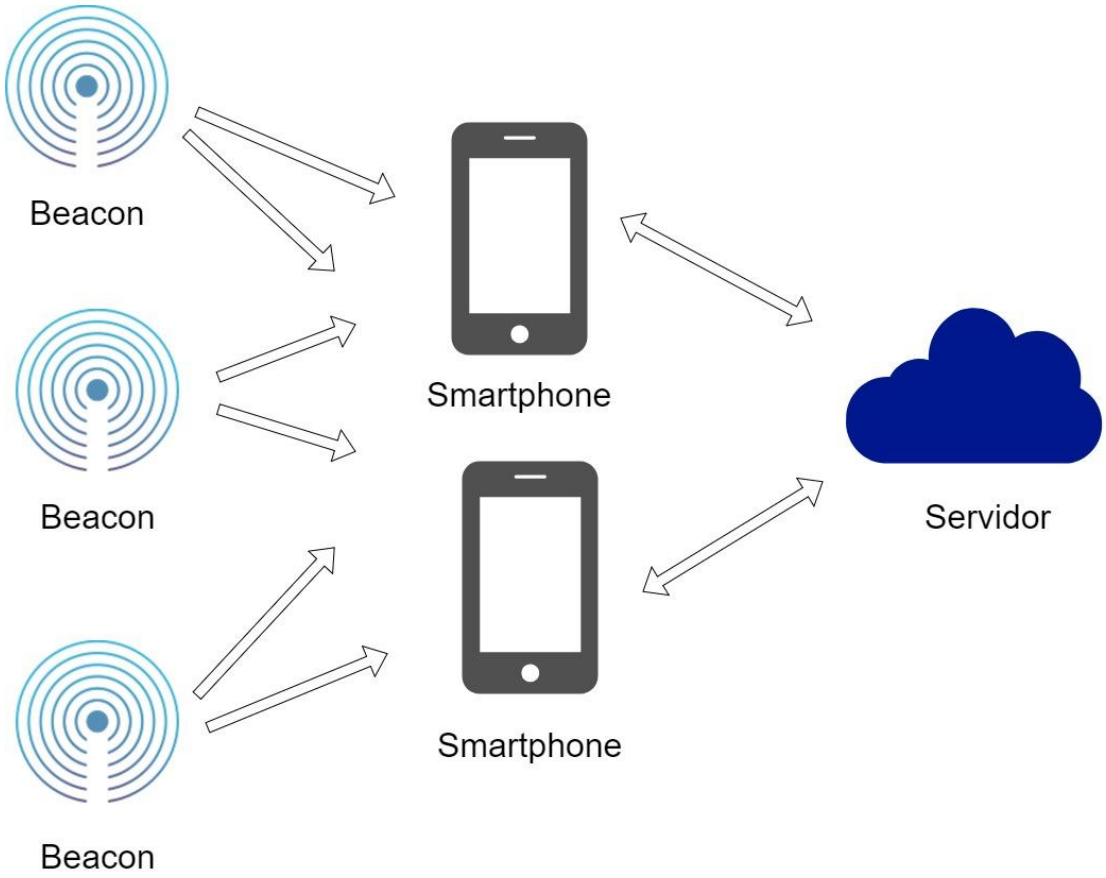


Figura 35. Diagrama Funcional Geral

Desta análise conseguem-se obter as tarefas e objetivos de cada um dos elementos. Os Beacons devem ser capazes de emitir informação que permita identificá-los e localizá-los. O Smartphone deve ser capaz de estar constantemente a obter informação, efetuar um primeiro processamento e filtragem para de seguida enviar para o Servidor, isto executado em segundo plano sem qualquer ação do utilizador. Este deve ainda ser capaz de atender e executar os pedidos do Servidor. O Servidor deve ser capaz de comunicar com Smartphone, guardar a informação, processar, aplicar a lógica de negócio e dar indicação ao mesmo das tarefas a executar. Ao nível técnico, todo o sistema deverá ser compatível com a aplicação YouBeep.

No próximo capítulo é apresentada a forma como foi desenhado e implementado o sistema, sendo apresentada a sua estrutura e também de todos os seus elementos.

5. DESENHO E IMPLEMENTAÇÃO

Depois de serem apresentados os vários fundamentos sobre o projeto, o presente capítulo começa por uma abordagem geral da sua arquitetura, passando por cada um dos módulos que o constituem, sendo logo apresentada a sua estrutura, a sua implementação e o resultado final.

5.1. ARQUITETURA GENÉRICA

Seguindo o raciocínio apresentado no capítulo "Especificação Funcional", este projeto é constituído por três grupos principais: os Dispositivos Detetáveis, o Smartphone e os Componentes da Rede. Tendo em conta as funcionalidades pretendidas e a especificidade das tecnologias utilizadas estes grupos subdividem-se, assumindo a estrutura apresentada na Figura 36.

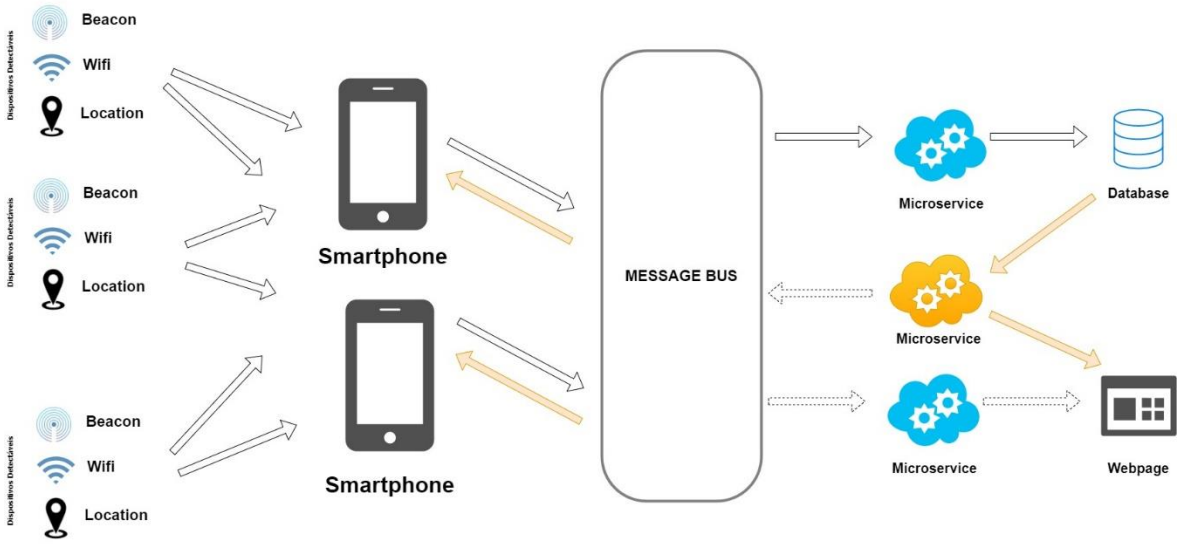


Figura 36. Arquitetura Geral do Sistema

Os dispositivos detetáveis são elementos externos ao sistema, contudo pretende-se conseguir detetá-los e identificá-los a partir do Smartphone, assim a este foram incorporadas funcionalidades de rastreamento de dispositivos *beacons*, de rastreamento de pontos de acesso de *Wifi* e de rastreamento da geolocalização. Assim consegue-se efetuar a localização do dispositivo a três níveis diferentes. O rastreamento de *beacons* e de pontos de acesso *Wifi* permite efetuar o posicionamento relativo destes dispositivos sendo possível a partir dos *beacons* ter um posicionamento mais refinado, dado o seu pequeno alcance, e a partir dos dispositivos *Wifi* ter uma cobertura superior, dado o grande número de dispositivos instalados e disponíveis. O uso da geolocalização dos dispositivos permite efetuar o mapeamento geográfico.

A estrutura da aplicação do Smartphone divide-se em duas partes. Uma é composta pelos recursos utilizados pelo Cordova, na outra temos os recursos nativos utilizados para efetuar o *scanner* dos dispositivos e envio de resultados. Estes encontram-se integrados num *plugin* Cordova.

Como interface, entre o Smartphone e os serviços na rede, existe um Message Bus, este é constituído por um *broker* RabbitMQ responsável pela gestão das ligações e o encaminhamento das mensagens.

Na rede existem Microserviços para apoiar a interligação dos elementos deste projeto e podem assumir dois tipos. Um dos tipos de microserviços é responsável por consumir as informações obtidas pelos Smartphones, a partir do Message Bus. Estes serviços têm informações em tempo real do mapeamento feito. São estes os serviços responsáveis por inserir os dados na base de dados, tendo sido também pensadas funcionalidades de representação dos resultados obtidos em tempo real. O outro tipo de microserviços é capaz de a partir de informações da base de dados responder a pedidos. Este serviço está interligado com a *webpage* sendo responsável por obter os dados para esta apresentar. Foi também pensado aproveitar este dispositivo para dar indicações ao *smartphone* das ações a executar, a partir do Message Bus.

A arquitetura deste projeto foi desenhada de forma a suportar a replicação dos módulos do mesmo, para garantir a escalabilidade do sistema, com a possibilidade de replicação e paralelismos na sua execução. Nos próximos pontos serão apresentados de forma mais pormenorizada todos os módulos que o constituem.

5.2. BEACON SERVICE *PLUGIN*

O Beacon Service *Plugin* é o *plugin* desenvolvido para efetuar a deteção de dispositivos *Wifi* e de dispositivos iBeacon em segundo plano e embebido numa aplicação móvel híbrida que utilize a *framework* Cordova no sistema operativo Android.

Este *plugin* faz a ponte entre a camada do Cordova e os mecanismos nativos utilizados para se conseguir efetuar a deteção de dispositivos. No diagrama da Figura 37 são apresentados os vários componentes que constituem o *plugin*, com as setas a indicarem o sentido da comunicação entre eles.

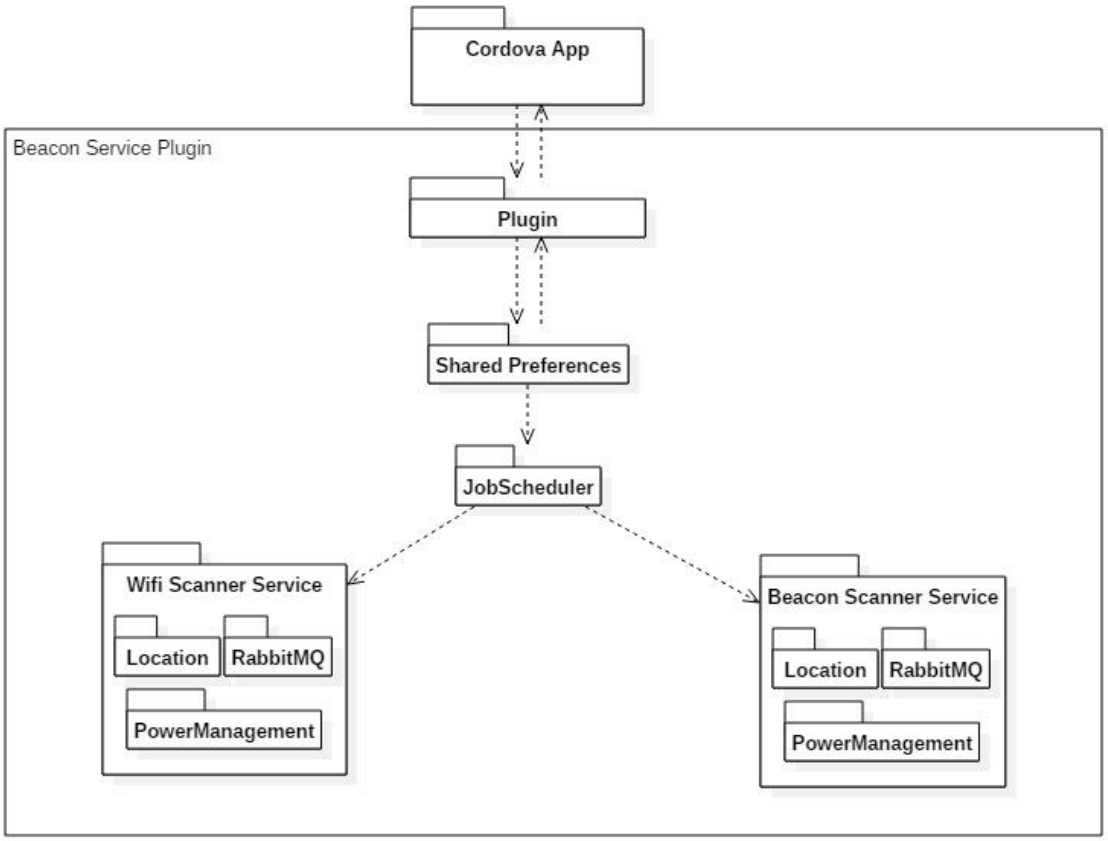


Figura 37. Diagrama de módulos do Beacon Service *Plugin*

Este *plugin* implementa, na aplicação, processos completamente independentes do processo principal. Apenas assim se consegue que as tarefas continuem em execução continuamente, mesmo com o processo principal desligado. Contudo, na realidade, os processos implementados pelos *plugin* não estão continuamente em execução, estando apenas "vivos" durante o tempo estritamente necessário para efetuar as tarefas necessárias. O controlo da execução destes serviços é efetuado a partir da utilização de um JobScheduler. O nível do Cordova apenas tem controlo e conhecimento do estado dos serviços.

O processo principal implementado pelo *plugin* é apresentado na Figura 38. Nos pontos seguintes serão apresentados de forma mais detalhadas todos os componentes assim como a interação entre eles.

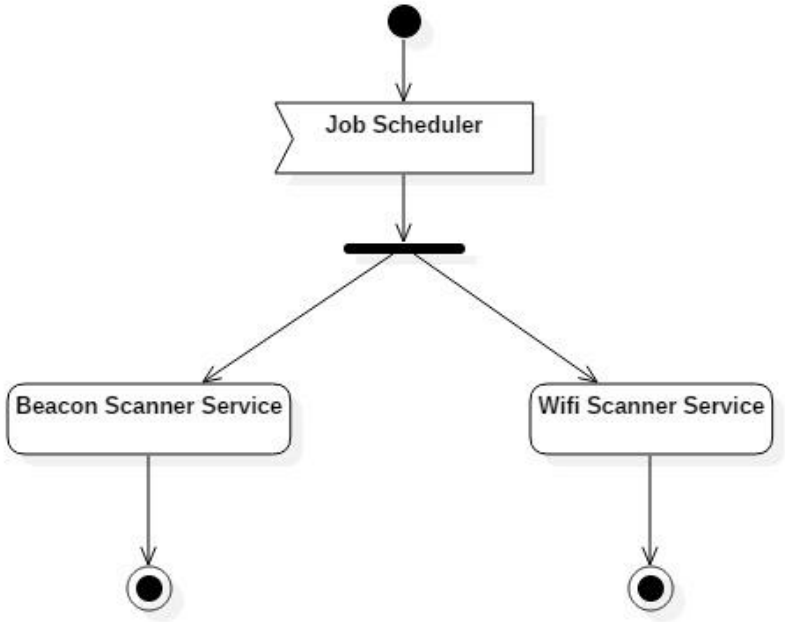


Figura 38. Diagrama de Atividade do processo principal do Beacon Service *Plugin*

5.2.1. JOB SCHEDULER

O Job Scheduler é um mecanismo do Android que permite invocar tarefas em segundo plano numa aplicação. Com a utilização deste mecanismo consegue-se ultrapassar as limitações do sistema operativo quanto à utilização desta abordagem.

Este mecanismo está implementado com o objetivo de ciclicamente iniciar os serviços de *scanner* a partir de um serviço com um tempo de execução muito curta, o JobService. Embora o período entre execuções seja um valor configurável o mesmo pode variar conforme o estado do dispositivo, como por exemplo: se o dipositivo estiver no estado de *Doze* as tarefas do JobService apenas são executadas dentro da janela de manutenção, tal como explicado no ponto 3.2.3. Na Figura 39, é apresentado um exemplo da execução ao longo do tempo do JobService, sendo que, "t" representa o tempo entre execuções.

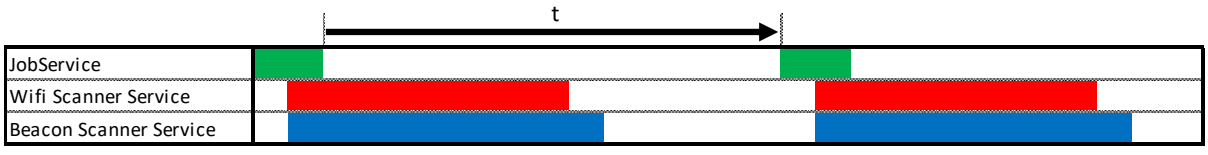


Figura 39. Diagrama Temporal representativo da execução do JobScheduler

Tal como apresentado no digrama de atividade da Figura 40, o JobService efetua o início dos serviços de *scanner* conforme o estado das configurações. Como o Android 7 não permite a execução deste mecanismo de forma periódica é necessário reconfigurar o JobService antes deste terminar a sua execução, assim é garantido que após o intervalo definido este irá reiniciar a sua execução.

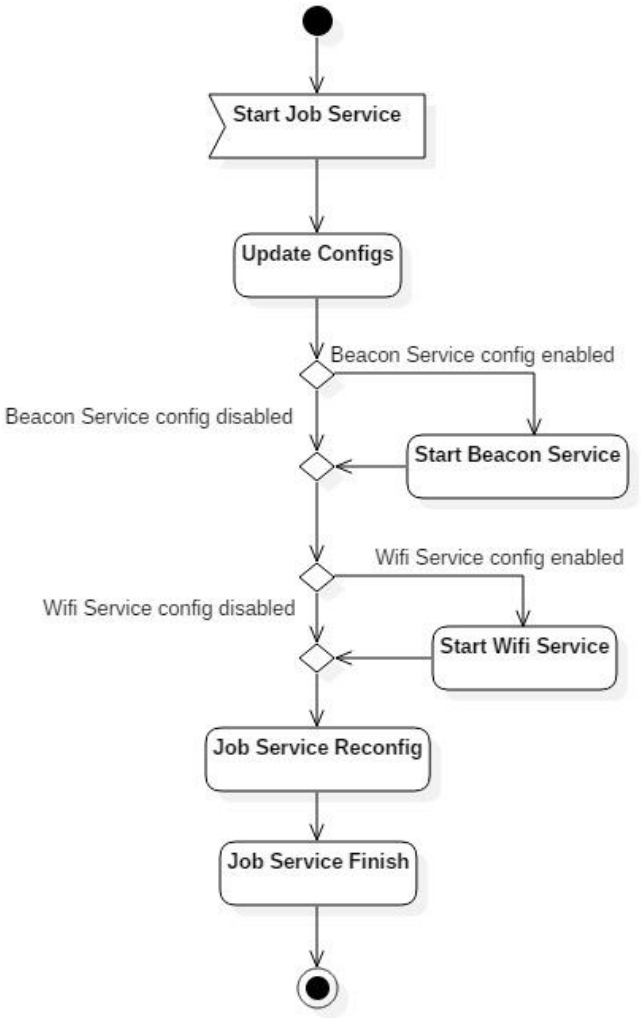


Figura 40. Digrama de Atividade do JobService

O JobService é um recurso que está em execução estritamente no tempo necessário para iniciar os serviços de *scanner*, efetuando execuções repetitivas logo a partir do *boot* do sistema operativo do dispositivo móvel, tal como apresentado no diagrama da Figura 41.

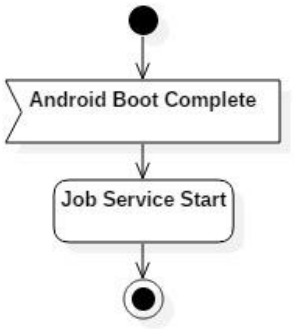


Figura 41. Diagrama de Atividade após o *boot* do Android

Ao completar o *boot*, o Android emite um sinal, para o receber é necessário implementar um Broadcast Receiver, tal como o apresentado abaixo e colocá-lo à escuta desse sinal.

```
public class YoubeepReceiver extends BroadcastReceiver {

    JobServiceUtils mJobService = new JobServiceUtils();

    public void onReceive(Context context, Intent intent) {
        mJobService.start(context);
    }
}
```

Para configurar o *receiver* é necessário identificá-lo no Android Manifest, assim como indicar o sinal que faz iniciar a sua execução. Implementado tal como o código apresentado de seguida.

```
<receiver android:enabled="true" android:exported="false"
android:name="com.beacon.plugin.YoubeepReceiver">
    <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

5.2.2. BEACON SCANNER SERVICE

O Beacon Scanner Service é responsável por efetuar o rastreamento de dispositivos iBeacon e a comunicação dos mesmos com o *broker* RabbitMQ que se encontra na rede. A sua

execução é feita em segundo plano. No diagrama de atividade da Figura 42 são apresentadas as fases de execução deste serviço.

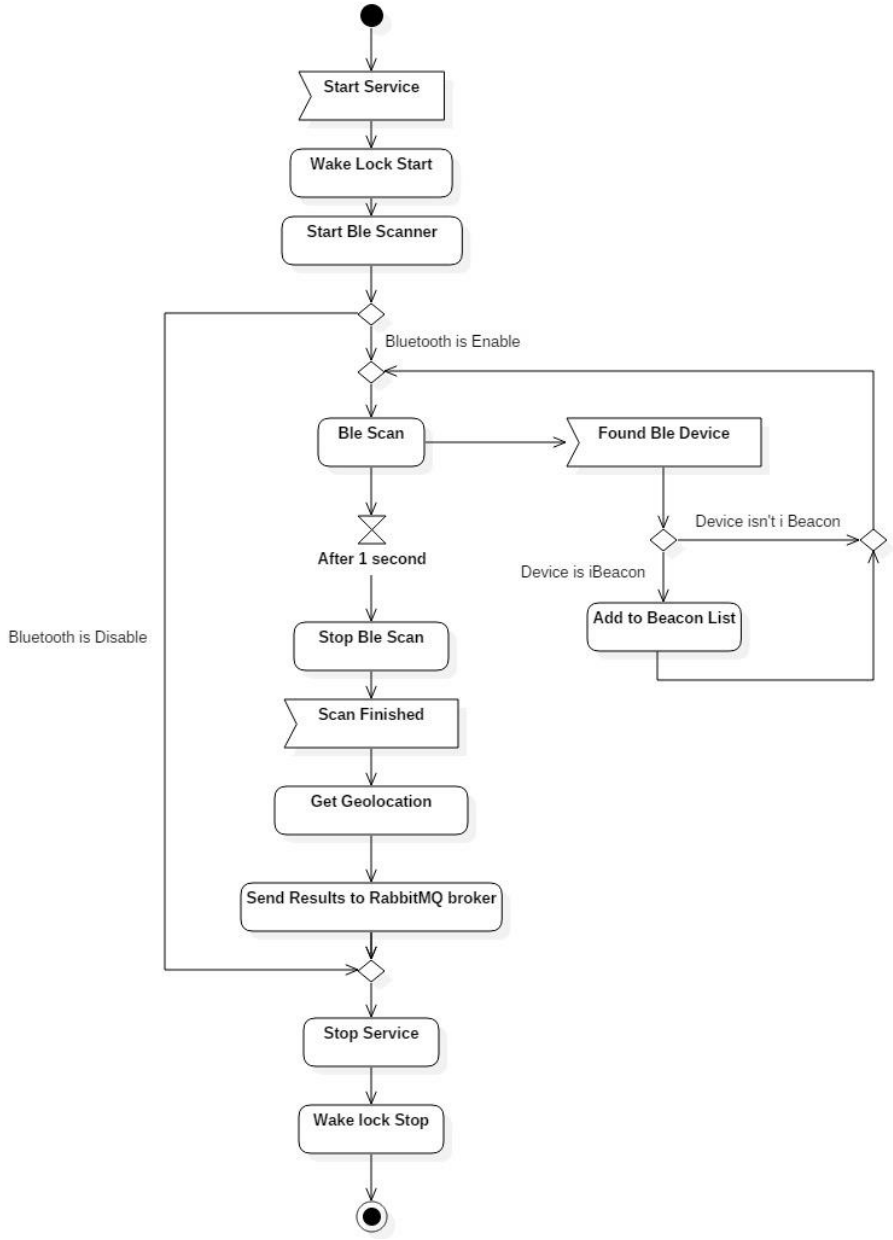


Figura 42. Diagrama de Atividade do Beacon Scanner Service

Logo que este serviço inicia, o mecanismo de Wake Lock é ativado para que o processador não entre em modo de repouso enquanto o serviço está em execução, evitando que este o tente destruir para poupar memória, processamento e bateria. De seguida é verificado se o *bluetooth* do dispositivo se encontra ativo, se este estiver desativado o serviço é prontamente parado e libertado o mecanismo de Wake Lock anteriormente utilizado. Se o *bluetooth* do dispositivo estiver ativo o *scanner* de dispositivos BLE é iniciado.

Após ser iniciado o rastreamento de dispositivos BLE, o dispositivo fica à escuta durante 1 segundo. Quando um dispositivo BLE é encontrado é verificado se este é um iBeacon, caso não se trate de um dipositivo compatível não é executada outra tarefa, dando continuidade ao *scanner*, caso contrário os dados de interesse do dispositivo são adicionados à lista dos resultados do *scanner*, continuando de seguida a tarefa até ao limite do tempo estipulado.

Quando o tempo dedicado a esta tarefa chega ao fim, é terminada a execução da tarefa de *scanner* sendo feito o envio dos resultados obtidos para o broker de RabbitMQ, juntamente com a geolocalização do dispositivo e outros argumentos necessários.

De seguida ao início do envio dos resultados é desbloqueado o mecanismo de Wake Lock e o serviço termina a sua execução. No fluxograma da Figura 43 é apresentado o fluxo de operações para a confirmação de que o dispositivo encontrado é o iBeacon.

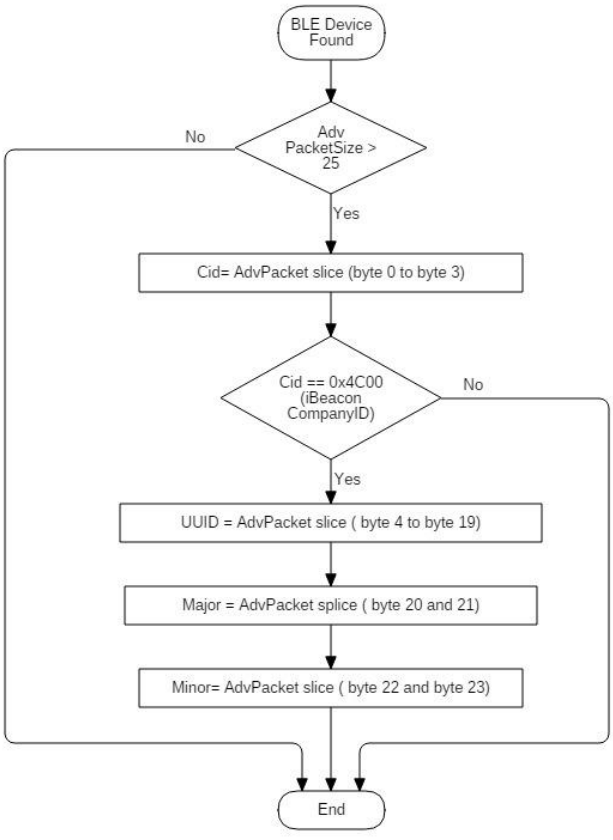


Figura 43. Fluxograma da verificação de dipositivo iBeacon

Assim, quando é detetado um dispositivo BLE, ou seja, quando é recebida uma trama de *advertise*, é feita a confirmação de que o dispositivo encontrado é um iBeacon. Essa verificação é feita através da comparação de dois parâmetros.

O primeiro parâmetro é a quantidade de informação recebida, ou seja, se a trama de *advertise* tiver um número de bytes inferior ao número de bytes de uma trama de advertisse de um iBeacon a mesma é rejeitada, caso seja superior é feita uma nova verificação. Se os dois primeiros bytes da trama recebida coincidirem com o Company ID da Apple, ou seja, tiverem o valor 0x4C00 isso quer dizer que o dispositivo é um iBeacon. Este método de confirmação corrobora com a informação apresentada na Figura 14, quando se aborda a estrutura das tramas de *advertise* do protocolo iBeacon.

5.2.3. WIFI SCANNER SERVICE

O Wifi Scanner Service é responsável por efetuar o rastreamento de Access Points *Wifi*, e o posterior envio dos resultados para o *broker* RabbitMQ, tendo um funcionamento idêntico ao Beacon Scanner Service.

Logo que este serviço inicia, o mecanismo de Wake Lock é ativado para que o processador não entre em modo de repouso enquanto o serviço está em execução, assim como o mecanismo de Wifi Lock, para que o sistema operativo não desligue o *wifi* para baixar o consumo. Assim tenta-se garantir que o serviço não é desligado de forma abrupta sem ter terminado as suas tarefas.

De seguida é efetuada a verificação do estado do *wifi* do dispositivo. Caso o dispositivo tenha o *wifi* desligado o serviço para imediatamente libertando os recursos anteriormente bloqueados. Caso o dispositivo tenho o *wifi* ativo é iniciado o rastreamento de dispositivos.

O scanner de *wifi* termina por ação de um sinal do sistema operativo indicando a existência de resultados (mesmo que não tenham sido encontrados dispositivos). Depois é feito o envio dos dados para o *broker* RabbitMQ. Após esta tarefa ser iniciada não é feito o término de serviço sem antes libertar os recursos de processamento e de *wifi*.

Neste caso não é necessário efetuar qualquer divisão dos dados obtidos, pois a API do Android devolve os dados numa estrutura em que por cada dispositivo é indicado o SSID, o BSSID (ou o endereço MAC), a frequência de funcionamento, nível de segurança, entre outros. Na Figura 44 é apresentado o diagrama de atividade deste serviço.

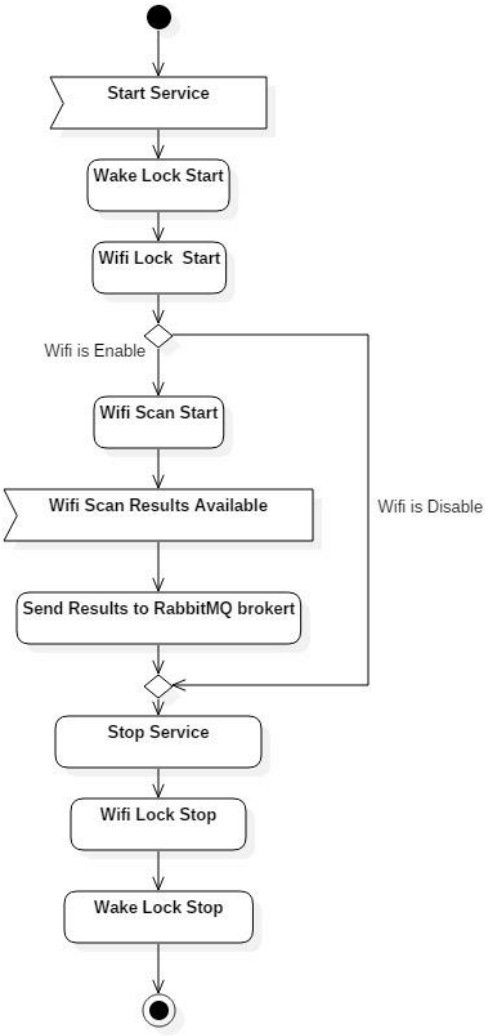


Figura 44. Diagrama de Atividade do Wifi Scanner Service

5.2.4. INTERFACE DE COMUNICAÇÃO COM A CAMADA DO CORDOVA

Outro dos módulos com maior peso neste *plugin* é a interface com a *framework* Cordova. Este módulo permite, por exemplo, controlar elementos nativos de h*ardware* a partir de linguagens web, como o javascript. Neste caso existe troca de mensagens entre a camada nativa e a camada Cordova quando é necessário ler ou alterar as variáveis de controlo dos serviços de *scanner*. É evidente que quando se aborda a comunicação entre estas camadas também se torna necessário conhecer a comunicação entre os módulos do *plugin*.

Nesta interface estão envolvidos vários módulos construídos em diferentes tecnologias. No Cordova a implementação do *plugin* comunica com a interface do Cordova e este último está em contacto direto com a camada nativa do sistema operativo.

A camada nativa tem uma interface, desenvolvida em Java, com a função de converter as mensagens vindas da camada do Cordova, assim como as de sentido inverso. A comunicação entre este módulo e os restantes dentro do *plugin* não é feita diretamente, mas sim a partir das Shared Preferences (ou preferências partilhadas). As Shared Preferences têm como função servir de banco de dados de pequeno volume de informação, aí são guardadas informações sobre os estados dos serviços e a frequência de *scanners*.

Na Figura 45, é apresentado o esquema de comunicação entre as duas camadas e a comunicação entre os módulos que as compõe, estando também assinalado o sentido da comunicação entre eles, assim como a tecnologia em que estão construídos.

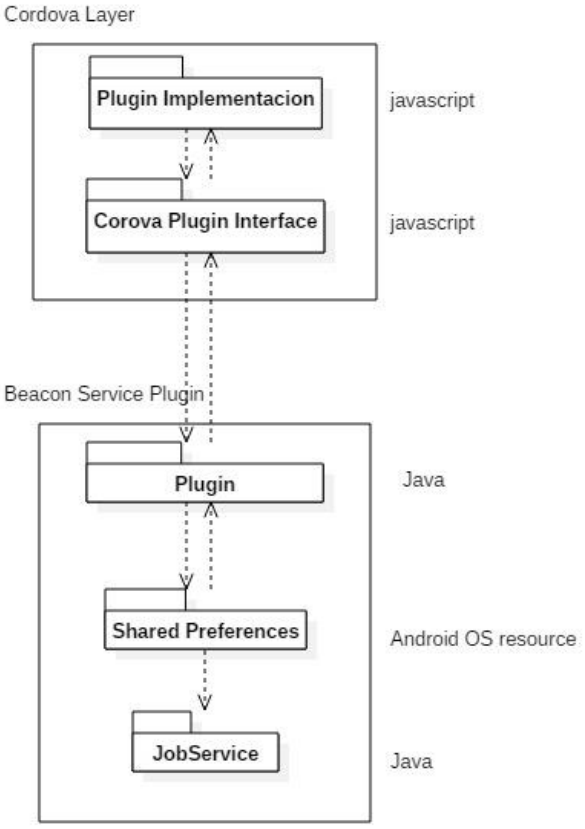


Figura 45. Representação dos módulos envolvidos na comunicação entre o *plugin* Beacon Service *Plugin* e a camada do Cordova

As mensagens trocadas entre as duas camadas são normalmente sobre os estados dos serviços, com estas mensagens é possível altera-los e conhecê-los, cada um dos serviços (o Beacon Scanner Service e o Wifi Scanner Service) são tratados independentemente. Também é possível consultar e alterar a frequência de funcionamento destes serviços.

Como exemplo da troca de mensagens entre as duas camadas, e os módulos que as compõe, é apresentado na Figura 46 a inicialização do Wifi Scanner Service. Assim, quando na implementação do *plugin,* é executada a função para iniciar o serviço é enviado pelo Cordova a mensagem com o valor "startWifiService" para a camada nativa. Aí é escrito nas Shared Preferences que o estado desse serviço é ativo. No final é retornada uma mensagem de sucesso para a camada do Cordova, sendo que isso permite executar um evento de sucesso na implementação do *plugin*.

Na execução seguinte do JobService, aquando a atualização das configurações ao requisitar o estado deste serviço, é indicado que o mesmo tem o valor de "verdadeiro", significando que deve executar o serviço nessa iteração.

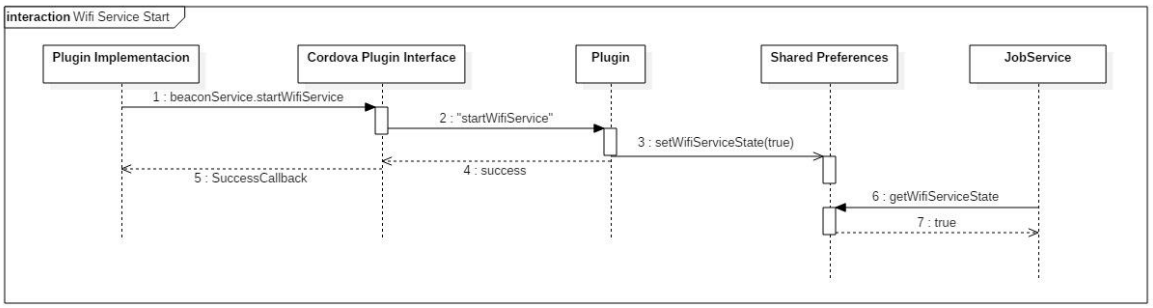


Figura 46. Diagrama de Sequência da inicialização de Wifi Scanner Service através do Cordova

A implementação desta interface obriga à construção de código e diferentes linguagens, mas cumprindo regras muito restritas de forma a eliminar todos os erros e incongruências. Utilizando o mesmo exemplo, no trecho de código abaixo é apresentado a invocação da inicialização do serviço de Wifi na implementação do *plugin*. Nele também são indicados os eventos realizados em caso de sucesso e em caso de erro durante a execução.

```
beaconService.startWifiService(xhockService.beacon.startWifiService.successCallback,
            xhockService.beacon.startWifiService.errorCallback);
```

Aquando a execução da função é realizada a transcrição da mesma para o formato de mensagem, sendo que é indicada a classe do Java no nativo que responde ao pedido e também uma mensagem com a ação a realizar. A estes dados são também indicados os eventos em caso de sucesso ou em caso de erro.

```
module.exports = {
    startWifiService: function(successCallback, errorCallback) {
        cordova.exec(
            successCallback,
            errorCallback,
            "BeaconPlugin",
            "startWifiService"
        );
    },
(…)
```

Através dos mecanismos implementados pelo Cordova, os parâmetros indicados no trecho anterior são convertidos em parâmetros de entrada na classe do Java. Aí são executadas as tarefas necessárias, sendo depois dado aos níveis anteriores a indicação de sucesso da operação.

```
public class BeaconPlugin extends CordovaPlugin {

    @Override
    public boolean execute(String action, JSONArray data, CallbackContext
callbackContext) throws JSONException {

        if (action.equals("startWifiService")) {

            boolean state= true;
            prefs.setWifiServiceState(context, state);
            jobServiceForceRunning();

            callbackContext.success();
            return true;
(…)
```

É importante salientar que o comportamento desta interface não se altera em qualquer que seja a aplicação que este *plugin* esteja integrado, desde que a mesma seja construída sobre a *framework* Cordova, permitindo assim que estas funcionalidades sejam portáveis.

5.2.5. MECANISMOS AUXILIARES

Na implementação de todas a funcionalidades principais do *plugin* são utilizados mecanismos que, embora não tenham uma função principal, apoiam no seu funcionamento, fazendo assim cumprir os requisitos pretendidos.

5.2.5.1. RABBITMQ *PUBLISHER*

No final de se efetuar a aquisição dos dados, estes são enviados para o Message Bus. Para tal tornou-se necessário implementar um *publisher* RabbitMQ, tarefa esta executada após a obtenção dos resultados sobre os dispositivos ao redor.

O processo de publicação de resultados é efetuado numa *thread* que executa em paralelo com o resto da implementação. Isto torna-se necessário devido a esta ser uma tarefa assíncrona, não sendo boa prática fazer com que o resto da aplicação espere que esta tarefa seja concluída, pois esta termina após findar o envio dos dados necessários. Sendo este tempo variável

De seguida é criada a ligação do canal de comunicação com o *broker*. Depois as mensagens que estão na fila de mensagens são enviadas até esta ficar vazia. Sendo que depois os canais e ligação criados anteriormente são terminados. Na Figura 47 é apresentado o diagrama de atividade do *publisher* implementado.

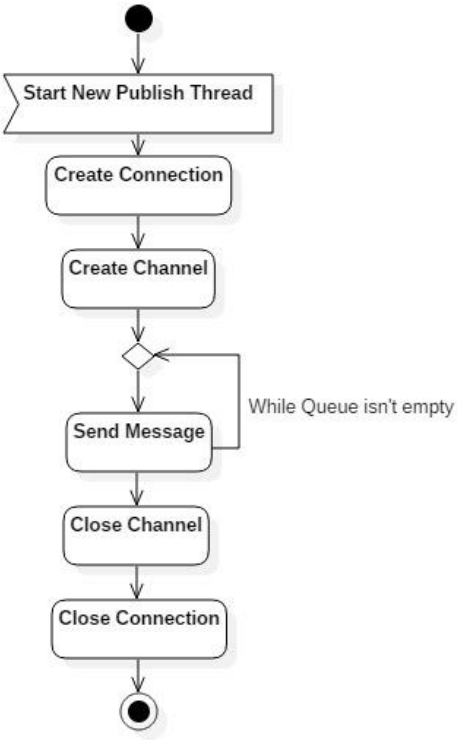


Figura 47. Diagrama de Atividade do *publisher* RabbitMQ

Para efetuar o envio das mensagens é necessário indicar a *Eechange* e a *routing key* e enviar a mensagem pretendida convertida em bytes, tal como no exemplo abaixo.

```
ch.basicPublish(CLOUDAMQP_EXCHANGE, routingKey, properties, message.getBytes());
```

5.2.5.2. CHECK PERMISSIONS/ CHECK STATE

Para o correto funcionamento de algumas das funções deste *plugin* existem requisitos ao nível das permissões de execução e dos estados dos componentes de *hardware* dos dispositivos, no caso o *Bluetooth*, o wifi e a localização. Como tal foi desenvolvido um mecanismo para confirmar os estados dos recursos acima referidos, tendo sido estas funcionalidades integradas com a camada do Cordova.

Dada a existência de uma dependência de ação do utilizador para alterar os estados das permissões ou dos componentes de *hardware* foram implementados encaminhamentos diretos para os menus quando os estados dos recursos não se encontram ativos

Qualquer que seja o recurso que se pretenda confirmar é seguido o mesmo esqueleto de atividade. Assim, tal como representado no diagrama da Figura 48, quando qualquer recurso

se encontra inativo é feito o encaminhamento do utilizador para a página referente ao recurso para este efetuar a sua ativação.

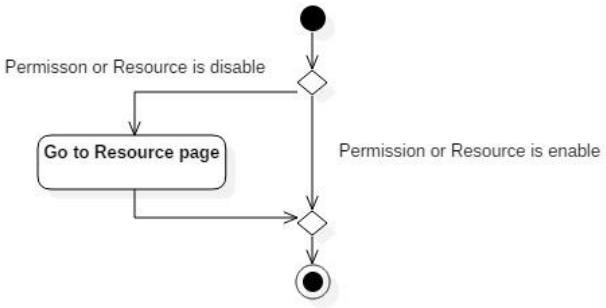


Figura 48. Diagrama de Atividade do Check Permissions / Check State

5.2.5.3. SHARED PREFERENCES

As Shared Preferences, ou preferências partilhadas, são um recurso do Android que permite efetuar o armazenamento de pequenas quantidades de informação. Quando se pretende inserir um valor neste mecanismo é necessário indicar um identificador da preferência, assim como o valor. Quando se pretende saber o valor de uma preferência já existente, apenas é necessário indicar o indentificador da mesma.

5.2.5.4. WAKE LOCK

O Wake Lock também é um recurso do Android. É utilizado para garantir que o equipamento não bloqueie o serviço antes deste terminar. Para tal este mecanismo bloqueia o funcionamento do processador para este não entrar no estado *Doze* contudo, se for utilizado para longos períodos de tempo perde o seu efeito. Com a utilização deste mecanismo pretende-se aumentar o tempo da janela de manutenção oferecida pelo Android.

Também são utilizados os mecanismos de Wifi Lock, com funcionamento idêntico, mas dedicado ao bloqueio do *wifi* no estado ativo.

5.2.5.5. LOCALIZAÇÃO

Para obter a geolocalização de cada *scanner* foi desenvolvida uma classe com a função de obter as coordenadas conforme os requisitos. Para tal é utilizada a API do Google Maps. Caso não seja possível efetuar a estimação da localização esta retorna a latitude e a longitude com valor igual a zero.

5.3. CORDOVA APP

A aplicação Cordova foi desenvolvida com o objetivo de conseguir provar o correto funcionamento do *plugin* anteriormente apresentado, assim como dos seus mecanismos, tendo assim, como principal funcionalidade o rastreamento de *beacons* e de wifi.

Esta aplicação além de originar os dados também pode ser um dos destinos destes, assim é possível ao utilizador ver em tempo real o que está à sua volta. Esta funcionalidade permite também servir de ferramenta de depuração de erros, permitindo saber se o *plugin* e se o broker RabbitMQ estão a funcionar corretamente.

Na Figura 49 é apresentado o posicionamento do Smartphone no sistema, assim como os elementos que comunicam diretamente com ele e quais, apresentando também o sentido das comunicações.

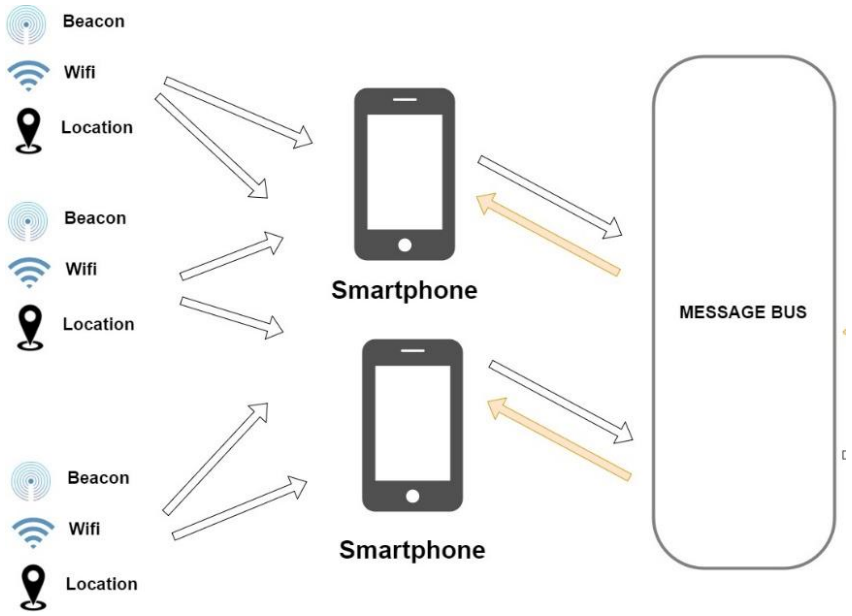


Figura 49. Diagrama representativo da posição do Smartphone no sistema.

Assim, a aplicação Cordova é constituída por uma camada composta por todos os *plugins* necessários para corresponder às necessidades da implementação, onde consta também o Beacon Scanner Service.

A camada seguinte é constituída pela implementação dos *plugins* e pelo consumidor RabbitMQ. O primeiro é composto por todos os recursos necessários para o seu correto

funcionamento criando, na camada da lógica de negócio, um nível de abstração em relação ao seu funcionamento. O consumidor RabbitMQ é construído a partir da sua API para *javascript*. Estes elementos devem conseguir corresponder aos requisitos da camada da lógica de negócio.

A camada da Lógica de Negócio é responsável por implementar as funcionalidades conforme os objetivos pretendidos. Esta camada engloba a comunicação com as camadas inferiores assim como gere a interface com o utilizador. A interface com o utilizador é constituída pelos ecrãs para o utilizador e pelos recursos necessários para o seu correto funcionamento, enquanto que as restantes camadas, á exceção dos componentes nativos dos *plugins*, são construídas em *Javascript*, esta camada é composta também por CSS e HTML. Na Figura 50 é apresentada a arquitetura básica da Aplicação Cordova.

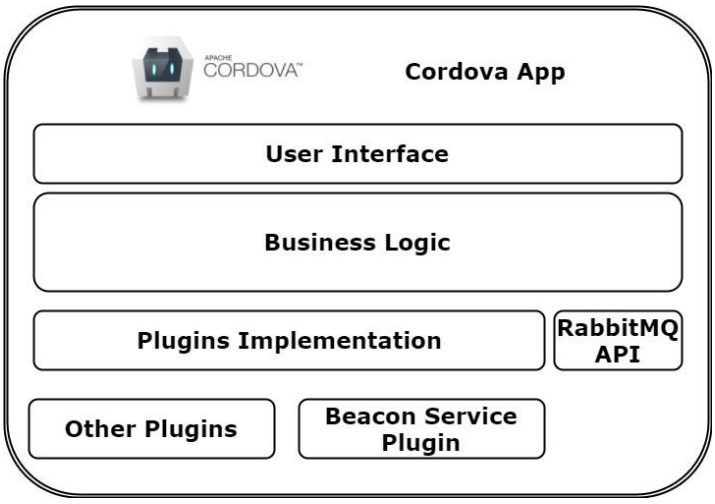


Figura 50. Arquitetura da Cordova App

5.3.1. INÍCIO DE EXECUÇÃO DA APLICAÇÃO

Após o início da aplicação é apresentado um menu ao utilizador, Figura 51, nesse menu são apresentadas duas opções. Numa é possível visualizar os resultados dos rastreamentos de *wifi*, no outro o resultado do rastreamento de *beacons.* Contudo, em segundo plano são executadas outras tarefas.

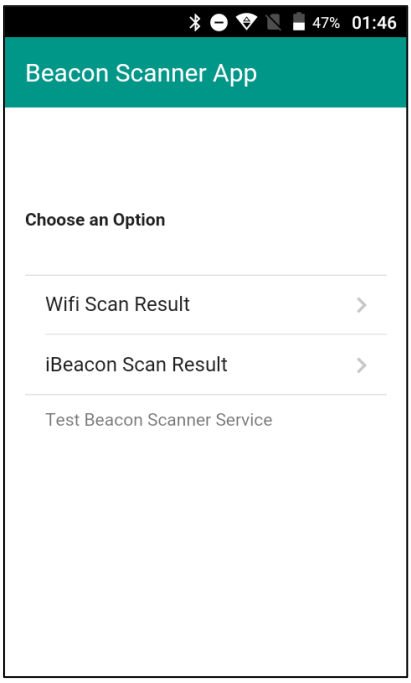


Figura 51. Menu Principal da aplicação Cordova

Logo que a aplicação é iniciada é verificado se esta tem ativas todas as permissões necessárias, caso não as tenha o utilizador é reencaminhado para as definições da aplicação até esta obter todas as permissões para o seu correto funcionamento, tal como já mencionado no ponto 5.2.5.2, de nome Check Permissions/Check State (Página 70).

Na Figura 52 é apresentada a sequência de ecrãs para ativação das permissões da aplicação. De denotar que na Figura 52 a) se apresenta o ecrã que aparece ao utilizador logo que este inicia a aplicação sem as permissões necessárias, sendo que depois de este premir o "Ok" do *popup* é reencaminhado para o ecrã representado ao lado em "b)". Nesse ecrã é que são ativas as permissões para a execução da aplicação.

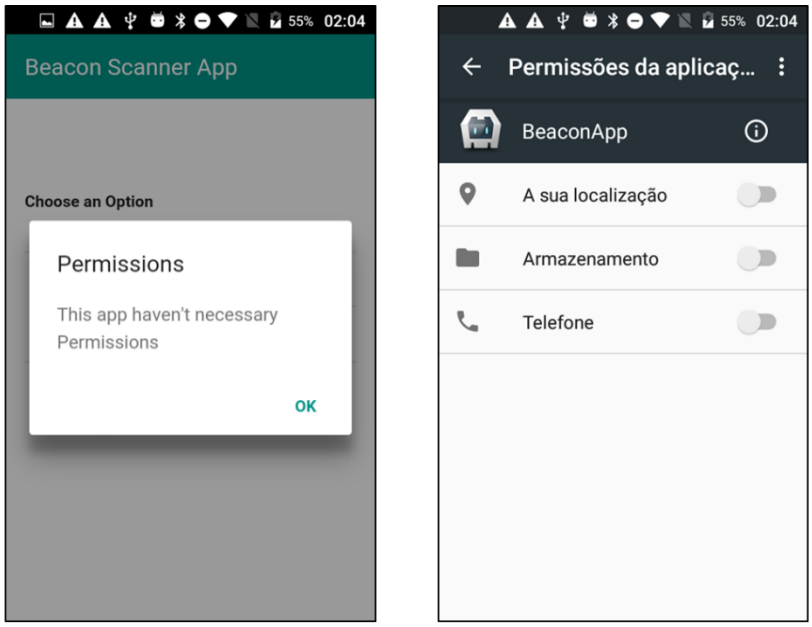


Figura 52. Sequência de ecrãs para ativação das permissões da aplicação a) Ecrã com o pop-up ao iniciar a aplicação b) Ecrã do Android com as permissões pedidas pela aplicação

Quando as permissões necessárias para a correta execução da aplicação estão ativas é então dado o início aos serviços de *scanner* da *plugin*, sendo utilizados os recursos de interligação com a parte nativa deste. Como estas tarefas são executadas sempre que a aplicação é iniciada, antes de ser dada a ordem de execução aos serviços é feita a paragem da sua execução. De seguida é apresentado o trecho de código referente à inicialização do serviço.

```
function initServices() {
    xhockService.beacon.stopWifiService();
    xhockService.beacon.stopBeaconService();
    xhockService.beacon.startWifiService();
    xhockService.beacon.startBeaconService();
    xhockService.beacon.setPeriodic(10000);
    xhockService.beacon.getServiceState();
}
```

Após esta ação os serviços de *scanner* ficam em execução de forma contínua, continuando a sua execução mesmo após a aplicação ser terminada. Numa primeira utilização os serviços não se encontram em funcionamento, sendo necessário iniciar a aplicação para iniciar o seu funcionamento.

5.3.2. APRESENTAÇÃO DOS RESULTADOS DOS SCANNERS

A apresentação dos resultados obtidos nos *scanners* é feita separadamente, ou seja, os resultados do rastreamento *wifi* são apresentados num ecrã, e o resultado do rastreamento de *beacons* é apresentado num outro. Contudo, estes têm um aspeto visual e funcionamento muito idênticos, sendo neles aprestada uma lista com as informações dos dipositivos que estão a ser reportados naquele momento, sendo atualizados conforme os dados sejam recebidos. Na Figura 53 são apresentados os dois ecrãs.

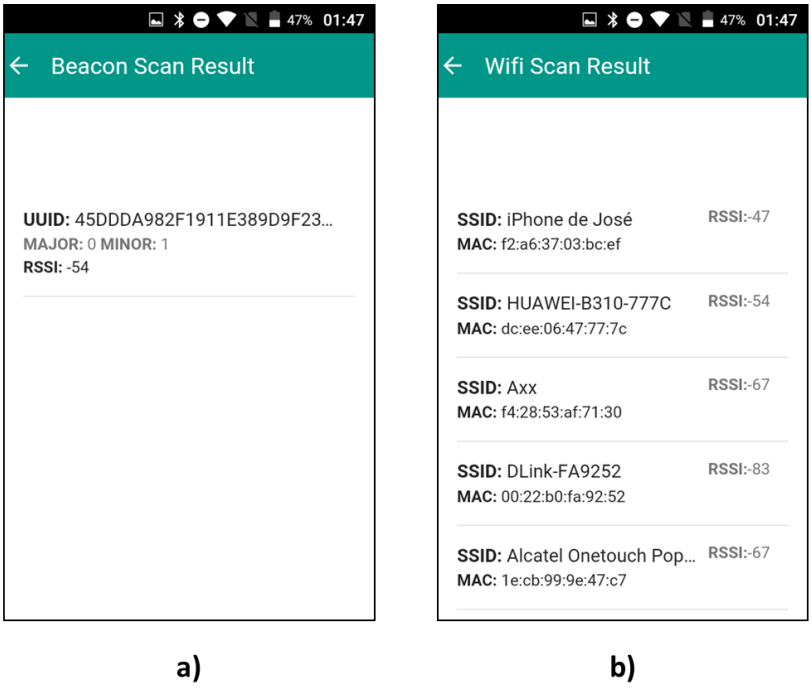


Figura 53. Ecrã de apresentação dos resultados dos *scanners* a) Apresentação dos resultados referentes ao rastreamento de *beacons* b) Apresentação dos resultados referentes ao rastreamento de *wifis*

Contudo os resultados apresentados no ecrã não são obtidos diretamente a partir do *plugin*, mas sim a partir do *broker* RabbiMQ. Ou seja, estes ecrãs mostram através de um *consumer a* informação enviada pelo *plugin* para a rede.

Quando ligado ao servidor este subscreve uma *queue* para onde são direcionadas as mensagens a si destinadas. Cada dispositivo tem a sua *queue*. Este fica à escuta de mensagens, não sendo uma tarefa bloqueante. Quando o dispositivo recebe uma mensagem envia a mesma para tratamento, colocando-se novamente à escuta.

Neste caso, o *consumer* RabbitMQ apenas está em funcionamento quando os ecrãs de apresentação de resultados estão abertos.

Quando estes ecrãs são iniciados é verificado se a localização está ativa, e se o *bluetooth* ou o *wifi* se encontram ativos, conforme os casos, sendo executado o processo mencionado no ponto 5.2.5.2.. Isto é, caso um deles não esteja ativo, aparece um *popup* e o utilizador é reencaminhado para um ecrã onde é possível executar a modificação do estado desse recurso. Na Figura 54 é apresentada a sequência de ecrãs efetuada quando a localização se encontra no estado desativado, para os restantes recursos mencionados a sequência é idêntica.

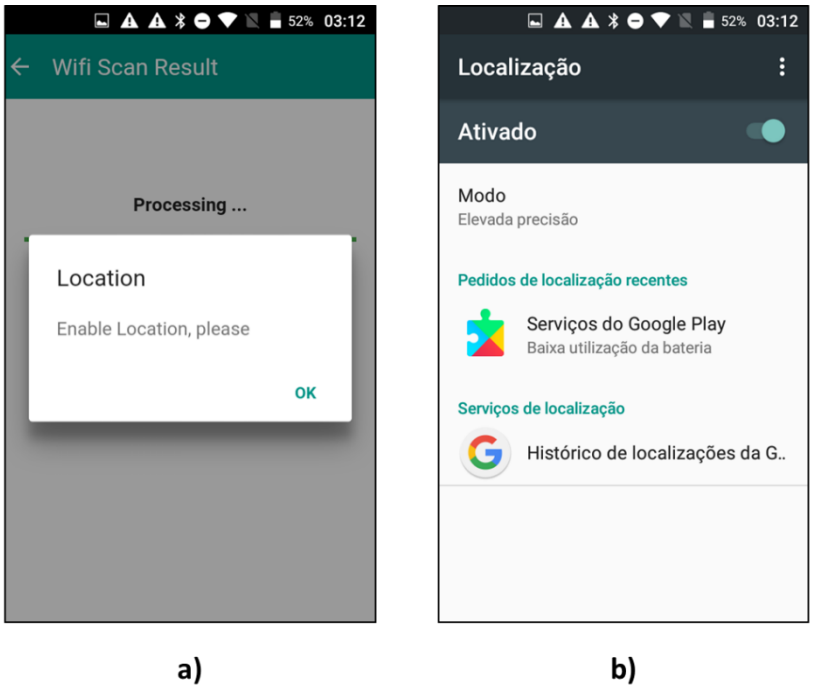


Figura 54. Sequência de ecrãs que ocorre quando a Localização se encontra no estado inativo a) Ecrã da aplicação com o pop-up indicando o recurso necessário b) Ecrã do Android das definições dos dispositivos para a localização

Como a origem dos dados não é local, mas da rede, e antes dos dados serem apresentados é apresentado um ecrã intermédio com uma barra de carregamento e com duas mensagens diferentes. Uma delas é “Connecting to server” indicando que o dispositivo ainda não está ligado ao broker RabbitMQ e à sua *queue* contudo, já está a efetuar esse processo. A outra mensagem é “Waiting Results”, indicando que o dispositivo já se encontra ligado ao *broker* e que já efetuou a subscrição da sua *queue*, contudo ainda não recebeu qualquer resultado.

A implementação deste ecrã, mostrando estes dois estados, serve para mostrar ao utilizador o estado da ligação permitindo que este consiga de forma simples depurar qualquer anomalia, e assim também foi possível eliminar um compasso em branco que existia entre o tempo de a página abrir até se obter um resultado para se mostrar ao utilizador. Na Figura 55 esses ecrãs são apresentados.

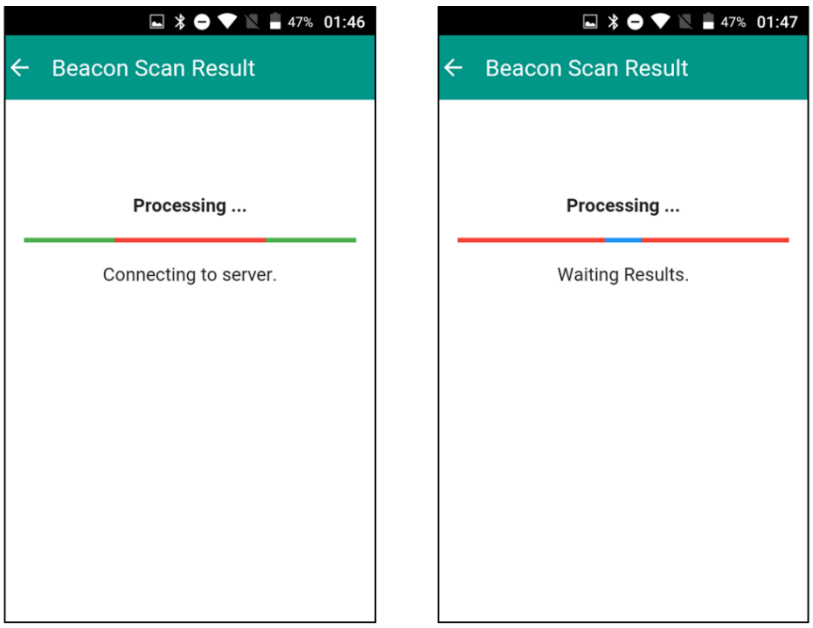


Figura 55. Sequência de ecrãs da aplicação até obter resultados a) Efetuando ligação ao servidor b) À espera de resultados

Com esta aplicação é demonstrado o correto funcionamento do *plugin* desenvolvido e dos seus mecanismos auxiliares, inclusive o *broker* RabbitMQ. Ainda são oferecidas ao utilizador algumas ferramentas para efetuar a depuração de anomalias no sistema. A partir desta aplicação fica demonstrado o cumprir dos objetivos propostos a este nível.

5.4. ENCAMINHAMENTO DE MENSAGENS DO RABBITMQ

Ao longo da apresentação dos vários módulos que constituem o sistema já se compreendeu a importância do broker RabbitMQ no encaminhamento de mensagens e na gestão das ligações. Neste subcapítulo pretende-se apresentar a estruturação do encaminhamento das mensagens.

Neste caso a troca de mensagens efetua-se nos dois sentidos, sentido smartphone-microserviço e no sentido inverso. Estas mensagens ao chegarem ao *broker* RabbitMQ são recebidas nas *exchanges*, que são do tipo *topic*, e são encaminhadas para as suas *queues* conforme as suas *routing keys*.

Quando o smartphone envia os resultados dos seus *scanners*, as mensagens são publicadas na *exchange* e de seguida são encaminhadas conforme o tipo de *scanner*. Caso a mensagem se trate do resultado de um scanner de *beacons* esta é encaminhada para a *queue* "*bluetooth*". Caso a mensagem se trate do resultado de um scanner de *hotspots Wifi*, esta é encaminhada para *queue* "*wifi*". O tipo de mensagem encontra-se descrito na *routing key* da mesma. Também existe um *queue* para onde são encaminhadas todas as mensagens que chegam a essa *exchange*. A mesma *queue* é normalmente subscrita pelo serviço que efetua o arquivamento dos dados na base de dados. Na Figura 56 é apresentada a estrutura do encaminhamento de mensagens neste sentido.

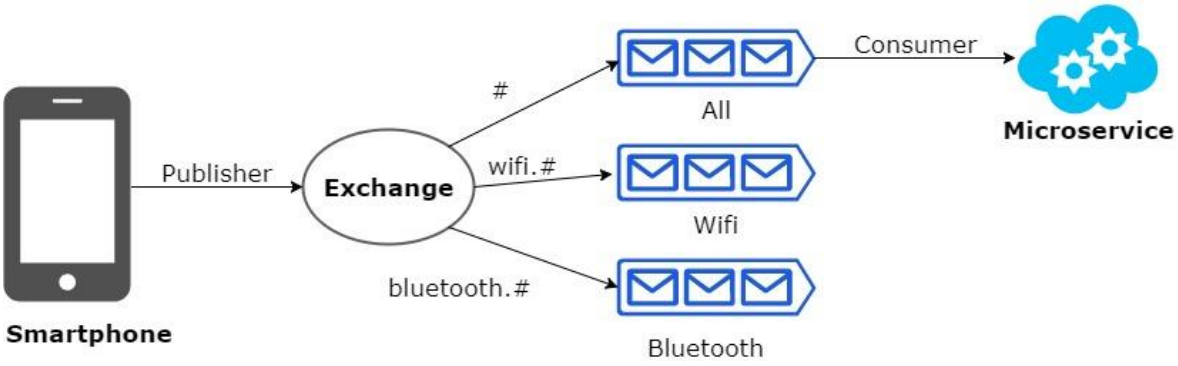


Figura 56. Estrutura do encaminhamento de mensagens no sentido *smartphone*-microserviço

No sentido inverso, ou seja, quando o microserviço publica mensagens com destino aos dispositivos, estas são divididas conforme o identificador do dispositivo. Isto é, quando o microserviço publica a mensagem indica na sua *routing key* o indentificador do dispositivo de destino da mensagem. Quando o dipositivo subscreve a *queue*, cujo nome é o seu identificador, consegue receber as mensagens dirigidas a si. Existindo um serviço que cria uma *queue* por cada dispositivo que utiliza a aplicação, sendo utilizado o os mecanismos já utilizados pela aplicação Youbeep para efetuar este algoritmo. Isto permite uma comunicação de forma diferenciada para um dispositivo ou para um grupo de dispositivos.

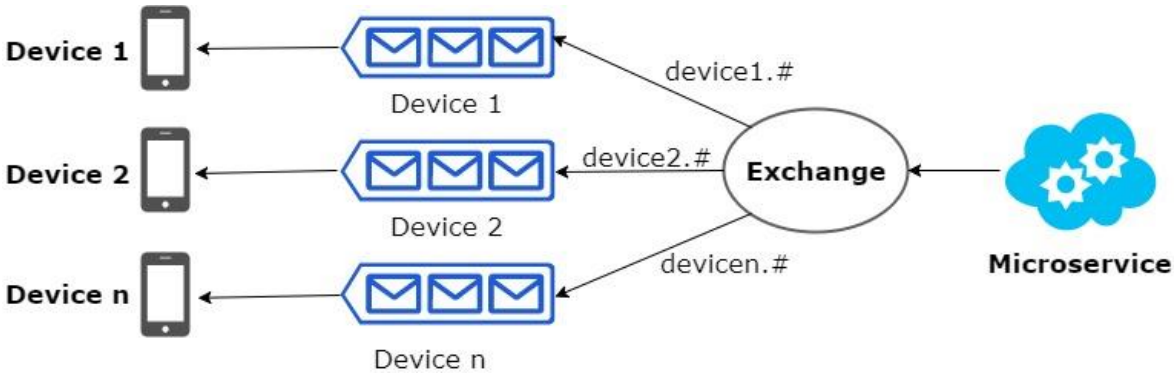


Figura 57. Estrutura do encaminhamento de mensagens no sentido microserviço-*smartphone*

5.5. WEBSERVICES

O módulo de *WebServices* é composto pelos vários serviços na rede. Estes serviços são responsáveis pelo tratamento dos dados resultantes dos *scanners*, o seu encaminhamento e armazenamento. Estes estão divididos em pequenos serviços, os microserviços, responsáveis apenas por uma tarefa facilitando a estabilidade do sistema conforme as necessidades demonstradas ao longo do tempo.

O armazenamento dos dados é feito numa base de dados contudo, para maior segurança, o sistema não comunica diretamente com a mesma, mas sim de uma API desenvolvida de forma a criar uma camada de proteção sobre a mesma.

Para efetuar o encaminhamento das informações recolhidas para a base de dados existe um serviço desenvolvido em node.js que comunica diretamente com *Message Bus* e com API da base de dados, implementando assim a ponte entre os dois pontos. Na Figura 58 é apresentado o enquadramento deste serviço no sistema.

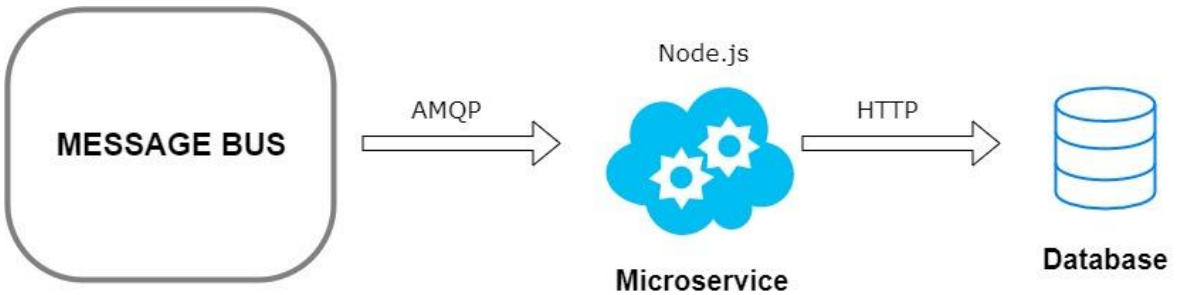


Figura 58. Enquadramento do Micro Serviço no sistema e das tecnologias utilizadas

Este serviço funciona de forma permanente e constante. A receção dos resultados dos *scanners* é feita através da subscrição da *queue* da *exchange* que recebe os dados originados

nos dispositivos. Aquando da receção de um resultado este é processado e encaminhado para a base de dados através do método de POST do protocolo HTTP (*Hypertext Transfer Protocol*).

Os dados encaminhados para a base de dados são armazenados em duas tabelas. Numa das tabelas, de nome *Beacon Signals*, são armazenados todos os resultados dos rastreamentos, na outra tabela, de nome *Beacon Spots*, são guardados os dados relativos a cada *beacon* e a cada *access point wifi* encontrado, ou seja, na tabela de *Beacon Signals* são armazenados todos os resultados. Na tabela *Beacon Spots* pretende-se guardar os dados de cada dispositivo diferente encontrado de forma a permitir efetuar o mapeamento de todos os spots encontrados e dos seus dados. Esta tabela é atualizada de forma iterativa com recurso a um serviço cujo seu funcionamento será abordado de seguida. As tabelas recebem dados de ambos os tipos de dispositivos, *Wifi* e *beacons*, sendo estes diferenciados através do campo *type*, os restantes campos são preenchidos conforme o tipo de dispositivo. Na Figura 59 é apresentada a estrutura das tabelas apresentadas anteriormente.

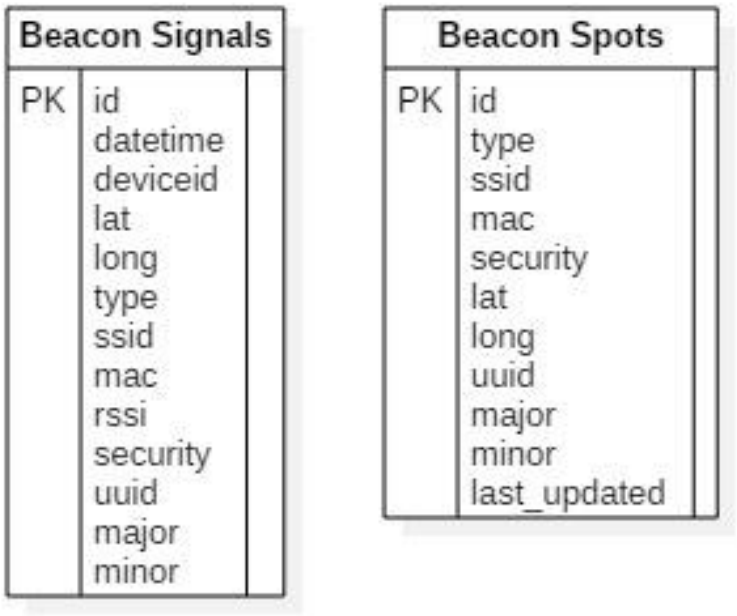


Figura 59. Estrutura das tabelas da base de dados

Para efetuar a inserção dos dados na base de dados é utilizado um serviço que dentro da API, e juntamente com os serviços da mesma, permite ainda processar as informações dos mesmos.

Aquando da receção de resultados, o serviço insere estes na tabela BeaconSignal, depois, caso o resultado seja do tipo Beacon, este procura na tabela BeaconSpots se existe algum registo de um *beacon* com os mesmos identificadores (UUID, Major, Minor). Caso não exista correspondência os dados sobre este dipositivo são adicionados à tabela. Se existir correspondência e o valor da potência de sinal do último resultado for superior ao correspondente existente na tabela então o valor da latitude, longitude e da potência de sinal é substituído no registo já existente. Caso o resultado seja do tipo *wifi*, este procura na tabela BeaconSpots se existe algum registo de um *Wifi* com o mesmo endereço MAC, aplicando-se de seguida o mesmo algoritmo apresentado para o caso anterior. Na Figura 60 é apresentado o diagrama de atividade deste serviço.

Para obter os dados dos *spots* armazenados na tabela BeaconSpots existe um serviço para atender aos pedidos direcionados a este efetuados através do método GET do protocolo HTTP.

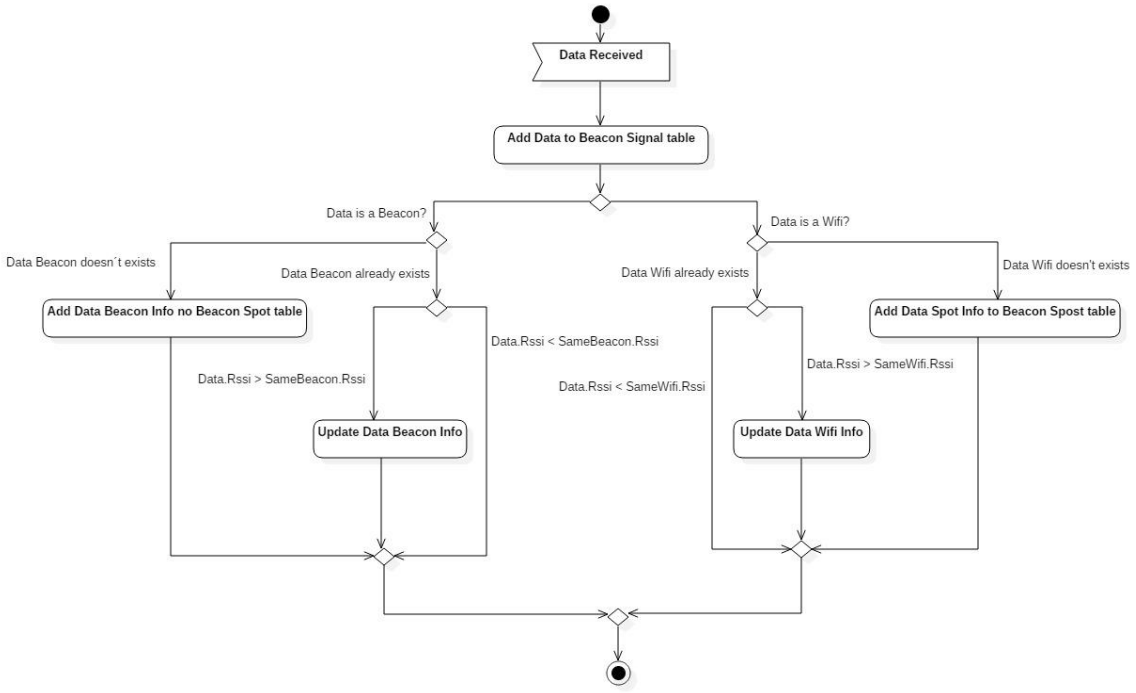


Figura 60. Diagrama de Atividade do Serviço auxiliar da base de dados

A implementação deste algoritmo permite mapear *spots* e de forma iterativa afinar a sua localização no espaço de um modo descentralizado e sem qualquer intervenção de um utilizador. Com as informações recolhidas a partir destes scanners é possível aplicar a lógica de negócio com algoritmos de análise de grande volume de dados e executar várias ações direcionadas.

5.6. REPRESENTAÇÃO GRÁFICA DOS SPOTS DETETADOS

Como forma de apresentar as potencialidades do mapeamento indireto dos *spots*, desenvolveu-se uma página *web* com a representação gráfica de todos os pontos detetados num mapa. Este mapa foi desenvolvido em HTML e Javascript com recurso à API do Google Maps.

Para obter os dados relativos aos *spots*, é efetuado um pedido HTTP GET aos Webservices, sendo retornado todos os registos da tabela Beacon Spots, já referida no ponto anterior, num vetor de objetos com formato apresentado de seguida.


```
[…,{"id":82,"type":1,"ssid":"","mac":"10:fe:ed:52:84:f6","rssi":-
86,"security":null,"latitude":41.1613,"longitude":-
8.60443,"uuid":null,"major":null,"minor":null,"last_updated":"2017-08-
11T00:00:00.000Z","last_deviceid":"f8879cdee5b8f64f"},{"id":80,"type":1,"ssid":"BYMAR
QUES","mac":"60:e3:27:8d:9d:a2","rssi":-
79,"security":null,"latitude":41.1613,"longitude":-
8.60443,"uuid":null,"major":null,"minor":null,"last_updated":"2017-08-
11T00:00:00.000Z","last_deviceid":"f8879cdee5b8f64f"}, …]
```


A partir desses dados são criados os marcadores com recurso à API do Google Maps. Nesses marcadores são adicionados todos os dados referentes ao mesmo, sendo estes diferenciados conforme o seu tipo, *beacon* ou *Wifi*. A cada marcador é adicionada uma janela com as suas informações que apenas é mostrada quando é feito um clique no marcador. Sendo que, caso seja um *beacon* é mostrado o tipo de dispositivo, o seu UUID, o Major, o Minor, o RSSI e a última atualização desses valores. Caso seja um dispositivo do tipo *wifi* é mostrado o nome da rede, o endereço MAC e o valor do RSSI, além do tipo de dispositivo. Na Figura 61 são apresentados exemplos da representação desses pontos.

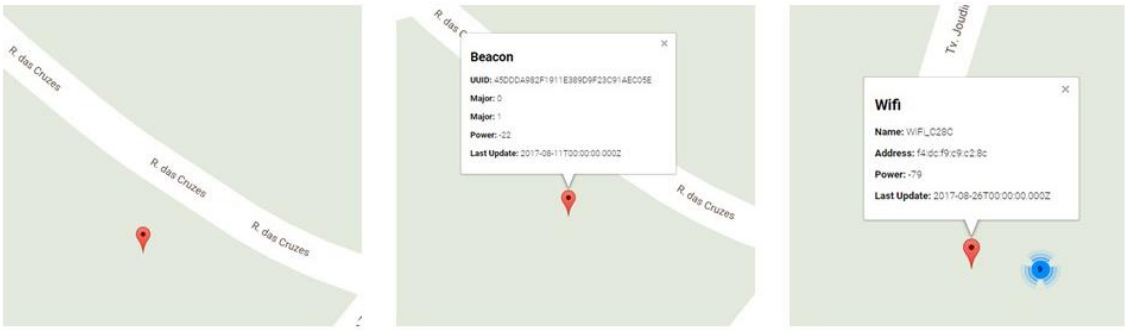


Figura 61. Representação gráfica de um spot no Mapa a) Representação normal b) Representação de um marcador *beacon* mostrando a sua informação c) Representação de um marcador *wifi* mostrando a sua informação

De seguida apresenta-se como é implementada a criação dos marcadores, com os dados obtidos através da resposta do pedido feito aos Webservices.

```
  // Criação dos Markers
  for (var i = 0; i < beaconSpot.length; i++) {
       if (beaconSpot[i].type == 1) {
            contentString = wifiContent(beaconSpot[i]); // #SE WIFI
       } else if (beaconSpot[i].type == 2) {
            contentString = beaconContent(beaconSpot[i]); // #SE BEACON
       } else {
            continue; //next loop
       }

       var latLng = new google.maps.LatLng(beaconSpot[i].latitude,
beaconSpot[i].longitude); //Coordenadas do ponto
        // Criação do MARKER
        var marker = new google.maps.Marker({
            position: latLng,
            map: map,
            contentString: contentString
         });
(…)
```

Logo após criação dos marcadores é adicionado um evento de clique em cada marcador para que aquando de um clique no mesmo, este mostre a sua informação.

```
(…)
// Cada Marker tem um evento de click
  marker.addListener('click', function() {
     if (infowindow) {
        infowindow.close();
     }
     infowindow = new google.maps.InfoWindow({});
     infowindow.setContent(this.contentString);
     infowindow.open(map, this);
     map.setCenter(this.getPosition());
});
(…)
```

Como a densidade de dispositivos num mesmo espaço pode ser elevada, levando a uma difícil visualização dos pontos, os marcadores foram agrupados, formando *clusters*, identificados com um ícone diferente dos marcadores, assumindo cores diferentes conforme o número de dispositivos que representam, tendo ainda o número de dispositivos que representa. Na Figura 62 são apresentados dois exemplos da representação gráfica dos marcadores em *clusters*.

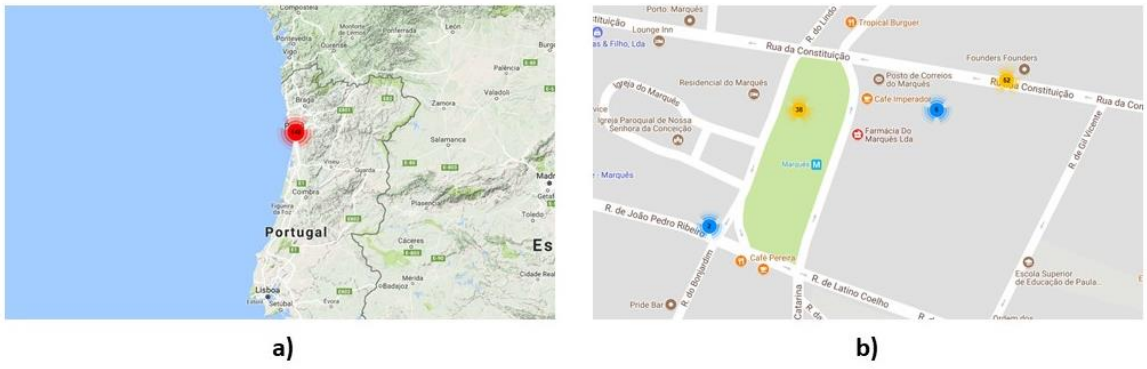


Figura 62. Representação dos marcadores *clusters* a) Representação com zoom baixo b) Representação com zoom elevado

Assim, para criar os grupos, os marcadores são todos inseridos num vetor, tal como apresentado de seguida.

```
markers.push(marker);
```

Esse vetor dá origem ao agrupamento dos marcadores em *clusters*, ficando a cargo da API a gestão do tamanho dos grupos, estando implementado da forma demonstrada abaixo.

```
var markerCluster = new MarkerClusterer(map, markers, { imagePath:'/markerclus/m' });
```

Para se conhecer os dispositivos que aquele *cluster* representa é apresentada, com o clique neste, uma janela idêntica à apresentada nos marcadores, mas com a informação de todos os marcadores daquele *cluster*, tal como representado na Figura 63. Esta informação apenas é mostrada quando o zoom do mapa está próximo do zoom máximo, se não com o clicar nos *clusters* apenas é feito zoom no mapa.

Figura 63. Representação gráfica de um *cluster* mostrando a informação dos seus elementos

Esta funcionalidade está implementada da seguinte forma.

```
google.maps.event.addListener(markerCluster, 'clusterclick', function(cluster) {
   if (map.getZoom() > map.maxzoom - 1) {
      var markers = cluster.getMarkers();
      var contentString = '';

      // Criação da janela de informações conforme o numero de MARKERS
      for (var i = 0; i < markers.length; i++) {
         var marker = markers[i];
         contentString += marker.contentString;
         contentString += ("<p> </p>");
      }
      // SE existir algum aberta, FECHA
      if (infowindow) {
          infowindow.close();
      }
      infowindow = new google.maps.InfoWindow();
      infowindow.setPosition(cluster.getCenter());
      infowindow.setContent(contentString);
      infowindow.open(map);
   }
});
```

Com a representação gráfica de todos os dipositivos encontrados é possível visualmente perceber que áreas que se encontram mapeadas e até fazer uma correlação entre os dispositivos encontrados e os pontos de interesse em redor. Permitindo demonstrar mais facilmente as potencialidades deste projeto.

5.7. CONSIDERAÇÕES FINAIS

Ao longo este capitulo foi apresentado como este projeto foi desenhado e implementado. Ao longo deste é apresentada a estrutura geral do projeto, seguindo-se do *plugin* e de todos os seus constituintes, passando de seguida pela aplicação Cordova, pelo encaminhamento de mensagens, pelos WebServices e pela representação gráfica dos *spots.*

É de salientar não ter sido apresentadas funcionalidades que exemplificam as possibilidades do projeto, assim como o resultado final, e não só as funções estritamente necessárias para o seu funcionamento.

No próximo capitulo pretende-se comprovar o funcionamento do Beacon Service *Plugin*, apresentado neste capitulo, e comparar a sua performance com outras abordagens.

6. TESTES E RESULTADOS

Após se conhecer como todos os elementos do sistema foram desenhados e implementados, neste capítulo pretende-se comprovar a abordagem escolhida para o desenvolvimento do *plugin* de *scanner* e comparar esta com outras abordagens. Relembrando que o grande objetivo do *plugin* desenvolvido é efetuar o rastreamento de *spots wifi* e de *beacons* de forma contínua, mesmo com a aplicação desligada, tendo esses resultados que ser encaminhados para um Message Bus, no caso, um *broker* RabbitMQ.

Deste modo foram criados 3 diferentes cenários de teste:

1. Scanner efetuado em 1º Plano
2. Scanner efetuado num Serviço
3. Solução Implementada

No próximo subcapítulo será apresentada a estrutura dos elementos que tomam parte nestes testes sendo depois analisados os resultados dos testes realizados.

6.1. SISTEMA DE TESTES

Este Sistema de Testes tem como objetivo conseguir perceber o comportamento da aplicação do *smartphone*, nos diferentes cenários através da quantificação do número de resultados chegados à rede. Em todos estes cenários é utilizada a mesma arquitetura de sistema, sendo que entre si apenas é alterada a aplicação do *smartphone*, o alvo dos testes.

A estrutura do Sistema de Teste foi desenhada a partir da estrutura implementada no sistema. Tal como representado na Figura 64, este é constituído pelos dipositivos detetáveis, no caso *wifi* e geolocalização, pelo smartphone e a sua aplicação, pelo Message Bus e pelo microserviço, sendo que o *output* dos testes é um ficheiro de texto.

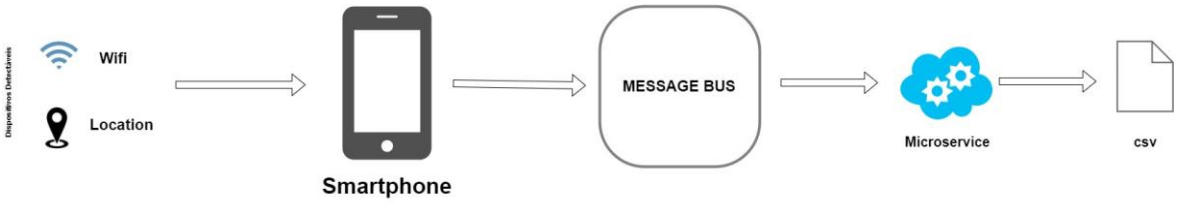


Figura 64. Estrutura do Sistema de Testes

Neste sistema o Smartphone é responsável por efetuar o rastreamento dos dipositivos *wifi* ao seu redor, e enviar esses resultados para o Message Bus, juntamente com a localização. No microserviço existe um consumidor RabbitMQ que faz a quantificação das mensagens recebidas no intervalo de tempo definido.

De seguida é apresentado com maior pormenor cada um dos elementos constituintes do sistema.

6.1.1. APLICAÇÃO DO SMARTPHONE

Em virtude destes foram desenvolvidas duas aplicações, que além da solução implementada fazem parte deste banco de testes. Essas duas soluções desenvolvidas são aplicações nativas Android, construídas a partir da solução implementada respeitando a abordagem pretendida em cada uma delas.

Assim, nas três aplicações o *scanner* de *wifi* é feito da mesma forma, ou seja, seguindo a sequência já exposta no ponto 5.2.3 (página 64).

Nos próximos pontos serão apresentadas as duas aplicações de teses desenvolvidas e as suas especificidades, sendo que a solução implementada já se encontra descrita no capítulo anterior.

6.1.1.1. SCANNER EFETUADO EM 1º PLANO

Nesta aplicação o *scanner* de *spots wifi* é efetuado em 1º plano e não num serviço em 2º plano, como na solução implementada. Neste caso o serviço de *scanner* foi convertido numa função, sendo esta função executada em intervalos de 1 segundo.

Assim, tal como representado no diagrama da Figura 65, ao arrancar a aplicação iniciado um ciclo em que de segundo a segundo é feito um novo *scanner*.

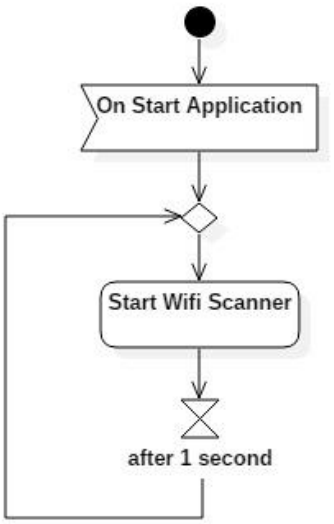


Figura 65. Diagrama de Atividade da Aplicação de Testes para scanner em 1º plano

6.1.1.2. SCANNER EFETUADO NUM SERVIÇO

Neste caso o scanner é feito num serviço em 2º plano que está em execução contínua efetuando o scanner dos dispositivos a cada segundo. O serviço é iniciado no arranque na aplicação ficando em execução ininterruptamente.

O serviço de *scanner* do *wifi* do *plugin* desenvolvido, foi aplicado integralmente nesta aplicação, com a diferença de que ao invés de o serviço ser arrancado sucessivamente pelo JobService este é iniciado apenas uma vez, sendo que os sucessivos *scanners* são iniciados com recurso a um timer. Isto também faz com que o serviço não possa terminar a sua execução quando termina a sua tarefa, tal como faz o serviço implementado pelo *plugin*.

No diagrama Figura 66 encontra-se representado o processo desta aplicação. De salientar que após a execução da sua tarefa o processo principal não termina, mas fica no estado de *idle*.

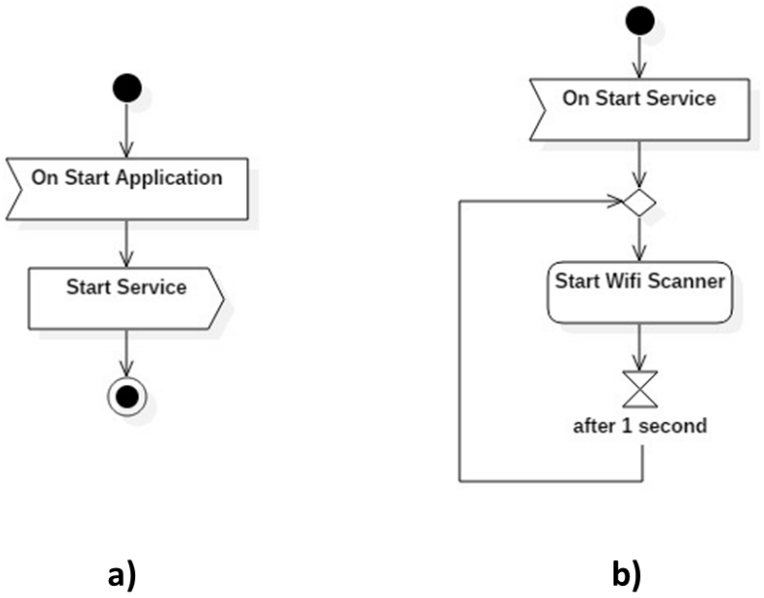


Figura 66. Diagrama de atividade da Aplicação de Testes para scanner num Serviço a) Processo Principal b) Serviço

6.1.2. WEBSERVICE

Neste serviço, desenvolvido em node.js, é feita a quantificação dos resultados obtido pelo smartphone. Esta quantificação podia ser feita localmente no *smartphone*, contudo assim é possível perceber o desempenho das diferentes abordagens inseridas no sistema completo, ou seja, numa implementação real.

Este Microserviço implementa um consumidor RabbitMQ, obtendo assim as mensagens enviadas pelo smartphone. Por cada mensagem recebida, existe um contador que é incrementado, sendo que a cada 10 segundos o valor desse contador é guardado num ficheiro de texto e o seu valor é reiniciado, tal como está representado no diagrama da Figura 67. De salientar que este serviço apenas considera mensagens obtidas do smartphone de testes, sendo estas mensagens filtrada a partir do campo de "deviceID" da mensagem, as restantes são ignoradas.

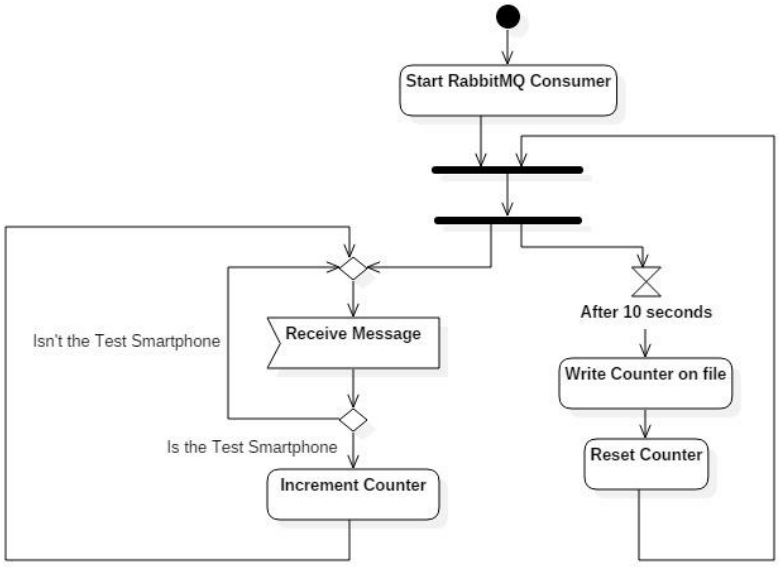


Figura 67. Diagrama de Atividade do Microserviço de Testes

6.2. ANÁLISE DE RESULTADOS

Para a recolha dos resultados, nos três cenários, foi utilizado o mesmo smartphone, um Alcatel Pixi, com a versão 6 do Android e a API23, colocado num local onde existia apenas um *spot wifi*. Nesses testes a aplicação arrancava, permanecendo em primeiro plano durante 3 minutos, sendo que no final desse tempo a aplicação era terminada e o smartphone era deixado em repouso. Desde o início do procedimento o serviço encontra-se a recolher e a enviar dados. O teste era terminado quando o dispositivo já não estivesse a enviar dados durante um longo período de tempo, comparando com a duração do teste.

De seguida vão ser analisados os resultados obtidos nos 3 cenários, terminando com uma comparação entre estes.

6.2.1. SCANNER EFETUADO EM 1º PLANO

Neste cenário a aplicação encontra-se em execução em primeiro plano, numa aplicação nativa, estando no gráfico da Figura 68 os resultados obtidos durante o teste.

A partir do gráfico é primordial salientar que não é reportada qualquer informação a partir dos três minutos, ou seja quando é terminada a execução da aplicação (marcado com uma reta vertical amarela).

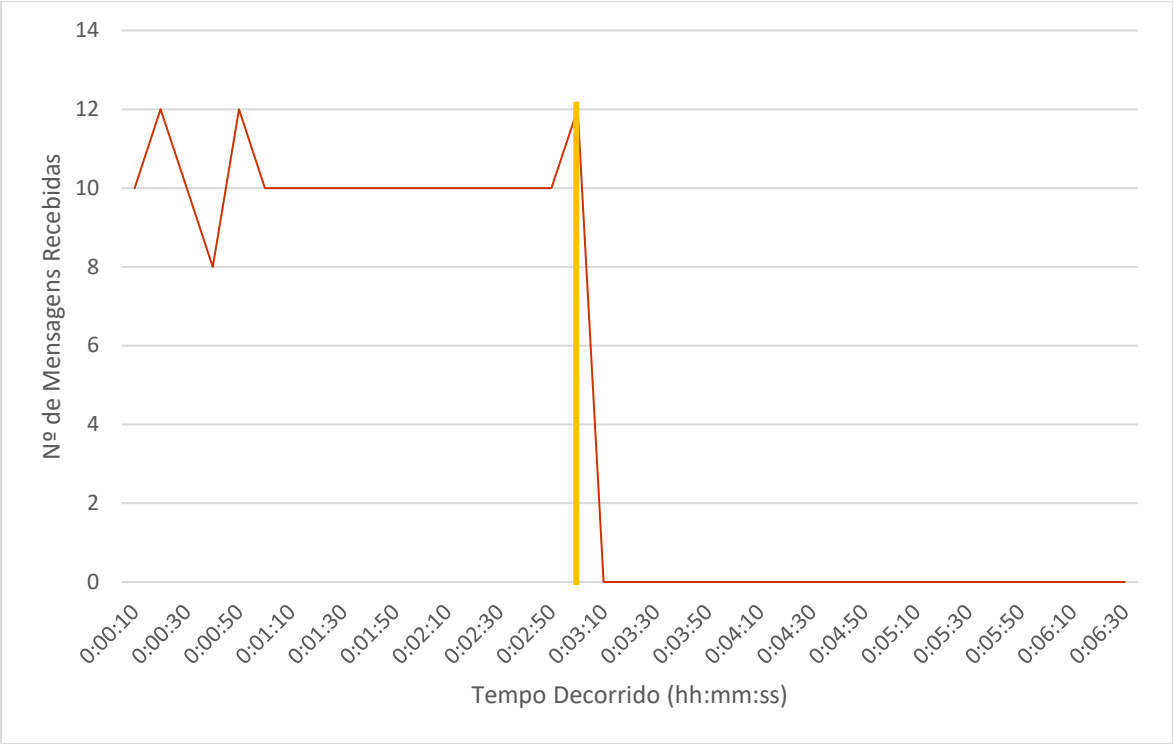


Figura 68. Representação gráfica dos resultados do teste no cenário de scanner efetuado em 1º plano

A partir dos resultados conclui-se que com esta abordagem apenas é possível efetuar o scanner de dispositivos enquanto a aplicação se encontra em execução. Logo que esta é terminada o rastreamento termina.

6.2.2. SCANNER EFETUADO NUM SERVIÇO

Neste cenário o scanner de dispositivos *wifi* é efetuado num serviço, em 2º plano, numa aplicação nativa, estando representados na Figura 69 os resultados obtidos durante o teste.

Analisando os resultados deste teste é importante salientar que neste caso o serviço continuou em execução após se ter terminado a execução da aplicação principal, aos 3 minutos (marcado a amarelo). Passado 15 minutos o serviço acaba por terminar a sua execução, pois aos 18 minutos e 10 segundos deixaram de ser reportados resultados. Denota-se que entre o término da aplicação e o término do serviço, existem 3 minutos em que não são reportados dados. De salientar que durante os testes o serviço nunca foi capaz de efetuar 10 scanners em 10 segundos, mostrando que a capacidade de execução se encontra limitada quando as tarefas são executadas num serviço em segundo plano

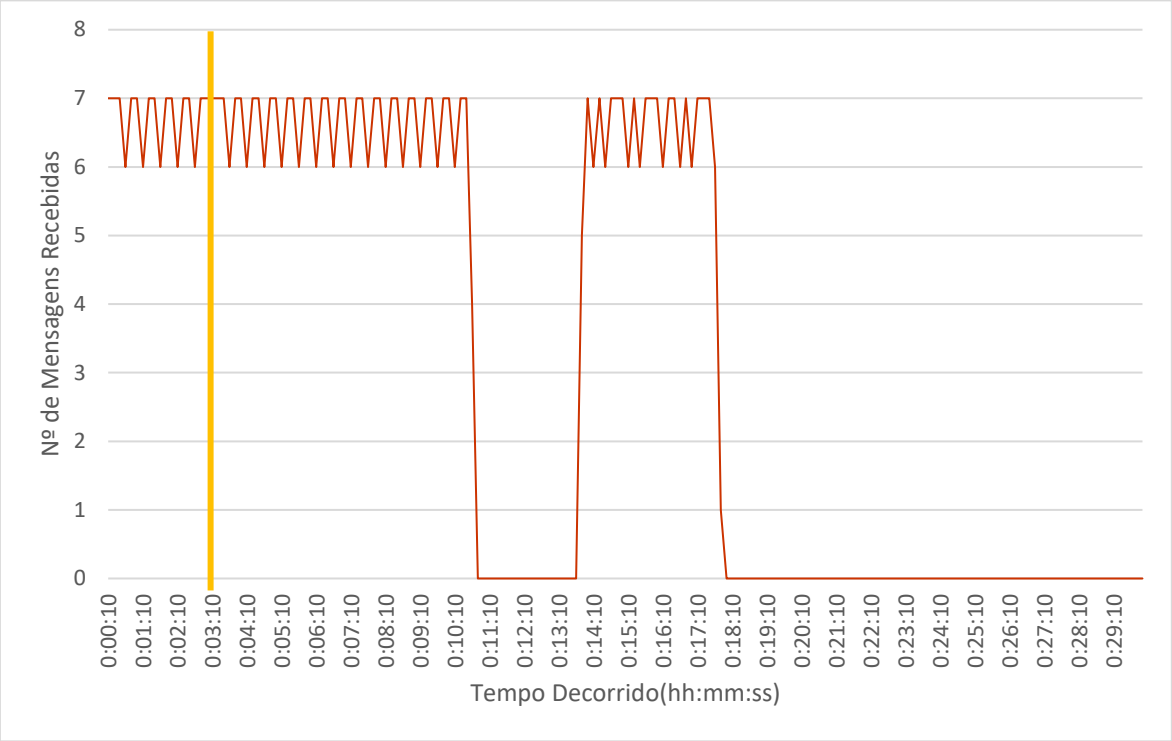


Figura 69. Representação gráfica dos resultados do teste no cenário de scanner efetuado num serviço

A partir destes resultados conclui-se que com a utilização de serviços em segundo plano é possível continuar a efetuar tarefas, mas em curtos espaços de tempo, pois o sistema operativo termina a execução em segundo plano quando pretende baixar o consumo de energia. Esta opção pode ser viável, quando se pretende executar pequenas tarefas depois da aplicação terminar.

É evidente que o tempo entre o término da aplicação e o término do serviço é variável, estando dependente de várias variáveis ligadas ao dispositivo e à gestão que o sistema operativo faz dos recursos que tem disponíveis.

6.2.3. SOLUÇÃO IMPLEMENTADA

Neste cenário foi utilizada a aplicação desenvolvida, mas com um intervalo entre scanners de 1 segundo. Relembrando, esta aplicação é desenvolvida em Cordova e tem integrada um *plugin* que utiliza serviços em 2º plano para efetuar o rastreamento, sendo estes iniciados por um JobScheduler.

No gráfico da Figura 70 estão representados os resultados obtidos durante este teste. A partir destes vê-se que durante uma hora são reportados resultados quase continuamente, embora existam algumas quebras. De salientar que o número de resultados obtidos é pouco contínuo, sofrendo constantemente flutuações. Outro ponto importante é o facto de, tal como acontece no cenário anterior, a latência de resultados a cada 10 segundos é sempre inferior a 10, evidenciando também que neste cenário a capacidade de execução em 2º plano se encontra limitada.

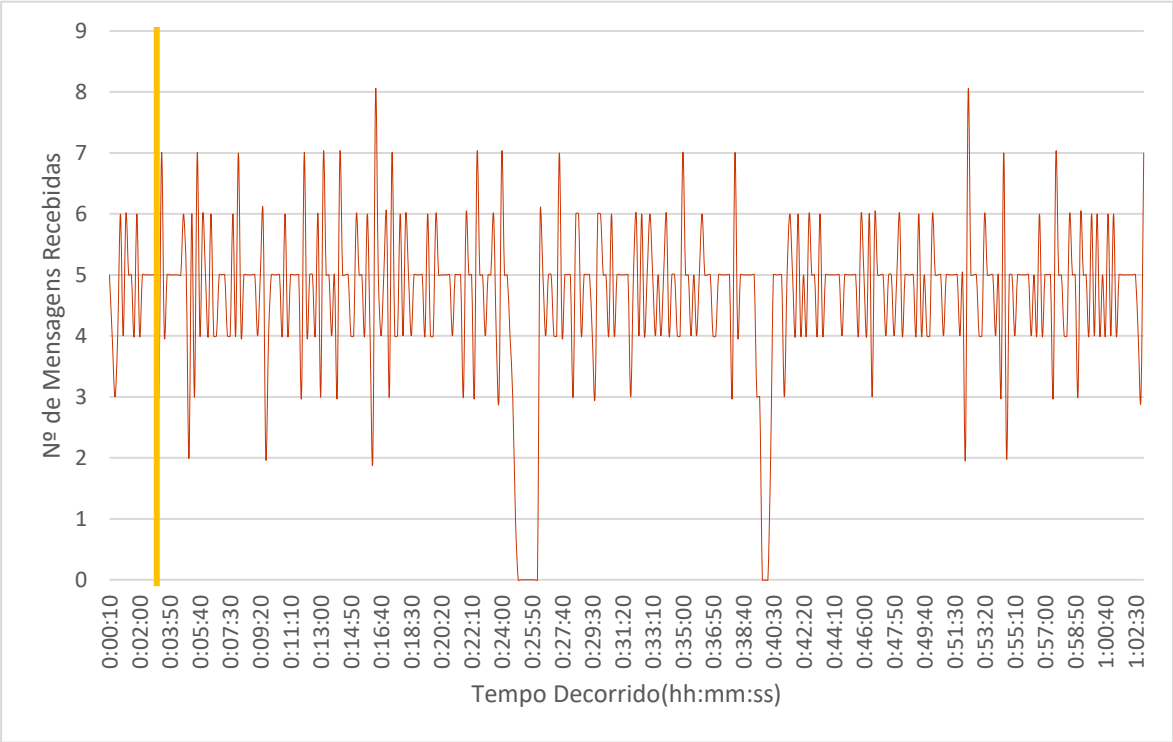


Figura 70. Representação gráfica dos resultados do teste no cenário da solução implementada durante 1 hora e 3 minutos

A partir destes resultados fica demonstrado que esta solução é capaz de efetuar o rastreamento contínuo de dipositivos e fica ainda mais evidente quando se olha para o gráfico da Figura 71. Neste gráfico estão representados os resultados obtidos durante um teste em que o dispositivo se encontra em repouso a recolher dados durante 10 horas após a aplicação principal ser terminada. Atenta-se para a grande variabilidade dos valores dos resultados obtidos ao longo do tempo, revelando as limitações de execução impostas pelo sistema operativo.

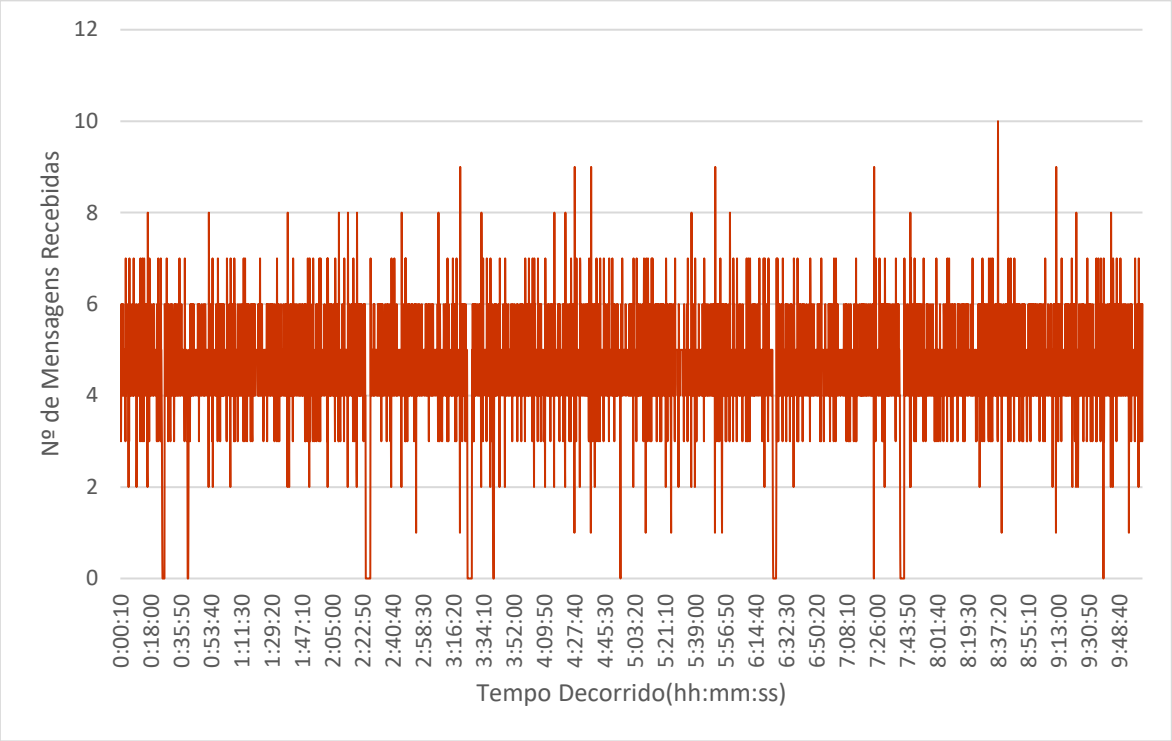


Figura 71. Representação gráfica dos resultados do teste no cenário da solução implementada durante 10 horas e 3 minutos

6.2.4. COMPARAÇÃO DE RESULTADOS

Com a análise feita nos pontos anteriores, das três diferentes abordagens, revela-se que a solução implementada é a única capaz de executar tarefas de forma contínua mesmo depois da aplicação terminar a sua execução. Com esses testes também se consegue perceber, na realidade, o comportamento do sistema operativo Android.

Assim, este sistema, quando a aplicação termina a sua execução também são terminadas todas as tarefas em primeiro plano. Os serviços em segundo plano continuam a sua execução mesmo após a aplicação terminar contudo, ao final de algum tempo estes são terminados para diminuição do consumo. Para contornar estas limitações um dos mecanismos possíveis é a utilização de um Jobscheduler, tal como foi implementado na solução final.

Embora a solução implementada garanta uma execução contínua, não garante o melhor desempenho na execução das tarefas, tal como se pode observar no gráfico da Figura 72. Neste gráfico consegue-se perceber que na execução da tarefa em 1º plano se tem o melhor

desempenho dos três cenários analisados, seguindo-se do cenário da execução da tarefa num serviço.

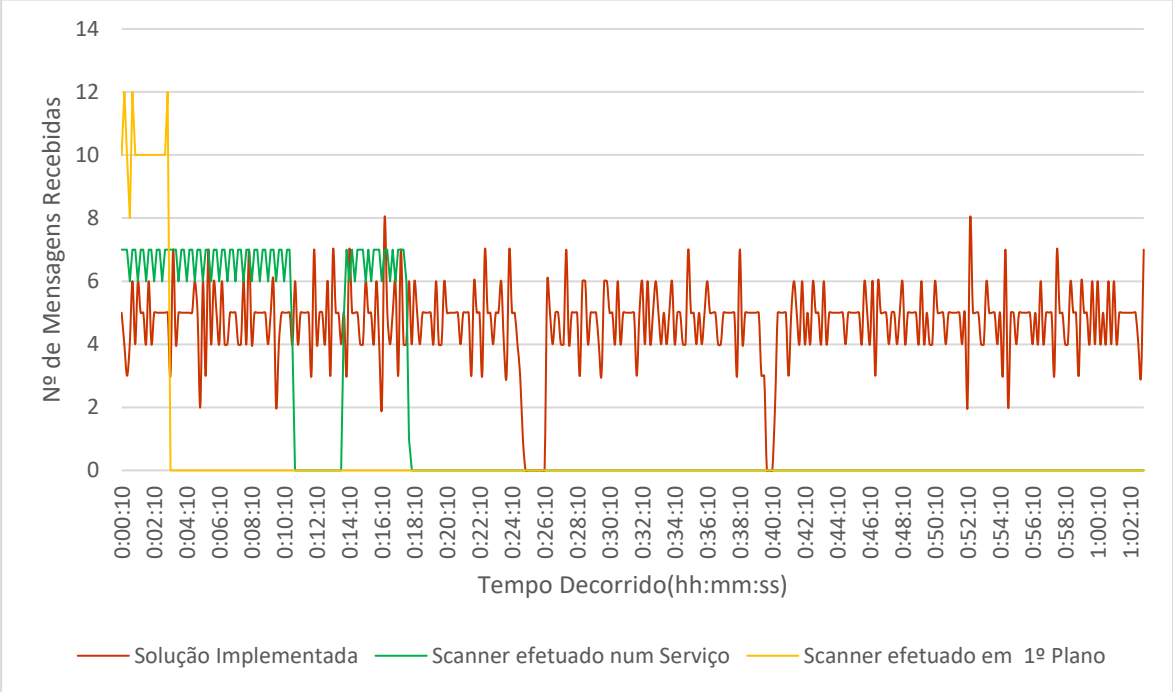


Figura 72. Representação gráfica dos resultados obtidos nos três testes

Com este teste, conseguiu-se demonstrar que a solução implementada consegue cumprir o principal objetivo a que se propunha, revelando-se a melhor opção para a execução contínua de uma tarefa, numa comparação entre as três soluções testadas.

A partir dos dados recolhidos pelas ferramentas do sistema operativo, o consumo de bateria da solução implementada significa 5% de um ciclo completo, como demonstrado na Figura 73a, sendo que num período de um mês o consumo de dados desta solução totaliza os 164 MB, resultando da soma dos valores apresentado na Figura 73b e Figura 73c. A partir de testes realizados com recurso à aplicação GlassWire, na visualização de um vídeo de 10 minutos com uma qualidade de 720p no YouTube existe um consumo de 134 MB, um valor muito próximo do valor mensal de consumo da aplicação. Fazendo com que esta solução não tenha um perfil de consumo que influencie a utilização normal do dispositivo

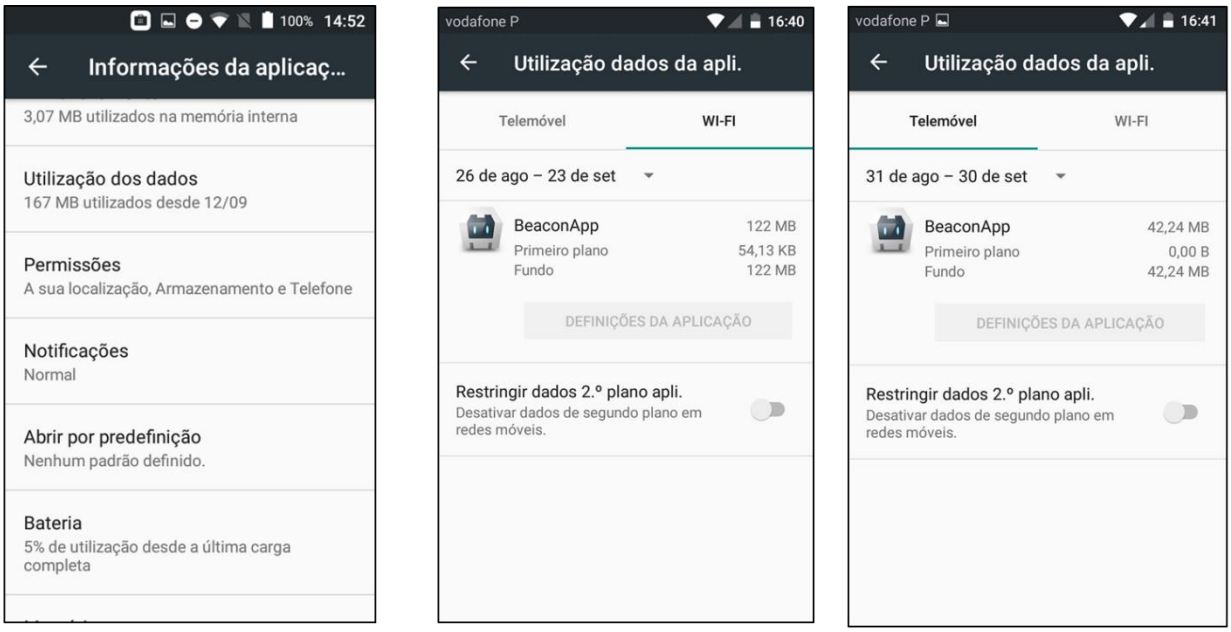


Figura 73. Ecrãs com informações relativas à utilização de recursos do *smartphone* na solução implementada a) Utilização de bateria b) Utilização de dados por Wifi c) Utilização de dados pela rede móvel

Ao longo dos resultados apresentados, neste capítulo, fica demonstrado que a solução implementada é capaz de cumprir os objetivos pretendidos sem influenciar a utilização normal do dispositivo.

7. CONCLUSÕES E TRABALHO FUTURO

Ao longo desta dissertação foi sendo apresentada a solução desenvolvida, demonstrando a pesquisa e desenvolvimento até à sua implementação, conseguindo-se demonstrar o seu funcionamento e comprovar o cumprimento dos objetivos propostos.

Conseguiu-se perceber o funcionamento dos dispositivos *beacon* e através da interceção das tramas de *advertise* do protocolo BLE foi possível identificar este tipo de dipositivos. Foi ainda possível explorar a tecnologia *Wifi* e a geolocalização (GPS) e utilizá-las no projeto.

Desenvolveu-se uma solução utilizando a *um Plugin* Cordova, para a plataforma Android, que permite executar a deteção destes dispositivos em segundo plano. Conseguiu-se assim ultrapassar a limitação do Cordova na execução destas tarefas levando por isso ao desenvolvimento de uma solução, que tomando partido das funcionalidades da componente nativa, consegue integrar esta solução neste tipo de aplicações móveis. Contudo foi necessário contornar as limitações impostas pelo sistema operativo Android, sendo este um dos maiores desafios de todo o projeto.

Implementou-se uma estrutura de comunicação entre os dispositivos móveis e os serviços na rede baseada no conceito de *middleware*, utilizando o RabbitMQ. A utilização deste tipo de conceito permite garantir a escalabilidade de todo o sistema sem afetar o seu desempenho.

Foi possível desenvolver micro-serviços, alojados na rede, que permitem demonstrar algumas funcionalidades do sistema, como o armazenamento de dados ou o mapeamento de pontos.

Este projeto demonstra um grande potencial ao permitir automatizar o processo de recolha de dados sobre o ambiente envolvente sem necessidade de ação do utilizador, a não ser iniciar a aplicação numa primeira vez. Contudo como este funciona em dispositivos em que não se tem influência direta é difícil garantir a sua total operacionalidade.

É possível integrar esta solução numa aplicação como a YouBeep. Os dados originados por esta implementação cruzados com outro tipo de dados como, por exemplo, os hábitos de consumo, permite desenvolver soluções de marketing direcionado e oferecer às marcas e aos retalhistas uma outra forma de comunicar com os seus clientes assim como conhecer outro género de métricas até agora desconhecidas.

De futuro, seria importante desenvolver a mesma solução para iPhones, conseguindo assim ser compatível com grande parte dos dipositivos móveis. Quanto ao plugin seria interessante alterar o algoritmo de recolha de dados de forma a minorar o consumo energético, assim como, diminuir a quantidade de mensagens enviadas filtrando resultados que possam não ter interesse. Desenvolver mecanismos de *feedback* e de alerta sobre o comportamento do *plugin* e do dispositivo. Em relação ao algoritmo de estimação de posição dos *spots* seria importante realizar um estudo nessa vertente e implementar um algoritmo mais competente.

A um nível mais pessoal, este projeto tornou-se interessante e desafiante pois não havia conhecimento de nenhuma das tecnologias estudadas, fazendo com que desenvolvesse algumas competências. Contudo este processo mostrou-se moroso, exatamente pelo processo de aprendizagem que foi necessário para o desenvolvimento da solução.

Referências Documentais

[1] SigiLabs, “Developing Beacons with Bluetooth® Low Energy (BLE) Technology,” 2016.

[2] Estimote, “Estimote Developer Docs.” [Online]. Available: http://developer.estimote.com/. [Accessed: 07-Sep-2017].

[3] Cisco, “iBeacon - Frequently Asked Questions,” pp. 1–13, 2014.

[4] S. Venkiteswaran, “iBeacon Reality Check - Essential Considerations for an iBeacon Deployment,” 2015.

[5] Beaconstac, “A Beginner’s Guide to Eddystone Beacon Pilots 101,” 2017.

[6] “National Slate Museum - Plan Your Visit - Interactive experience.” [Online]. Available: https://museum.wales/slate/visit/ibeacon/. [Accessed: 08-Sep-2017].

[7] *Amgueddfa Cymru - National Museum Wales pilots a “world first” iBeacon programme*. Walles, 2014.

[8] “Estimote Products.” [Online]. Available: https://estimote.com/products/. [Accessed: 09-Sep-2017].

[9] “Kontakt.io Store.” [Online]. Available: https://store.kontakt.io/. [Accessed: 09-Sep-2017].

[10] “Radius Networks Store.” [Online]. Available: https://store.radiusnetworks.com/collections/all. [Accessed: 09-Sep-2017].

[11] LitePoint, “Bluetooth ® Low Energy - Whitepaper,” 2012.

[12] K. Townsend, “Introduction to Bluetooth Low Energy,” 2014.

[13] Joakim Lindh, “Bluetooth ® Low Energy Beacons,” 2015.

[14] Apple Inc., “Getting Started with iBeacon,” 2014.

[15] “Eddystone format | Beacons | Google Developers,” 2017. [Online]. Available: https://developers.google.com/beacons/eddystone. [Accessed: 15-Jun-2017].

[16] M. Ashbridge, “Eddystone Protocol Specification,” 2015.

[17] D. Helms, “AltBeacon Protocol Specification v1.0,” 2014.

[18] S. Staler, *Beacon Technologies: The Hitchhiker’s Guide to the Beacosystem*. San Diego: Apress, 2016.

[19] Lighthouse, “Indoor Location Technologies Compared: GPS, WiFi, iBeacon; NFC/RFID.” 2017.

[20] F. Hakimpour and A. Z. Zardiny, “Location Based Service in Indoor Environment Using Quick Response Code Technology,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XL-2/W3, no. November, pp. 137–140, 2014.

[21] S. Amatya, “Degree project Cross-Platform Mobile Development : An Alternative to Native Mobile Development,” Linnaes University - Sweden, 2013.

[22] W. Felix, “6 aspectos essenciais para decidir entre aplicações mobile híbridas e nativas,” 2015. [Online]. Available: https://medium.com/@waldyrfelix/6-aspectos-essenciais-para-decidir-entre-aplicações-mobile-híbridas-e-nativas-51bce0dace68. [Accessed: 13-Jul-2017].

[23] Open Soft, “APPS NATIVAS E MOBILE WEB APPS : QUAIS AS DIFERENÇAS ?”

[24] “Cross-Platform Mobile Development in Visual Studio.” [Online]. Available: https://msdn.microsoft.com/library/dn771552.aspx. [Accessed: 18-Jun-2017].

[25] “Architectural overview of Cordova platform - Apache Cordova.” [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/index.html. [Accessed: 18-Jun-2017].

[26] “WebView | Android Developers.” [Online]. Available: https://developer.android.com/reference/android/webkit/WebView.html. [Accessed:

18-Jun-2017].

[27] “Adobe PhoneGap Products.” [Online]. Available: https://phonegap.com/products/. [Accessed: 04-Sep-2017].

[28] I. Documentation, “Chapter 1: All About Ionic.” .

[29] A. Bradley, “Where does the Ionic Framework fit in?,” *Ionic Blog*, 2013. .

[30] “Ionic Pro Products.” [Online]. Available: http://ionicframework.com/products/. [Accessed: 03-Sep-2017].

[31] “Xamarim - Developer Documentation - Application Fundamentals.”

[32] “Serviços | Android Developers.” [Online]. Available: https://developer.android.com/guide/components/services.html. [Accessed: 03-Jul-2017].

[33] “Limites da execução em segundo plano | Android Developers.” [Online]. Available: https://developer.android.com/preview/features/background.html. [Accessed: 08-Jul-2017].

[34] “Otimização para soneca e aplicativo em espera | Android Developers.” [Online]. Available: https://developer.android.com/training/monitoring-device-state/doze-standby.html. [Accessed: 03-Jul-2017].

[35] C. Teixeira, “Middleware for Large-scale Distributed Systems,” Instituto Superior de Engenharia do Porto, 2015.

[36] “Advanced Message Queuing Protocol Specification v0-9-1,” 2008.

[37] M. Albano, L. L. Ferreira, L. M. Pinho, and A. R. Alkhawaja, “Computer Standards & Interfaces Message-oriented middleware for smart grids,” *Comput. Stand. Interfaces*, vol. 38, pp. 133–143, 2015.

[38] Johansson Lavisa, *Getting Started with RabbitMQ*. 84 CaodesAB, 2017.

[39] A. Videla and J. J. W. Williams, *RabbitMQ in action*. 2012.