Universidade de Aveiro
2012
Departamento de Electrónica, Telecomunicações e Informática

Emanuel José Vieira Santana

Mapeamento de Energia Electromagnética utilizando dispositivos móveis

Universidade de Aveiro
2012
Departamento de Electrónica, Telecomunicações e Informática

Emanuel José Vieira Santana

Mapeamento de Energia Electromagnética utilizando dispositivos móveis

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações (Mestrado Integrado), realizada sob a orientação científica do Dr. Nuno Borges de Carvalho, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho aos meus pais por todo o esforço e apoio.

o júri

presidente

Professor Doutor Rui Manuel Escadas Ramos Martins
Professor auxiliar da Universidade de Aveiro

vogais

Doutora Ana Collado Garrido
Investigadora do Centre Tecnològic de Telecomunicacions de Catalunya

Professor Doutor Nuno Miguel Gonçalves Borges de Carvalho
Professor associado com agregação da Universidade de Aveiro

agradecimentos

Começo por agradecer a todos aqueles que me acompanharam durante os últimos anos na cidade de Aveiro e durante todo o percurso académico que agora chega ao fim. Sem o apoio e amizade de todos eles este caminho teria sido muito mais difícil.
À minha namorada, Mariana, pelo apoio e paciência durante todo o percurso deste trabalho que múltiplas vezes fez com que não lhe pudesse dar a atenção merecida.
Agradeço a toda a minha família pelo apoio dado em especial aos meus pais que sempre me apoiaram mesmo quando nem tudo correu bem, pela compreensão e dedicação, e que fizeram de mim aquilo que sou hoje, a eles o meu muito obrigado.
Ao meu orientador, professor Nuno Borges de Carvalho, agradeço pela disponibilidade sempre que necessário, pelas opiniões e ideia dadas e por permitir que realiza-se este trabalho.
Agradeço também ao Ludimar Guenda pela paciência e disponibilidade sempre que surgiu a necessidade de resolver qualquer problema.
A todos o meu muito obrigado.

palavras-chave *Energy Harvesting*, Computação móvel, Android, Localização

resumo A colheita de energia, ou *energy harvesting*, é uma tecnologia emergente nos dias de hoje. A possibilidade de criar sistemas que conseguem colher energia do meio ambiente para se auto carregarem abre um vasto leque de oportunidades para a colocação de sensores em qualquer sítio e ambiente. Por outro lado, o crescimento de sistemas de comunicação sem fios tem sido enorme nos últimos anos, sistemas esses, que, para comunicar, utilizam energia electromagnética que é enviada para o ar. Assim, um dos ramos do *energy harvesting* tem como objectivo aproveitar essa energia, que não estando a ser utilizada na comunicação pode ser canalizada para outros fins. No meio deste processo, umas das dúvidas que se levanta é a quantidade de energia que nos rodeia e em que frequência se encontra. Assim, o objectivo do trabalho desenvolvido e apresentado neste documento foi o de criar uma base de dados pública que contenha essa informação. O objectivo passa por apresentar a quantidade de energia presente em cada ponto da Terra. Para que seja possível recolher essa informação, foi desenvolvida uma aplicação para dispositivos móveis que permite que qualquer pessoa possa contribuir com informação para o sistema e que mede a energia presente nas frequências de comunicação móvel e de redes *Wi-Fi*. Para que a informação esteja visível a qualquer um foi ainda criado um sítio na Internet onde é possível ter acesso detalhado a toda a informação recolhida.

keywords Energy Harvesting, Mobile Computing, Android, Location

abstract Energy harvesting is an emerging technology nowadays. The possibility of create systems that can harvest energy of environment to self-charge makes it a wide open of possibilities to placing many sensors anywhere. On the other hand, the rising of systems of communication wireless has been huge in the last few years, systems that, to communicate, use electromagnetic energy that is sent to the air. So, one of the sides of energy harvesting aims rent this energy, that when are not in use in communication could be channeled to other purposes. In the middle of this process, one of the doubts is the quantity of energy which involve us and in which frequency is it. So, the purpose of this developed work and presented in this document is to create a public database that contains that information. The aim is presenting the quantity of energy present in which point of the earth. To make possible receive that information, has been developed an application to mobile devices which allows that anyone contributes with information for the system and measure the energy present in the frequencies of mobile communication and Wi-Fi. So that, to make that information visible to everyone was also created website where is possible have the detailed access to all information collected.

Conteúdo

Lista de Figuras 3
Acrónimos 5
1 Introdução 7
1.1 Motivação e Objectivos 7
1.2 Estrutura da Dissertação 8
2 Energy Harvesting 11
2.1 RF Energy Harvesting 11
2.1.1 Estrutura Geral de um Circuito 12
2.1.2 Exemplo de Aplicação 14
3 Computação Móvel 17
3.1 Breve História da Computação Móvel 17
3.2 Sistema Operativo Móvel 19
3.3 Android 20
3.3.1 Breve história 21
3.3.2 Arquitectura 21
3.3.3 Aplicação 22
4 Tecnologias 27
4.1 *Software* 27
4.1.1 Java 27
4.1.2 HTML 30
4.1.3 CSS 31
4.1.4 PHP 33
4.1.5 JavaScript 35
4.1.6 XML 36
4.1.7 MySQL 37
4.2 Comunicação 38
4.2.1 UDP 38
4.2.2 HTTP 39
4.3 Sistemas de Localização 42
4.3.1 GPS 42
4.3.2 Localização pela rede 46

5 Implementação 49
5.1 Aplicação Móvel 49
5.1.1 Separador Info 50
5.1.2 Separador Spectrum 55
5.1.3 Separador Map 56
5.2 Sítio na Internet 57
5.3 Base de Dados 62
5.4 Comunicação entre plataformas 63
5.4.1 Envio de dados para o servidor 63
5.4.2 Acesso à Base de Dados pela Aplicação Móvel 72
6 Conclusão e Trabalho Futuro 77
6.1 Conclusão 77
6.2 Trabalho Futuro 78
Referências 79

Lista de Figuras

Fig. 1.1 - Mapa de leituras de energia electromagnética 8

Fig. 2.1 - Sistema de *RF energy harvesting*[1] 12
Fig. 2.2 - *Rectenna* espiral [2] 13
Fig. 2.3 - Circuito para *RF energy harvesting* sem malha de adaptação[3] 14
Fig. 2.4 - Gráfico de potência de saída para banda larga de frequências em função da resistência de carga e para uma potência de entrada de -42dBm [3] 15
Fig. 2.5 - Circuito para *RF energy harvesting* utilizando malha de adaptação[3] 15
Fig. 2.6 - Potência recolhida estimada para o sistema de banda estreita [3] 16

Fig. 3.1 - Kenbak-1 [6] 18
Fig. 3.2 - Osborne I [7] 18
Fig. 3.3 - Apple Macintosh 1984 [8] 18
Fig. 3.4 - *Simon*, o primeiro *smartphone* [9] 19
Fig. 3.5 - Gráfico da evolução na venda dos sistemas operativos móveis [10] 20
Fig. 3.6 - Logotipo Android [11] 20
Fig. 3.7 - Arquitectura do sistema operativo móvel Android [13] 21
Fig. 3.8 – Excerto de código de um ficheiro *manifest* com declaração de uma *activity* 24
Fig. 3.9 – Excerto de código de um ficheiro *manifest* com declaração de um *intent-filter* numa *activity* 25

Fig. 4.1 - Excerto de código *Java* 27
Fig. 4.2 - Estrutura de um documento HTML 30
Fig. 4.3 - Estrutura de uma regra CSS 32
Fig. 4.4 - Exemplo de uma *pseudo-class* em CSS 33
Fig. 4.5 - Exemplo de *script* PHP embutido num documento HTML 33
Fig. 4.6 - Evolução da utilização do PHP [21] 34
Fig. 4.7 - Exemplo de *script* de *JavaScript* num documento HTML 35
Fig. 4.8 - Pequeno documento em XML 36
Fig. 4.9 - Camadas TCP/IP [25] 38
Fig. 4.10 - *Header* de um pacote UDP [26] 39
Fig. 4.11 - Esquema de funcionamento do protocolo HTTP [28] 40
Fig. 4.12 - Exemplo de troca de mensagens HTTP entre cliente e servidor [28] 41
Fig. 4.13 - Representação dos segmentos do sistema GPS [30] 43
Fig. 4.14 - Segmento espacial do sistema GPS [31] 44
Fig. 4.15 - Sistema de controlo do GPS [32] 45
Fig. 4.16 - Exemplo de receptor GPS comercial [33] 45

Fig. 5.1 - Separadores presentes na aplicação móvel 49
Fig. 5.2 - Separador Info 50
Fig. 5.3 - Menu do separador Info 52
Fig. 5.4 - Janela de configuração do *refresh* periódico 52

Fig. 5.5 - Código de criação de nova *Thread* para *Refresh* periódico 53
Fig. 5.6 - Janela de informação 54
Fig. 5.7 - Janela de configuração das opções do servidor 55
Fig. 5.8 - Separador Spectrum 55
Fig. 5.9 - Excerto de código para utilização de Canvas 56
Fig. 5.10 - Separador Map e detalhe de informação de uma leitura 57
Fig. 5.11 - Mapa presente no sítio na *Internet* 58
Fig. 5.12 - Balão de informações de uma leitura presente no mapa 59
Fig. 5.13 - Excerto de código do *script* que coloca o mapa no sítio da *Internet* 60
Fig. 5.14 - Detalhes de uma leitura 61
Fig. 5.15 - Estrutura da base de dados 62
Fig. 5.16 - Envio de dados do dispositivo móvel para o servidor 63
Fig. 5.17 - Excerto do código de envio da trama UDP pela aplicação móvel 64
Fig. 5.18 - Estrutura da trama de envio de dados do dispositivo móvel para o servidor 64
Fig. 5.19 - Exemplo de ficheiro XML a enviar para o servidor 66
Fig. 5.20 - Excerto do código de recepção do pacote UDP no servidor 67
Fig. 5.21 - Estrutura das classes *Scan*, *NetworkCell, NeighborCell* e *WifiSpot* 68
Fig. 5.22 - Excerto do código para realizar o *parse* do documento XML no servidor 69
Fig. 5.23 - Excerto de código para colocação de dados na base de dados no servidor 71
Fig. 5.24 - Esquema de pedido HTTP do dispositivo móvel e resposta do servidor 72
Fig. 5.25 - Excerto de código presente na aplicação móvel para realizar pedido http 73
Fig. 5.26 - Código presente no servidor que retira os dados presentes na tabela *Scan* da base de dados para um determina dispositivo 74
Fig. 5.27 - Captura de pedido HTTP recebido no servidor 75
Fig. 5.28 - Captura de envio de resposta HTTP por parte do servidor 75

Acrónimos

API – Application Programming Interface

ASCII – American Standard Code for Information Interchange

BPSK – Binary Phase-Shift Keying

BSSID – Basic Service Set Identification

CDMA – Code Division Multiple Access

CEO – Chief Executive Officer

CID – Cell ID

CGI – Common Gateway Interface

CSS – Cascading Style Sheets

DC – Direct Current

GPS – Global Position System

GSM – Global System for Mobile

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IBM – International Business Machines Corporation

IP – Internet Protocol

LAC – Location Area Code

MIME – Multipurpose Internet Mail Extensions

PDA – Personal Digital Assistant

PHP – HyperText Preprocessor

RF – Radiofrequency

RFID – Radiofrequency IDentification

SDK – Software Development Kit

SQL – Structured Query Language

SSID – Service Set IDentifier

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

UMTS – Universal Mobile Telecommunications System

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

UWB – Utra-Wide Band

XML – Extensible Markup Language

1 Introdução

1.1 Motivação e Objectivos

O *Energy Harvesting* é uma das tecnologias que mais tem emergido e ganho relevo nos últimos anos. O objectivo é proporcionar a possibilidade de criar dispositivos móveis que sejam autónomos e não tenham a necessidade, ou grande necessidade, de serem recarregados por influência humana, mas sim por capacidade própria. Além deste facto, existe cada vez mais a tendência de colocar sensores nos mais variados tipos de locais, sensores esses que não podem muitas vezes estar constantemente a ser alimentados e que necessitam de ter a capacidade de se auto sustentar.

Uma questão que se levanta a quem pretende desenvolver dispositivos e realizar *Energy Harvesting* é que quantidade de energia se encontra disponível em cada zona. Tendo em conta essa questão, o Instituto de Telecomunicações, pólo de Aveiro, e o Centro Tecnológico de Telecomunicações da Catalunha decidiram desenvolver uma base de dados que permita guardar esta informação e que esteja disponível através de um servidor *web*. Através do sítio na *Internet* desenvolvido para o efeito é possível um utilizador registar-se e, a partir do momento em que esse esteja aceite, contribuir com leituras para essa base de dados através da ferramenta de *upload* na qual podem ser inseridos ficheiros com a extensão *.csv*. Depois de ser inserida na base de dados, a informação recolhida passa a estar acessível a qualquer pessoa através de um mapa presente no sítio da *Internet* onde é possível verificar a posição em que foi efectuada a leitura e detalhes acerca da mesma, nomeadamente um gráfico com o nível de potência para cada uma das frequências existentes no local. É possível visualizar esse mapa e o detalhe de uma leitura na Fig. 1.1.

A ferramenta desenvolvida e aqui descrita apresenta o problema de apenas permitir contribuições por parte de entidades que possuam as ferramentas necessárias para efectuar medidas de energia electromagnética presente no ambiente. Esta limitação leva a que a informação recolhida e presente no sistema seja necessariamente limitada e dependente da contribuição de um número reduzido de utilizadores.

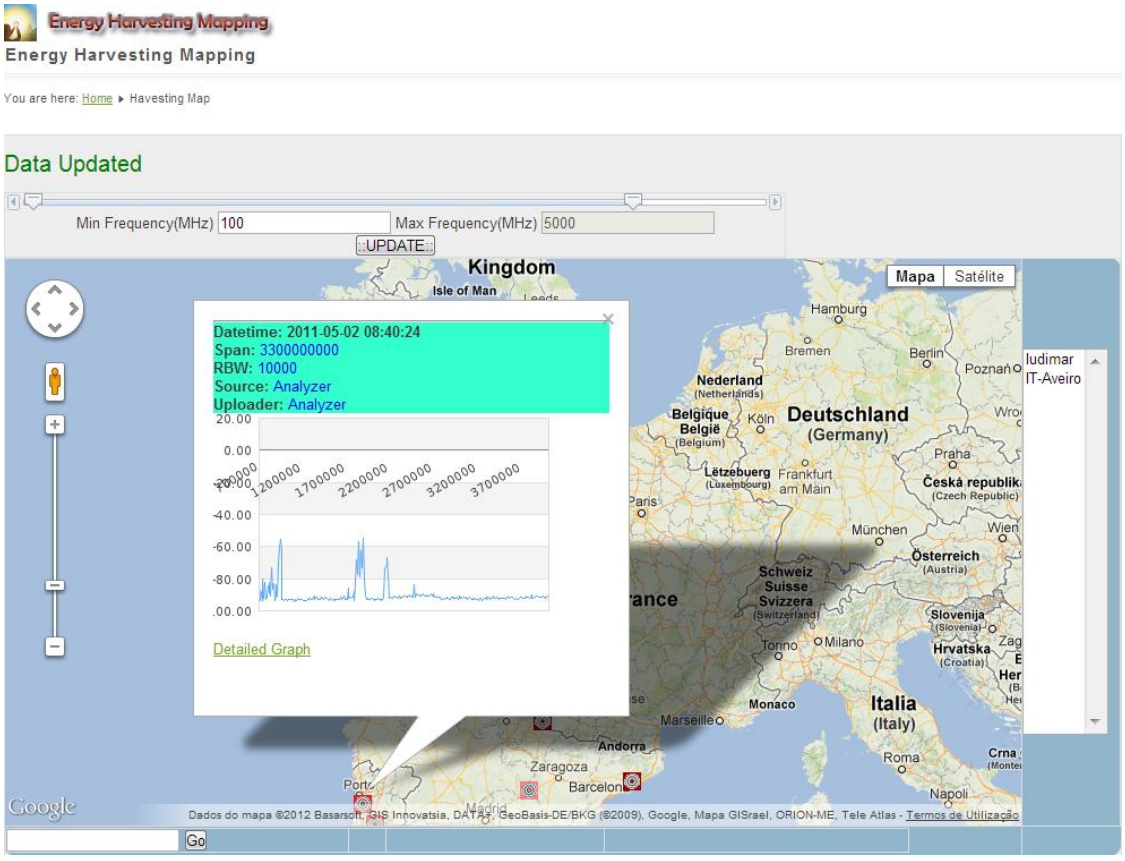

Fig. 1.1 - Mapa de leituras de energia electromagnética

Com o objectivo de ultrapassar essa limitação e permitir que qualquer utilizador em qualquer parte do Mundo possa contribuir com informação para o sistema foi pensada a criação de uma aplicação para dispositivos móveis que permita a recolha de informação da localização onde se encontra o dispositivo assim como da quantidade de potência presente nas bandas de frequência a que este tem acesso, redes móveis e *Wi-Fi*. A proliferação deste tipo de dispositivos permite aumentar o número de utilizadores a contribuir com informação para o sistema de uma forma quase exponencial. Assim, o trabalho desenvolvido e apresentado neste documento visa a criação dessa aplicação assim como o envio de informação por parte da mesma para a base de dados e apresentação da mesma através do mapa.

1.2 Estrutura da Dissertação

A presente dissertação encontra-se organizada por capítulos de forma a facilitar a sua leitura e a separar a informação que se pretende transmitir ao leitor.

O capítulo 2, *Energy Harvesting*, tem como objectivo dar a conhecer o conceito e o propósito que levou ao aparecimento desta tecnologia. É abordado de uma forma mais específica e aprofundada a vertente de *RF Energy Harvesting* por se tratar do ramo que mais interesse tem para o projecto desenvolvido e apresentado neste documento.

O capítulo 3, Computação Móvel, apresenta uma visão sobre a evolução dos dispositivos móveis e de como se chegou até aos dispositivos de que dispomos nos dias de hoje. É

apresentado o conceito de sistema operativo móvel e de uma forma mais particular o sistema operativo móvel Android por se tratar do sistema que dá suporte ao trabalho desenvolvido.

O capítulo 4, Tecnologias, tem a função de apresentar todas as tecnologias utilizadas no desenvolvimento do projecto apresentado. Começam por ser apresentadas todas as linguagens de programação utilizadas e o tipo de base de dados. Seguidamente são apresentadas as tecnologias utilizadas para permitir a comunicação entre o dispositivo móvel e o servidor que aloja a base de dados e o sítio na *Internet* e, por fim, são apresentadas as duas formas de localização utilizadas, visto que a obtenção da localização é aspecto fundamental no trabalho apresentado.

O capítulo 5, Implementação, é onde se apresenta o trabalho desenvolvido para cumprir os objectivos propostos. Inicialmente é apresentada a aplicação desenvolvida para dispositivos móveis, seguindo-se a apresentação do sítio na *Internet*. É depois apresentada a estrutura da base de dados onde todos os dados serão guardados e, por fim, é descrita a forma como as duas estruturas, aplicação móvel e servidor, comunicam entre si para envio e recepção de dados.

2 Energy Harvesting

A utilização de dispositivos móveis de vários tipos alimentados por uma bateria é uma realidade cada vez mais presente nos nossos dias e em todas as actividades. Um dos grandes inconvenientes deste tipo de dispositivos, se não o inconveniente maior, é a necessidade de recarregar a bateria dos mesmos num período de tempo relativamente pequeno. O ideal seria que a bateria de tais dispositivos tivesse um período de vida relativamente longo sem que o utilizador tivesse de a colocar a recarregar. Se o dispositivo possui-se uma forma de auto recarregar a sua própria bateria o tempo entre recarregamentos seria necessariamente menor. Mais importante do que para os dispositivos móveis utilizados por nós, e os quais podemos facilmente colocar a recarregar, essa capacidade ganha grande importância no caso de sensores que comunicam por radiofrequência e se encontram em sítios de difícil acesso ou em número demasiado elevado para que possam ser recarregados facilmente. Para que tal fosse possível era necessário que o dispositivo tivesse a capacidade de recolher energia do meio ambiente. A essa capacidade chamamos *energy harvesting*, ou, em português, colheita de energia. Este conceito é hoje em dia a base dos dispositivos RFID, utilizados para várias aplicações, em que a *tag* recebe energia vinda do leitor.

Assim, *energy harvesting*, é um processo que permite recolher energia, de fontes externas, utilizando dispositivos autónomos. A energia recolhida pode ter origem em variadas fontes, tais como, energia térmica, energia eólica ou, a que mais nos interessa neste documento, energia electromagnética transportada por ondas de radiofrequência.

2.1 RF Energy Harvesting

Actualmente muitos são os sistemas de comunicação presentes no nosso dia-a-dia que comunicam por radiofrequência, telemóveis, televisão, rádio, entre outros. Para que estes sistemas possam funcionar, e realizar a comunicação entre os seus vários elementos, possuem antenas que enviam para o ar energia electromagnética na forma de ondas rádio, dentro da banda de frequências em que operam contendo informação que é transmitida variando a amplitude, a frequência e a fase da onda de rádio dentro dessa banda. Ao contactar com um condutor, como uma antena, a radiação electromagnética induz corrente eléctrica na

superfície desse mesmo condutor. Muita desta energia acaba por se encontrar à nossa volta sem que lhe seja dado uso. É a pensar nisto que surge o *RF energy harvesting*. Este ramo do *energy harvesting* pretende criar dispositivos que tenham a capacidade de recolher a energia electromagnética, presente nas várias frequências, que se encontra no meio ambiente. Pode assim definir-se *RF energy harvesting* como sendo um processo pelo qual a energia de radiofrequência emitida por fontes que geram campos electromagnéticos de alta intensidade, tais como sinais de TV, redes *wireless* e torres de células de redes móveis de telefone, é capturada através de uma antena de recepção e convertida em voltagem do tipo DC para ser utilizada.

Olhando para os dispositivos que foram referidos anteriormente, dispositivos móveis para comunicação ou sensores presentes em diversos locais, esta pode ser a fonte de energia que mais se adeqúe ao seu recarregamento autónomo, uma vez que o acesso a luz solar ou a força do vento nem sempre está disponível, ao invés da energia criada por uma fonte de radiofrequência. Tendo por base esta ideia, além da tentativa de aproveitamento da energia que se encontra presente no ambiente, enviada pelos vários sistemas de comunicação sem fios existentes, existe também a ideia de projectar sistemas em que a rede de sensores é alimentada por *RF energy harvesting*, mas sendo a energia enviada por uma estação própria para o efeito e para alimentar esses sensores especificamente.

2.1.1 Estrutura Geral de um Circuito

A Fig. 2.1 tem representado o esquema com os vários componentes que fazem parte de um circuito que tem como objectivo realizar *RF energy harvesting*.

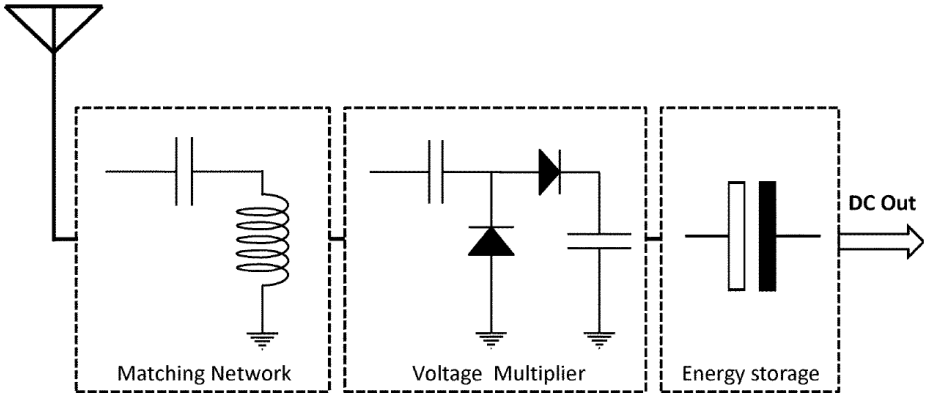

Fig. 2.1 - Sistema de *RF energy harvesting*[1]

O primeiro componente é, como seria de esperar, uma antena. De seguida está uma rede de adaptação que é composta por elementos capacitivos e indutivos e que assegura que é entregue o máximo de potência da antena para o multiplicador de voltagem. Por sua vez, o multiplicador de voltagem tem a função de transformar a energia que chega ao circuito na forma de ondas de radiofrequência em voltagem do tipo DC. Por fim, o componente de armazenamento de energia assegura que é fornecida à carga um nível de potência contínuo e que, quando não existe energia exterior a ser recolhida, esta não fica sem o fornecimento de energia [1].

Começando pelo primeiro componente, a antena, é necessário ter em conta que a energia electromagnética disponível no ambiente é de quantidade limitada. Desta forma torna-se necessário, regra geral, o aproveitamento da energia presente em frequências de diferentes valores. Para que tal seja possível é usual a utilização de UWB ou multibanda *rectennas* para que seja possível capturar sinais de várias frequências [2]. Na Fig. 2.2 pode ser visualizada uma *rectenna* com formato espiral.

Fig. 2.2 - *Rectenna* espiral [3]

Apesar de uma antena com as características das apresentadas anteriormente permitir obter energia de várias frequências e, por consequência, à partida maior quantidade de energia, nem sempre é esse o objectivo. Em muitos dos circuitos construídos o objectivo passa por utilizar a energia presente dentro de uma banda de frequências específica. Por exemplo, no caso de um sensor colocado nas proximidades de uma torre de suporte a uma rede móvel será preferível e mais eficiente o desenho de um circuito que capte energia dentro da banda de frequências utilizada para esse tipo de comunicação. Nesse caso, a antena é desenhada de forma a optimizar a recolha de energia dentro da banda pretendida.

Um dos pontos cruciais no circuito é o passo em que a energia de radiofrequência é convertida em voltagem DC. É necessário ter em conta nesta transformação que o sinal de radiofrequência recebido é, regra geral, muito fraco e com um valor de pico bastante menor do que o valor de *threshold* do díodo, são assim preferíveis díodos em que a tensão mínima de funcionamento seja o menor possível. Por outro lado, por norma a energia a ser colhida encontra-se em bandas de frequência elevada. Assim, é necessária a utilização de díodos com um tempo de comutação bastante rápido. Dessa forma é preferível a utilização de díodos *Schottky*, uma vez que estes utilizam uma junção metal-semicondutor ao invés de uma junção semicondutor-semicondutor, esta característica permite a este tipo de díodos operar de forma bastante mais rápida além de ter uma queda de tensão na ordem dos 0.15 V [1].

Na secção seguinte será possível visualizar dois circuitos diferentes para a realização de *RF energy harvesting*.

2.1.2 Exemplo de Aplicação

Para que seja possível uma melhor percepção e visualização de um circuito capaz de realizar a colheita de energia de ondas de radiofrequência e comparar a eficiência de um sistema que permite a recolha de uma banda larga de frequências com um que permite uma banda estreita vai de seguida dar-se atenção ao trabalho apresentado em [4]. Além disso, será ainda possível observar a eficiência dos dois circuitos e perceber a potencialidade deste tipo de solução para alimentar circuitos ou baterias.

Antes de partir para o desenho de ambos os circuitos foram efectuadas, pelos autores do trabalho, medidas, em diferentes pontos de um ambiente urbano, que possibilitam perceber a quantidade de energia disponível em cada um dos casos. Assim, no caso de banda larga estimou-se que a densidade de energia disponível dentro da banda de frequências entre os 680MHz e os 3500MHz varia entre -60dBm/m^2 e -14.5dBm/m^2. No caso da banda estreita de frequências, que se situa entre os 1800MHz e os 1900MHz, a densidade de energia disponível estimou-se ser em média de -14.5dBm/m^2.

Olhe-se agora para os circuitos projectados para recolher energia dentro das bandas referidas anteriormente e para as simulações efectuadas para os mesmos.

No primeiro caso, circuito para recolha de energia numa banda larga de frequências, para que seja possível obter energia de diversas frequências foi projectada uma antena semelhante à apresentada na Fig. 2.2. O desenho desta antena é de grande importância, uma vez que é necessário que exista adaptação óptima com o circuito de rectificação para um número elevado de frequências. A antena de forma espiral, além de permitir essa adaptação e de ser uma antena de banda larga, possibilita ainda uma polarização dupla e possui um padrão de radiação omnidireccional. Em termos de captação de energia, a antena projectada tem a capacidade máxima de capturar -42dBm [4].

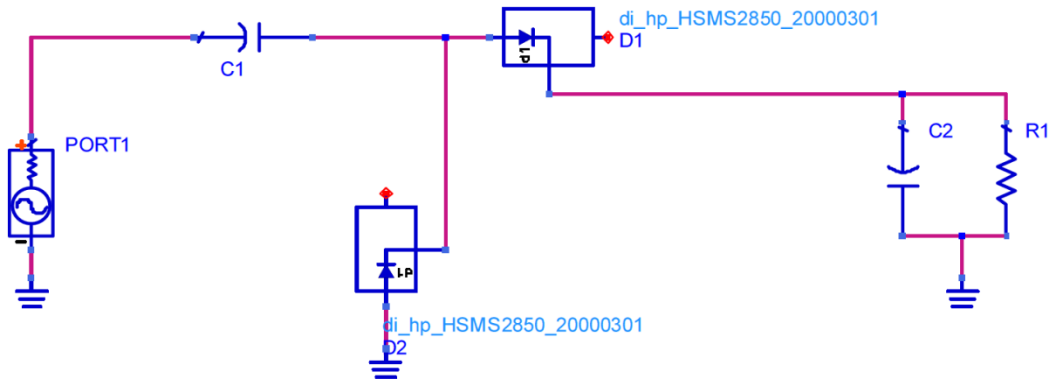

Fig. 2.3 - Circuito para *RF energy harvesting* sem malha de adaptação[4]

Tal como referido anteriormente, neste caso o circuito não possui uma malha de adaptação e o sinal já está adaptado quando sai da antena. Assim, a antena é conectada directamente ao circuito que faz a rectificação e que se encontra representado na Fig. 2.3. Devido às condicionantes referidas na secção 2.1.1 os díodos utilizados são díodos *Schottky*.

Para que se possa perceber a potência que é possível obter com o circuito é apresentado na Fig. 2.4 um gráfico onde se visualiza o valor da potência de saída, em pW, em função da resistência de carga e para a potência máxima de entrada que a antena é capaz de recolher, -42dBm. Olhando para o gráfico apresentado é possível verificar que os maiores valores de potência são obtidos para a frequência de 1GHz, e para uma resistência de carga situada entre os 15KΩ e os 20KΩ, além disso o valor da potência de saída situa-se entre os 5pW e os 10pW.

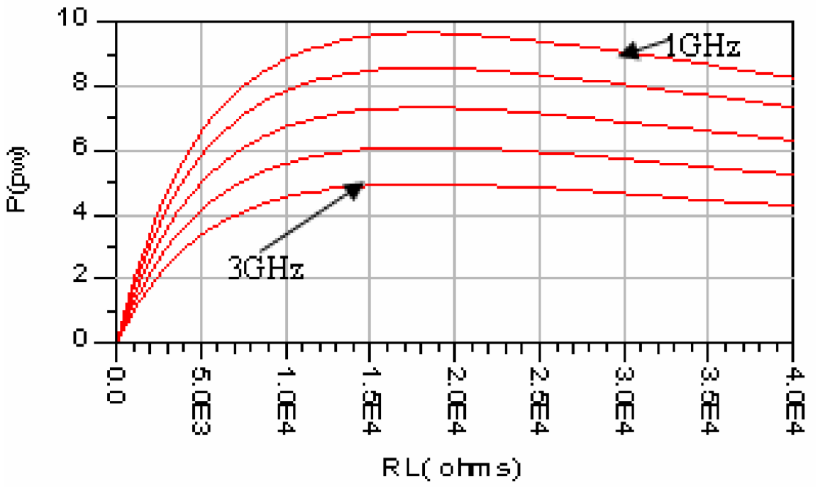

Fig. 2.4 - Gráfico de potência de saída para banda larga de frequências em função da resistência de carga e para uma potência de entrada de -42dBm [4]

Atente-se agora no segundo caso, um circuito para recolha de energia numa banda estreita de frequências. Neste caso, a construção da antena é de uma complexidade menor visto que apenas pretende atingir uma pequena banda de frequências, o que é possível com uma antena de construção simples para a banda em questão. O facto de se utilizar uma antena normal obriga a que se tenha uma malha de adaptação no circuito construído após a mesma, esse circuito pode ser visualizado na Fig. 2.5.

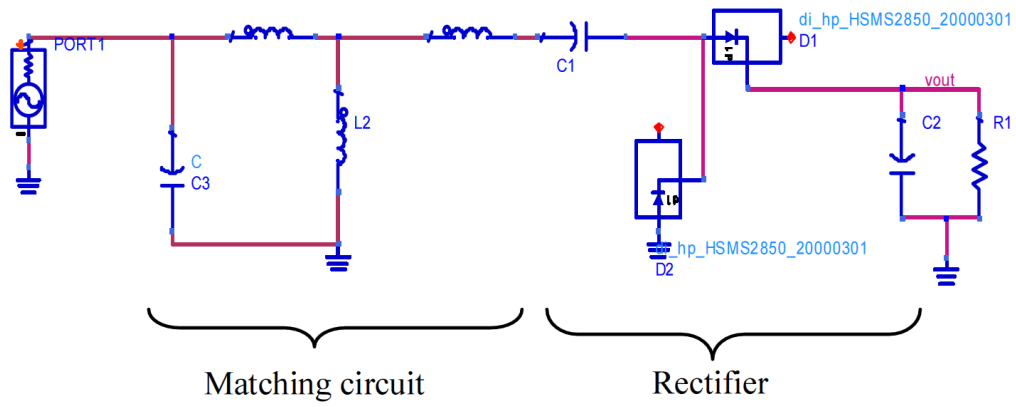

Fig. 2.5 - Circuito para *RF energy harvesting* utilizando malha de adaptação[4]

Além da malha de adaptação é também necessário mais uma vez o circuito rectificador que segue a mesma estrutura do que foi apresentado para o caso anterior.

Mais uma vez interessa perceber a quantidade de energia que é possível obter utilizando o circuito apresentado anteriormente. Para isso apresenta-se na Fig. 2.6 um gráfico onde é possível ver a quantidade de potência recuperada, em pW, em função da frequência, em GHz. Mais uma vez considera-se como potência de entrada no circuito, ou seja a potência máxima que é recuperada pela antena, -42dBm. Olhando para o gráfico anterior é verifica-se que é possível recuperar uma potência de perto de 400 pW para uma frequência de 1.8GHz.

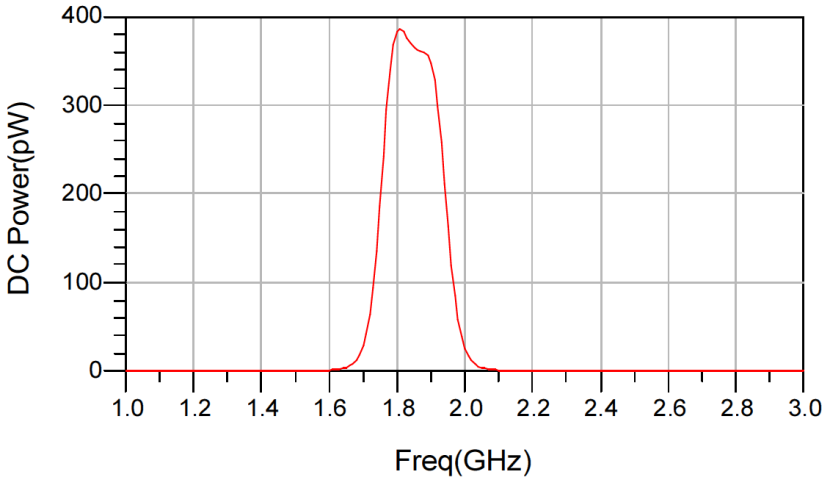

Fig. 2.6 - Potência recolhida estimada para o sistema de banda estreita [4]

Olhando para os resultados obtidos anteriormente é possível verificar que a quantidade de energia que é obtida por cada um dos sistemas é de valor bastante baixo, no entanto, esta energia pode ser utilizada para alimentar baterias utilizadas para alimentar circuitos com consumos bastante reduzidos.

Termina-se assim um pequeno resumo acerca do que é o *RF energy harvesting*, as aplicações em que pode ser utilizado assim como diferentes formas de o realizar.

3 Computação Móvel

A busca pela mobilidade total sempre foi uma das grandes ambições do Homem. Este desejo levou à criação dos vários meios de transporte de que hoje dispomos. Com o aparecimento do computador pessoal esse desejo em relação ao mesmo não foi menor. Foi devido a este desejo que hoje temos ao nosso dispor vários dispositivos móveis, tais como computadores portáteis, telemóveis e *smartphones.* O objectivo destes dispositivos é o de fornecer a liberdade de informação aos utilizadores.

Assim, a computação móvel tem como objectivo a criação de uma plataforma de gestão de informação livre de restrições espaciais e temporais. Estando livre destas restrições, o utilizador está livre para gerir a informação quando e onde quiser, esteja parado ou em movimento[5].

No presente capítulo iremos olhar de forma breve para a história deste tipo de dispositivos, olhando de forma preferencial para a evolução dos *smartphones* por se tratarem dos dispositivos com maior interesse no nosso documento, e perceber a sua evolução ao longo do tempo. Iremos ainda olhar para os vários sistemas operativos utilizados em *smartphones* e com especial atenção para aquele que deu suporte ao trabalho desenvolvido, o sistema operativo móvel Android.

3.1 Breve História da Computação Móvel

Assim que foi criado o primeiro computador pessoal a mobilidade sempre foi um factor presente. O primeiro computador pessoal era, de facto, facilmente transportável, apesar se não ter sido criado com esse objectivo. Esse computador, o Kenbak-1, Fig. 3.1, surgiu para venda em Setembro de 1971 com o custo de $750.00 [6]. No entanto, o número de vendas deste dispositivo foi baixo, cerca de 40 unidades, o que levou ao fecho da companhia que o produziu em 1973.

Fig. 3.1 - Kenbak-1 [7]

Desta data para a frente vários foram os computadores pessoais a surgir. Em Janeiro de 1975 surge o Altair 8800 vendido como um kit através da revista norte-americana *Popular Electronics*, e, em Abril de 1976, os fundadores da Apple, Steve Jobs e Steve Wozniak, lançam o seu primeiro computador, o Apple 1.

Em 1981 surge aquele que é considerado o primeiro computador pessoal portátil, o Osborne I, Fig. 3.2. Apesar de ser tido como um computador portátil a sua portabilidade não era a maior, já que o seu peso era de 10.6 Kg. No entanto, este já dispunha de um monitor agregado, não sendo apesar de tudo as suas dimensões as melhores, tinha 5 polegadas. Mesmo com todas as condicionantes este modelo foi um sucesso de vendas.

Fig. 3.2 - Osborne I [8]

O próximo marco importante na era dos computadores e de os tornar portáteis, já depois do aparecimento de alguns modelos, dá-se em 1984 com o lançamento do Apple Macintosh, Fig. 3.3. Este foi o primeiro computador a ter uma interface gráfica organizada em janelas e em que se utilizou o rato como ferramenta para a navegação nas mesmas. Outra grande novidade foi a capacidade de reprodução de áudio, algo nunca antes visto. A apresentação por Steve Jobs do Apple Macintosh é umas das mais arrebatadoras e surpreendentes de sempre na área dos computadores pessoais. O objectivo de Steve Jobs era que o computador fosse portátil, segundo este um dos requisitos é que deveria ser um bloco único que pudéssemos levantar e levar até casa. A ideia de portabilidade era tal que o computador era vendido dentro de uma mala e, aquando da sua apresentação, uma das frases reproduzidas pelo mesmo foi "Nunca confie num computador que não consiga transportar".

Fig. 3.3 - Apple Macintosh 1984 [9]

Após o surgimento de computadores pessoais que tentavam ser portáteis, o grande avanço na computação móvel foi o aparecimento do PDA, *Personal Digital Assistant*. Este termo foi utilizado pela primeira vez em 1992 pelo então CEO da Apple, John Sculley, aquando do anúncio do projecto Newton.

Após 1992 vários foram os modelos de PDA a surgir, nomeadamente com a criação da empresa *Palm Computing* nesse mesmo ano. É no fim desse mesmo ano que surge o primeiro protótipo daquilo a que viríamos a chamar um *smartphone*. O dispositivo foi criado pela IBM e tinha o nome de *Simon*, Fig. 3.4. Este foi o primeiro dispositivo a reunir as capacidades de um telemóvel e de um PDA num só.

Fig. 3.4 - *Simon*, o primeiro *smartphone* [10]

A partir desse momento vários foram os dispositivos a surgir com tais capacidades e, no ano de 2007, dois acontecimentos importantes acontecem para a evolução destes dispositivos. A 9 de Junho desse mesmo ano Steve Jobs apresenta o primeiro iPhone e, em Novembro é criada a *Open Handset Alliance* com o objectivo de dar suporte ao sistema operativo móvel Android. O primeiro *smartphone* a surgir no mercado com este sistema operativo foi lançado a 30 de Outubro de 2008 e era o modelo *Desire* da HTC. Daí até aos dias de hoje a evolução no segmento dos *smartphones* não parou de acontecer e muito haverá ainda para evoluir.

3.2 Sistema Operativo Móvel

Um sistema operativo móvel é um sistema operativo que é utilizado para controlar um dispositivo móvel, tal como um *smartphone* ou um *tablet*. Estes sistemas assemelham-se, hoje em dia, bastante aos sistemas operativos existente para os computadores pessoais, apesar disso têm de lidar com as limitações que um dispositivo móvel acarreta, tais como, menor capacidade de processamento, menor memória, o facto de ser alimentado por uma bateria e as suas dimensões físicas, principalmente no que respeita ao ecrã.

Actualmente vários são os sistemas operativos móveis existentes no mercado. Os sistemas que mais se destacam actualmente são o iOS, o sistema operativo móvel da Apple, o Android, o sistema da Google, o Windows Phone, sistema criado pela Microsoft, o RIM, o sistema da BlackBerry e o Symbian, o sistema operativo da Nokia, sendo que este último tem vindo a perder dimensão depois da Microsoft e a Nokia terem assinado um acordo para que os dispositivos da marca Finlandesa começassem a estar equipados com o Windows Phone.

A evolução na venda de dispositivos contendo cada um dos sistemas operativos móveis mais utilizados actualmente pode ser vista no gráfico da Fig. 3.5.

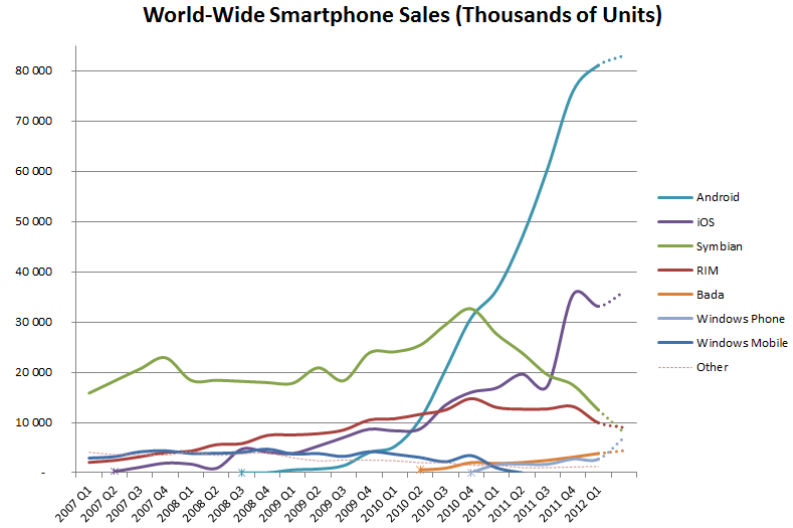

Fig. 3.5 - Gráfico da evolução na venda dos sistemas operativos móveis [11]

Através do gráfico anterior é possível observar o aumento quase exponencial na venda de dispositivos móveis utilizando o sistema operativo Android desde o seu aparecimento. Por outro lado é ainda possível verificar a queda nas vendas de dispositivos com o sistema operativo Symbian. Foi devido a esta queda acentuada que a Nokia decidiu alterar a sua política e voltar-se para o sistema operativo da Microsoft uma vez que este se encontra num nível de desenvolvimento mais avançado que o seu sistema operativo e com maior capacidade de ombrear no futuro com os líderes neste segmento, o Android e o iOS.

Uma vez que o trabalho referente a este documento foi desenvolvido para o sistema operativo móvel Android, iremos dar atenção ao mesmo na próxima secção.

3.3 ANDROID

Tal como referido anteriormente, o Android é um sistema operativo móvel cujo logotipo se encontra representado na Fig. 3.6. Na presente secção será apresentada uma breve história sobre o mesmo, apresentada a sua arquitectura e funcionamento e os fundamentos de uma aplicação deste sistema operativo móvel.

Fig. 3.6 - Logotipo Android [12]

3.3.1 BREVE HISTÓRIA

Em Outubro de 2003 nos Estados Unidos da América foi fundada por Andy Rubin, Rich Miner, Nick Sears e Chris White a Android Incorporation. Inicialmente, apenas se sabia que esta companhia estava a trabalhar em *software* para dispositivos móveis, sem se saber qual o seu propósito[13].

Em Agosto de 2005 a Android Incorporation é adquirida pela Google. Com esta aquisição fica claro que a Google se preparava para entrar no mundo das plataformas móveis.

É então em Novembro de 2007 que se dão dois marcos importantes no desenvolvimento do sistema operativo móvel Android. No dia 5 desse mês é criada a *Open Handset Alliance* formada por um grupo de companhias, tais como, Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile e Texas Instruments, apresentando o seu novo produto, um sistema operativo móvel. No dia 12 é então lançado o Android SDK, *Software Develompment Kit*, que passa a permitir a construção de aplicações.

Tal como referido anteriormente, é a 23 de Setembro de 2008 que é lançado o primeiro *smartphone* utilizando o sistema operativo móvel Android na sua versão 1.0, o HTC *Dream*.

Desde a primeira versão várias têm sido as versões lançadas pela Google do sistema operativo móvel Android, destacando-se o facto da versão 3 ser especificamente construída para a utilização em *tablets*. Actualmente, a versão mais recente é a 4.1. A família da versão 4 tem a particularidade de poder ser utilizada tanto em *smartphones* como em *tablets*.

3.3.2 ARQUITECTURA

A arquitectura do sistema operativo Android pode ser vista na Fig. 3.7. É possível observar que esta se encontra dividida em cinco camadas, o núcleo Linux, o tempo de execução Android, as bibliotecas, a camada de *framework* da aplicação e a camada das aplicações.

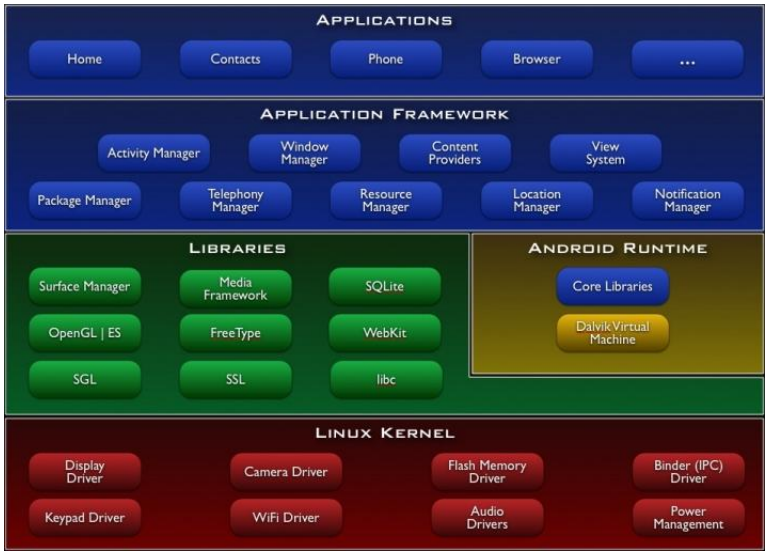

Fig. 3.7 - Arquitectura do sistema operativo móvel Android [14]

O núcleo do sistema é a versão 2.6 do Linux e o Android depende dele para serviços como a segurança ou a gestão de memória, entre outros. O núcleo é também responsável por actuar como camada de abstracção entre o *hardware* e as restantes camadas de *software*.

Na camada de bibliotecas encontram-se as bibliotecas nativas desenvolvidas na linguagem de programação C ou C++. Entre as várias bibliotecas temos uma biblioteca de C do sistema, bibliotecas de media para suporte de reprodução de vários formatos de áudio e vídeo, biblioteca SQLite para o fornecimento de base de dados para todas as aplicações, entre outras.

Dentro da camada de bibliotecas encontra-se a camada de tempo de execução do Android. Nesta camada estão contidas as bibliotecas Java e a máquina virtual Dalvik. Esta máquina virtual foi desenvolvida de forma a que cada uma das aplicações seja executada no seu próprio processo e com a sua própria instância, sendo que é possível a um dispositivo correr várias máquinas virtuais com eficiência.

De seguida existe a camada de *framework* da aplicação. Esta camada é escrita em linguagem Java e é a camada que permite aos programadores tirar proveito de todas as capacidades de *hardware* do dispositivo. Através desta camada é possível a qualquer programador ter acesso aos mesmos recursos que as aplicações do sistema. Permite ainda que as aplicações sejam desenhadas por forma a poder aproveitar os recursos já fornecidos por outras aplicações, tais como o envio de mensagens escritas, o envio de e-*mails* e muito mais.

Por fim temos a camada de aplicações. Esta é a camada onde se encontram todas as aplicações existentes no dispositivo e que são utilizadas pelo utilizador. Todas as aplicações presentes nesta camada são escritas na linguagem de programação Java.

3.3.3 APLICAÇÃO

As aplicações do sistema operativo móvel Android são escritas na linguagem de programação Java [17]. Após construída a aplicação o código, assim como outros ficheiros de dados, é compilado num ficheiro .apk que é depois utilizado para instalar a aplicação nos dispositivos.

Após se encontrar instalada num dispositivo Android cada aplicação vive no seu próprio mundo:

- Sendo o Android um sistema Linux multi-utilizador, cada aplicação é considerada um utilizador diferente e é executada num processo próprio e independente;
- Cada aplicação corre isolada de todas as outras, uma vez que cada processo tem a sua máquina virtual;
- Por defeito, a cada aplicação é atribuído um ID de utilizador. O sistema define as permissões para todos os ficheiros numa aplicação de modo que apenas o ID de utilizador atribuído a essa aplicação possa aceder-lhes.

Apesar de cada aplicação correr num processo único e ter acesso apenas aos seus componentes, existem mecanismos que possibilitam a troca de dados entre aplicações. É possível duas aplicações partilharem o mesmo ID de utilizador, o que permite a que cada uma aceda aos recursos da outra. Por outro lado, é possível a uma aplicação aceder a dados do

dispositivo em que foi instalada, desde que essas permissões tenham sido concedidas e aceites pelo utilizador aquando da instalação da mesma.

Uma das características mais importantes de uma aplicação Android é o facto de esta não possuir uma função *main()*, ou seja, uma aplicação não tem um ponto de inicio único. Esta característica permite que uma aplicação aceda a um componente de outra aplicação, desde que tenha permissão para tal, sem ter de iniciar a aplicação na sua totalidade, mas apenas o componente que lhe é necessário. Desta forma, não é necessário a uma aplicação conter todo o código de que necessita para executar todas as funções desejadas.

Existem quatro tipos de componentes diferentes, sendo que cada um serve um diferente propósito e tem o seu ciclo de vida próprio que define como o componente é criado e destruído[18]. De seguida explica-se cada um dos quatro componentes.

Activity
Uma *activity* é um componente que apresenta um ecrã simples ao utilizador e com o qual este tem a possibilidade de interagir. Na grande maioria dos casos, uma aplicação é composta por várias *activities*. Por exemplo, numa aplicação que funcione como agenda pode existir uma *activity* para mostrar a lista das várias tarefas a realizar, uma para mostrar os detalhes da tarefa e uma outra para criar uma nova tarefa.

Service
Um *service* é um componente que não fornece uma interface de utilizador. Este componente tem como tarefa a realização de tarefas em *background*, tarefas essas, por norma, de longa duração. Um *service* pode, por exemplo, ser um componente com a função de avisar o utilizador sempre que existe um golo numa partida de futebol, enquanto o utilizador está a utilizar outras aplicações ou tem simplesmente o dispositivo em modo de *standby.* Existe a possibilidade de outro componente, como por exemplo uma *activity*, iniciar um *service* ou conectar-se a ele para interagirem.

Content Provider
Um *content provider* permite que um conjunto de dados de uma determinada aplicação esteja disponível para que outras aplicações lhe acedam, desde que tenham permissão para tal. Esses dados podem estar armazenados no sistema de ficheiros, numa base de dados ou em qualquer outra localização de armazenamento a que seja possível aceder. Um exemplo de um *content provider* é o que é fornecido pelo sistema operativo Android para gerir as informações dos contactos presentes no dispositivo. Desde que tenha permissão, uma aplicação pode aceder a um contacto para ver a sua informação ou até alterá-la.

Broadcast Receiver
Um *broadcast receiver* tem a função de responder a anúncios de *broadcast*. Um *broadcast* pode ser o anúncio de que a bateria do dispositivo está em baixo, que houve uma alteração da língua utilizada no dispositivo, entre outros. Numa aplicação podem existir vários *broadcast receivers* para responder a alterações ou anúncios. Este componente não dispõe de uma interface com o utilizador, mas pode criar uma notificação na barra de estado do dispositivo de forma a alertar o utilizador da ocorrência de algo.

Após olhar para os quatro componentes que podem estar presentes numa aplicação é agora importante perceber como é que é possível activar cada um deles, quer pela própria aplicação, quer pelas aplicações exteriores que os pretendam utilizar.

No caso de se tratar de um *content provider*, a activação é feita através de um pedido por parte de um *Content Resolver*. O *content resolver* é utilizado para manipular as transacções de informação entre o *content provider* e o método que o chamou.

No caso dos outros três componentes, a sua activação é realizada através de uma mensagem assíncrona de nome *Intent*. Esta mensagem permite activar componentes da própria aplicação ou de aplicações externas. Existem dois tipos de *Intent*, o explícito e o implícito. No primeiro caso a mensagem contém informação sobre o componente a ser activado, no segundo caso a mensagem contém informação sobre o tipo de componente a ser activado. Por exemplo, no primeiro caso podemos definir que queremos uma certa actividade para realizar a tarefa pretendida, já no segundo caso apenas é pedida uma actividade que realize essa tarefa, sem especificar qual a actividade. No segundo caso, no caso de existir mais do que uma actividade capaz de realizar a mesma tarefa, será perguntado ao utilizador qual das actividades, ou aplicação, pretende para a sua realização.

Após conhecer e perceber como é que os componentes são activados é agora necessário entender como é que uma aplicação sabe da existência dos mesmos. Essa informação encontra-se declarada num ficheiro de nome *Manifest*. O ficheiro *Manifest* está presente em todas as aplicações e é escrito na linguagem *xml*. Neste ficheiro, além de estarem declarados todos os componentes que fazem parte da aplicação, estão declaradas coisas como as permissões necessárias ao funcionamento da aplicação, a versão mínima de sistema operativo Android em que é possível ser instalada, entre outros dados.

De seguida, apresenta-se um excerto de código presente num ficheiro *manifest* em que é declarada uma *activity*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity
               android:name="com.example.project.FreneticActivity"
               android:icon="@drawable/small_pic.png"
               android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Fig. 3.8 – Excerto de código de um ficheiro *manifest* com declaração de uma *activity*

Como dito anteriormente, várias são as características e informações da aplicação presentes dentro do ficheiro *manifest*. No que aos componentes presentes na aplicação diz respeito, estes encontram-se declarados dentro do elemento <application>, onde é declarado um elemento para cada um dos componentes e que varia consoante o tipo de componente. Tem-se assim os elementos:

- <activity> para as *activities;*

- <service> para os *services*;
- <receiver> para os *broadcast receivers;*
- <provider> para os *content providers*.

Dentro de cada um destes elementos são depois declaradas as características de cada um dos componentes. No exemplo apresentado em cima é possível ver que a *activity* declarada tem o nome com.example.project.FreneticActivity, declarado por android:name, tem um ícone que está declarado em android:icon e que o nome que é apresentado no ecrã está presente em android:label.

Após os componentes estarem declarados é preciso que as outras aplicações saibam as capacidades que eles têm e em que funções os podem utilizar. A declaração dessas capacidades é feita através de um atributo declarado dentro do atributo de cada componente. Esse atributo tem o nome de *Intent Filter* e é declarado como <intent-filter>. A sua representação pode ser vista na Fig. 3.9.

```
<activity android:name="com.example.project.FreneticActivity"
          android:icon="@drawable/small_pic.png"
          android:label="@string/freneticLabel"
          . . .  >
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category
                   android:name="android.intent.category.LAUNCHER"
              />
          </intent-filter>
</activity>
```

Fig. 3.9 – Excerto de código de um ficheiro *manifest* com declaração de um *intent-filter* numa *activity*

A forma como o sistema identifica o componente que pode responder a um *intent* é comparando o *intent* recebido com os *intent filters* fornecidos no ficheiro *manifest* de outras aplicações no dispositivo.

Por fim, falta perceber onde se encontram os recursos que não código e que também são necessários para o funcionamento da aplicação. No que a recursos de *layout* diz respeito, todos eles são escritos em linguagem *xml*. É possível ter cada um dos ecrãs de cada actividade separados, assim como um ficheiro apenas para conter todas as *strings* presentes na aplicação ou um ficheiro para cada um dos menus a ser apresentados. Com recursos como imagens ou áudio o processo é o mesmo, sendo que cada um deles se encontra na sua pasta própria. Este facto facilita uma possível mudança no futuro, visto que cada um dos recursos é utilizado através de um ID que lhe é atribuído pelo SDK Android.

Termina-se assim um breve olhar sobre o que é e o funcionamento básico de uma aplicação do sistema operativo móvel Android.

4 Tecnologias

No presente capítulo pretende-se dar a conhecer todas as tecnologias utilizadas no desenvolvimento do trabalho que é apresentado neste documento. Estas tecnologias irão ser separadas tendo em conta a sua natureza.

4.1 *SOFTWARE*

Na presente secção pretende dar a conhecer-se todas as linguagens de programação utilizadas para a realização do trabalho apresentado neste documento assim como o tipo de base de dados utilizado.

4.1.1 JAVA

Java é uma linguagem de programação de alto nível orientada a objectos desenvolvida na década de 90 pela *Sun Microsystems* e que tem uma forte inspiração em linguagens orientadas a objectos anteriores como o C++ e *Smalltalk* [19].

Na Fig. 4.1 podemos ver um exemplo do código Java, nomeadamente do método *main()* onde o programa é iniciado.

```
public class OlaMundo {
  public static void main(String[] args) {
    System.out.println("Olá, Mundo!"); //Imprime na tela a frase
  }
}
```

Fig. 4.1 - Excerto de código *Java*

Algumas das propriedades mais importantes da linguagem Java são:

- Orientada a objectos;
- Simplicidade;
- Coleta de lixo automática;
- Portabilidade;

- Programação *multi-threaded*;
- Segurança;
- Sensibilidade para a *Internet*.

Vejamos agora cada uma destas propriedades ao pormenor.

Programação orientada a objectos
A programação orientada a objectos foi pensada e criada com o objectivo de aproximar a programação à forma como pensamos sobre as coisas no nosso dia-a-dia.

Em termos práticos, e para simplificar, vamos comparar um objecto a algo do nosso dia-a-dia, uma bicicleta. A bicicleta será o nosso objecto. Uma bicicleta tem características, tais como a cor, altura, entre outras. Em termos de programação estas são as propriedades do objecto. Além das características, uma bicicleta tem funções tais como, andar, virar ou colocar no descanso. No nosso objecto isto são os métodos. Através de métodos é possível fazer com o objecto as acções necessárias.

A programação orientada a objectos tem a grande vantagem de permitir uma grande facilidade na reutilização de código, o que permite trabalhar a um nível de abstracção superior. Por outro lado tem uma maior complexidade de aprendizagem e um maior uso de memória.

Simplicidade
Apesar do facto de se tratar de uma linguagem orientada a objectos ser já uma vantagem e uma simplificação, isso por si só não faz de uma linguagem de programação algo mais simples.

Ao contrário do que acontece no C++, em Java não existem ponteiros para acesso à memória. O programador apenas pode aceder a um objecto por referências a esse objecto. Além de não possuir ponteiros para aceder a memória, uma outra simplificação reside no facto de o Java não possuir herança múltipla. Assim, cada objecto apenas contém uma classe pai. Estas características fazem do Java uma linguagem mais simples que as suas antecessoras.

Coleta de lixo automática
No caso de linguagens como o C e o C++ é necessário que o programador indique que necessita de um espaço de memória para guardar dados. Assim, a memória é gerida por ele e é ele que coloca e retira as variáveis da memória. Este processo pode ser bastante complexo e criar uma repetição enorme de colocar e retirar dados de memória se existirem várias variáveis que são utilizados por espaços de tempo muito curtos.

No caso da linguagem Java este processo é feito automaticamente sem a interferência do programador. Quando uma nova variável ou objecto é declarado, é a máquina virtual do Java que tem a preocupação de o colocar em memória e, após um objecto deixar de estar a ser utilizado, a referência para o mesmo passa a ser nula e este é retirado de memória quando esta estiver a ser necessária para guardar um novo objecto.

Esta característica facilita em muito o trabalho do programador, uma vez que este não tem de ter preocupações sobre o gerenciamento da memória disponível. Além da maior facilidade

para o programador, ainda tem a vantagem de eliminar o risco de a memória estar preenchida por dados que já não estão a ser utilizados nem vão mais ser necessários. Por outro lado, esta característica tem o contra de fazer com que uma maior quantidade de recursos computacionais seja utilizada para decidir que parte da memória pode ou não ser apagada.

Portabilidade
Em termos gerais, a portabilidade de um *software* significa que este pode ser compilado e executado em diferentes arquitecturas, tanto de *hardware* como de *software*.

No caso do Java, após o código ser compilado este pode ser executado em qualquer ambiente, desde que este possua instalada a máquina virtual Java. Este facto simplifica bastante a vida do programador que quando está a criar um programa não necessita de ter a preocupação de saber em que ambiente este vai ser utilizado nem de o refazer se este for necessário para um ambiente diferente.

Apesar de trazer grandes vantagens, a portabilidade em Java também tem desvantagens. Visto que o código depois de ser compilado irá correr na máquina virtual Java, este é compilado em *bytecode* ao invés de ser compilado em código máquina. Este facto faz com que a velocidade a que o programa é executado seja menor em relação a programas que estejam compilados em código máquina.

Programação *multi-threaded*
Em linguagens como o C existe a possibilidade de dividir o processamento de um processo em mais do que uma parte. No entanto, isto implica uma maior ocupação de memória, uma vez que para cada parte do processo é necessário uma duplicação de dados, tais como variáveis.

Uma solução alternativa a esta e com mais benefícios é a programação *multi-threaded*. Neste caso, existe processamento concorrente, mas com partilha dos dados e do código da aplicação. Desta forma, é possível que as *threads* conservem a memória e interajam entre si se for necessário [19].

Segurança
Actualmente, praticamente todos os computadores têm uma ligação à *internet*. Este facto aumenta a necessidade de todo o *software* presente num dispositivo possuir um grau de segurança bastante elevado, de forma a evitar ser corrompido.

É naturalmente possível que cada programador crie uma forma e mecanismos de proteger o programa que desenvolveu, mas o facto de essas ferramentas já estarem implementadas na linguagem facilita o seu trabalho.

De forma a evitar que um programa seja corrompido, o Java implementa a filosofia da caixa de protecção. Isto implica que todas as classes desconhecidas que sejam descarregadas para um dispositivo fiquem isoladas e que tenham de corresponder a certos requisitos para que possam ter acesso a todos os dados.

Sensibilidade para a *Internet*
Ao contrário de outras linguagens em que para se desenvolver uma aplicação que funcione através da *Internet* é necessário importar bibliotecas específicas e que alteram segundo o sistema operativo em que se encontram, o Java possui uma API para desenvolver aplicações que funcionem em rede sem qualquer problema de onde estas vão ser utilizadas.

Em particular, o Java oferece classes para os seguintes recursos de rede:

- Endereço IP;
- Pacotes UDP*;*
- *Streams* TCP;
- Pedidos HTTP;
- *Multicasting* de pacotes de dados.

4.1.2 HTML

HTML é a abreviatura para *HyperText Markup Language*. O HTML é uma linguagem de marcação que tem como objectivo possibilitar a formatação de textos e inserção de imagens na produção de páginas *web*. Posteriormente, o *web browser*, tem a responsabilidade de identificar as marcações presentes e apresentar o documento conforme o especificado.

A primeira especificação do HTML surgiu de forma pública em 1991. Desde ai várias foram as versões a surgir. A versão mais recente actualmente é o HTML 5.

A estrutura básica de um documento HTML é a apresentada na Fig. 4.2.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="description" content="a descrição do site">
        <meta name="keywords" content="palavras-chaves curtas”>
        <title>Título do Documento</title>
    </head>
    <body>
       <div>Tag para criar-se uma 'caixa', um bloco</div>
       <img src="endereço_de_uma_imagem.jpg" />
       <a href="http://www.wikipedia.org">Wikipedia, A
Enciclopédia Livre</a>
    </body>
</html>
```

Fig. 4.2 - Estrutura de um documento HTML

A partir do exemplo apresentado é possível desde logo notar que todos os elementos num documento HTML são apresentados numa *tag*, <tag> </tag>, são estas *tags* que definem a formatação do texto, ou outro objecto, que se encontra dentro delas. Dentro de uma *tag* é ainda possível ter atributos, como é o caso da *tag* <img>, apresentada no exemplo, que possui o atributo src.

Em termos de estrutura, o documento HTML apresenta duas secções principais, o *head* e o *body*.

Na primeira secção, o *head*, são colocadas informações relativas ao documento. Olhado para o exemplo é possível notar que na secção *head* está definido o título e alguns atributos como por exemplo as palavras-chave que dizem respeito ao documento.

Algumas das *tags* que podem estar presentes na secção *head* são:

- <title> - é a *tag* onde é definido o título da página, título esse que é depois apresentado na barra do *web browser*;
- <style> - *tag* onde podem ser definas folhas de estilo para a página;
- <script> - *tag* onde podem ser definidos *scripts* em diferentes linguagens e que podem ser depois utilizados na página;
- <link> - *tag* que define ligações da página com outros arquivos;
- <meta> - *tag* que define propriedades da página.

A segunda secção, o *body*, é onde são colocados todos os elementos que se pretendem que sejam apresentados na página. É possível a apresentação de vários tipos de elementos, tais como texto, imagens, tabelas ou *links*. A forma como cada um desses elementos é apresentado pode ser definida através dos atributos existentes na linguagem e dos vários tipos de *tag* existentes.

Dentro da secção *body* podem encontrar-se várias *tags*, sendo algumas das principais:

- <h1>, <h2> até <h6> - estas *tags* são utilizadas para definir títulos no documento com diversos tamanhos;
- <p> - é a *tag* que define um novo parágrafo;
- - esta *tag* determina uma divisão na página que pode possuir várias formatações próprias dessa mesma divisão e diferentes das definidas para o restante documento;
- <img> - *tag* utilizada para colocar uma imagem;
- <a> - *tag* utilizada para definir uma hiperligação.

4.1.3 CSS

CSS, abreviatura de *Cascading Style Sheets*, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em linguagem de marcação, como por exemplo o HTML [20].

Esta linguagem surge com o objectivo de simplificar a formatação de páginas escritas em HTML. Antes do uso de CSS, todo o conteúdo de um *site* escrito em HTML era formatado no documento em que estava escrito. Isto criava um trabalho extra e uma série de problemas. Imaginando o caso de um parágrafo. Se o programador quisesse que todos os parágrafos presentes na página tivessem um tamanho de letra diferente do que é padrão, teria de definir esse facto em cada um dos parágrafos que estivessem contidos em todas as páginas que fazem parte de um determinado *site*. Além disso, se um dia resolve-se alterar esse dado, o trabalho a ter seria imenso, pois teria de percorrer todos os parágrafos e alterar a informação em todos eles. Com a criação do CSS este problema está resolvido e muito facilitado. Todas as

formatações são definidas num documento à parte e, no nosso caso específico, apenas teríamos de definir que o tamanho de letra de parágrafos seria o que desejávamos e, para o alterar mais tarde, bastaria alterar um simples valor na folha de CSS.

Importa agora levantar uma questão. O que acontece se um mesmo tipo de *tag* tiver mais do que uma formatação definida. Imagine-se, por exemplo, que no caso do parágrafo, este estava definido com um tamanho de letra na folha CSS, mas que no documento HTML tinha sido definido outro tamanho para um parágrafo específico. É nestes casos que entra a regra de efeito cascata. Esta regra define qual a prioridade para definir a formatação a um determinado elemento.

A prioridade para o efeito cascata, em ordem crescente, é a seguinte:

- Folha de estilo padrão do navegador do utilizador;
- Folha de estilo do utilizador;
- Folha de estilo do programador:
  - Estilo externo, importado;
  - Estilo incorporado, definido na secção *head* do documento HTML;
  - Estilo *inline*, estilo que é definido dentro de um elemento HTML;
- Declarações do programador com o código !important;
- Declarações do utilizador com o código !important.

Depois de perceber como funciona o CSS e qual a importância da sua existência veja-se de seguida a sua sintaxe e como é declarado cada elemento. Na Fig. 4.3 encontra-se a estrutura seguida no código CSS.

```
selector [, selector2, ...][:pseudo-class] {
  property: value;
 [property2: value2;
  ...]
}
/* comment */
```

Fig. 4.3 - Estrutura de uma regra CSS

Olhando para o código acima comecemos pelo mais básico. O caso mais básico é aquele em que teremos um selector e, dentro desse selector, uma ou mais propriedades. O selector é o elemento HTML, normalmente identificado pela sua *tag*, e a propriedade é o atributo do elemento HTML ao qual será atribuído um valor. Dando um exemplo, o selector poderia ser p, *tag* para o parágrafo em HTML, a propriedade *font-size*, que simboliza o tamanho de letra, e, por fim, o valor 12px, letra com o tamanho de 12 pixéis. Se existir uma propriedade que se pretenda válida para vários selectores, estes podem ser agrupados. Poderíamos assim ter o selector p, seguido de uma vírgula e depois o selector h1, por exemplo. Esta representação evita a repetição da escrita da mesma propriedade para vários selectores diferentes.

Por fim, temos a *pseudo-class*. A utilização da *pseudo-class* permite adicionar efeitos a um selector ou a parte de um selector. Olhemos para o exemplo presente na Fig. 4.4.

```
p {
  font-size: 12px
}
p:first-letter {
  font-size:15px;
}
```

Fig. 4.4 - Exemplo de uma *pseudo-class* em CSS

O tamanho de letra definido para um parágrafo é de 12px, segundo o que está especificado pelo selector p, mas, mais abaixo, temos a regra *p:first-letter*. Esta regra define que a primeira letra de cada parágrafo deve ter o tamanho de 15px. Assim, *first-letter* é a *pseudo-class* que permite dar características diferentes à primeira letra de um selector.

4.1.4 PHP

PHP é o acrónimo recursivo para *PHP: Hypertext Preprocessor* e é uma linguagem de programação de *script Open Source* especialmente utilizada para desenvolvimento em plataforma *web* e que pode ser incorporada em documentos HTML [21]. Na Fig. 4.5 está o exemplo de um pequeno trecho de código PHP embutido no documento HTML que imprime a frase "*Hi, I'm a PHP script*".

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <?php
            echo "Hi, I'm a PHP script!";
        ?>
    </body>
</html>
```

Fig. 4.5 - Exemplo de *script* PHP embutido num documento HTML

O PHP foi criado por *Rasmus Lerdorf* em 1994 com o objectivo de obter a informação de quem acedia à sua página pessoal. Por esta altura, tratava-se de um simples conjunto de binários CGI (*Common Gateway Interface*) escritos em linguagem C. No ano seguinte, 1995, *Rasmus* acrescenta algumas funcionalidades e lança a primeira versão pública da linguagem.

Em Abril de 1996 é lançada a versão 2 do PHP. Esta versão, além de inserir mais ferramentas, traz a grande novidade de permitir trabalhar com bases de dados. Passa a ser possível colocar directamente no documento HTML *queries* mSQL. A meio de 1997, e tratando-se de um projecto de código aberto, a versão 2 já tinha crescido bastante e atraído um vasto número de utilizadores. No entanto, continuava a ser um projecto construído por apenas um desenvolvedor. É neste ponto que começam a voluntariar-se contribuidores para melhorar e fazer evoluir o projecto, passando assim este a ser um projecto de código aberto real. A partir deste ponto grande passou a ser o crescimento e evolução do PHP, tornando-se num projecto de escala mundial. Actualmente este encontra-se na versão 5.

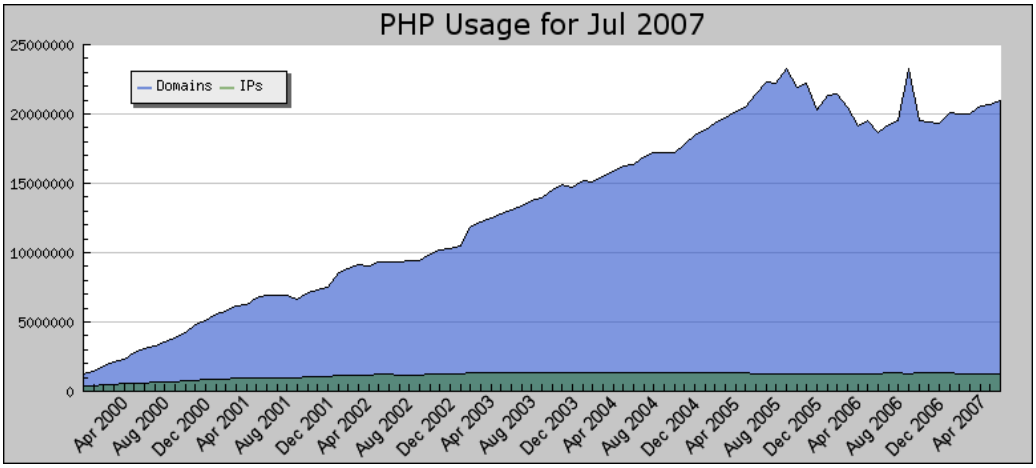

Fig. 4.6 - Evolução da utilização do PHP [22]

Vejamos agora quais as principais características da linguagem PHP.

Código de fonte aberto
Tal como referido anteriormente, o PHP é uma linguagem de código aberto. Isto significa que, desde que não sejam violados os direitos de autor da linguagem, pode ser feita qualquer modificação ao código. O código fonte e a documentação do mesmo encontram-se no *site* oficial do PHP. Este facto permite que os *bugs* que vão surgindo possam ser reparados mais rapidamente.

Velocidade e estabilidade
O PHP é uma linguagem com uma execução bastante rápida, além de que exige poucos recursos do sistema. Este facto é potenciado pelo facto de ter sido desenvolvido de forma a ser um pequeno invólucro à volta de muitas chamadas do sistema operacional. Além disso, o facto de possuir o seu próprio sistema de gestão de recursos e um método eficiente para manipular variáveis faz do PHP muito estável.

Orientado a objectos
O PHP é uma linguagem de programação orientada a objectos. As características deste tipo de linguagem já foram descritas na secção referente à linguagem de programação Java.

Sintaxe similar a C e C++
Para programadores com experiência na linguagem C ou C++, ou até mesmo em Java, a adaptação ao PHP é bastante simples devido à semelhança que este tem com estas linguagens.

Baseado no servidor
Uma das vantagens para o utilizador que aceda a páginas que contenham PHP é o facto de este ser baseado no servidor. Isto significa que o código do *script* PHP é executado no servidor onde se encontra a página alojada e não será carregado na memória do computador do utilizador como acontece com outras linguagens de *script*. Evita-se assim o consumo de recursos no computador pessoal do utilizador. Além da vantagem para o utilizador, este facto também tem uma importante vantagem para o programador, uma vez que isto faz com que os

scripts de PHP não possam ser vistos por ninguém que acede a uma página *web*. Assim, a segurança é maior no que a cópias diz respeito.

Portabilidade
A portabilidade é outra das características que o PHP tem em comum com o Java. Tal como foi visto aquando da descrição do Java, esta característica permite-lhe ser executado em diferentes ambientes quer em termos de *software* quer em termos de *hardware*.

Tipo de variáveis dinâmico
Esta característica faz com que o tipo de dados de uma variável não necessite de ser declarado. Assim, é a própria linguagem de programação que escolhe o tipo de dados a utilizar para cada variável, podendo este ser alterado dinamicamente durante a execução do *script*.

Acesso a base de dados
O PHP possui a capacidade de aceder e manipular vários tipos de base de dados, possuindo várias funções para o efeito. Vários são os tipos de base de dados suportados, entre eles temos o *SQL Server*, *MySQL*, *PostgreSQL*, entre outros.

4.1.5 JAVASCRIPT

O *JavaScript* é uma linguagem de *script* utilizada para desenvolver aplicações para a *Internet*. Actualmente, é considerada a principal linguagem para programação *cliente-side* para *web browsers*. Na Fig. 4.7 encontra-se um pequeno exemplo de um *script* contido num documento HTML.

```
<html>
  <head>
    <script language="javascript">
      var nome;
      nome = window.prompt("Digite o seu nome:");
      document.write("Bom dia, " + nome + "!<BR>");
    </script>
  </head>
  <body>
  </body>
</html>
```

Fig. 4.7 - Exemplo de *script* de *JavaScript* num documento HTML

JavaScript foi desenvolvido inicialmente na *Netscape* por *Brendan Eich*. O primeiro lançamento desta linguagem deu-se em 1995 na versão beta do *web browser Netscape* 2.0 tendo na altura o nome oficial *LiveScript*. No entanto, ainda no mesmo ano, no mês de Dezembro, o nome é alterado para *JavaScript* em acordo com a *Sun Microsystems*. Esta alteração de nome é caracterizada por muitos como uma estratégia de *marketing*, visto que à altura tinha sido lançada a linguagem de programação *Java*.

Em 1996 e após uma grande aceitação do *JavaScript*, a *Microsoft* decide desenvolver uma linguagem clone de seu nome *JScript* que é lançada no *Internet Explorer* 3.0.

Após o lançamento por parte da *Microsoft* surge o problema de existir mais do que uma versão para uma mesma linguagem, sendo que essa linguagem não tinha um padrão. Assim, em Novembro de 1996 a *Netscape* anuncia que submetera o *JavaScript* para a ECMA, *European Computer Manufacters Association*, como candidato a padrão industrial. Daqui surgiu um padrão definindo uma nova linguagem de *script* de nome ECMAScript e que se encontra actualmente na sua 5ª versão.

O *JavaScript* contém algumas das características já vistas em linguagens anteriormente descritas neste documento, tais como, portabilidade, com uma sintaxe similar a C e C++ e tipo de variáveis dinâmico.

Uma das características que contrasta com a linguagem de *scripts* descrita anteriormente, o PHP, é o facto de o *JavaScript* ser uma linguagem baseada no lado do cliente. Isto significa que os *scripts* são processados no computador do utilizador. Este facto apresenta um contra em termos de suporte, uma vez que é necessário que o *web browser* que está a aceder à página *web* contenha suporte para *JavaScript*. É de notar que, hoje em dia, a maioria dos *web browsers* contém esse suporte. Além desta diferença, ainda existe outra em relação às linguagens anteriores. O *JavaScript* é baseado em objectos e não orientado a objectos. Isto significa que a linguagem contém os seus próprios objectos embutidos, ao invés de estes serem construídos através de classes.

4.1.6 XML

O XML, *Extensible Markup Language*, é um formato baseado em texto simples para representar informações estruturadas: documentos, dados, configurações, livros, transacções, facturas e muito mais. Foi derivada a partir de um formato mais antigo padrão chamado SGML, a fim de ser mais apropriado para o uso na *web*[23].

O XML surgiu em 1996 com o objectivo de ser utilizável através da *Internet*, com uma linguagem de fácil compreensão e legibilidade para humanos, suportado por um vasto leque de aplicações e de ser de grande facilidade criar um documento. Actualmente, o XML é utilizado principalmente para compartilhar dados através da *Internet.*

Na Fig. 4.8 apresenta-se um pequeno documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<contactslist>
    <contact>
        <name>João</name>
        <phone>999191911</phone>
        <email>joao@ua.com</email>
    </contact>
    <contact>
        <name>Pedro</name>
        <phone>989291911</phone>
        <email>pedro@ua.com</email>
    </contact>
</contactslist>
```

Fig. 4.8 - Pequeno documento em XML

Tal como é possível observar no excerto acima, o XML tem uma grande similaridade com o HTML, ambos se tratam de linguagens de marcação. Importa então perceber as reais diferenças entre as duas linguagens. Uma das diferenças é logo notada à partida quando se percebe quais os objectivos de cada uma, o HTML tem como objectivo a exibição de dados e organiza a forma como estes são apresentados, já o XML tem como objectivo a descrição dos dados. Por outro lado, é possível verificar que, em relação ao XML, as *tags* existentes não têm limite e são definidas pelo construtor do documento, já em relação ao HTML as *tags* a utilizar estão limitadas às pré-definidas e suportadas pela linguagem.

Por fim, veja-se as características e vantagens da linguagem XML, muitas delas são coincidentes com aqueles que eram os requisitos aquando da criação da linguagem.

Simplicidade
A linguagem XML está construída de uma forma que se torna bastante simples de ser lida tanto por humanos como por máquinas. Além disso, a sua sintaxe e regras básicas permitem que o programador rapidamente tenha facilidade em trabalhar com a linguagem.

Extensibilidade
Tal como já referido aquando da comparação com a linguagem HTML, a linguagem XML não contém *tags* previamente definidas ou um conjunto limitado das mesmas, o que permite ao programador criar um número ilimitado de *tags* consoante as necessidades que tem.

Interoperabilidade
A linguagem XML pode ser utilizada sobre uma grande variedade de sistemas e ser interpretada por uma grande variedade de ferramentas [24].

Abertura
O processo padrão da linguagem XML é completamente aberto e está disponível na *Internet* para qualquer utilizador de forma gratuita [24].

4.1.7 MySQL

O *MySQL* é um dos sistemas de gestão de bases de dados mais utilizados em todo o mundo e utiliza como interface a linguagem SQL.

Uma base de dados é um conjunto de dados que podem ou não estar relacionados. Esses dados podem ser de vários tipos, texto, números ou mesmo ficheiros binários. Há vários anos que as bases de dados se tornaram a principal ferramenta de qualquer sistema de informação. Um sistema de gestão de bases de dados não é mais do que o *software* que é responsável por gerir a base de dados.

O *MySQL* foi desenvolvido por David Axmark, Allan Larsson e Michael Widenius tendo surgido a sua primeira versão a 23 de Maio de 1995. No ano de 2008 a companhia que desenvolve o *MySQL* foi adquirida pela *Sun Microsystems*. Actualmente goza de grande popularidade, contribuindo muito para esse facto a sua fácil integração com a linguagem de programação PHP.

4.2 COMUNICAÇÃO

Nesta secção pretende-se dar a conhecer as tecnologias de comunicação utilizadas para enviar os dados da aplicação móvel para o servidor, onde se encontra a base de dados que contém a informação, e para retirar informação presente no servidor para ser apresentada ao utilizador na aplicação móvel.

4.2.1 UDP

O UDP, acrónimo de *User Datagram Protocol*, é um protocolo que pertence à camada de transporte dos protocolos de *Internet*. A camada de transporte é a camada responsável pela transferência de dados entre a máquina de origem e a máquina destino, independentemente do tipo, topologia ou configuração das redes físicas existentes entre elas[25]. Na Fig. 4.9 é possível verificar o modelo TCP/IP e a localização do protocolo UDP no mesmo.

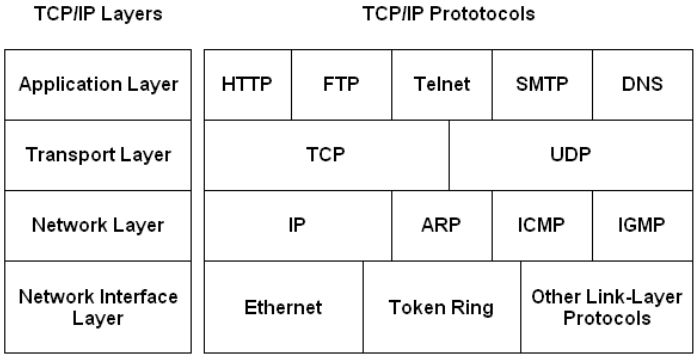

Fig. 4.9 - Camadas TCP/IP [26]

Vejamos de seguida o funcionamento do protocolo e a estrutura de uma trama do mesmo.

Funcionamento do protocolo
O protocolo UDP complementa o endereçamento que é fornecido na camada de rede pelo protocolo IP. Enquanto o protocolo IP tem como função o endereçamento lógico e permite a transmissão de dados entre duas máquinas presentes na mesma rede, o protocolo UDP fornece um endereçamento adicional que permite que sejam criadas várias conexões para várias aplicações numa mesma máquina. A conexão para cada uma das aplicações é feita através de uma ligação à rede, sendo atribuída a cada uma dessas ligações um número que é definido como número de porto.

Contrariamente ao outro protocolo presente na camada de transporte, o protocolo TCP, o protocolo UDP não é orientado para a ligação, visto que não há necessidade de manter uma relação entre o cliente e o servidor. Assim, este protocolo é pouco confiável, uma vez que não existem garantias de que todos os dados chegam ao destino ou que a sua recepção é feita pela mesma ordem com que foram enviados.

Pacote UDP
Na Fig. 4.10 está representado o esquema do *header* de um pacote UDP. Vejamos de seguida o significado e função de cada um dos campos representados.

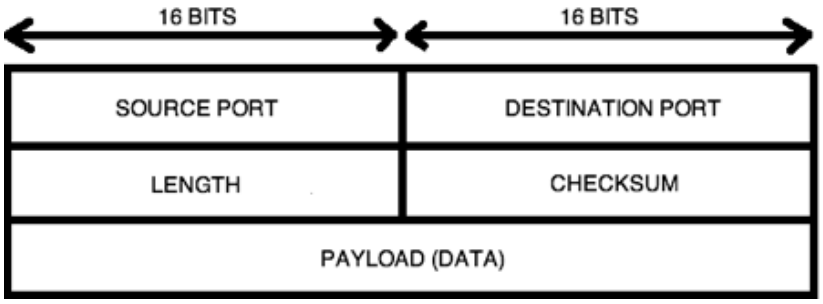

Fig. 4.10 - *Header* de um pacote UDP [27]

Cada campo presente tem o tamanho de 16 bits, excepto o campo referente aos dados a serem enviados, sendo que no caso dos campos *Source Port* e *Checksum* a sua utilização é opcional no caso do IPv4, no caso do IPv6 apenas o campo *Source Port* é de utilização opcional.

Source Port
Este campo destina-se a indicar o número de porto de onde os dados foram enviados e, no caso de ser necessário proceder a uma resposta, deve ser o porto considerado para a enviar. Se não for utilizado, deve estar definido com o valor zero.

Destination Port
Neste campo é inserido o número do porto do processo para o qual deve ser enviada a informação no campo referente a dados.

Length
No campo *Length* é inserida a informação do tamanho total do pacote a ser enviado. O valor pode variar entre 0 e 65535, mas, devido ao número de bytes presentes no *header*, o seu valor mínimo será sempre 8.

Checksum
Este campo é utilizado para testar se existiram erros na transmissão do pacote UDP. Vejamos agora como é efectuado esse teste. Para o cálculo a efectuar é adicionando um *pseudoheader* com os dados referentes ao protocolo IP. Depois, todos os dados, *pseudoheader* IP, *header* UDP e dados, são organizados em blocos de 16 bits. É então feita a soma de todos esses blocos e feito o complemento de um que é então armazenado no campo *Checksum*. Quando um pacote chega ao destino, o procedimento é refeito. Tendo em conta os dados recebidos são somados blocos de 16 bits. Esse valor é depois somado ao valor presente no campo *Checksum* e se o resultado dessa soma for igual a 0 é considerado que não existiram erros na transmissão. É de referir que no caso do IPv4 este campo não é obrigatório e pode estar preenchido a zeros. No entanto, pode acontecer que o complemento de um da soma dos 16 bits seja igual a 0, estando a ser utilizado o *Checksum*, por esta razão, quando tal acontece, o campo é todo preenchido a um.

4.2.2 HTTP

O HTTP, acrónimo de *Hypertext Transfer Protocol*, é um protocolo de comunicação que pertence à camada de aplicação do modelo de protocolos TCP/IP, como se pode verificar na

Fig. 4.9. A camada aplicação é a que tem a responsabilidade de fornecer serviços para as aplicações de modo a que estas se abstraiam da existência de uma rede por trás da comunicação entre processos presentes em diferentes dispositivos.

Funcionamento do Protocolo

O HTTP é um protocolo de pedido/resposta, isto significa que o cliente envia ao servidor um pedido, obtendo uma resposta por parte do servidor. Esse processo está esquematizado na Fig. 4.11.

O pedido enviado ao servidor contém um URI, a versão do protocolo, seguidos de uma mensagem MIME que contém modificadores de requisição, informações sobre o cliente e, possivelmente, conteúdo no corpo da mensagem[28].

O servidor responde ao pedido vindo do cliente com uma linha de estado, incluindo uma mensagem com a versão do protocolo e um código de sucesso ou erro, tudo isto seguido de uma mensagem MIME contendo informação do servidor, metainformação da entidade e, possivelmente, conteúdo no corpo da mensagem.

MIME, sigla de *Multipurpose Internet Mail Extensions*, é um protocolo que foi estabelecido com o objectivo de permitir a inclusão via *email* de dados que não estejam no formato ASCII.

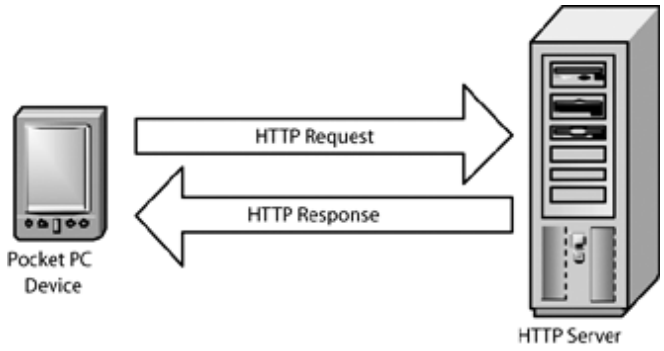

Fig. 4.11 - Esquema de funcionamento do protocolo HTTP [29]

Mensagem HTTP

A comunicação, entre cliente e servidor, atrás descrita é feita pelo meio de mensagens. As mensagens de requisição e resposta são ambas constituídas por uma linha inicial, nenhuma ou várias linhas de cabeçalho às quais se segue uma linha em branco de carácter obrigatório, que indica o fim do campo de cabeçalhos e, por fim, o corpo da mensagem que é opcional.

A linha inicial tem diferenças no caso de se tratar de uma requisição ou de uma resposta. No primeiro caso, a linha é constituída pelo nome do método HTTP, pelo caminho local do recurso requerido e, por fim, pela versão de HTTP que está a ser utilizada. Todas as informações se encontram separadas por um espaço. No caso da resposta, a linha é constituída inicialmente pela versão de HTTP utilizada, seguida de um código indicando o resultado do pedido feito anteriormente e, por fim, uma descrição legível do significado do código anterior. Mais uma vez todas as partes se encontram separadas por meio de um espaço.

Em relação às linhas de cabeçalho, estas fornecem informação acerca do pedido e da resposta ou acerca do conteúdo enviado no corpo da mensagem. Os tipos de cabeçalho existentes são quatro, sendo eles *general-header, request-header, response-header* e *entity-header*. O formato pelo qual estes são descritos é "header-name: value".

Por último o corpo da mensagem. O corpo da mensagem, como seria de esperar, tem diferentes utilidades no caso da requisição e da resposta. Aquando do pedido, o corpo da mensagem pode conter dados de utilizador ou arquivos a serem enviados para o servidor. Já no caso da resposta, o corpo da mensagem contém o recurso que foi anteriormente requerido ao servidor ou, caso essa requisição não tenha sido respondida com sucesso, contém uma mensagem de erro. No corpo da mensagem podem ser incluídos cabeçalhos para dar informações acerca do mesmo. Por exemplo, o cabeçalho *Content-Type* indica o tipo MIME do conteúdo presente, sendo que o cabeçalho *Content-Lenght* informa a quantidade de bytes presentes no corpo da mensagem.

O conteúdo das mensagens de requisição e resposta está representado na Fig. 4.12, onde se pode observar um exemplo. Por fim, iremos ver de seguida os métodos que podem ser utilizados na mensagem de requisição.

Fig. 4.12 - Exemplo de troca de mensagens HTTP entre cliente e servidor [29]

Métodos HTTP

Os métodos são definidos para indicar qual a acção a ser executada sobre o recurso que é indicado na mensagem de requisição. Os métodos são oito e segue em seguida uma descrição de cada um deles.

OPTIONS

Este método tem como função requerer informação acerca dos métodos suportados pelo recurso indicado presente no servidor.

GET
O presente método indica que qualquer que seja a informação presente no caminho indicado, esta deve ser retornada para o cliente. No caso da informação do caminho presente na requisição se tratar de um conjunto de formas para produção de dados, a informação devolvida deve ser esses mesmos dados produzidos.

HEAD
O método HEAD é semelhante ao que foi descrito anteriormente, o método GET, com a diferença que neste caso a informação presente no caminho não deve ser enviada na resposta à requisição, apenas é enviada a metainformação acerca da informação. Essa metainformação vai estar presente no cabeçalho de resposta. Isto permite que se saiba a informação acerca do conteúdo do caminho sem que esse conteúdo tenha de ser transferido.

POST
Este método tem como objectivo o envio de dados por parte do cliente para o servidor, dados esses a serem utilizados pelo recurso presente no caminho especificado pela mensagem de requisição.

PUT
A função do presente método é a de enviar um recurso a ser armazenado no caminho especificado.

DELETE
Tal como o próprio nome indica, o método DELETE tem como função enviar um pedido para que o servidor apague o recurso especificado pelo caminho fornecido.

TRACE
Este método é utilizado para invocar um controlo remoto.

CONNECT
Este método é reservado para utilizar com um *proxy* que pode dinamicamente alterar-se para ser um túnel.

4.3 Sistemas de Localização

Nesta secção pretende-se dar a conhecer os dois tipos de localização utilizados para marcar a localização da informação recolhida.

4.3.1 GPS

O GPS, *Global Positioning System*, é um sistema de navegação por rádio que foi desenvolvido pelo departamento de defesa dos Estados Unidos da América. O seu objectivo inicial era tornar-se o principal sistema de navegação do exército americano, mas devido à sua exactidão e grau tecnológico presente nos receptores a sua aplicação passou para várias actividades de índole civil.

O primeiro passo para o surgimento do GPS deu-se com o lançamento, por parte da Rússia, do satélite *Sputnik*, em 1957. Depois de monitorarem o sinal de rádio desse mesmo satélite, dois

físicos americanos perceberam que através da força desse sinal podiam saber a posição em que o satélite se encontrava. Assim, em 1960 surgiu o primeiro sistema de navegação por satélite, o TRANSIT, utilizado pela marinha dos Estados Unidos e que tinha como objectivo fornecer a posição dos submarinos. A constelação era constituída por cerca de 5 satélites e era necessário às unidades presentes em terra esperar um tempo aproximado de uma hora para obter uma nova localização. Em 1967, e ainda dentro do projecto TRANSIT, foi dado um passo importante na evolução da tecnologia que viria a ser a base do GPS. Nesse ano foi desenvolvido o satélite *Timation* que provou a capacidade de colocar no espaço relógios precisos. A sua lógica era transmitir uma referência de tempo precisa que pudesse ser utilizada pelos receptores presentes em terra.

Aproveitando os avanços feitos anteriormente, em 1973 é iniciado um projecto, por parte do Departamento de Defesa dos Estados Unidos, liderado por Ivan Getting e Bradford Parkinson, com o objectivo de fornecer um serviço de informação de navegação contínuo. Ainda antes de serem enviados satélites, no ano de 1977, foram feitos testes a transmissores instalados na superfície terrestre. Esses transmissores receberam o nome de *Pseudolites*[30]. O primeiro satélite foi enviado no ano de 1978 e até ao ano de 1985 onze satélites tinham sido enviados para o espaço.

Apesar de ter sido pensado para uso do sistema de defesa dos Estados Unidos, em 1983 o sistema passou a estar disponível para uso público depois de um avião Americano ter sido abatido no espaço aéreo Soviético devido a se ter perdido neste espaço. Daí em diante, o sistema apenas esteve interdito para uso público entre 1990 e 1993 devido à necessidade do exército dispor de mais receptores em razão da ocorrência da primeira guerra do Golfo.

No ano de 1995 é tomada a decisão de que o sistema GPS devia ser utilizável e gratuito para o mundo inteiro e, nesse mesmo ano, é activado o último dos 24 satélites da constelação. Desde esse ano até aos dias de hoje a evolução do sistema não tem parado e mais satélites foram enviados para o espaço a fim de melhorar o sistema quer em precisão quer em disponibilidade.

4.3.1.1 Tecnologia

O sistema GPS é composto por três segmentos separados, o segmento espacial, onde se encontram os satélites, o segmento de controlo, onde estão as estações terrestres que controlam todo o sistema e, por fim, o segmento de utilizador, onde se inserem todos os utilizadores do sistema. Estes segmentos encontram-se representados na Fig. 4.13 e a sua explicação em detalhe será dada de seguida.

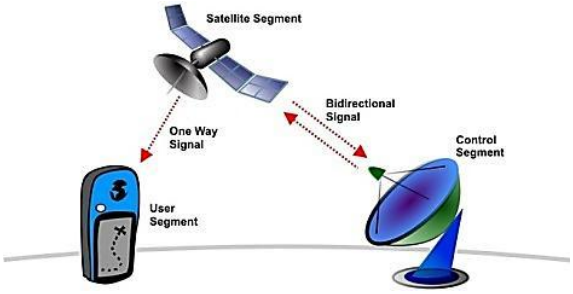

Fig. 4.13 - Representação dos segmentos do sistema GPS [31]

Segmento Espacial

O segmento espacial do sistema é constituído por um grupo de 24 satélites, mais três para o caso de existir uma falha em algum dos anteriores, dispersos por 6 órbitas sendo que cada uma delas dispõe de um total de 4 satélites. Cada um desses satélites está a orbitar a cerca de 20000 Km da superfície terrestre e o seu plano orbital tem um grau de inclinação em relação ao equador de 55 graus. Cada um dos satélites demora aproximadamente 12 horas a percorrer a sua órbita. Esta forma de arranjo permite que cada ponto da superfície terrestre esteja, a qualquer instante, a ser visto por quatro satélites.

São duas as frequências em que os satélites emitem sinais, uma designada de Link 1 a 1575.42MHz e outra designada Link 2 a 1227.67MHz, utilizando a técnica CDMA, *Code Division Multiple Access* e a modulação BPSK. O nível mínimo do sinal recebido na superfície terrestre é aproximadamente de -160dBW, sendo o máximo de -153dBW.

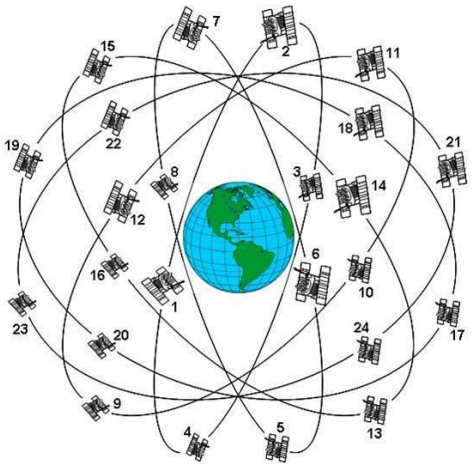

Fig. 4.14 - Segmento espacial do sistema GPS [32]

Segmento de Controlo

O segmento de controlo consiste numa estação de controlo principal situada no estado do Colorado e em várias estações de monotorização espalhadas pela superfície terrestre na vizinhança do equador.

Este segmento tem como principais funções monitorar os relógios dos satélites e prever o seu comportamento, obter informação acerca da órbita de cada satélite, saber o estado em que se encontra o satélite em termos de funcionamento, entre outras funções. Se necessário, a estação de controlo envia para o satélite dados de correcção para que este não saia da sua rota e mantenha o seu relógio actualizado.

A constante monotorização por parte do segmento de controlo tem como objectivo que o nível de precisão do sistema não se degrade com o passar do tempo ou pela ocorrência de factores externos.

Na Fig. 4.15 é possível visualizar a localização das várias estações do segmento de controlo do sistema GPS.

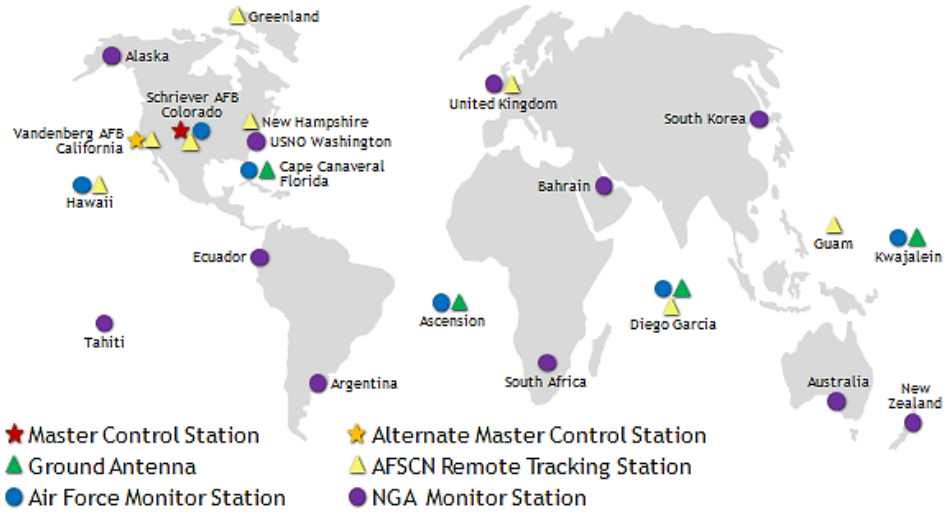

Fig. 4.15 - Sistema de controlo do GPS [33]

Segmento de utilizador

Este é o segmento onde se encontra presente o receptor GPS a que qualquer utilizador pode ter acesso. É aqui que é feita a recepção dos sinais enviados constantemente pelos satélites e, após o seu processamento, é indicada a localização em coordenadas e em altura assim como a velocidade.

O uso de receptores GPS está hoje presente nas mais variadas áreas. Qualquer meio de transporte aéreo ou marítimo está munido de receptores de GPS e, hoje em dia, é cada vez mais usual a sua inclusão em automóveis assim como em dispositivos móveis de comunicação, como telemóveis ou *smartphones*. Além disso, a disponibilidade de receptores é grande no mercado actual. Há ainda a possibilidade de utilizar o sistema GPS como base temporal o que, em sistemas que necessitam de funcionar com grande precisão de sincronismo, se torna numa ferramenta de grande importância.

Fig. 4.16 - Exemplo de receptor GPS comercial [34]

4.3.1.2 Obtenção da Localização

A obtenção de localização por parte de um receptor de GPS baseia-se na informação que este recebe vinda de satélites. O princípio é simples, o satélite envia um sinal a dizer que satélite é, em que posição se encontra e quando foi enviado o sinal. O receptor recebe o sinal com estas informações e utiliza-o para perceber onde se encontra, mas vamos por partes.

Inicialmente é recebida no receptor informação vinda de um satélite que diz onde está e quando foi enviada a mensagem. Fazendo a diferença entre quando a mensagem foi enviada e quando foi recebida é possível ao receptor saber a distância que o separa do satélite. Considerando essa distância como sendo o raio de uma esfera, o receptor pode estar em qualquer um dos pontos dessa superfície esférica.

De seguida, este processo é repetido para um segundo satélite. Mais uma vez é calculada a distância do satélite ao receptor e, mais uma vez, podemos imaginar uma esfera criada pelo raio igual a essa distância. Com mais esta informação, os pontos onde pode estar o nosso receptor são reduzidos. O receptor pode estar em qualquer um dos pontos em que estas duas esferas se interceptam, formando essa intercepção uma circunferência.

Por fim, é necessário perceber qual o ponto em que se encontra o receptor de todos os pontos presentes nessa circunferência. Para isso é utilizado um terceiro satélite. O mesmo processo feito para cada um dos satélites anteriores é repetido e, após o mesmo, sobram apenas dois pontos onde todas as esferas se intersectam. Basta agora perceber qual desses pontos é o correcto. Para esse efeito basta considerar a superfície terrestre como sendo uma quarta esfera e apenas um ponto é válido, estando assim encontrada a localização do receptor.

Olhando para todo o processo explicado anteriormente é fácil perceber que o aspecto fundamental para uma medição precisa é a perfeita medição entre o tempo que a mensagem é enviada pelo satélite e que é recebida pelo receptor. Para que esta medida seja exacta é necessário que os relógios dos dois dispositivos se encontrem sincronizados de forma perfeita.

Para que a sincronia entre o receptor e cada um dos satélites fosse perfeita era necessário que, tal como acontece nos satélites, o receptor possui-se um relógio atómico. Se tal acontece-se o preço que teria um receptor seria enorme comparado com o que tem hoje em dia. Para evitar que tal seja necessário é utilizado um quarto satélite para que o erro de relógio do receptor possa ser corrigido. Este facto permite que o receptor seja equipado com um relógio de *quartzo* normal sem que a precisão da sua medida seja afectada.

4.3.2 LOCALIZAÇÃO PELA REDE

A Google disponibiliza no seu sistema operativo Android a possibilidade de os utilizadores saberem a sua localização através das redes *Wi-Fi* ou da sua rede móvel.

Para ser possível dar informação sobre localização sem utilizar o sistema GPS a Google criou um método engenhoso aproveitando o facto da grande maioria dos dispositivos móveis actuais possuírem o sistema de localização GPS. Quando um dispositivo está a utilizar uma aplicação que requer o Google Maps o dispositivo tem necessariamente de estar conectado à Internet e, se a localização que se quer obter for com resultado preciso, deve estar também com o sistema GPS activo. Assim, o que o sistema criado pela Google faz é recolher os dados de ID da célula do operador móvel e, se existirem, as redes *Wi-Fi* a que o dispositivo está conectado, ao mesmo tempo recolhe os dados acerca da localização do dispositivo fornecidos pelo sistema GPS. Depois de recolhidos, estes dados são guardados numa base de dados num servidor da Google para puderem ser utilizados posteriormente.

Quando um dispositivo se encontra a utilizar a localização por via das redes móveis e de *Wi-Fi*, o que acontece é que o ID da célula de rede móvel e das redes *Wi-Fi*, se existirem, a que se encontra conectado são enviados para o servidor da Google. Depois de receber esses dados o servidor cruza-os com a base de dados existente e, comparando esses dados com os lá existentes, envia para o dispositivo a localização mais precisa que contém relativamente a estes. É facilmente perceptível que, para utilizar este serviço, o dispositivo móvel tem de estar conectado à *Internet*.

O método descrito anteriormente tem uma precisão variável e que depende muito do facto de o dispositivo ter acesso a uma rede *Wi-Fi* ou não. Se na busca pela localização apenas estiver presente o ID da célula de rede móvel, uma vez que uma célula contém um raio de distância considerável, o ponto dentro da circunferência criado por esse raio em que o dispositivo móvel se pode encontrar é bastante variável. Mas se a informação anterior for cruzada com a de uma rede *Wi-Fi* as possibilidades de localização diminuem substancialmente, uma vez que o raio de acção de uma rede *Wi-Fi* é bastante menor que o de uma célula de uma rede móvel.

A precisão do sistema anteriormente descrito e a sua globalização é tanto maior quanto maior for o número de dados presentes na base de dados da Google. Com o crescimento da utilização de dispositivos móveis utilizando o sistema operativo móvel Android, e com o grande uso de aplicações que utilizam a ferramenta Google Maps, é de esperar que o sistema esteja cada vez mais optimizado e com possibilidade de dar localização em qualquer ponto do Mundo.

5 Implementação

Neste capítulo irá ser apresentado o trabalho desenvolvido. A descrição será dividida em quatro partes. Inicialmente será descrita a aplicação desenvolvida para *smartphones* do sistema operativo móvel Android, de seguida o *site* desenvolvido para apresentar os dados através da *Internet*, a estrutura da base de dados criada e, por fim, a forma como é feita a comunicação entre as duas plataformas.

5.1 Aplicação Móvel

Tal como já referido anteriormente, a aplicação desenvolvida tem como objectivo fazer um mapeamento da energia electromagnética presente nas frequências a que um *smartphone* consegue ter acesso durante o seu normal funcionamento.

A aplicação é composta por um ecrã simples que contém três separadores, os quais podem ser vistos na Fig. 5.1. De seguida iremos perceber a informação presente em cada um dos ecrãs e as opções a que o utilizador tem acesso.

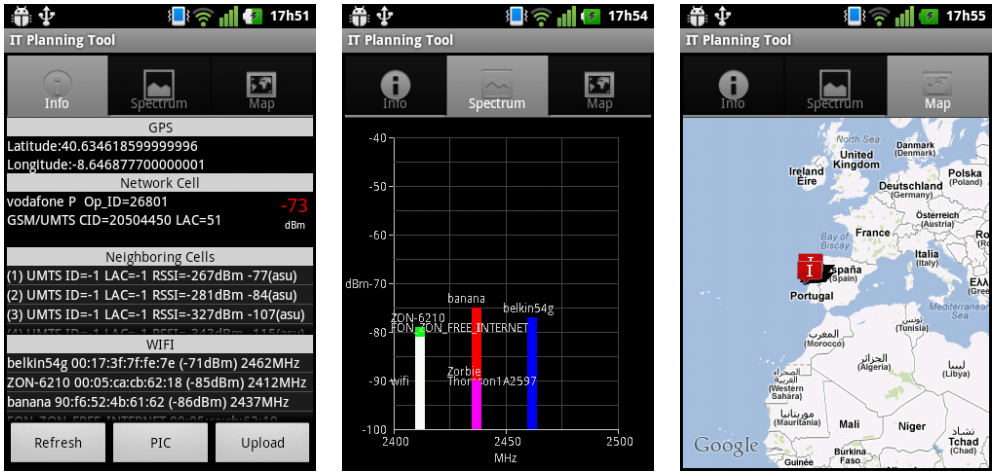

Fig. 5.1 - Separadores presentes na aplicação móvel

5.1.1 SEPARADOR INFO

Comecemos por dar atenção ao primeiro separador, visto que é neste que é contida a informação que pretendemos obter do local onde nos encontramos. Veja-se então a Fig. 5.2 onde este separador se encontra representado com maior pormenor.

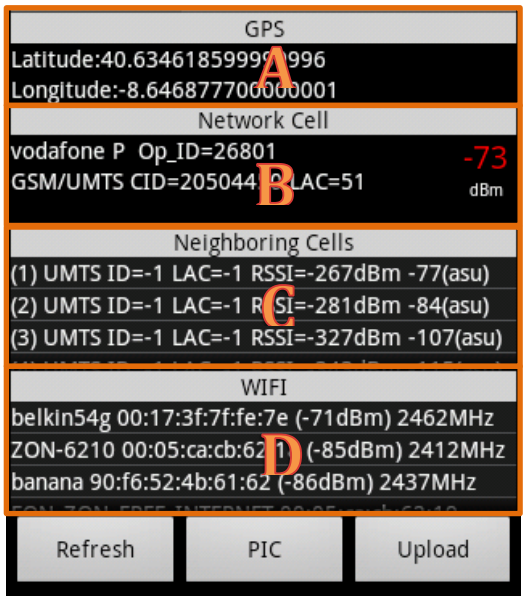

Fig. 5.2 - Separador Info

Desde logo é possível distinguir quatro áreas diferentes no ecrã, estando elas representadas pelas letras A, B, C e D. Em cada uma destas áreas está representado um grupo de informações.

A área representada pela letra A tem o título de GPS. Nesta área encontram-se as coordenadas em que o dispositivo se encontra no preciso momento em que a aplicação começou a ser executada. De forma a ser possível ter esta informação na maioria do tempo, visto que esta é de grande importância para o que se deseja obter no final, quando o GPS não se encontra activo no dispositivo, a sua localização é obtida através da informação da rede.

De seguida encontra-se a área representada pela letra B. Essa área tem o nome de *Network Cell*. Tal como o próprio nome indica, nesta zona está representada a informação relativa à célula de rede móvel a que o dispositivo se encontra conectado. Na primeira linha está presente a informação relativa ao operador, o seu nome e o seu ID. Na linha seguinte está a informação relativa à estação base. Por ordem temos o tipo de rede, o CID e o LAC. O CID é o ID da célula, sendo que este é um número único e que representa a célula, já o LAC, *Location Area Code*, é o código que identifica uma determinada célula em relação às células pertencentes ao operador. Por fim, do lado direito do ecrã, nessa mesma zona, é possível ver o valor de potência, em dBm, que o dispositivo está a receber vinda dessa estação base.

Na Fig. 5.2, para a zona B, podemos observar que o operador móvel do dispositivo é Vodafone P e que o seu ID é 26801. Em relação à célula em que este se encontra é possível saber que a estação base é do tipo GSM/UMTS, o seu CID é 20504450 e o seu LAC é 51. Por fim, observa-se que o dispositivo está a receber uma potência de -73dBm vindos da estação base.

Relativamente à informação acerca da rede móvel temos ainda a área representada pela letra C. Nesta zona encontram-se todas as células vizinhas que o dispositivo consegue detectar. Cada linha da lista representa uma dessas células e contém informação acerca da mesma. A informação apresentada é em tudo similar à que é visível no caso da *Network Cell*. Olhando da esquerda para a direita tem-se o tipo de rede da célula, o CID, a LAC e, por fim, a potência que o dispositivo consegue receber de cada uma dessas células. No exemplo em cima é possível observar a informação de 3 células vizinhas. O valor de -1 para o CID e a LAC refere que não é possível saber estes valores para a célula em questão.

Por fim temos a zona D. Olhando para a imagem facilmente se conclui que nesta zona se encontram todas as redes *Wi-Fi* a que é possível o dispositivo se conectar. A lista é apresentada de forma semelhante à anterior, mas como é natural a informação dada é diferente. Cada uma das linhas representa uma rede e, para cada rede, tem-se a seguinte informação olhando da esquerda para a direita, SSID, BSSID, potência e frequência.

O SSID, *Service Set Identifier*, é um conjunto de caracteres que identifica a rede *Wi-Fi*, já o BSSID representa o MAC address, *Media Access Control Address*, que é o identificador de um dispositivo que tenha capacidade para se conectar a uma rede, tal como um router, uma placa de rede de um computador, entre outros.

Olhando para a Fig. 5.2 podemos reparar que, no caso da primeira rede *Wi-Fi*, o SSID é belkin54g, o BSSID é 00:17:3f:7f:fe:7e, que a potência é de -71dBm e, por fim, que a frequência a que rede opera é de 2462 *MHz*.

Por fim, é possível observar que neste separador dispomos de três botões, sendo eles *Refresh, PIC* e *Upload*. Vejamos agora para que serve cada um deles.

Botão *Refresh*
Quando o botão *Refresh* é pressionado, a informação presente no ecrã é actualizada, ou seja, os dados referentes a cada uma das redes, móvel e *Wi-Fi*, sofrem uma actualização. Além desta actualização no ecrã, ao carregar neste botão a informação é guardada pela aplicação num ficheiro XML. Desta forma, o utilizador pode guardar várias leituras em vários locais. A estrutura desse ficheiro será vista mais à frente neste documento.

Botão PIC
Este botão dá a possibilidade ao utilizador de tirar uma fotografia. O seu objectivo é dar a possibilidade ao utilizador de retirar uma fotografia do sítio onde está a fazer uma leitura.

Botão *Upload*
O botão *Upload* tem a função de, quando premido, enviar todas as leituras feitas pelo utilizador até então para o servidor onde se encontra a base de dados do sistema, esta base de dados será apresentada mais à frente neste documento. A informação enviada é a previamente guardada no ficheiro XML aquando do pressionar no botão *Refresh* pelo utilizador. A forma como a informação é guardada no ficheiro XML e enviada para o servidor será descrita com pormenor na secção 5.4.1 do presente documento.

Ficam assim descritas todas as informações e funcionalidades presentes no separador Info e que se encontram visíveis ao utilizador.

Importa agora ver o que contém o menu que surge quando o utilizador prime o botão de opções presente em todos os dispositivos Android. Esse menu está representado na Fig. 5.3.

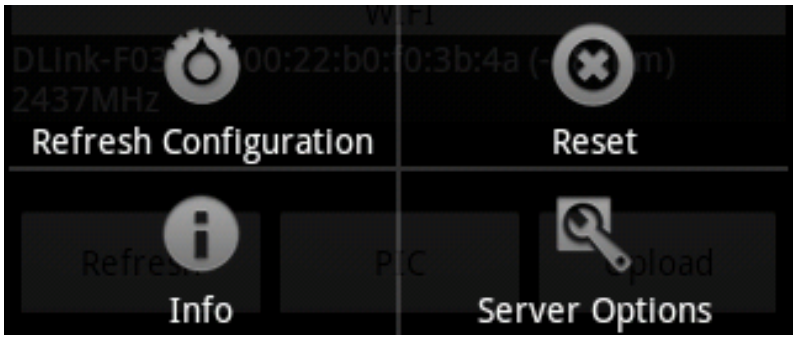

Fig. 5.3 - Menu do separador Info

É possível verificar que o menu contém quatro opções. Vejamos de seguida o que faz cada uma delas.

Refresh Configuration

O objectivo desta opção é dar a possibilidade ao utilizador de configurar uma actualização e recolha de informação periódica. Assim, é possível obter um maior número de leituras e de recolha de dados sem que o utilizador tenha de estar a utilizar a aplicação constantemente.

Depois de pressionada a opção *Refresh Configuration* a janela apresentada é a representada na Fig. 5.4.

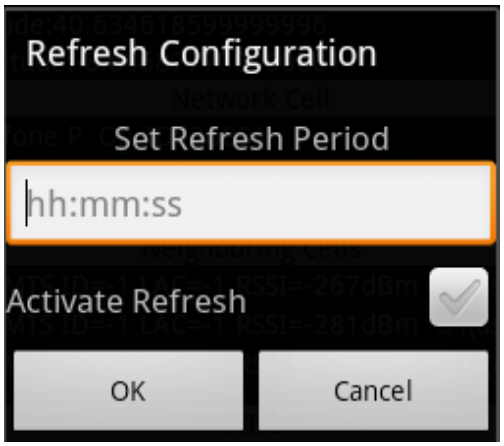

Fig. 5.4 - Janela de configuração do *refresh* periódico

Através desta janela é possível colocar o período de recolha de dados desejado. Este período é escrito na forma horas:minutos:segundos. Depois de definido um período o utilizador apenas necessita de assinalar a opção *Activate Refresh* e, enquanto esta opção não for desactivada, a cada intervalo definido é feita uma recolha de dados e guardada no ficheiro XML.

Para efectuar essa recolha de dados periodicamente é criada no dispositivo uma nova *Thread* que corre de forma independente da restante aplicação para que esta não precise de estar activa durante a recolha de informação. O código responsável pela criação da mesma é o presente na Fig. 5.5. De seguida segue uma explicação do mesmo.

```
Handler threadHandler = new Handler(){
  @Override
  public void handleMessage(Message msg){
    if(test_getUpdates){
      getUpdates(true, 5000, 20);
      telMng.listen(listener, PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
      telMng.listen(listener, PhoneStateListener.LISTEN_CELL_LOCATION);
    }
    getCellData();
    scanWifi();
    xmlCreater(info, wifi_result);
  }
};
Thread refreshThread = new Thread(){
  @Override
  public void run(){
    try {
      while(refresh_set){
        if(perio>60000){
          long temp = perio-30000;
          sleep(temp);
          threadHandler.sendEmptyMessage(0);
          sleep(30000);
          getUpdates(false, 0, 0);
          telMng.listen(listener, PhoneStateListener.LISTEN_NONE);
          test_getUpdates = true;
        }else{
          sleep(perio);
          threadHandler.sendEmptyMessage(0);
          test_getUpdates = false;
        }
      }
    } catch (Exception e) {

    }
  }
};
```

Fig. 5.5 - Código de criação de nova *Thread* para *Refresh* periódico

Inicialmente é criado um *Handler* que contém uma mensagem com as acções a realizar quando este for chamado. Essas acções são a obtenção da localização, a activação da detecção de mudanças nos sinais de rede móvel assim como todos os dados referentes a essa rede e às redes *Wi-Fi*. Na *Thread* propriamente dita, e mais especificamente no método *run()* que é onde se encontra o que é executado quando a *Thread* está em execução, tem-se como elemento principal um ciclo *while*. Este ciclo está ou não a ser executado consoante a variável *booleana refresh_set* é verdadeira ou falsa. Para essa variável ser verdadeira, o campo *Activate*

Refesh apresentado anteriormente tem de estar assinalado. Após a entrada no ciclo existem duas hipóteses, uma para quando o período de *refresh* é superior a 60 segundos e outra para quando é inferior. Esta divisão acontece porque se o período for inferior a 60 segundos considera-se que não existe a necessidade de estar constantemente a activar e desactivar a detecção na alteração da localização e da força dos sinais da rede móvel, ficando assim esse processo sempre activo após a primeira execução. Esse teste, de necessidade de reactivação das alterações de localização e sinal, é feito pela variável *booleana test_getUpdates*. Olhando agora com mais pormenor, no caso de o período ser superior a 60 segundos, a *Thread* começa por ser adormecida durante esse período menos 30 segundos, de seguida é chamada a mensagem presente no *Handler*, onde são activadas as detecções de alterações na localização de força do sinal e retiradas todas as informações que são guardadas no ficheiro XML através da função *xmlCreater*. Após isto, a *Thread* adormece os 30 segundos retirados anteriormente ao período definido após os quais são desactivadas as detecções de mudanças na localização e força do sinal. Por fim, é colocada a variável *test_getUpdates* com o valor *true* para que na próxima vez que a mensagem do *Handler* seja chamada esta saiba que é necessário reactivar a detecção de alterações nos sinais. No caso de o período ser menor a 60 segundos, a detecção de alterações é activada na primeira vez que a *Thread* é executada e não volta a ser desactivada, daí a colocação da variável *test_getUpdates* com o valor *false*.

Reset

Esta opção tem como função o eliminar de todas as leituras que foram realizadas até então e que ainda não foram enviadas para o servidor. Após um certo número de leituras, o utilizador pode decidir que não as pretende enviar para que constem na base de dados, assim esta opção ao ser premida elimina todos os dados recolhidos até então.

Info

A função desta opção é simplesmente abrir uma janela com informações ao utilizador, essa janela pode ser vista na Fig. 5.6.

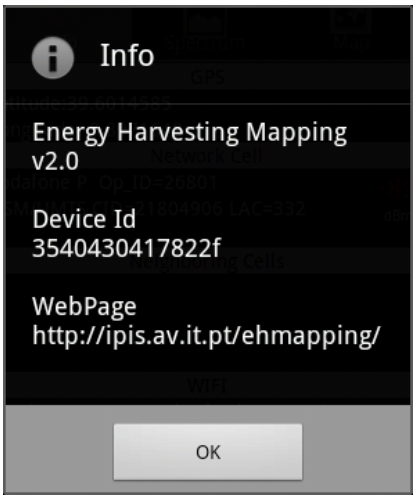

Fig. 5.6 - Janela de informação

A janela de informação indica a versão do *software*, o número de identificação do dispositivo dentro do sistema e a página *web* onde podem ser consultadas todas as leituras presentes na base de dados realizadas por todos os utilizadores.

Server Options

O objectivo desta opção é dar ao utilizador a possibilidade de alterar os dados que dizem respeito ao servidor para onde é enviada a informação. O objectivo da opção é evitar que uma mudança de servidor onde se encontra a base de dados não obrigue a que todos os utilizadores necessitem de voltar a instalar a aplicação. A janela que surge quando a opção é escolhida é a presente na Fig. 5.7.

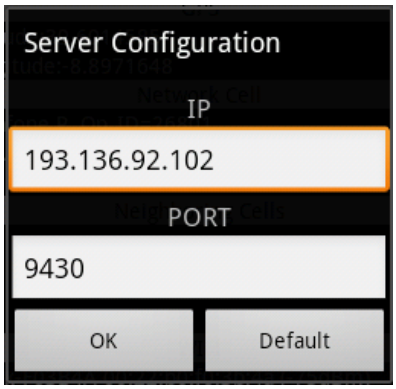

Fig. 5.7 - Janela de configuração das opções do servidor

No campo referente ao IP o utilizador pode inserir o valor de IP do servidor e, no campo seguinte, inserir o porto dentro desse servidor que é utilizado para receber os dados referentes à aplicação.

Para evitar que, depois de serem alterados os dados erradamente, o utilizador fique sem a possibilidade de enviar dados para o servidor, existe a opção *Default* que volta a configurar como dados do servidor os inicialmente definidos aquando da construção da aplicação.

5.1.2 SEPARADOR SPECTRUM

De seguida é apresentado o separador Spectrum, este separador encontra-se representando em pormenor na Fig. 5.8.

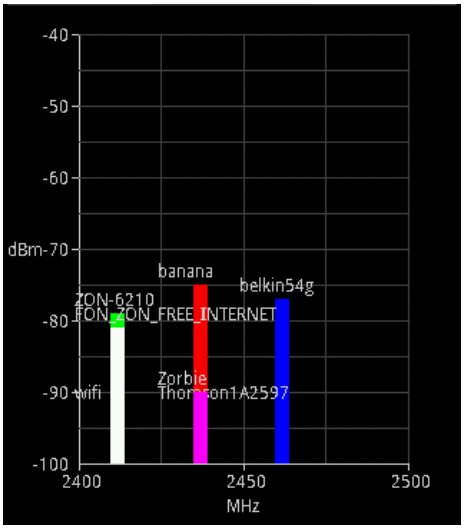

Fig. 5.8 - Separador Spectrum

Tal como o próprio nome indica, o objectivo neste separador é apresentar os valores num espectro de todas as redes *Wi-Fi* a que o dispositivo consegue ter acesso. Assim, é desenhado um gráfico de duas dimensões onde estão representados no eixo das abcissas os valores de frequência, em MHz, e no eixo das ordenadas o valor da potência, em dBm. O tipo de gráfico escolhido foi de barras devido ao facto de, na maioria das vezes, as redes *Wi-Fi* a que o dispositivo consegue ter acesso não serem de grande número ou até uma única.

O gráfico apresentado dá a possibilidade ao utilizador de ver com maior clareza as redes a que se pode conectar, em que frequências se encontram e a sua potência.

O gráfico apresentado é desenhado utilizando a classe *Canvas* disponibilizada pelo sistema operativo Android. Esta classe permite efectuar o desenho específico das formas pretendidas e a sua personalização quer em aspecto quer em animação. Um excerto de código que utiliza a classe *Canvas* encontra-se na Fig. 5.9.

```
private static class GraphView extends View{

  @Override
  protected void onDraw(Canvas c){
    super.onDraw(c);

    Paint p = new Paint();
    p.setColor(Color.LTGRAY);
    …
    c.drawLine(startX, startY, startX, endY, p);
    c.drawLine(startX, endY, endX, endY, p);
  }
}
```

Fig. 5.9 - Excerto de código para utilização de Canvas

No código apresentado é criada uma classe, *GraphView*, que é uma extensão de uma classe do tipo *View*. A classe do tipo *View* representa um bloco que permite a construção de componentes para a interface de utilizador. No método *onDraw()*, que recebe como parâmetro de entrada um objecto *Canvas*, estão representados os elementos que se pretendem desenhar na interface.

No caso apresentado é criado inicialmente um objecto do tipo *Paint*. Através deste objecto é possível definir as características visuais dos objectos que vão ser desenhados. No código acima é definido que a cor será cinzento claro. Por fim, são desenhadas duas linhas através do método *drawLine()* da classe *Canvas*. Este método possui cinco parâmetros de entrada. Os dois primeiros indicam o ponto onde se deve iniciar a linha, o primeiro representa a coordenada x e o segundo a coordenada y. O terceiro e quarto representam o ponto onde a linha termina, seguindo a mesma lógica do anterior. Por fim, o quinto elemento, é o objecto *Paint* onde estão definidas as características de aspecto para a linha.

5.1.3 Separador Map

Por último tem-se o separador Map que se encontra representado em pormenor na Fig. 5.10, do lado esquerdo da imagem, estando do lado direito a informação de uma leitura.

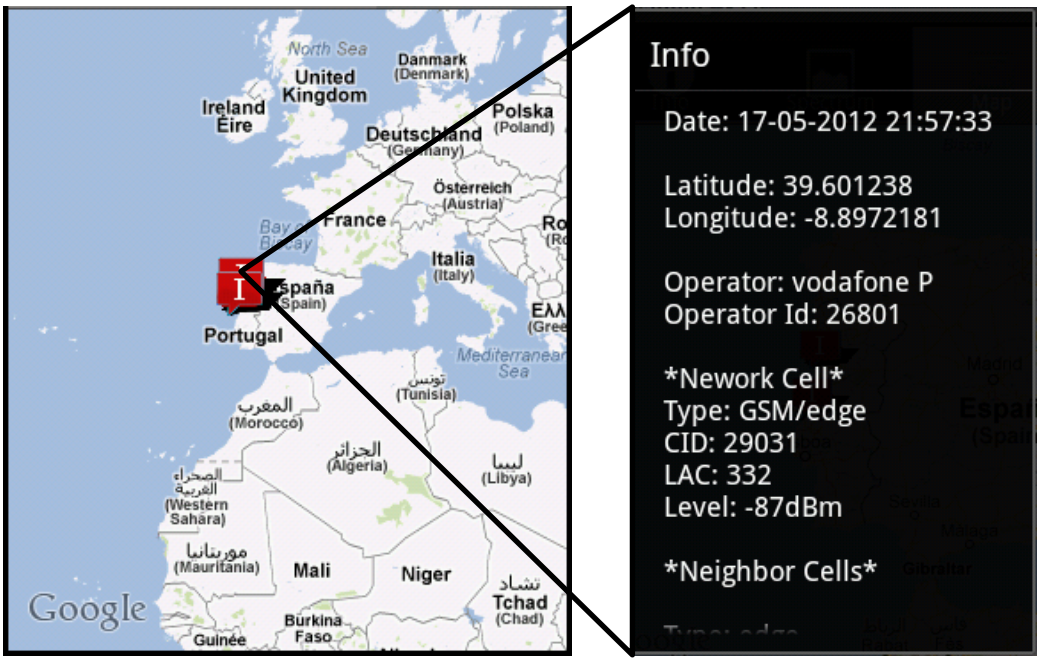

Fig. 5.10 - Separador Map e detalhe de informação de uma leitura

Tal como o próprio nome indica neste separador o objectivo é representar um mapa. Neste mapa estão todas as leituras efectuadas pelo utilizador e que se encontram na base de dados presente no servidor.

Depois de ter a possibilidade de ver todos os pontos onde foram efectuadas leituras, o utilizador tem ainda a possibilidade de ver todos os pormenores acerca de cada uma das leituras efectuadas por si. Para ter acesso a essa informação, basta ao utilizador carregar num dos pontos presentes no mapa para que apareça uma janela com todas as informações, essa janela encontra-se representada na Fig. 5.10 do lado direito.

Na janela que contém as informações o utilizador pode ver em que data e hora a leitura foi efectuada, as coordenadas em que se encontrava, os dados referentes ao operador que o dispositivo estava a utilizar, todos os dados da célula a que se encontrava conectado, das células de rede móvel de que conseguia captar sinal e, por último, todas as redes *Wi-Fi* a que tinha possibilidade de se conectar. Resumindo, o utilizador tem acesso a todos os dados que podia ver quando olhava para o separador Info na altura em que a leitura foi efectuada e guardada.

Para que seja possível a apresentação do mapa é necessária a importação de uma API externa ao sistema operativo Android de nome Google Maps Android. Através desta API é possível ter acesso a um conjunto de classes que nos permitem construir um mapa, colocar pontos sobre o mesmo entre outras opções.

5.2 Sítio na Internet

Para que todos os dados recolhidos pelos utilizadores da aplicação móvel estejam disponíveis e qualquer um os poder consultar foi criado um sítio na *Internet* onde estes são apresentados.

No sítio presente na *Internet* é possível ao utilizador ver uma pequena explicação acerca dos objectivos do projecto, uma apresentação da aplicação móvel e, a parte que mais interessa, o mapa onde se encontram todos os dados recolhidos e presentes na base de dados. Uma vez que o mapa é o que maior importância tem, apresenta-se de seguida essa informação ao pormenor.

Na Fig. 5.11 pode ver-se o mapa que é apresentado ao utilizador. Neste mapa estão contidos todos os pontos enviados até ao momento para a base de dados e é ainda possível ver apenas as leituras efectuadas por um determinado dispositivo, isto dá a possibilidade ao utilizador de ver e controlar as leituras que foram feitas por si sem ter de as procurar entre todas as outras.

Para efectuar a pesquisa das leituras efectuadas por determinado dispositivo existe um campo por baixo do mapa apresentado. Nesse campo, o utilizador pode inserir o ID correspondente ao dispositivo em questão, a informação referente ao ID está presente na aplicação móvel, tal como apresentado anteriormente. Após inserido o ID e premido o botão *Search* são apresentadas no mapa apenas as leituras referentes a esse dispositivo. Para voltar a ver todas as leituras, de todos os dispositivos, basta ao utilizador premir o botão que se encontra ao lado do botão *Search* e que tem a designação de *All Devices*.

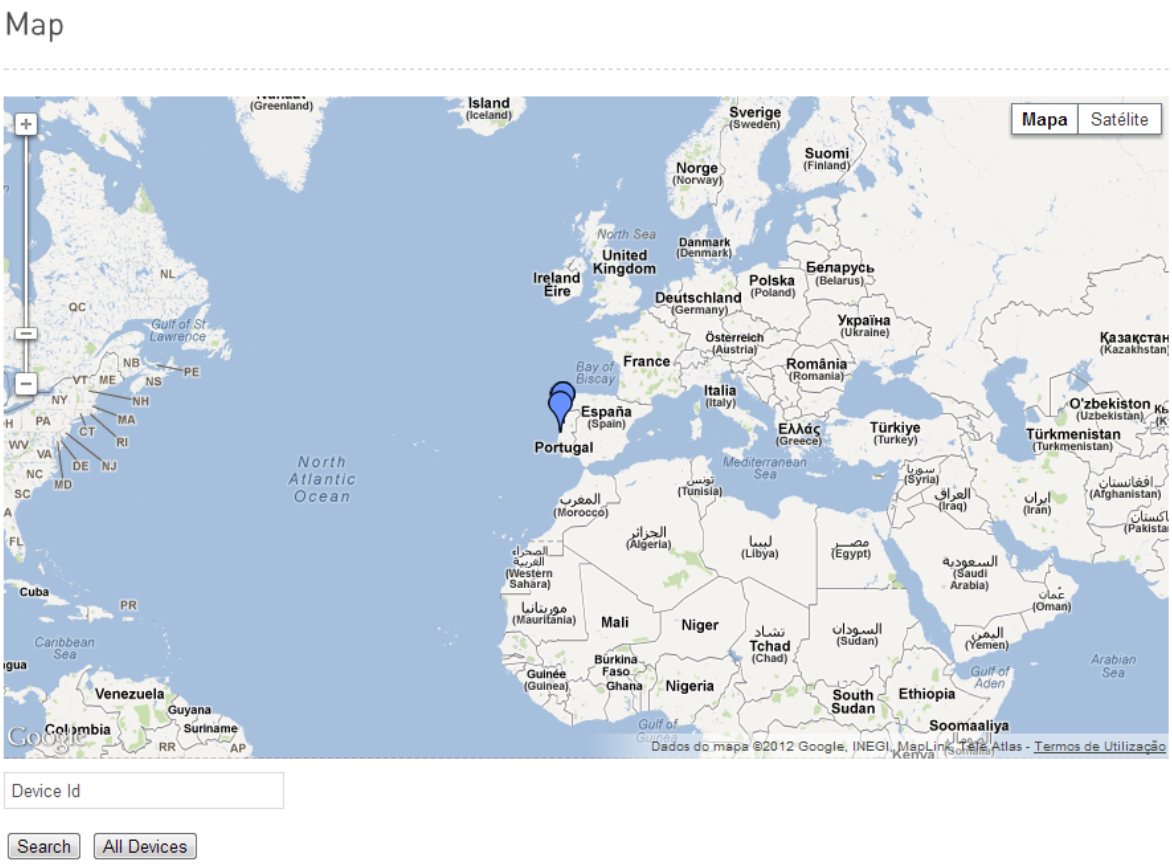

Fig. 5.11 - Mapa presente no sítio na *Internet*

Após ter acesso aos pontos presentes na base de dados e colocados no mapa, importa agora ter acesso à informação presente na base de dados relativa a cada um desses pontos.

Para ter acesso a toda a informação basta ao utilizador clicar em cima do ícone que marca o sítio no mapa onde foi efectuada a leitura. Após clicar em cima do ícone aparece ao utilizador um pequeno balão que contém alguma informação, essa representação pode ser vista na Fig. 5.12. Nesse balão está presente a informação da data e hora a que foi efectuada a leitura, a localização onde foi efectuada, através do valor de latitude e longitude, o tipo de localização utilizado, visto que, como visto anteriormente, esta tanto pode ser por GPS como através das redes a que o dispositivo tem acesso, o operador que fornece o serviço móvel ao dispositivo e por fim o ID do dispositivo que efectuou a leitura.

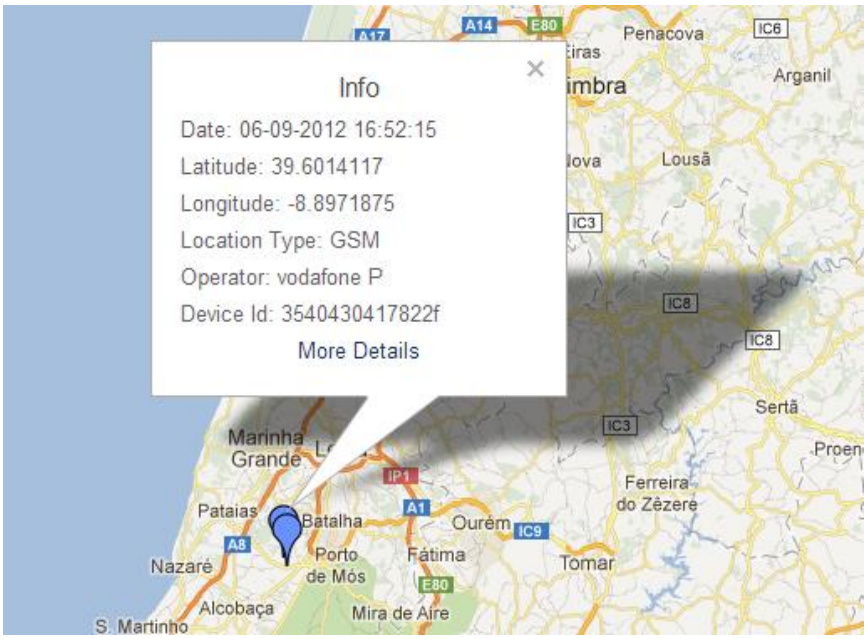

Fig. 5.12 - Balão de informações de uma leitura presente no mapa

No fim do balão que aparece ao se carregar num ícone está a expressão *More Details* que é uma hiperligação à qual o utilizador pode aceder. Ao aceder a essa hiperligação, o utilizador é direccionado para uma nova página. Essa nova página, representada na Fig. 5.14, tem como objectivo mostrar ao utilizador todos os detalhes acerca da leitura seleccionada.

Antes de passar à página que mostra os detalhes da leitura ao utilizador importa perceber como é feita a representação do mapa e a colocação nos pontos no mesmo. Esta operação é realizada através de um *script* escrito na linguagem de programação *JavaScript* e encontra-se representado um excerto do mesmo na Fig. 5.13.

No *script* existem duas funções, a primeira tem como função a adição de um novo ponto ao mapa, enquanto a segunda função tem como objectivo iniciar o mapa dentro do sítio da *Internet*. Para que seja possível realizar as acções pretendidas é necessário utilizar ferramentas presentes na API Google Maps. Assim, o ficheiro que contém todas as ferramentas necessárias é carregado antes de ser criado o *script* apresentado.

Para retirar da base de dados os pontos a adicionar ao mapa é utilizado um ficheiro escrito na linguagem PHP. Um ficheiro desse tipo será apresentado na secção 5.4.2.

```
var map = null;
var bounds = new google.maps.LatLngBounds();

function addMarker(lat, lng, info) {
   var pt = new google.maps.LatLng(lat, lng);
   bounds.extend(pt);
   var marker = new google.maps.Marker({
      position: pt,
      icon: icon,
      map: map
   });

   var popup = new google.maps.InfoWindow({
      content: info,
      maxWidth: 300
   });

   google.maps.event.addListener(marker, "click", function() {
      popup.open(map, marker);
   });

   google.maps.event.addListener(popup, "closeclick", function() {
      currentPopup.close();
   });
}

function initialize_map(){
   var latlng = new google.maps.LatLng(40.634, -8.647);
   var myOptions = {
      zoom: 4,
      center: latlng,
      zoomControl : true,
      streetViewControl : false,
      mapTypeControl: true,
      panControl: false,
      mapTypeId: google.maps.MapTypeId.ROADMAP,
   };

   map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
}
```

Fig. 5.13 - Excerto de código do *script* que coloca o mapa no sítio da *Internet*

Veja-se agora o conteúdo que está presente na página que tem como função mostrar ao utilizador os detalhes de uma determinada leitura, tal como referido anteriormente essa página encontra-se representada na Fig. 5.14.

Na página relativa aos detalhes o utilizador tem acesso a todos os dados recolhidos e de várias formas. Inicialmente a informação apresenta é a que já se encontrava presente no balão mostrado no mapa, mas além dessa informação é possível ao utilizador ver em maior pormenor a localização onde a leitura foi executada e, aplicando menor *zoom* pode mesmo

ver onde se encontra tendo em conta o espaço global sem a presença de mais nenhuma leitura.

Após essa informação está descrita toda a informação relativa às redes rádio. Esta informação está separada em duas colunas, na coluna da esquerda está presente a informação relativa à rede móvel do dispositivo e, na coluna da direita, encontra-se a informação relativa às redes *Wi-Fi*.

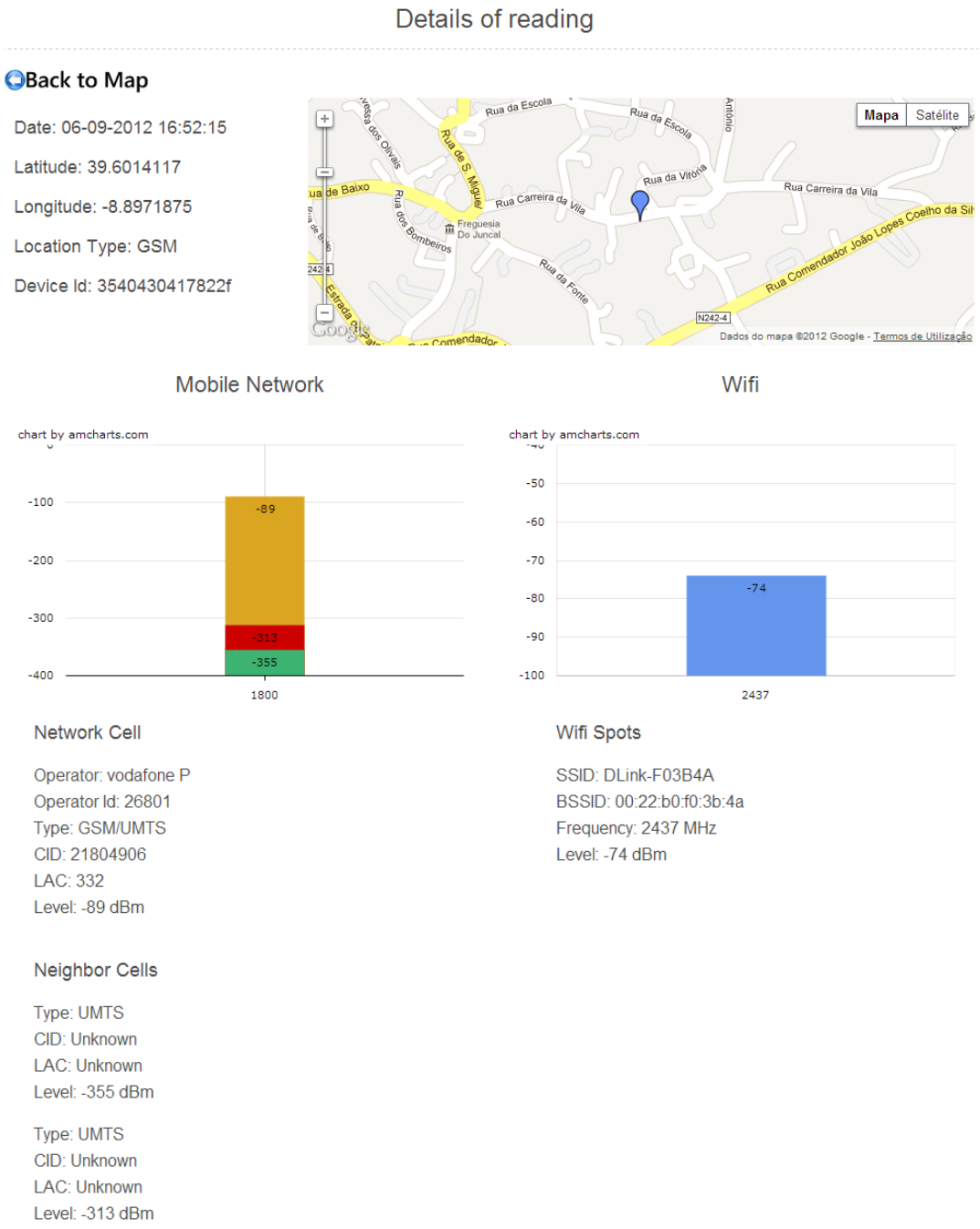

Fig. 5.14 - Detalhes de uma leitura

No caso da rede móvel começa por ser apresentada a informação relativa à célula a que se encontra conectado o dispositivo. A informação apresentada é aquela que era visível na

aplicação móvel quando foi guardada a leitura. Começa por ser apresentada a informação relativa ao operador móvel, nome e ID, seguindo-se a informação relativa à célula, o tipo de rede, o CID, o LAC e por fim o nível de potência recebida pelo dispositivo vinda da estação em causa. Seguidamente encontra-se a informação relativa a cada uma das células vizinhas a que o dispositivo consegue ter acesso. Para cada uma dessas células é apresentado o tipo de rede, o CID, o LAC e, por fim, o nível de potência. A informação relativa à potência recebida pelo dispositivo vinda de cada uma das células encontra-se resumida num gráfico de barras presente acima de todas as informações.

Em relação às redes *Wi-Fi* a informação é apresenta para cada uma em separado. Para cada uma das redes é apresentado o SSID, o BSSID, a frequência e o nível de potência a que o dispositivo consegue ter acesso no ponto em que se encontra. A informação relativa à potência de cada uma das redes encontra-se resumida num gráfico de barras acima da lista. Além de conter o valor da potência para cada uma das redes o gráfico agrupa ainda cada uma das redes tendo em conta a frequência em que estão a operar.

5.3 BASE DE DADOS

Para guardar todos os dados foi criada uma base de dados *MySQL* com a estrutura que é apresentada na Fig. 5.15. Como é possível observar a base de dados em questão é composta por três tabelas, scan_table, wifispots_table e neighborcells_table. Cada uma das tabelas segue um propósito e será detalhada de seguida.

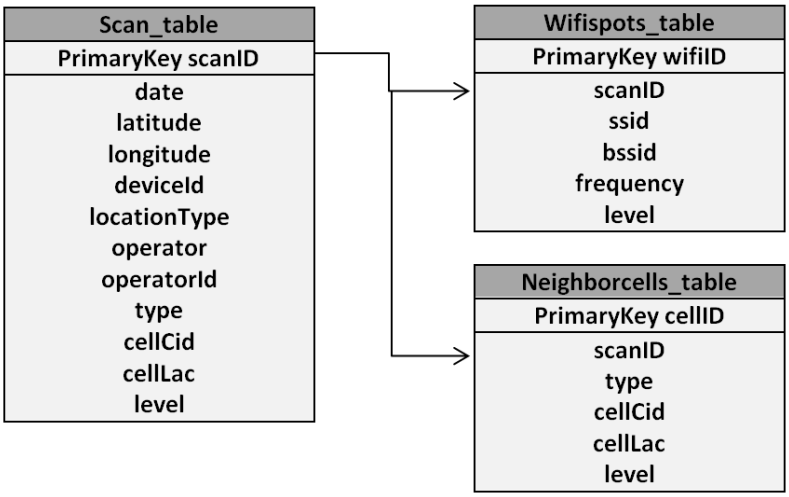

Fig. 5.15 - Estrutura da base de dados

A primeira tabela, scan_table, tem a função de guardar dados relativos à data da leitura, à localização onde foi efectuada e à célula de rede móvel a que o dispositivo se encontra conectado. Os campos presentes nesta tabela são:

- date – data e hora a que foi efectuada a leitura;
- latitude – valor da latitude em que foi efectuada a leitura;
- longitude – longitude em que foi efectuada a leitura;
- deviceId – ID do dispositivo que efectuou a leitura;
- locationType – tipo de localização utilizado para obter a latitude e longitude;

- operator – operador de serviço móvel utilizado pelo dispositivo;
- operatorId – ID do operador de serviço móvel utilizado pelo dispositivo;
- type – tipo de rede da célula a que o dispositivo se encontra conectado;
- cellCid – CID da célula a que o dispositivo se encontra conectado;
- cellLac – LAC da célula a que o dispositivo se encontra conectado;
- level – nível de potência a que o dispositivo tem acesso vindo da célula a que se encontra conectado.

A segunda tabela, wifispots_table, tem como propósito guardar os dados que dizem respeito às redes *Wi-Fi*. Os campos presentes na tabela são:

- scanID – ID que identifica a que leitura pertence a rede;
- ssid – SSID da rede *Wi-Fi*;
- bssid – BSSID da rede *Wi-Fi*;
- frequency – frequência a que está a operar a rede *Wi-Fi*;
- level – nível de potência.

Por fim, a tabela neighborcells_table tem como função guardar os dados que se referem às células vizinhas de cada leitura. Os seus campos são os seguintes:

- scanID – ID que identifica a que leitura pertence a célula vizinha;
- type – tipo de rede da célula vizinha;
- cellCid – CID da célula vizinha;
- cellLac – LAC da célula vizinha;
- level – nível de potência.

5.4 COMUNICAÇÃO ENTRE PLATAFORMAS

A presente secção tem como objectivo a explicação de como é feita a comunicação entre as duas plataformas apresentadas anteriormente, a aplicação móvel e o servidor. Irá começar por ser apresentado o envio de dados do dispositivo móvel para o servidor e, por fim, o acesso por parte do mesmo dispositivo à base de dados presente no servidor.

5.4.1 ENVIO DE DADOS PARA O SERVIDOR

O envio de dados por parte do dispositivo móvel para o servidor é feito através de um pacote UDP. Este envio de dados é feito, como já referido anteriormente aquando da apresentação da aplicação móvel, quando o utilizador pressiona o botão *Upload* presente no separador *Info*. O esquema deste envio está representado na Fig. 5.16.

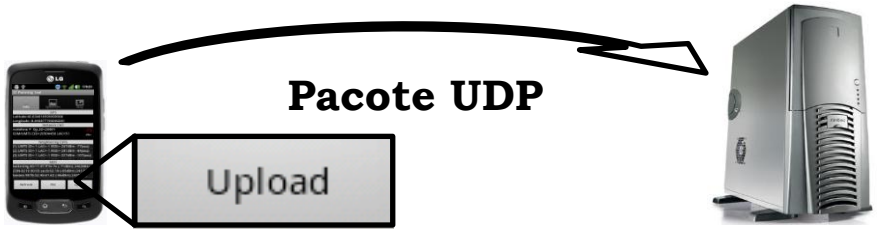

Fig. 5.16 - Envio de dados do dispositivo móvel para o servidor

Assim que o botão *Upload* é pressionado é chamado o código presente, de forma reduzida, na Fig. 5.17. Nesse código é criado o *DatagramSocket* que tem como função a implementação de um *socket* para o envio de um pacote UDP. A informação a ser enviada é então contida num *DatagramPacket* que mais não é do que uma classe que permite enviar e receber dados através de um *DatagramSocket*. Na construção dessa classe é possível ver que estão contidos quatro argumentos, *outData, outData.length, serverIP* e *port*. O primeiro argumento, *outData*, é o *array* de *bytes* onde se encontra a informação que irá ser enviada. O segundo argumento, *outData.length*, não é mais do que o tamanho do *array* anterior, no fundo o tamanho em número de *bytes* do pacote a ser enviado. Os dois argumentos seguintes dizem respeito a informação referente ao servidor para onde será enviada a informação. O primeiro, *serverIP*, representa o endereço IP para onde o pacote deve ser enviado e, o segundo, *port*, representa o número de porto dentro do servidor para onde o pacote deve ser encaminhado. Por fim, para enviar o pacote é chamado o método *send()* da classe *DatagramSocket* contendo um argumento que se trata do *DatagramPacket* definido.

```
public void sendUDPMessage() throws java.io.IOException {
  DatagramSocket socket = new DatagramSocket();
  InetAddress serverIP = InetAddress.getByName(ip);
  ...
  DatagramPacket out = new DatagramPacket(outData, outData.length, serverIP, port);
  socket.send(out);
  socket.close();
}
```

Fig. 5.17 - Excerto do código de envio da trama UDP pela aplicação móvel

Para que o envio seja efectuado de uma forma mais segura e fiável foi criado um protocolo a utilizar pela aplicação e que é reconhecido pela aplicação criada para receber os pacotes UDP no servidor. Assim, no campo para dados do pacote UDP, ou seja, no *array outData* apresentado anteriormente, é inserida uma trama de dados com a estrutura presente na Fig. 5.18.

Header 2 bytes	Length 2 bytes	ID 7 bytes	Data ...	End 2 bytes	Checksum 2 bytes

Fig. 5.18 - Estrutura da trama de envio de dados do dispositivo móvel para o servidor

Importa agora perceber qual a função que cada um dos campos representados na figura anterior desempenha.

O campo *Header*, composto por 2 bytes, tem como função sinalizar a que aplicação se destina a trama recebida. Este campo é necessário devido à coexistência de mais do que uma aplicação no servidor em que se encontra a base de dados onde são alojados os dados enviados. No caso presente, este campo tem o valor 0x21 0x21.

O campo *Length,* composto igualmente por 2 bytes, indica o tamanho da trama enviada.

O campo *ID*, que é composto por 7 bytes, tem como função o envio do ID do dispositivo que está a enviar a presente trama.

O campo *Data* é onde vai estar presente a informação que se pretende enviar. O tamanho deste campo é variável dependendo da quantidade de informação que o utilizador tenha armazenado durante a utilização da aplicação e que não tenha sido ainda enviada para o servidor.

O campo *End*, composto por 2 bytes, tem a função de sinalizar o fim do campo de dados. Como o campo anterior é de tamanho variável é importante no servidor perceber até onde deve ser extraída a informação que a dados diz respeito para que não sejam cometidos erros de tradução dessa mesma informação. Este campo é preenchido com os valores 0x0d 0x0a.

Por fim, o campo *Checksum*, composto igualmente por 2 bytes, tem como função o controlo de erros. Tal como descrito aquando da apresentação do protocolo UDP, este protocolo não é totalmente confiável devido a não ser orientado para a ligação. Tendo esse facto por base é acrescentado na trama a enviar o campo *Checksum* de forma a garantir a integridade dos dados enviados. O algoritmo utilizado é o CRC16. Este algoritmo é um código de detecção de erros desenhado para detectar alterações acidentais em canais de transmissão e é utilizado normalmente em redes digitais e dispositivos de armazenamento de dados.

Para organizar e sintetizar os dados a serem enviados é criado pela aplicação um ficheiro XML. Na Fig. 5.19 está presente um exemplo deste ficheiro. É o conteúdo desse ficheiro que é colocado no campo *Data* da trama apresentada anteriormente.

Detalhe-se agora a função de cada uma das *tags* presentes no ficheiro XML a enviar:

- *Info* – delimita a totalidade de informação enviada no ficheiro;
- *Scan* – contém todos os dados de uma leitura efectuada;
    - *Date* – data e hora a que foi efectuada a leitura;
    - *Latitude* – valor da latitude onde foi efectuada a leitura;
    - *Longitude* – valor da longitude onde foi efectuada a leitura;
    - *Locationtype* – tipo de localização utilizado;
    - *Device_id* – ID do dispositivo que efectuou a leitura;
    - *Network_Cell* – contém todos os dados da célula onde se encontrava o dispositivo conectado;
        - *Operator* – operador móvel utilizado no dispositivo;
        - *Operator_id* – ID do operador móvel utilizado no dispositivo;
        - *NetworkCelltype* – tipo de rede utilizado na célula;
        - *NetworkCellcid* – CID da célula;
        - *NetworkCelllac* – LAC da célula;
        - *NetworkCellSignalstrength* – nível de potência recebida no dispositivo vindo da célula;
    - *Neighbor_Cell* – contém todos os dados de uma célula vizinha;
        - *NeighborCelltype* – tipo de rede utilizado na célula vizinha;
        - *NeighborCellcid* – CID da célula vizinha;
        - *NeighborCelllac* – LAC da célula vizinha;
        - *NeighborCellSignalstrength* – nível de potência visto vindo da célula vizinha;
    - *Wifi_Spot* – contém todos os dados de uma rede *Wi-Fi;*

- *Ssid* – SSID da rede *Wi-Fi*;
- *Bssid* – BSSID da rede *Wi-Fi*;
- *Level* – Nível de potência recebido vindo da rede *Wi-Fi*;
- *Frequency* – frequência em que opera a rede *Wi-Fi*.

```
<?xml version='1.0' encoding='UTF-8'?>
<Info>
    <Scan>
        <Date>17-10-2012 11:53:38</Date>
        <Latitude>40.6343205</Latitude>
        <Longitude>GPS</Longitude>
        <Locationtype>GSM</Locationtype>
        <Device_id>3540430417822f</Device_id>
        <Network_Cell>
            <Operator>vodafone P</Operator>
            <Operator_id>26801</Operator_id>
            <NetworkCelltype>GSM/UMTS</NetworkCelltype>
            <NetworkCellcid>20734272</NetworkCellcid>
            <NetworkCelllac>51</NetworkCelllac>
            <NetworkCellSignalstrength>-
77</NetworkCellSignalstrength>
        </Network_Cell>
        <Neighbor_Cell>
            <NeighborCelltype>UMTS</NeighborCelltype>
            <NeighborCellcid>-1</NeighborCellcid>
            <NeighborCelllac>-1</NeighborCelllac>
            <NeighborCellSignalstrength>-
307</NeighborCellSignalstrength>
        </Neighbor_Cell>
        <Wifi_Spot>
            <Ssid>eduroam</Ssid>
            <Bssid>00:0f:f7:71:ad:20</Bssid>
            <Level>-90</Level>
            <Frequency>2462</Frequency>
        </Wifi_Spot>
        <Wifi_Spot>
            <Ssid>OpenEPCLTE</Ssid>
            <Bssid>c4:64:13:0c:7e:d1</Bssid>
            <Level>-80</Level>
            <Frequency>2437</Frequency>
        </Wifi_Spot>
    </Scan>
</Info>
```

Fig. 5.19 - Exemplo de ficheiro XML a enviar para o servidor

Veja-se agora de que forma é recebida a trama no servidor e como é processada a informação presente no ficheiro XML apresentado anteriormente e, por fim, colocada a informação na base de dados. À semelhança do código desenvolvido para a aplicação móvel, também o código presente no servidor desenvolvido para executar as tarefas referidas anteriormente foi escrito na linguagem de programação Java.

Para executar a recepção do pacote UDP enviado pela aplicação móvel, o código desenvolvido é bastante semelhante com o que foi apresentado para o envio. Um excerto desse código encontra-se na Fig. 5.20.

```
public class Server {
  public static void main(String args[]) throws Exception{
    DatagramSocket socket = new DatagramSocket(port);
    …
    while(true){
      DatagramPacket packet = new DatagramPacket(data, data.length);
      socket.receive(packet);
      String dados = new String(packet.getData());
      …
    }
  }
}
```

Fig. 5.20 - Excerto do código de recepção do pacote UDP no servidor

Tal como é possível verificar, é novamente criado um *DatagramSocket*, com a diferença que na sua construção é referenciado qual o porto pelo qual irá ser recebido o pacote. É também criado, mais uma vez, um *DatagramPacket* que neste caso terá a função de conter o pacote recebido. Assim, na sua construção apenas estão especificados dois argumentos, *data* e *data.length*. O primeiro é o *array* de *bytes*, definido anteriormente, onde ficará a informação recebida e o segundo é o seu tamanho. Para a recepção dos dados é chamado o método *receive()* da classe *DatagramSocket* onde é colocado um argumento que é o *DatagramPacket* onde a informação recebida deve ficar armazenada. Por fim é recolhida essa informação do *DatagramPacket* através do método *getData()*.

Após recolhida a informação e feita a sua descodificação para criar um ficheiro XML igual ao que se encontrava no dispositivo móvel, é necessário descodificar, retirar e guardar a informação presente nesse ficheiro. O primeiro passo para concretizar esse objectivo foi a criação de quatro classes que representam cada uma das informações recolhidas pela aplicação. Foi então criada a classe *Scan*, *NetworkCell*, *NeighborCell* e *WifiSpot*, estas encontram-se representadas na Fig. 5.21. Tal como o próprio nome indica, as três últimas têm o objectivo de guardar todos os dados referentes à célula a que o dispositivo se encontra conectado, a uma célula vizinha que está ao seu alcance e a uma rede *Wi-Fi* a que o dispositivo consegue aceder, respectivamente. Os dados a guardar por cada uma são os já referidos anteriormente para cada uma dessas estruturas. Por fim, tem-se a classe *Scan* onde estão contidos todos os dados referentes a uma leitura. Nesta classe está, além da informação referente à data, localização e que dispositivo fez a leitura, a célula a que este estava conectado e duas listas, uma contendo todas as células vizinhas e outra contendo todas as redes *Wi-Fi* da respectiva leitura. A utilidade desta estrutura irá ser vista já de seguida.

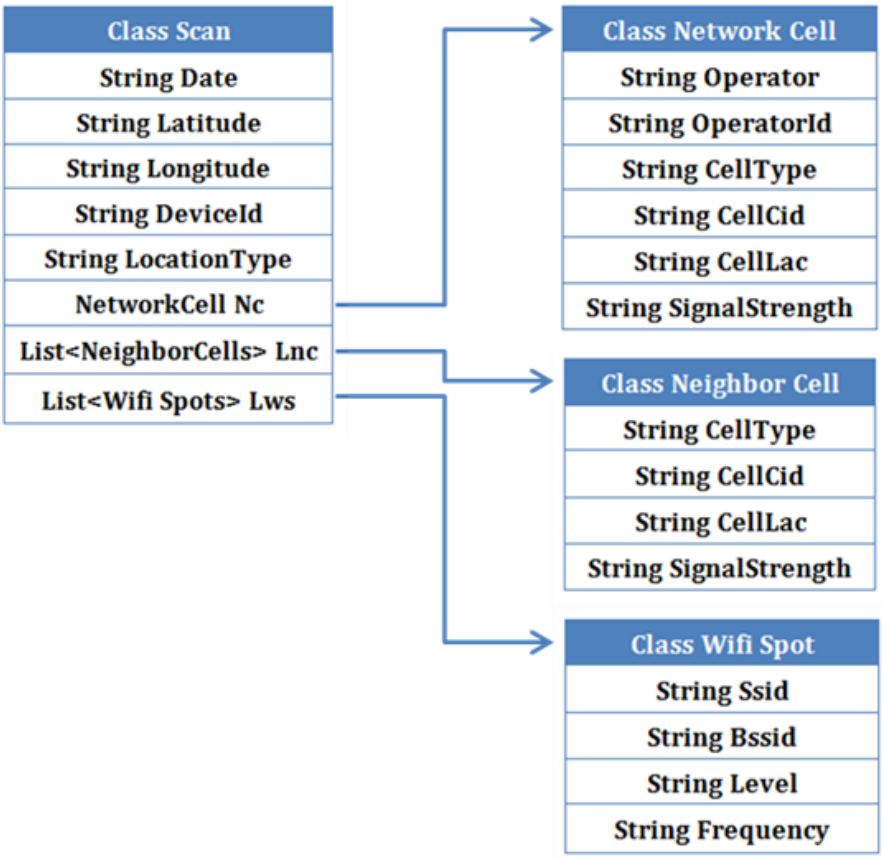

Fig. 5.21 - Estrutura das classes *Scan*, *NetworkCell*, *NeighborCell* e *WifiSpot*

É agora necessário efectuar o *parse* do ficheiro. Para esse efeito é criada uma nova classe de nome *XmlParser*. Um excerto do código desenvolvido nessa classe está presente na Fig. 5.22. De seguida, é detalhado o funcionamento do mesmo.

Como é possível verificar através do código apresentado, a classe *XmlParser* é uma extensão da classe *DefaultHandler*. Esta classe permite-nos fazer o *parse* de um documento XML através da detecção do início e fim do espaço delimitado por uma certa *tag*. É possível ainda ver que estão implementados quatro métodos nessa mesma classe, métodos esses que serão explicados detalhadamente de seguida e que são as ferramentas que nos permitem extrair a informação do documento XML apresentado anteriormente.

O primeiro método, *parseDocument()*, é aquele que é chamado pela classe principal quando se pretende fazer o *parse* do documento XML e extrair a informação que nele está contida. Neste método começa-se por criar uma instância da classe *SAXParserFactory*. Esta classe permite configurar um SAX *parser* para fazer o parse de documentos XML. De seguida, é chamado o método *newSAXParser()* dessa mesma classe. Este método cria um *SAXParser*, classe que vai fazer o *parse* ao documento quando chamado o seu método *parse*(). Neste método são passados dois argumentos. O primeiro indica o caminho dentro do disco rígido do servidor onde se encontra o ficheiro a processar e o segundo o *DefaultHandler* a utilizar que, no nosso caso, é o próprio onde o método está a ser chamado.

```
public class XmlParser extends DefaultHandler{
  public void parseDocument(){
    SAXParserFactory spf = SAXParserFactory.newInstance();
    try{
      SAXParser sp = spf.newSAXParser();
      sp.parse(path, this);
    }catch(SAXException se){
      se.printStackTrace();
    }catch(ParserConfigurationException pce){
      pce.printStackTrace();
    }catch(IOException ie){
      ie.printStackTrace();
    }
  }
  @Override
  public void startElement(String uri, String localName, String qName, Attributes
attributes) throws SAXException{
    tempVal = "";
    if(qName.equalsIgnoreCase("Scan")){
      tempScan = new Scan();
    }else if(qName.equalsIgnoreCase("Network_Cell")){
      tempCell = new NetworkCell();
    }…
    …
  }
  @Override
  public void characters(char[] ch, int start, int length) throws SAXException{
    tempVal = new String(ch, start, length);
  }
  @Override
  public void endElement(String uri, String localName, String qName) throws
SAXException{
    if(qName.equalsIgnoreCase("Scan")){
      scanlist.add(tempScan);
    }else if(qName.equalsIgnoreCase("Date")){
      tempScan.setDate(tempVal);
    }…
    …
  }
}
```

Fig. 5.22 - Excerto do código para realizar o *parse* do documento XML no servidor

Os restantes três métodos são utilizados para definir como é feito o *parse* do documento. Comecemos pelo método *startElement()*. Este método recebe notificação do início de uma nova *tag*. É assim possível definir dentro do método o que fazer cada vez que uma determinada *tag* é detectada. São quatro os elementos de entrada do método, sendo o mais

importante no nosso caso específico o terceiro, uma vez que é este que representa o nome da *tag* detectada. Dentro do método é então construída uma estrutura do tipo *if…else* onde é decidido o que realizar consoante o início de *tag* encontrado. No nosso caso são utilizadas quatro hipóteses, estando duas representadas no código da Fig. 5.22. Quando surge o início de uma *tag Scan* é criado um novo objecto da classe *Scan*. O mesmo procedimento é feito para quando surge uma *tag Network_Cell, Neighbor_Cell* ou *Wifi_Spot*, sendo criado um objecto *NetworkCell*, *NeighborCell* ou *WifiSpot* respectivamente.

Passemos agora ao método *endElement()*. Este método apresenta semelhanças com o método anterior, mas neste caso é detectada a terminação de uma determinada *tag*. Mais uma vez é definida uma estrutura *if…else* onde se decide o que fazer cada vez que uma *tag* termina. Como forma de exemplo pode referir-se que, quando é encontrada a terminação da *tag* <*Date*> é definido o atributo *Date* do objecto *Scan* com o seu valor, através do método *setDate()*. Este procedimento é efectuado para todos os restantes atributos do objecto *Scan* e dos restantes objectos, sendo que cada vez que é detecta o fim de uma *tag* <*Neighbor_Cell*> o objecto dessa classe é adicionado à lista presente na classe Scan, acontecendo o mesmo para o caso da *tag* <*Wifi_Spot*>. Por fim, detectando-se o fim de uma *tag Scan*, o objecto *Scan* é adicionado a uma lista que irá conter todas as leituras a colocar futuramente na base de dados.

O último método é o *characters()*. Este método é utilizado para recolher os caracteres que se encontram entre o início e o fim de cada *tag*, formando uma *String* com os mesmos. É assim através deste método que é recolhido o valor colocado em cada uma das *tags*.

Agora que já foi possível obter a informação presente no ficheiro XML falta o último passo, a sua colocação na base de dados presente no servidor. Para efectuar essa tarefa foi criada a classe *mysqlDatabase*, da qual se encontra um excerto de código na Fig. 5.23. A classe apresenta dois métodos, um que realiza a conexão com a base de dados, *databaseConnect()*, e outro que insere uma leitura nessa mesma base de dados, *insertScanInDB()*.

A conexão à base de dados, efectuada pelo primeiro método, é conseguida através de um *DriveManager* que utiliza o método *getConnection()*, que cria um objecto do tipo *Connection*, para o efeito. Neste método são inseridos três argumentos, todos do tipo *String*. O primeiro argumento é o URL para estabelecer conexão com a base de dados, o segundo o nome de utilizador que pretende aceder à mesma e, por fim, o terceiro é a palavra passe desse mesmo utilizador.

O segundo método, utilizado para colocar uma leitura na base de dados, recebe como argumento um objecto da classe *Scan*. Para proceder à inserção de uma nova linha na tabela é criado um objecto da classe *Statement*, através do método *createStatement()* da classe *Connection*. A classe *Statement* permite executar instruções SQL, o que nos permite inserir os dados na nossa base de dados MySQL. A execução da instrução é feita através do método *executeUpdate()*. É possível observar que é executada uma instrução para inserir dados na tabela referente aos dados da leitura e que, para inserir dados referentes às células vizinhas e redes *Wi-Fi,* é executada uma instrução por cada um dos objectos presentes nas listas pertencentes ao objecto *Scan* que está a ser processado.

```
public class mysqlDatabase {
  public void databaseConnect(){
    try {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      conn = DriverManager.getConnection(url+dbname, username, password);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void insertScanInDB(Scan s){
    try {
      Statement st = conn.createStatement();
      scanId = st.executeUpdate("INSERT INTO scan (date, latitude, longitude, deviceId,
locationType, operator, operatorId, type, cellCid, cellLac, level)
VALUES('"+date+"','"+latitude+"','"+longitude+"','"+deviceId+"','"+locationType+"','"+operato
r+"','"+operatorId+"','"+cellType+"','"+cellCid+"','"+cellLac+"','"+level+"')",
Statement.RETURN_GENERATED_KEYS);
      ResultSet keys = st.getGeneratedKeys();
      if (keys.next()) {
        scanId = keys.getInt(1);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    for(int i=0; i<celllist.size(); i++){
      try {
        Statement st1 = conn.createStatement();
        int val = st1.executeUpdate("INSERT INTO neighborcell (scanId, type, cellCid, cellLac,
level) VALUES('"+scanId+"','"+ncelltype+"','"+ncellCid+"','"+ncellLac+"','"+nlevel+"')");
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    for(int i=0; i<wifilist.size(); i++){
      try {
        Statement st = conn.createStatement();
        int val = st.executeUpdate("INSERT INTO wifispot(scanId, ssid, bssid, frequency,
level) VALUES('"+scanId+"','"+ssid+"','"+bssid+"','"+frequency+"','"+wlevel+"')");
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
}
```

Fig. 5.23 - Excerto de código para colocação de dados na base de dados no servidor

Terminado este processo, os dados ficam armazenados nas tabelas correspondentes da base de dados apresentada anteriormente. A tabela *scan* recebe uma linha com os dados de localização da leitura e da célula a que o dispositivo se encontra conectado. A tabela *neighborcell* recebe uma linha por cada uma das células vizinhas presentes no objecto scan, o mesmo se passando para a tabela *wifispot*, mas neste caso para cada uma das redes *Wi-Fi*. Para que seja definido o *scanId,* para cada uma das linhas das duas tabelas anteriores, é obtido o *id* da nova linha da tabela *scan* e colocado em cada uma das novas linhas das restantes tabelas.

Fica assim descrita a forma como é colocada a informação recolhida pela aplicação móvel na base de dados presente no servidor. De seguida será apresentada a forma como a aplicação móvel acede a essa mesma base de dados para apresentar informação ao utilizador.

5.4.2 Acesso à Base de Dados pela Aplicação Móvel

O acesso à base de dados presente no servidor por parte da aplicação móvel é feito através de um pedido HTTP. Esse acesso é feito cada vez que o utilizador acede ao separador Map com o objectivo de observar as leituras por ele realizadas, esse acesso está representado esquematicamente na Fig. 5.24.

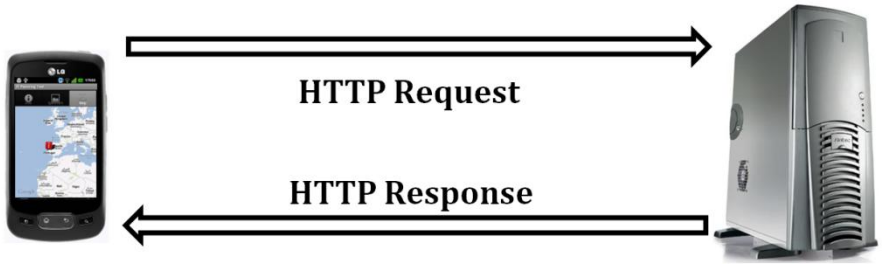

Fig. 5.24 - Esquema de pedido HTTP do dispositivo móvel e resposta do servidor

Assim que o utilizador acede ao separador Map é necessário que a aplicação obtenha todas as leituras que este efectuou. Para que tal aconteça é, tal como já referido, efectuado um pedido HTTP ao servidor. Uma parte do código utilizado para efectuar esse pedido encontra-se representado na Fig. 5.25, sendo detalhado seguidamente.

O primeiro passo é criar um *array* de objectos do tipo *NameValuePair*. Esta classe vai permitir a adição de dados na mensagem de pedido HTTP a enviar. No caso específico do código apresentado é enviado o ID do dispositivo que está a realizar o pedido. Para realizar o pedido propriamente dito é criado um objecto da classe *HttpClient*, esta classe cria um cliente HTTP encapsulando tudo o que é requerido para que se possa proceder ao pedido. Após ter um cliente criado são dados os passos necessários para lhe fornecer a informação necessária para o pedido ser correctamente executado. É assim criado um objecto *HttpPost* contendo o caminho para onde o pedido deve ser executado e os dados que se pretendem enviar na mensagem. Após tudo isto é chamado o método *execute()* da classe *HttpClient*. Este método executa o pedido HTTP e tem como retorno a resposta por parte do servidor a esse pedido, na forma de um objecto da classe *HttpResponse*, que mais não é que uma classe que representa uma resposta HTTP. Após obtida a resposta é agora necessário retirar a informação enviada

por parte do servidor. O primeiro passo é criar um objecto da classe *HttpEntity*, que representa a entidade contida na mensagem HTTP, através do método *getEntity()* da classe *HttpResponse*. Após obtida essa informação é-lhe retirado o conteúdo e, utilizando várias ferramentas, essa informação é passada para o formato de um *JSONArray* que nos vai permitir obter a informação pretendida utilizando a referência de cada coluna da tabela presente na base de dados.

```
ArrayList<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("idofdevice",id));
InputStream is = null;
try{
   HttpClient httpclient = new DefaultHttpClient();
   HttpPost httppost = new
HttpPost("http://ipis.av.it.pt/ehmapping/service/getScanToId.php");
   httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
   HttpResponse response = httpclient.execute(httppost);
   HttpEntity entity = response.getEntity();
   is = entity.getContent();
}catch(Exception e){
   Log.e("log_tag", "Error in http connection "+e.toString());
}
try{
   BufferedReader reader = new BufferedReader(new InputStreamReader(is,"iso-8859-
1"),8);
   StringBuilder sb = new StringBuilder();
   String line = null;
   while ((line = reader.readLine()) != null) {
      sb.append(line + "\n");
   }
   is.close();
   result=sb.toString();
}catch(Exception e){
   Log.e("log_tag", "Error converting result "+e.toString());
}
try{
   JSONArray jArray = new JSONArray(result);
}catch(JSONException e){
   Log.e("log_tag", "Error parsing data "+e.toString());
}
```

Fig. 5.25 - Excerto de código presente na aplicação móvel para realizar pedido http

O procedimento realizado anteriormente é repetido para o caso das tabelas onde se encontra a informação relativa às células vizinhas de rede móvel e às redes *Wi-Fi*. Nesse caso, a informação enviada através do pedido HTTP é o *scanID*.

Após toda a informação ser obtida é organizada para que se coloquem os pontos no mapa a apresentar ao utilizador da aplicação móvel, assim como a informação existente para cada um dos pontos.

É importante de seguida perceber como é obtida a informação pelo pedido HTTP. Dessa forma, apresenta-se na Fig. 5.26 o ficheiro para o qual é feito esse pedido.

```
<?php
   $dbcnx = mysql_connect ("$dbserver", "$dbuser", "$dbpass");
   mysql_select_db("$dbname") or die(mysql_error());
   $q = mysql_query("SELECT * FROM scan WHERE deviceId LIKE
'%".$_REQUEST['idofdevice']."%'");

   while($e=mysql_fetch_assoc($q))
      $output[]=$e;

   print(json_encode($output));

   mysql_close();
?>
```

Fig. 5.26 - Código presente no servidor que retira os dados presentes na tabela *Scan* da base de dados para um determina dispositivo

O ficheiro apresentado é escrito na linguagem PHP e está presente no servidor onde se encontra a base de dados. É para esse ficheiro que é feito o pedido HTTP e é o caminho para o mesmo que é colocado no objecto *HTTPPost* apresentado anteriormente.

No ficheiro apresentado começa-se por realizar uma conexão à base de dados. Inicialmente é feita a conexão ao servidor onde se encontra através do método *mysql_connect()* após a qual é seleccionada a base de dados dentro do servidor a que nos queremos conectar através do método *mysql_select_db()*. Realizada a ligação é criada uma instrução SQL que irá permitir retirar da tabela desejada as leituras referentes ao ID de utilizador que se encontrava na mensagem HTTP recebida. Depois de todas essas leituras estarem retiradas são colocadas num *array* e codificadas numa *String* com representação JSON. Ao colocar a informação nesta codificação está-se a permitir que mais tarde esta seja passada para um *JSONArray* na aplicação móvel, tal como foi visto anteriormente.

O mesmo tipo de ficheiro apresentado anteriormente para o caso da tabela *scan* está também presente no servidor para o caso das tabelas *neighborcell* e *wifiSpot*. São esses ficheiros que são utilizados para retirar dessas tabelas os elementos referentes ao *scanID* enviado pela aplicação móvel.

Fica assim terminado o processo de obtenção por parte da aplicação móvel de dados provenientes da base de dados presente no servidor. Para melhor visualizar o que realmente acontece estão representadas na Fig. 5.27 e Fig. 5.28 capturas do pedido e resposta HTTP. É possível observar os endereços IP para onde são enviadas ou recebidas as mensagens assim como o conteúdo de cada uma delas. Na primeira imagem é possível verificar o ID do

dispositivo enviado e, na segunda imagem, é possível observar que é enviado para esse dispositivo a informação contida na base de dados.

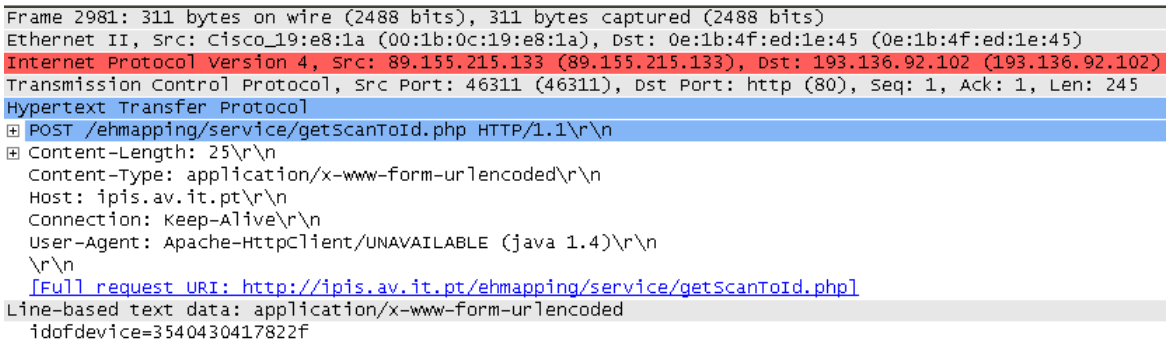

Fig. 5.27 - Captura de pedido HTTP recebido no servidor

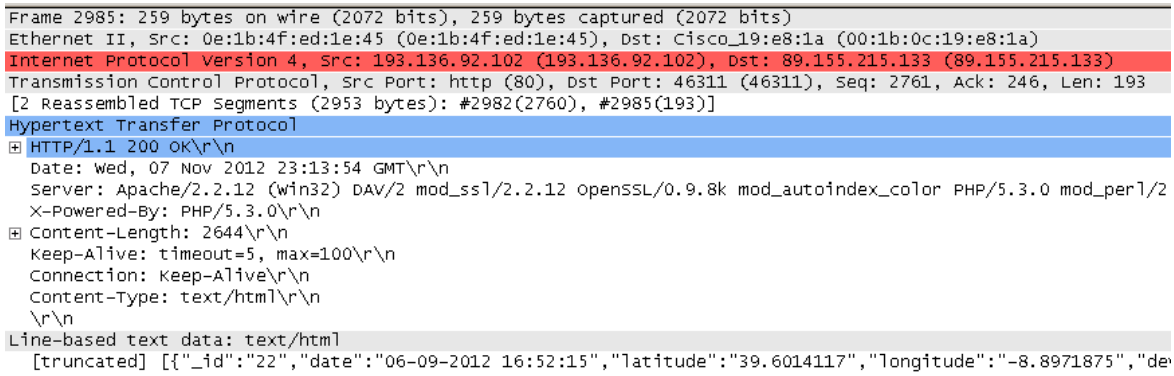

Fig. 5.28 - Captura de envio de resposta HTTP por parte do servidor

6 Conclusão e Trabalho Futuro

Serve este capítulo para fazer um balanço de todo o trabalho realizado e apresentado neste documento. Importa perceber quais os objectivos iniciais e se os mesmos foram cumpridos e quais foram alterados durante a realização do trabalho. Serve ainda para olhar para o trabalho que pode vir a ser desenvolvido no futuro de forma a melhorar o que foi apresentado.

6.1 Conclusão

Tendo em conta os objectivos iniciais e o resultado final pode afirmar-se que na generalidade foram cumpridos. No que ao projecto diz respeito foi cumprido o objectivo de criar uma aplicação para dispositivos móveis, que estivesse ao alcance de qualquer utilizador que a pretende-se utilizar, que permitisse a recolha de informação acerca da energia presente em cada uma das frequências a que esse tipo de dispositivos tem acesso.

Ao contrário do que era objectivo inicial, a informação recolhida pela aplicação não é colocada na base de dados que já existia. Após se perceber durante o desenvolvimento da aplicação móvel que a informação recolhida continha bastantes diferenças de a obtida por um aparelho existente para esse fim e que era possível obter mais dados que podiam ser uma mais-valia, foi decidida a criação de uma nova base de dados que contém apenas as leituras efectuadas pelos dispositivos móveis. Além da base de dados foi ainda criado um novo sítio na *Internet* para conter apenas as leituras efectuadas pela aplicação de forma a que as duas informações não se confundam.

Em termos de aprendizagem o objectivo inicial foi cumprido, a aprendizagem de desenvolvimento de aplicações móveis para o sistema operativo móvel Android. O desenvolvimento da aplicação obrigou à utilização de um variado leque de ferramentas o que levou à criação de um vasto conhecimento sobre a construção de uma aplicação e que possibilitou criar as bases para criar aplicações de outra natureza e complexidade futuramente. Além do conhecimento no sistema operativo móvel Android e na construção de aplicações para o mesmo, foi ainda obtido conhecimento sobre o funcionamento, criação e

gestão de uma base de dados e ainda de linguagens de programação utilizadas para criar sítios para a *Internet*, tais como, PHP, HTML, CSS e JavaScript.

6.2 TRABALHO FUTURO

Como trabalho futuro pode ser encarada a possibilidade de estender a aplicação a outros sistemas operativos móveis, tais como o *Windows Phone* ou o *iOS*, de forma a permitir que um maior número de utilizadores tenha acesso à mesma e possa contribuir para o aumento da informação presente na base de dados.

Por outro lado, no que ao sítio na *Internet* diz respeito pode ser encarado o aproveitamento do botão PIC para colocar na informação apresentada no mapa uma foto do local onde a informação foi recolhida. Esta funcionalidade foi pensada aquando da realização do trabalho, mas a sua implementação acabou por não ser desenvolvida.

Referências

1. Prusayon Nintanavongsa, et al., “Design Optimization and Implementation for RF Energy Harvesting Circuits”, in *IEEE Journal on emerging and selected topics in circuits and systems*, vol 2, Março, 2012.

2. A. Georgiadis, A. Collado, S. Via e C. Meneses, "Flexible Hybrid Solar/EM Energy Harvester for Autonomous Sensors", in *Proc*. *2011 IEEE MTT-S Intl*. *Microwave Symposium (IMS)*, Baltimore, US, Junho 5-10, 2011.

3. UTPA - The University of Texas-Pan American. (Visitado a 12 de Novembro de 2012). http://faculty.utpa.edu/hfoltz/papers.htm

4. D. Bouchouicha, F. Dupont, M. Latrach, e L. Ventura, “Ambient RF energy harvesting,” in *IEEE International Conference on Renewable Energies and Power Quality*, Março 2010.

5. Vijay Kumar, *A Brief History of Personal Communication System*.: University of Missouri, Kansas City.

6. Kenbak-1 Computer (Visitado a 9 de Agosto de 2012). http://www.kenbak-1.net/

7. Maximum PC – The Most 25 Important PCs in History. (Visitado a 9 de Agosto de 2012). http://www.maximumpc.com/files/imagecache/futureus_imagegallery_fullsize/gallery/kenbak-1_01_0.jpg

8. Old Computers – Osborne 1 computer. (Visitado a 9 de Agosto de 2012) http://oldcomputers.net/osborne.html

9. TekWik – Macintosh Computer Day. (Visitado a 10 de Agosto de 2012) http://tekwik.com/2012/01/macintosh-computer-day/

10. Tecnodrop – IBM Simon, o primeiro smartphone do mundo. (Visitado a 9 de Agosto de 2012). http://www.tecnodrop.com/2012/09/ibm-simon-o-primeiro-smartphone-do-mundo.html

11. HubPages - Android and the Open Source Revolution. (Visitado a 13 de Agosto de 2012). http://gamerelated.hubpages.com/hub/The-Future-of-Android

12. Realmdigital. (Visitado a 14 de Agosto de 2012) http://www.realmdigital.co.za/images/uploaded_images/android-logo.png

13. Manish Yadav, History of Android (Visitado a 13 de Agosto de 2012). http://www.tech2crack.com/history-android/

14. Buildall – Arquitectura do Google Android (Visitado a 14 de Agosto de 2012). http://buildall.wordpress.com/2009/11/15/arquitetura-do-google-android/

15. Stefan Brähler, "Analysis of the Android Architecture", 2010.

16. Application Fundamentals (Visitado a 15 de Agosto de 2012). http://eagle.phys.utk.edu/guidry/android/applicationFundamentals.html

17. Android – Application Fundamentals (Visitado a 16 de Agosto de 2012). http://www.linuxtopia.org/online_books/android/devguide/guide/topics/fundamentals.html

18. Android Developers – App Components (Visitado a 8 de Agosto de 2012). http://developer.android.com/guide/components/index.html

19. Michael Reilly, *Java network programming and distributed computing*.: Addison-Wesley, 2002, ch. 2.

20. Wikipédia - Cascading Style Sheets (Visitado a 28 de Julho de 2012). http://pt.wikipedia.org/wiki/Cascading_Style_Sheets

21. PHP – What is PHP? (Visitado a 1 de Setembro de 2012). http://pt1.php.net/manual/en/intro-whatis.php

22. PHP – Usage Stats for April 2007 (Visitado a 19 de Setembro de 2012). http://php.net/usage.php

23. W3C – XML Essentials (Visitado a 9 de Agosto de 2012). http://www.w3.org/standards/xml/core

24. iMasters – As principais características do XML (Visitado a 20 de Outubro de 2012). http://imasters.com.br/artigo/163/xml/as-principais-caracteristicas-do-xml

25. Wikipédia – Camada de Transporte (Visitado a 29 de Setembro de 2012). http://pt.wikipedia.org/wiki/Camada_de_transporte

26. SourceDaddy – The TCP/IP Protocol Framework (Visitado a 1 de Outubro de 2012). http://sourcedaddy.com/windows-xp/the-tcp-ip-protocol-framework.html

27. ServerFault – What is the difference between UDP and TCP? (Visitado a 2 de Outubro de 2012). http://serverfault.com/questions/8981/what-is-the-difference-between-udp-and-tcp

28. R. Fielding, et al. *HyperText Transfer Protocol – HTTP/1.1.* W3C/MIT, 1999.

29. etutorials – HyperText Transfer Protocol (HTTP and HTTPS) (Visitado a 4 de Outubro de 2012). http://etutorials.org/Programming/Pocket+pc+network+programming/Chapter+2.+WinInet/Hypertext+Transfer+Protocol+HTTP+and+HTTPS/

30. GPS History - How it all started (Visitado a 10 de Outubro de 2012). http://www.maps-gps-info.com/gps-history.html

31. Azosensores – Understanding the Global Positioning System (Visitado a 10 de Outubro de 2012). http://www.azosensors.com/article.aspx?ArticleId=29

32. Núcleo Infanto-Juvenil de Aviação – Navegação Aérea (Visitado a 14 de Outubro de 2012). http://ninja-brasil.blogspot.pt/2010/09/navegacao-aerea.html

33. GPS – Control Segment (Visitado a 14 de Outubro de 2012). http://www.gps.gov/systems/gps/control/

34. Tecnodrop – Navegador GPS Garmin Nuvi 885t com tela touchscreen (Visitado a 10 de Outubro de 2012). http://www.tecnodrop.com/2009/01/navegador-gps-garmin-nuvi-885t-com-tela-touchscreen.html

35. F. Kattan, Dynamic Cell-ID: Clever way to Block Google, but will it Backfire? (Visitado a 19 de Outubro de 2012). http://franciscokattan.com/2010/02/06/dynamic-cell-id-clever-way-to-block-google-but-will-it-backfire/

36. M. Fasel, A Good Look at Android Location Data (Visitado a 19 de Outubro de 2012). http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/