

mtAndroid

Aplicação Móvel Android de Apoio a Percursos Pedestres *Outdoor*

Ricardo José Andrade Lopes Perdigão e Silva

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Professor Doutor José Carlos Alves Pereira Monteiro

Orientador: Professor Doutor Alberto Manuel Rodrigues da Silva

Vogal: Professor Doutor Paulo Jorge Pires Ferreira

Março 2013

Agradecimentos

Os meus primeiros agradecimentos são destinados ao Professor Doutor Alberto Silva, pela oportunidade que me deu em abordar o tema desta dissertação, pelo apoio e orientação incondicionais, pela dedicação e excelência, e por toda a aprendizagem e conhecimento que me permitiu adquirir no decorrer deste trabalho.

Uma palavra de agradecimento para os colegas de Engenharia Informática, aqueles com quem tive a oportunidade de fazer trabalhos de grupo nas cadeiras do MEIC-Alameda, e sobretudo para os companheiros de sempre da LEIC98, pela amizade e pelo apoio durante este projeto.

Agradeço aos colegas e Professores do POSI-X, pelo conhecimento e pela experiência adquiridos nesta excelente pós-graduação do IST, que me permitiu renovar muitos conhecimentos nas áreas das Tis e dos SIs, e me ajudou a decidir em avançar para o Mestrado.

Obrigado a todos os amigos e amigas mais próximos, que me apoiaram e me motivaram durante os períodos mais difíceis.

Esta dissertação é dedicada à minha irmã (Cláudia), aos meus pais (Maria Fernanda e Reinaldo), e aos meus filhos (Leonor Maria e Rodrigo José).

Resumo

O número de utilizadores de *Smartphones* está a aumentar, e a criar oportunidades de desenvolvimento de inúmeras aplicações móveis, para as mais diversas áreas. A maior parte dos *Smartphones* vêm munidos de ligação à Internet, antena GSM/GPRS/3G/HSPDA/4G, sensores GPS, *Bluetooth*, Wi-Fi, e ainda outros componentes de *hardware* como os sensores magnetómetro, giroscópio e acelerómetro. Estes componentes facilitam a localização geográfica, o cálculo da posição geográfica, o fornecimento de notificações sobre proximidade de pontos de interesse e apoiam o utilizador na navegação de percursos. Nesta dissertação analisam-se os desafios no desenvolvimento de aplicações móveis para a plataforma Android com serviços de localização e apoio à navegação com mapas em percursos pedestres *outdoor*, recorrendo apenas e só às capacidades de um *Smartphone* e das tecnologias disponíveis no meio envolvente. Além das principais técnicas e tecnologias de localização existentes nos *Smartphones*, e nos equipamentos Android em particular, esta dissertação analisa estratégias de poupança de energia, recorrendo por exemplo à técnica de fusão de sensores, tipos de mapas, e consumo de *web services*. O objetivo desta investigação é discutir e propor melhores práticas e referências para desenvolvimento de aplicações móveis em Android, com serviços de localização e de apoio à navegação em percursos pedestres *outdoor*, sem criar uma infraestrutura de localização dedicada, que sejam eficientes energeticamente, e que comuniquem via *web services* com um servidor HTTP.

Palavras-chave

Aplicações móveis, *Smartphones*, GPS, Sensores, Localização, Android

Abstract

The number of Smartphones users is increasing, and creating opportunities for the development of many applications in various application domains. Most Smartphones come with Internet connection, GSM/GPRS/3G/HSPDA/4G antenna, GPS, Bluetooth, *Wi-Fi*, and other sensors such as magnetometer, gyroscope and accelerometer, facilitating geographic localization, geographic position calculation, the supply of points of interest proximity and supporting the user in trails navigation. This research analyzes the challenges in developing Android mobile applications that contain location services and map navigation support for outdoor pedestrian trails, using only the Smartphones native features and technologies available. Besides the main location techniques and technologies available in Smartphones, and particularly in Android, this research analyses power saving strategies, using for instance sensor fusion, map types, and web services consuming. The objective of this research is to discuss and propose best practices and references to the development of mobile applications in Android, with location services and pedestrian outdoor navigation support, without creating a dedicated location infrastructure, that are energy-efficient, and that communicate through web services with a HTTP server.

Keywords

Mobile Applications, Smartphones, GPS, Sensor, Location, Android

Índice

ÍNDICE	V
LISTA DE FIGURAS	VI
LISTA DE TABELAS	VI
LISTA DE ABREVIATURAS	VII
1 INTRODUÇÃO	1
1.1 MOTIVAÇÃO E ENQUADRAMENTO	1
1.2 OBJECTIVOS.....	1
1.3 METODOLOGIA DE INVESTIGAÇÃO E DESENVOLVIMENTO	3
1.4 ORGANIZAÇÃO DA DISSERTAÇÃO	4
2 TRABALHO RELACIONADO	6
2.1 TÉCNICAS E TECNOLOGIAS DE LOCALIZAÇÃO E SENSORES DOS SMARTPHONES	6
2.2 EFICIÊNCIA ENERGÉTICA NOS SERVIÇOS BASEADOS EM LOCALIZAÇÃO EM SMARTPHONES	16
2.3 WEB SERVICES EM APLICAÇÕES MÓVEIS COM SERVIÇOS DE LOCALIZAÇÃO.....	22
2.4 MAPAS EM APLICAÇÕES MÓVEIS.....	24
3 SUPORTE NO ANDROID	27
3.1 PLATAFORMA ANDROID	27
3.2 DESENVOLVIMENTO DE APLICAÇÕES EM ANDROID	28
3.3 TÉCNICAS E TECNOLOGIAS DE LOCALIZAÇÃO E FUSÃO DE SENSORES NO ANDROID	33
3.4 EFICIÊNCIA ENERGÉTICA NOS SERVIÇOS BASEADOS EM LOCALIZAÇÃO NA PLATAFORMA ANDROID	36
3.5 WEB SERVICES NA PLATAFORMA ANDROID	37
3.6 MAPAS NA PLATAFORMA ANDROID	38
4 MTANDROID – ARQUITETURA, DESENHO E IMPLEMENTAÇÃO	40
4.1 REQUISITOS GERAIS	40
4.2 ARQUITETURA	42
4.3 MODELO DE DOMÍNIO	43
4.4 CONTEÚDOS	44
4.5 MAPAS E REPRESENTAÇÃO DE TRILHOS E PONTOS DE INTERESSE.....	48
4.6 TÉCNICAS E TECNOLOGIAS DE LOCALIZAÇÃO COM EFICIÊNCIA ENERGÉTICA	49
4.7 DIFICULDADES NO DESENVOLVIMENTO EM ANDROID.....	53
5 AVALIAÇÃO	56
5.1 AVALIAÇÃO DE REQUISITOS FUNCIONAIS	56
5.2 AVALIAÇÃO DE REQUISITOS NÃO-FUNCIONAIS.....	62
6 CONCLUSÕES E TRABALHO FUTURO	68
6.1 CONCLUSÕES	68
6.2 TRABALHO FUTURO	70
REFERÊNCIAS	71

Lista de Figuras

FIGURA 1. METODOLOGIA ACTION-RESEARCH	3
FIGURA 2. TAMANHOS E LOCALIZAÇÕES DAS ÁREAS COBERTAS POR DARDOS NUM ALVO (PRECISÃO VS. EXATIDÃO)	7
FIGURA 3. CÁLCULO DE UMA POSIÇÃO 2D USANDO A TÉCNICA DE LATERALIZAÇÃO	8
FIGURA 4. CÁLCULO DE UMA POSIÇÃO 2D COM ANGULAÇÃO	8
FIGURA 5. LONGEVIDADE DA BATERIA (HORAS:MINUTOS:SEGUNDOS) DE CADA ESQUEMA DE TESTE DO RAPS	18
FIGURA 6. CONSUMO MÉDIO DE ENERGIA ESTIMADO PARA CADA ESQUEMA DE TESTE DO RAPS	19
FIGURA 7. FLUXOGRAMA DA LÓGICA DO CLIENTE ENTRACKED	21
FIGURA 8. ARQUITETURA ANDROID	28
FIGURA 9. CICLO DE VIDA DE UMA ACTIVITY NA PLATAFORMA ANDROID	30
FIGURA 10. PROTÓTIPO MTANDROID.....	40
FIGURA 11. ARQUITETURA DO PROTÓTIPO MTANDROID	42
FIGURA 12. DIAGRAMA UML DO MODELO DE DOMÍNIO.....	44
FIGURA 13. EXEMPLO DO FICHEIRO CATEGORIESPOIS.XML	45
FIGURA 14. EXEMPLO DO FICHEIRO POIS.XML	46
FIGURA 15. EXEMPLO DO FICHEIRO TRAILS.XML.....	47
FIGURA 16. ARRANQUE DA APLICAÇÃO MTANDROID	56
FIGURA 17. VISUALIZAÇÃO DE PONTOS DE INTERESSE E EVENTOS	57
FIGURA 18. INTERFACE ESPÉCIES.....	58
FIGURA 19. SELEÇÃO DE TRILHO	59
FIGURA 20. INICIAR PERCURSO	60
FIGURA 21. PERCURSOS	61
FIGURA 22. MEDIDAS POWERTUTOR DA APLICAÇÃO MTANDROID (GPS, 5 SEG., 10M)	63
FIGURA 23. TESTES COM 10, 50, 200 E 400 POIS SOBRE O MAPA NO PROTÓTIPO MTANDROID	66

Lista de Tabelas

TABELA 1. COMPARAÇÃO ENTRE TECNOLOGIAS DE LOCALIZAÇÃO EM SMARTPHONES	14
TABELA 2. VANTAGENS E LIMITAÇÕES DAS TECNOLOGIAS DE LOCALIZAÇÃO EM SMARTPHONES	15
TABELA 3. SEIS ESQUEMAS DIFERENTES PARA AVALIAÇÃO DO RAPS	17
TABELA 4. ESTRATÉGIAS DE EFICIÊNCIA ENERGÉTICA DO RAPS E DO ENTRACKED.....	22
TABELA 5. SENSORES SUPORTADOS PELA PLATAFORMA ANDROID.....	34
TABELA 6. TESTES AOS CONSUMOS MÉDIOS DE ENERGIA DA APLICAÇÃO MTANDROID.....	64
TABELA 7. TEMPO DE VISUALIZAÇÃO DO MAPA E TEMPO DE APRESENTAÇÃO DA DIALOG BOX DE UM POI.....	67

Lista de Abreviaturas

3G	3rd Generation (Mobile Telecommunications)
4G	4th Generation (Mobile Telecommunications)
API	Application Programming Interface
DOM	Domain Object Model
GPRS	General Packet Radio Service
GPS	Ground Position System
GSM	Global System for Mobile communication
HTTP	HyperText Transfer Protocol
INS	Inertial Navigation Systems
IPC	Inter Process Communication
JSON	JavaScript Object Notation
LLC	Localized Location Computation
POI	Point of Interest
REST	REpresentational State Transfer
RPC	Remote Procedure Call
RSS	Received Signal Strength
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
Wi-Fi	Wireless Fidelity
WPS	Wi-Fi Positioning System
XML	eXtensible Markup Language

1 Introdução

1.1 Motivação e Enquadramento

Com os *Smartphones* a tornarem-se cada vez mais ubíquos, a disponibilidade da Internet e o aumento da capacidade de processamento e das funcionalidades dos terminais móveis estão a potenciar novos desenvolvimentos de aplicações móveis baseadas em localização para diferentes domínios de aplicação tais como turismo [36], saúde [37], comunidades sociais [61] ou transportes [65]. O número de *Smartphones* está a aumentar, com custos cada vez mais reduzidos (nomeadamente os *Smartphones* com o sistema operativo Android) e a dotar cada vez mais utilizadores com terminais capazes de usar aplicações que tiram partido da Internet e da geo-referenciação [4,6,7,36,37].

O desafio mais evidente prende-se com a existência de várias plataformas móveis, como a Symbian, BlackBerry, Windows Phone (Microsoft), Android (Google) ou iOS (Apple), que dificulta o desenvolvimento de qualquer tipo de aplicação móvel. Para cada uma das plataformas, as ferramentas (SDK) e as linguagens de programação são diferentes, aumentando a dificuldade de pôr em produção uma aplicação que abranja todo o tipo de terminais. Para minimizar esta dificuldade, existem ferramentas que permitem desenvolver aplicações móveis transversalmente, ou seja, que geram código para várias plataformas móveis [64].

A plataforma Android, promovida pela Google, é a plataforma móvel com maior crescimento nos últimos anos [38], por disponibilizar o SDK a custo zero à comunidade de programadores, e por integrar com as populares APIs da Google, como as *Google Maps APIs*.

Com base em investigação nas áreas de computação móvel, informação geográfica e localização espacial, pretende-se com esta dissertação investigar as técnicas e tecnologias para desenvolvimento de aplicações móveis baseadas em localização para o sistema operativo Android.

1.2 Objectivos

Pretende-se com esta dissertação identificar os desafios no desenvolvimento de aplicações móveis na plataforma Android, que satisfaçam os seguintes requisitos:

- Dar apoio à navegação sobre mapas digitais, em percursos pedestres *outdoor*, em trilhos pré-definidos;
- Determinar a posição geográfica (*outdoor*) em tempo-real recorrendo apenas a um *Smartphone* Android e às redes públicas disponíveis (GPS, GSM/GPRS/3G/4G, Wi-Fi), com as melhores precisão e exatidão possíveis - os termos precisão e exatidão serão explicados em detalhe no Capítulo 2;
- Fornecer notificações sobre proximidade de pontos de interesse;
- Minimizar o consumo de bateria do *Smartphone*;
- Comunicar com um servidor HTTP remoto para consultar e atualizar dados sobre trilhos, posição geográfica e pontos de interesse.

Os objetivos desta dissertação são os seguintes:

- Identificar e analisar os desafios inerentes ao desenvolvimento de uma aplicação móvel com serviços de localização para *Smartphones* Android com os requisitos atrás enunciados;
- Conceber e implementar um protótipo de uma aplicação Android que satisfaça esses mesmos requisitos;
- Avaliar os resultados da aplicação protótipo com base em cenários controlados.

No contexto desta investigação, com o propósito de identificar e discutir melhor os desafios referidos no primeiro objetivo, foi desenhado e implementado o protótipo **mtAndroid**. O cenário de utilização do protótipo mtAndroid é baseado na aplicação MobileTrails [62, 64]. No Capítulo 4 é apresentada a arquitetura, desenho e implementação do mtAndroid.

O primeiro desafio identificado é determinar quais as técnicas e tecnologias de localização mais adequadas, e vantagens e desvantagens de cada um delas. Existem várias técnicas de localização, e diversas tecnologias que usam essas mesmas técnicas nas plataformas móveis - GPS, WPS, GSM e o *Bluetooth* - sendo a mais usada o sistema GPS, por ser a que fornece melhor precisão e exatidão.

As tecnologias de localização podem, por várias razões, ficar indisponíveis (p.ex. por falta de sinal GPS em ambientes *indoor*). Podem também não garantir a precisão ou exatidão exigidas conforme o tipo de aplicação. Felizmente, os *Smartphones* atuais estão munidos com sensores úteis no apoio à determinação da posição, como o acelerómetro, o magnetómetro e o giroscópio. Estes sensores podem ser incluídos em técnicas de localização, usando a técnica de fusão de sensores (*Sensor Fusion*) [1], e melhorar assim a precisão e exatidão na determinação da posição geográfica. Todavia, a fusão de sensores tem limitações; primeiro porque os sensores não estão incluídos nativamente nas funções de localização, é necessário programar os algoritmos de localização para os utilizar nestas funções; segundo porque podem ser muito imprecisos e pouco exatos devido aos graus de liberdade de movimento dos *Smartphones*.

O problema mais referido relativo ao uso do GPS é o consumo de bateria [3,4,7,25,26,32]. Por conseguinte, o segundo desafio identificado é analisar estratégias de poupança de bateria que consigam garantir a precisão e a exatidão exigidas no cálculo da posição.

Por outro lado, para facilitar a navegação em aplicações móveis, é frequente recorrer-se a mapas digitais. Por isso, o terceiro desafio é determinar os tipos de mapas que podem ser usados em aplicações móveis Android para apoio à navegação.

Por fim, as aplicações móveis ficam mais funcionais se tiverem interação com serviços disponibilizados na *cloud*, por exemplo para consulta ou atualização de dados. A técnica atual mais usada para comunicação entre terminais móveis e servidores é a dos *web services*, os mais adequados à heterogeneidade da Internet.

Consequentemente o quarto desafio é determinar as técnicas mais adequadas para comunicar entre *smartphones* e servidores usando *Web Services*.

1.3 Metodologia de Investigação e Desenvolvimento

A metodologia de investigação utilizada foi a *Action-Research* [56], que consiste em provar se a solução resolve o problema. Na Figura 1 é ilustrada a metodologia, que é composta por um ciclo de cinco passos interativos. O primeiro passo é o de Diagnóstico, onde o problema será analisado num contexto específico. O passo seguinte, o Plano de Ação, serve para propor uma solução e alterações necessárias para resolver o problema encontrado no passo anterior. O terceiro passo, Tomar Ação, consiste em pôr em prática o plano de ação resultante do passo anterior no contexto da investigação. O passo seguinte é o de Avaliação, que consiste em avaliar se a ação tomada resolve o problema no contexto da investigação. Por último, tem-se o passo de Aprendizagem, que permite recolher o conhecimento resultante do ciclo, e preparar novo ciclo caso seja necessário.

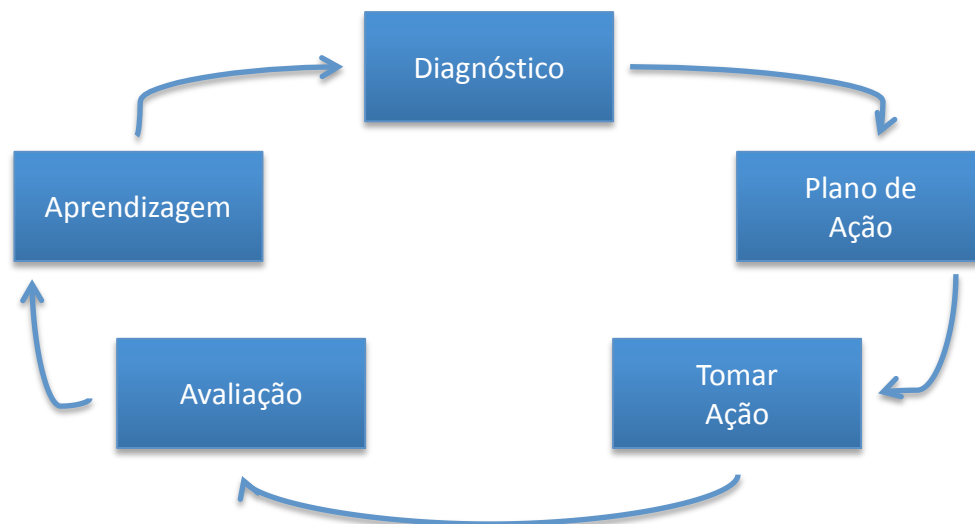


Figura 1. Metodologia Action-Research¹

No passo de Diagnóstico foi usada preferencialmente a informação disponível em diferentes artigos científicos relativos a computação móvel, serviços baseados em localização, técnicas e tecnologias de localização em *smartphones*, fusão de sensores em *smartphones*, técnicas de localização eficientemente energeticamente em *smartphones*, *web services* em *smartphones*, mapas digitais para serviços de localização em *smartphones* e plataforma Android. Foi usada a informação presente nos sites oficiais dos fornecedores de serviços de localização, como por exemplo a Google [21,42] e a ESRI [11].

Para definição dos Planos de Ação, foram analisados os requisitos da aplicação, a informação recolhida no passo anterior, e criados e atualizados os esquemas UML (Modelo de Domínio) e Relacionais (Base de Dados SQLite).

¹ Adaptado de Baskerville *et al* [56]

² Adaptado de Davis *et al*[47]

³ Adaptado de Hightower *et al*[48]

⁴ Adaptado de Hightower *et al*[48]

Para o passo Tomar Ação (desenvolvimento de testes e dos primeiros protótipos) foi usado o IDE Eclipse Helios SR2 em Mac OS X Snow Leopard 10.6.8, com o *plugin* Android SDK instalado, e as Android Google APIs Level 8 (Android 2.2), 9 (Android 2.3) e 15 (Android 4.0). O último protótipo foi desenvolvido em Mac OS X Lion 10.7.5, com o *Eclipse Juno Service Release 1*, Android Google API's Level 15 (Android 4.0.3). Como apoio ao despiste, depuração de erros, e problemas de configuração do SDK Android no Eclipse, foram consultados os sites *Android Developers* [5], *Stackoverflow* [43] e *Vogella.com* [57], onde programadores partilham conhecimento sobre várias plataformas e linguagens de programação. Para avaliação dos vários protótipos foram usados vários *Smartphones*. Os protótipos iniciais, para avaliar a importação de dados, a base de dados e a interface, foram corridos no emulador Android do SDK integrado no Eclipse. Os testes e os protótipos seguintes, com funcionalidades de localização, foram executados nos dispositivos Android LGE P-500 (Android 2.2), Sony Ericsson Xperia X10 (Android 2.3.3) e Samsung Galaxy Note GT-N7000 (Android 4.0.3). Foi utilizado preferencialmente o trilho de São Pedro do Estoril para avaliar a aplicação.

Após obter-se o conhecimento resultante de um ciclo, era iniciado novo ciclo para efetuar as alterações necessárias para se obter a solução melhorada do problema, e avaliar o novo protótipo.

O primeiro ciclo correspondeu à exploração do Android, aprendizagem das tecnologias e técnicas de localização, e à definição de requisitos da aplicação *mtAndroid* (entre Setembro de 2011 e Janeiro de 2012).

O segundo ciclo consistiu na definição do modelo de dados, em avaliar a melhor forma de representação de pontos de interesse e trilhos, e na interação com mapas em Android (entre Janeiro de 2012 e Março de 2012).

No terceiro ciclo (entre Março de 2012 e Agosto de 2012), foi implementada a interface normalizada da aplicação para validar os principais requisitos funcionais e o modelo de domínio. Neste ciclo foram também estudadas quais as melhores formas de implementar os mapas *offline*, e sua implementação.

O quarto ciclo (entre Agosto de 2012 e Dezembro de 2012) consistiu em analisar as estratégias de poupança de bateria na aplicação *mtAndroid*, nomeadamente recorrendo ao acelerómetro e fusão de sensores, e ao ciclo de vida das atividades em Android. Por fim, foi efetuada a avaliação do protótipo final a nível de requisitos funcionais e não-funcionais.

1.4 Organização da Dissertação

Esta dissertação encontra-se organizada em seis capítulos, cujos conteúdos estão descritos nos próximos parágrafos.

Capítulo 1 – Introdução: introduz a motivação e enquadramento, os objetivos gerais que orientam este trabalho de investigação, e aspectos relacionados com a metodologia utilizada.

Capítulo 2 – Trabalho Relacionado: é efetuada uma revisão da literatura existente relevante para o projeto proposto, nomeadamente o estado da arte das técnicas e tecnologias de localização, sensores em *smartphones*, das estratégias de eficiência energética na localização nos *Smartphones*, dos *web services* em *Smartphones*, e dos mapas usados em aplicações móveis

Capítulo 3 – Suporte no Android: é discutido o suporte na plataforma Android das tecnologias de localização,

estratégias de poupança de bateria, mapas digitais e *web services*, referidos no capítulo 2.

Capítulo 4 – *mtAndroid* – Arquitetura, Desenho e Implementação: é introduzido um cenário de avaliação das técnicas e das tecnologias descritas no capítulo 3, a aplicação *mtAndroid*. É definida a arquitetura de software, o modelo de domínio, o formato dos conteúdos, a implementação dos mapas, as técnicas de localização com eficiência energética, e as dificuldades sentidas no desenvolvimento.

Capítulo 5 – Avaliação: apresenta a avaliação da aplicação protótipo *mtAndroid* a nível de requisitos funcionais (validação das funcionalidades) e requisitos não-funcionais (eficiência energética e escalabilidade).

Capítulo 6 – Conclusão e Trabalho Futuro: o último capítulo apresenta a conclusão, considerações finais e aspectos em aberto para trabalho futuro.

Bibliografia. Lista de referências bibliográficas relevantes tanto para a fase de pesquisa como para a fase do desenvolvimento do protótipo *mtAndroid*.

2 Trabalho Relacionado

Neste capítulo efetua-se uma revisão da literatura existente que se considera relevante para o projeto proposto. Efetua-se uma análise dos desenvolvimentos mais recentes das áreas que estão diretamente relacionados com os objetivos principais, enunciados no capítulo 1.

Nesta dissertação são usados frequentemente os termos exatidão e precisão. As definições detalhadas destes termos podem ser consultadas na secção 2.1. De seguida, nesta mesma secção, são analisadas as técnicas e tecnologias de localização mais usadas nas aplicações móveis, e quais as formas de usar os sensores presentes nos *Smartphones*. A secção 2.2 analisa estratégias que permitem minimizar o consumo de bateria em aplicações que fornecem serviços baseados em localização. É apresentada na secção 2.3 a análise dos *Web Services* em aplicações móveis baseadas em localização. Por fim, na secção 2.4 é apresentada uma análise sumária do uso de mapas em aplicações móveis.

2.1 Técnicas e Tecnologias de Localização e Sensores dos Smartphones

Nesta secção começa-se por definir os conceitos precisão e exatidão. Depois são analisadas as técnicas de localização mais usadas. A seguir são descritas as tecnologias de localização existentes nos *Smartphones*, como o GPS, o WPS, as Torres de Células e o *Bluetooth*. De seguida são descritos os sensores adicionais que permitem ajudar a calcular a posição ou a melhorar a experiência de navegação em mapas (acelerómetro, magnetómetro e giroscópio), e introduz-se o conceito de fusão de sensores.

2.1.1 Precisão versus Exatidão

É frequente os termos exatidão (*accuracy*) e precisão (*precision*) serem confundidos ou usados como sinónimos. Todavia, a nível científico têm significados distintos [47].

Exatidão refere-se à proximidade das medidas para o valor correto ou aceitável da quantidade medida.

Precisão refere-se à proximidade de um conjunto de medidas da mesma quantidade efetuadas da mesma forma.

Uma analogia muito usada para explicar o significado e a diferença entre os termos precisão e exatidão, é o exemplo do alvo de dardos [47] (*dartboard*), conforme ilustrado na Figura 2.

Quanto mais próximos os dardos estiverem do alvo, com maior exatidão foram atirados. Quanto mais próximos os dardos estiverem entre si, com maior precisão foram atirados. Na Figura 2(a) os resultados são precisos e exatos porque os dardos estão perto do alvo e próximos uns dos outros. Na Figura 2(b) os resultados são precisos porque os dardos estão muito próximos entre si, mas pouco exatos porque estão distantes do alvo. Na Figura 2(c) os resultados são imprecisos porque os dardos estão distantes entre si, e pouco exatos porque estão distantes do alvo. Na Figura 2(d) os resultados são exatos, quando comparado com (c), pois os dardos estão distribuídos igualmente em torno do alvo, mas imprecisos por os dardos estarem longe entre si.

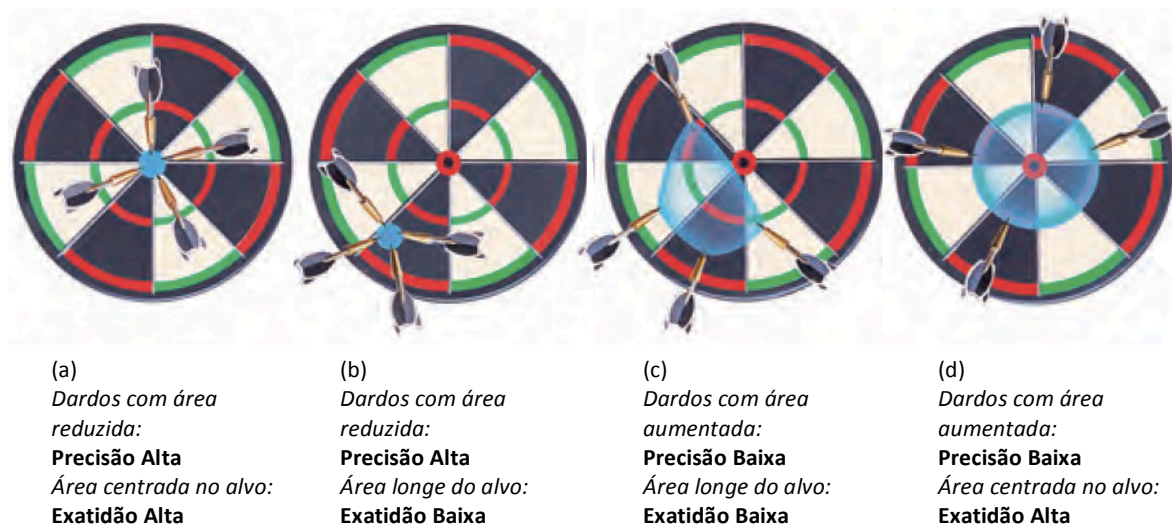


Figura 2. Tamanhos e localizações das áreas cobertas por dardos num alvo (precisão vs. exatidão)²

Em Hightower *et al* [1], é apresentada a definição destes dois conceitos usando como exemplo dispositivos GPS: Dispositivos GPS mais baratos podem localizar posições dentro de 10 metros para aproximadamente 95% das medidas; Dispositivos GPS mais caros conseguem exatidões entre 1 e 3 metros 99% das vezes.

As distâncias apresentadas nos exemplos correspondem à exatidão, ou granularidade. A percentagem corresponde à precisão, ou seja, quantas vezes estamos à espera de ter aquela mesma exatidão.

2.1.2 Técnicas de Determinação da Localização

Conforme descrito em Hightower *et al* [1,48], as três principais técnicas de determinação da localização (*Location Sensing*) são a Triangulação, a Análise de Cenários (*Scene Analysis*) e a Proximidade.

A técnica de *Triangulação* usa as propriedades geométricas dos triângulos para calcular a posição de objetos. A triangulação divide-se em duas categorias: a lateralização e a angulação.

A *Lateralização*, também designada por *Trilateração* [49], calcula a posição através da medida das distâncias a pontos de referência. Para calcular a posição em 2D a lateralização necessita de 3 pontos de referência, e em 3D de quatro pontos de referência. Na Figura 3 é ilustrado o cálculo de uma posição 2D usando lateralização. A intersecção dos três círculos determinam a posição do ponto que se pretende localizar. Os círculos envolvem as localizações conhecidas e os raios dos círculos (r_1 , r_2 , r_3) são usados para calcular a posição do objecto a localizar. Outras técnicas de localização baseadas em lateralização são as *Time of Arrival* (TOA), *Time of Difference of Arrival* (TDOA), e *Received Signal Strength* (RSS), usadas em tecnologias de redes sem fios como o GSM, Wi-Fi e o *Bluetooth*, descritas nas secções seguintes. As técnicas TOA e TDOA envolvem cálculos de distância com base em tempos de propagação entre a antena receptora e a antena emissora, o RSS usa para calcular a posição a força do sinal recebido [23]. A técnica RSS é a de mais simples implementação, e a que consegue mais exatidão é a TOA [24]. O problema da lateralização é o de que, na realidade, as medidas das

² Adaptado de Davis *et al*[47]

distâncias entre os pontos nunca são perfeitas e a intersecção dos círculos costuma resultar em vários pontos possíveis.

Uma das tecnologias de localização mais usadas em *Smartphones* e descrita na secção seguinte, o GPS, usa a lateralização para determinar a posição, usando como pontos de referência os satélites.

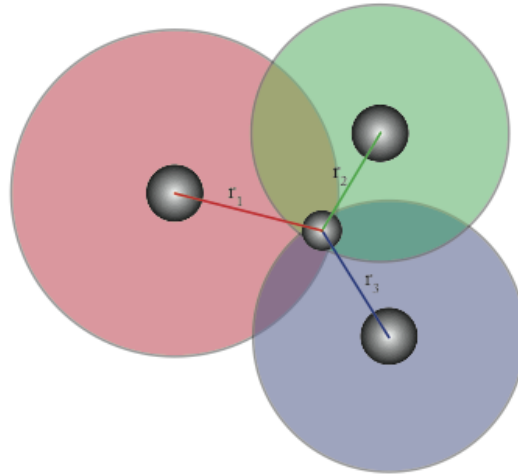


Figura 3. Cálculo de uma posição 2D usando a técnica de lateralização³

A *Angulação* é semelhante à lateralização, com a diferença de usar ângulos invés de distâncias para calcular a posição. Na Figura 4 é ilustrado um exemplo de localização em 2D recorrendo à angulação. O objecto que se pretende localizar é X. É indicado na ilustração o referencial para os zero graus, a distância entre os dois pontos de referencia, e os dois ângulos necessários ao cálculo da posição.

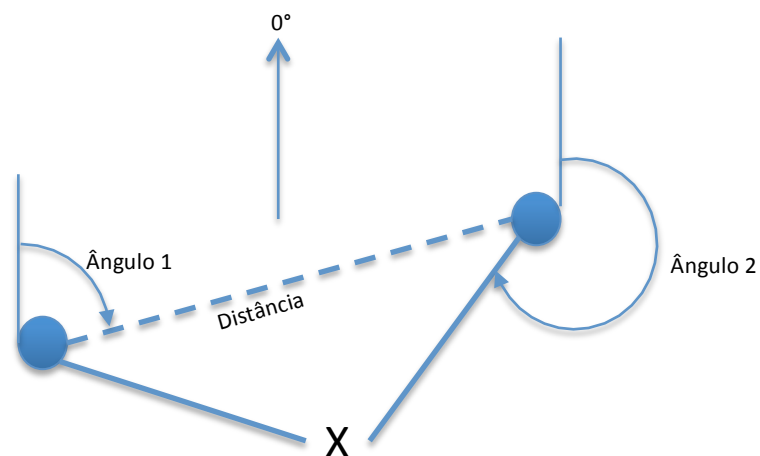


Figura 4. Cálculo de uma posição 2D com angulação⁴

³ Adaptado de *Hightower et al*[48]

⁴ Adaptado de *Hightower et al*[48]

Para calcular uma posição 2D é necessário conhecer os dois ângulos e a distância entre os pontos de referência. Para calcular uma posição 3D é necessária também a leitura da medida do azimute (distância angular sobre o horizonte). É uma técnica muito menos usada que a lateralização. Uma técnica baseada em angulação bastante aplicada em redes sem fios é a técnica *Angle of Arrival* (AOA), que utiliza o ângulo de propagação da onda de rádio. Esta técnica tem como limitação necessitar de antenas emisoras direcionais. É usada, por exemplo, para determinar posições de *Smartphones* em redes GSM (tecnologia Torres de Células, ver subsecção 2.1.5).

A *Análise de Cenários* é uma abordagem de localização totalmente diferente da triangulação. Para as redes sem fios, baseia-se na ideia de que o ambiente rádio tem assinaturas características, que podem ser medidas em antecipação, guardadas e comparadas com as leituras do momento. A forma mais evidente de análise de cenários é, com base numa fotografia tirada com uma câmara a um local, tentar a partir da imagem determinar a posição. Outra forma é usar características específicas de uma localização, como padrões de propagação de ondas de rádio. Uma das vantagens desta técnica em relação à triangulação é não precisar de cálculos baseados em distâncias ou ângulos, ou seja, é computacionalmente menos pesado se forem usados dados históricos computacionalmente rápidos a processar e a comparar com as leituras. Por outro lado, se a determinação da posição utilizar processamento de imagem, poderá ser computacionalmente mais pesado. Para processamento de imagem faz mais sentido usar um servidor robusto para processar os dados *offline* invés de um *Smartphone* em tempo-real. Uma das tecnologias que usa a análise de cenários para determinar a localização de *Smartphones* é o Wi-Fi, mais especificamente através do sistema WPS (*Wireless Positioning System*), descrito na subsecção 2.1.4.

A terceira e última técnica genérica de localização é a *Proximidade*. Existem três abordagens [48] para determinar a localização usando proximidade do objeto que se pretende localizar: deteção de contacto físico, pontos de acesso sem fios de monitorização e sistemas de identificação automática.

A deteção de proximidade através de contacto físico é conseguida através de sensores de pressão, sensores de tacto ou detectores de campos electromagnéticos (ou campos capacitivos).

A abordagem dos pontos de acesso sem fios serve para detectar o objecto (um *Smartphone*, p.ex.) quando este se aproxima do raio de cobertura de rede da antena do ponto de acesso. Um sistema de localização que usa esta abordagem é *Active Badge* descrito em Hightower *et al* [1]. É uma abordagem totalmente diferente do sistema WPS (subsecção 2.1.4), em que é o próprio *Smartphone* que analisa o meio envolvente (técnica de análise de cenários), ou seja, os pontos de acesso são passivos na determinação da posição.

Os sistemas baseados em ID são, por exemplo, os terminais POS de pagamentos multibanco, os registos autenticação de um utilizador num computador, registos de chamadas telefónicas de uma linha de rede fixa ou registos electrónicos de passagem em parques de estacionamento ou portagens. Todas as abordagens devem ser complementadas com um sistema de identificação automática se a própria tecnologia não o incluir, por

forma a conseguir localizar o objecto. Sistemas baseados na tecnologia RFID⁵ (*Radio Frequency Identification*) ou NFC⁶ (*Near Field Communication*) são exemplos de sistemas que usam a proximidade como técnica de localização e que incluem sistema de identificação automática.

Conforme facilmente se pode verificar, a técnica baseada em proximidade exige na maior parte dos casos a criação de uma infraestrutura dedicada, que está fora do âmbito desta dissertação. Uma forma de usar a técnica de proximidade num *Smartphone* sem recorrer a instalação de sensores ou pontos de acesso é tirar partido da interface *Bluetooth* (subsecção 2.1.6) para determinar a proximidade de outros *Smartphones*.

Nas subsecções seguintes serão analisadas as principais tecnologias presentes na maioria dos *Smartphones*, que tiram partido das técnicas de localização descritas, e que servem os objetivos desta dissertação.

2.1.3 GPS

O GPS foi inventado em 1973 pelo Departamento de Defesa dos Estados Unidos da América, entidade ainda responsável pela sua manutenção. Começou por ser conhecido pelo sistema NAVSTAR, e ficou operacional em 1995 com a constelação de 24 satélites. No 1º de Maio de 2000, o presidente dos EUA Bill Clinton retirou as restrições militares ao sistema de posicionamento global, e anunciou que a opção de degradar o sinal GPS durante situações de emergência iria ser retirada, faseadamente, até 2010. Desde então, o nº de aplicações comerciais e de utilizadores do GPS aumentou exponencialmente. Segundo uma previsão da RNCOS⁷, serão comercializados mundialmente entre 2011 e 2013 cerca de 900 milhões de dispositivos GPS.

O Sistema de posicionamento global (GPS) é o sistema mais referenciado e usado nos serviços baseados em localização. A maior parte das aplicações móveis para *Smartphones* usa o GPS como tecnologia preferencial para cálculo da posição geográfica, por ser mais exato e mais preciso, quanto comparado com outras tecnologias de localização, como as Torres de Células ou o WPS. De acordo com a taxonomia de Hightower *et al* [1], o GPS fornece a localização física, absoluta, é escalável (não existe limitação de receptores GPS) e fornece computação de localização localizada (LLC), assegurando assim privacidade, uma vez que apenas o próprio receptor GPS conhece a sua posição.

Um receptor de GPS necessita do sinal e dados de efemérides de 3 satélites para calcular o tempo e a posição em latitude, longitude (mapas 2D), e de 4 para também calcular a altitude (mapas 3D), através da técnica de lateralização (referida na secção 2.1.2). Com pelo menos 4 satélites, a exatidão será de 15 metros. As fontes do erro, neste caso, são: i) efeitos ionosféricos (~5 metros), ii) deslocamentos nas orbitas do satélite (~2,5 metros), iii) erros de relógio do satélite (~2 metros), iv) efeitos *multipath* (~1 metros), v) efeitos troposféricos (0,5 metros), e vi) erros de cálculo e arredondamentos (~1 metros) [2].

Uma das limitações do GPS quando comparado com as técnicas alternativas de cálculo da posição nos *Smartphones* é o consumo de bateria [3,4,7,25,26,32], mais elevado do que quando é usado o WIFI, *Bluetooth*

⁵ http://www.pcmag.com/encyclopedia_term/0,1237,t=RFID&i=50512,00.asp (2013/01/11)

⁶ <http://www.mobileburn.com/definition.jsp?term=NFC> (2013/01/11)

⁷ <http://www.rncos.com/> (2013/01/11)

ou antena rádio do *Smartphone*. Na secção 2.2 são apresentadas algumas estratégias para minimizar este problema.

Outra limitação do GPS é a perda de sinal em ambientes *indoor* ou meios urbanos com fraco sinal de satélite, como por exemplo, entre edifícios altos, perdendo-se assim precisão. Uma das técnicas para resolver esta situação é recorrer às alternativas do GPS, como as Torres de Células ou o WPS, ou seja, as mesmas que são usadas para otimizar o consumo de bateria. Técnicas mais caras envolvem instalar repetidores de sinal GPS *indoor* ou nas fachadas dos prédios [1].

2.1.4 Wi-Fi (WPS)

A tecnologia Wi-Fi pode ser usada em várias técnicas de localização nos *Smartphones*. É usada, por exemplo, em sistemas que tirem partido de pontos de acesso para detecção de proximidade ou determinação da posição com técnicas de triangulação. No âmbito desta dissertação, escolheu-se analisar para esta tecnologia o sistema WPS, por ser o sistema de localização mais usado em *Smartphones* quando não existe infraestrutura Wi-Fi dedicada. Este sistema insere-se na técnica de localização baseada em análise de cenários, referida na secção 2.1.2.

O WPS (*Wi-Fi Positioning System*) é usado como alternativa de localização quando o GPS não está disponível. É frequentemente usado em ambientes *indoor* ou ambientes urbanos com prédios altos, onde é frequente a perda de sinal GPS. Este sistema tira partido dos pontos de acesso Wi-Fi públicos, cujo número está a crescer exponencialmente nos meios urbanos, e é usado por vários fornecedores de localização como a *Google*, a *Navizon*⁸ ou a *Skyhook Wireless*⁹ (fornecedor do serviço de localização da Apple).

Para determinar a localização dos pontos de acesso Wi-Fi, é usada a técnica *Wardriving* [16], que consiste na procura de acessos *Wireless* num veículo munido com um cliente Wi-Fi e GPS. Os técnicos registam numa base de dados o *Mac Address* (e potencialmente o SSID se não estiver escondido), sinal, ruído, tipo de encriptação e a posição GPS do ponto de acesso Wi-Fi [18]. Técnicas similares são o *Warwalking* e o *Warbiking*. A *Google* recolheu esta informação através dos carros de captura de imagens para o *StreetView* [21]. Esta técnica envolve várias iterações, ou seja, é necessário atualizar regularmente a base de dados repetindo os *Site Surveys*. Os pontos de acesso podem ser desligados ou mudados de local, introduzindo erros no cálculo da posição, e baixando assim a precisão [50].

Ao contrário do GPS, o WPS não garante privacidade. Existem várias técnicas para recolher informação sobre trilhos percorridos por utilizadores, locais visitados, acesso a dados transferidos e sites de internet acedidos. Os terminais Android, Apple e Microsoft enviam informação de posição via Internet, quando se tem o mecanismo de localização ativo (*Google Latitude*¹⁰ nos *Smartphones* e *Tablets* Android), atualizando assim as bases de dados WPS sem recorrer ao *Wardriving*. Esta funcionalidade põe ainda mais em causa a privacidade, porque se está a explicitamente a fornecer informação do terminal aos fornecedores privados de serviços de

⁸ <http://www.navizon.com/> (2013/01/11)

⁹ <http://www.skyhookwireless.com/> (2013/01/11)

¹⁰ <http://www.google.com/latitude> (2013/01/11)

localização, até agora acessível só aos operadores de telecomunicações, e com regulação do estado [17]. De acordo com um artigo da ZDNET [20], os gigantes Google, Apple e Microsoft, podem estar a recolher esta informação, mesmo com o serviço de localização desativado pelo utilizador.

Os problemas de segurança e ataques típicos a computadores pessoais conhecidos nas redes Wi-Fi, aplicam-se da mesma forma aos *Smartphones*. Para tirar partido do WPS, é necessário ativar o Wi-Fi no *Smartphone*. Ao fazer isto o equipamento fica disponível para ser atacado via Wi-Fi.

Além de ser mais inseguro que o GPS, o WPS é também menos exato. Em Cheng *et al* [16] foi medida em laboratório uma exatidão entre 13 e 20 metros, em ambientes urbanos a exatidão cai para os 40 metros. Em Schilit *et al* [19] são encontradas referências para experiências com resultados de exatidões entre 3 e 100 metros, dependendo, por exemplo, do número de pontos de acesso e da força de sinal Wi-Fi. É também menos preciso devido à volatilidade da posição dos pontos de acesso Wi-Fi.

Apesar de menos seguro, menos exato e menos preciso, o WPS é melhor que o GPS no consumo de bateria [25,26].

2.1.5 Torres de Células (*Cell-ID* e *Cell Tower-ID*)

As redes celulares foram as impulsionadoras do conceito de LBS (*Location-Based Services*). As estações de redes móveis (BTS- *Base Transceiver Station*) comportam uma série de redes celulares que vão desde o GSM até ao recente 4G. Estas tecnologias de localização são mais eficientes *indoor* e em alguns locais urbanos do que o GPS.

Existem várias técnicas de localização em redes de operadores de telecomunicações móveis, e estas podem ser classificadas como [22]: *i*) baseadas no terminal (*hardware* do terminal ou cartão SIM); *ii*) baseadas na rede ou *iii*) híbridas; classificação que varia conforme o local onde as medidas são efetuadas (terminal e/ou rede). A mais usada é a técnica baseada em rede, pois não depende do terminal, e tem uma taxa de penetração de 100%. As técnicas baseadas em rede incluem *Time of Arrival* (TOA), *Time of Difference of Arrival* (TDOA), *Angle of Arrival* (AOA) e *Received Signal Strength* (RSS). Estas técnicas, já referidas na secção 2.12, podem ser aplicadas a outras redes sem fios que não as celulares, por exemplo, às redes Wi-Fi usadas no sistema descrito na subsecção anterior (WPS).

Nos *Smartphones* atuais, esta tecnologia é mais usada com a técnica de análise de cenários, usando um cadastro das posições conhecidas das *Cell-ID* ou da *Cell Tower-ID*, de forma semelhante ao WPS.

Em comparação com o GPS e com o WPS, esta tecnologia revela-se menos exata. A exatidão pode rondar os 1000 metros. Outra limitação é a necessidade de cadastro da *Cell-ID* ou da *Cell Tower-ID*, ou seja, da identificação das células ou torres de células e da sua localização. Além disso, as torres de células são propriedade dos operadores de telecomunicações. Numa aplicação móvel o uso desta informação está limitado às torres do operador ao qual o equipamento se liga. Outro problema é a manutenção do cadastro atualizado, se uma torre for desativada ou mudar de posição (a mesma dificuldade com a alteração da localização dos pontos de acesso Wi-Fi no WPS, embora menos volátil), diminuindo assim a precisão. O problema da privacidade também se coloca neste caso, embora menos crítico do que no WPS, pois o *tracking* pode ser

efetuado apenas pelo operador de telecomunicações móveis, que já está condicionado por legislação sobre privacidade em defesa dos cidadãos [17].

Uma das vantagens é a melhor cobertura, na maior parte dos casos, em ambientes *indoor* ou ambientes urbanos com fraca cobertura GPS ou Wi-Fi. A grande vantagem em relação às duas tecnologias anteriores é o consumo de bateria, mais reduzido quando comparado com qualquer uma delas [3,25,26]. Uma forma de usar as torres de células para poupar bateria, é a técnica usada pelo sistema RAPS [3], que usa a informação das torres de células e histórico (ID da estação GSM com a informação da existência ou não do sinal GPS) para determinar se o sinal satélite está indisponível, evitando assim ativar o GPS desnecessariamente. Mais recente, o sistema CAPS [26] tem igualmente o objetivo de reduzir as consultas GPS, e propõe armazenar sequências de pares <Cell-ID, posição>, tendo também em conta a transição entre células, e aumentando assim a exatidão da localização em relação ao RAPS na técnica de *Cell Tower-ID*.

2.1.6 Bluetooth

O *Bluetooth*¹¹ é uma especificação para ligações sem fios de curto alcance. O *Bluetooth* não fornece qualquer informação de localização, mas em conjunto com as outras técnicas atrás referidas pode ajudar a corrigir a posição geográfica de *Smartphones*, e aumentar a exatidão. Ao uso conjunto de sensores distintos denomina-se fusão de sensores [1], e será explicado na secção seguinte.

O *Bluetooth* como tecnologia de apoio à determinação da posição geográfica aplica-se em ambientes *indoor*, uma vez que o alcance máximo entre o emissor e o receptor é cerca de 10 metros, usando a técnica de localização baseada em proximidade. Sendo o *Bluetooth* um protocolo usado nas redes sem fios, podem ser aplicadas as técnicas de triangulação descritas na secção 2.1.2 (AOA, TDOA, etc.) para determinar a localização de um *Smartphone*. Em Kotanen *et al* [27] é descrita uma experiência que avalia uma estratégia de localização baseada em *Bluetooth*. Contudo, no âmbito desta dissertação, a melhor forma de usar o *Bluetooth* para determinar a posição é usando a técnica da proximidade, em relação a outros *Smartphones*.

Em ambientes *outdoor*, o *Bluetooth* pode ser útil para comunicar com outros *Smartphones* que usem a mesma aplicação móvel, não só para trocar informação no âmbito da comunidade de utilizadores, mas também para fornecer informação de localização com base na posição de outros terminais. O sistema RAPS [3] é um exemplo de uma aplicação que tira partido do *Bluetooth* para reduzir a incerteza de posição de um *Smartphone*. No RAPS, sempre que o terminal recebe uma atualização da posição, envia-a para os terminais vizinhos, poupando a estes uma ativação do sensor GPS. Se o terminal receber uma posição com maior incerteza do que a sua, responde com o seu valor atual (mais exato). O objetivo desta funcionalidade é evitar múltiplas consultas via GPS que irão reduzir rapidamente o tempo de vida da bateria do terminal. Os problemas relativamente ao uso *Bluetooth* referidos no artigo que descreve o RAPS, são a segurança, pelo facto de ter o *Bluetooth* ativo, e a privacidade, por partilhar com outros terminais a posição geográfica. No mesmo artigo, é demonstrado que o *Bluetooth* é mais eficiente a nível energético que o GPS.

¹¹ <http://www.bluetooth.org> (2013/01/11)

A maior limitação divulgada em relação ao *Bluetooth* são as vulnerabilidades de segurança, pois ao estar ativo pode permitir que o *Smartphone* seja atacado por um *hacker*. O tipo de ataques está descrito em Wong [28]. O *Bluetooth* encontra-se atualmente na versão 4, e desde a versão 2.1¹² (2007) que foi introduzido pelo *Bluetooth SIG* o *SSP (Secure Simple Parsing)*, introduzindo melhorias significativas na segurança do protocolo. Além da segurança foram melhoradas a simplicidade de configuração do emparelhamento, a velocidade e a eficiência energética (menor consumo de bateria).

2.1.7 Comparação das Tecnologias de Localização em Smartphones

Após a análise das tecnologias de localização em Smartphones relevantes para o âmbito desta dissertação, é apresentado na Tabela 1 um comparativo entre estas. Para cada tecnologia é apresentada a técnica de localização, qual a exatidão e nível de precisão, se é uma tecnologia segura a nível da proteção dos dados e privacidade, e qual o nível de consumo de bateria. As técnicas de localização apresentadas são as identificadas para cada tecnologia como as mais adequadas para atingir os objectivos deste trabalho, pois conforme já referido, podem ser usadas várias técnicas de localização para a mesma tecnologia.

Tecnologia / Sistema	Técnica de Localização	Exatidão	Precisão	Seguro?	Consumo de Bateria
GPS	Triangulação	Entre 3 e 15 metros	Alta	Sim	Alto
WPS	Análise de Cenários	~100 metros	Média	Não, usam pontos de acesso e cadastros não controlados	Médio
Cell and Tower ID	Análise de Cenários	~1000 metros	Alta	Sim, apenas os operadores tem acesso ao histórico	Baixo
Bluetooth	Proximidade	~10 metros	Alta	Sim, desde a versão 2.1	Baixo, desde a versão 2.1

Tabela 1. Comparação entre Tecnologias de Localização em Smartphones

Na Tabela 2 é apresentado o resumo das vantagens e limitações das tecnologias de localização adequadas ao projeto que se pretende desenvolver.

¹² <http://www.differencebetween.net/technology/difference-between-bluetooth-2-0-and-bluetooth-2-1/> (2013/01/11)

Tecnologia / Sistema	Vantagens	Limitações
GPS	<ul style="list-style-type: none"> ➤ Melhor Precisão e Exatidão que WPS e Cell and Tower ID ➤ Garantia de Privacidade 	<ul style="list-style-type: none"> ➤ Consumo de Bateria Elevado comparado com WPS e Cell and Tower ID ➤ Fraca Cobertura em ambientes <i>indoor</i> ou ambiente <i>outdoor</i> com edifícios altos
WPS	<ul style="list-style-type: none"> ➤ Consumo de bateria inferior ao GPS ➤ Melhor Cobertura <i>indoor</i> e em ambientes urbanos <i>outdoor</i> do que o GPS 	<ul style="list-style-type: none"> ➤ Inseguro (usa Wi-Fi) ➤ Não garante Privacidade ➤ Volatilidade das posições dos pontos de acesso ➤ Precisão e Exatidão inferiores ao GPS
Cell and Tower ID	<ul style="list-style-type: none"> ➤ Consumo de bateria inferior ao GPS e ao WPS ➤ Melhor Cobertura <i>indoor</i> e em ambientes urbanos <i>outdoor</i> do que o GPS e o WPS 	<ul style="list-style-type: none"> ➤ Não garante Privacidade ➤ Muito pouco exata (~1000 metros)
Bluetooth	<ul style="list-style-type: none"> ➤ Fácil emparelhamento com outros terminais ➤ Permite reduzir a incerteza de posição através da consulta de outros terminais ➤ Baixo consumo de energia para versões > 2.1 	<ul style="list-style-type: none"> ➤ Inseguro para versões < 2.1 ➤ Só permite reduzir a incerteza da posição / localização, não fornece informação de localização se não existir uma infraestrutura dedicada (técnica baseada em proximidade)

Tabela 2. Vantagens e Limitações das Tecnologias de Localização em *Smartphones*

2.1.8 Magnetómetro, Giroscópio, Acelerómetro e Fusão de Sensores nos *Smartphones*

A maioria dos *Smartphones* vem munida de um sensor magnético, ou magnetómetro. O magnetómetro, também designado como bússola ou sensor de orientação na literatura estudada [29,30,39,40,41], serve para medir a força e/ou a direção do campo magnético da área envolvente. Este sensor permite usar o telefone como uma bússola electrónica. Outro sensor também comum nos *Tablets* e *Smartphones* é o giroscópio [29,30,39,40,41]. Este sensor permite determinar ou manter orientação, através da velocidade angular. Por último, um componente também presente em quase todos os *Smartphones* é o acelerómetro [29,30,39,40,41], que fornece dados sobre a aceleração própria, ou seja, a aceleração relativa à gravidade.

Um *Survey* sobre sensores em *Smartphones* está disponível em Lane *et al* [51]. A fusão de sensores [1,29,30,31,39,40,41] tem o objetivo de melhorar a precisão e a exatidão, integrando várias fontes de dados sensoriais.

A fusão dos sensores acelerómetro, giroscópio e magnetómetro é frequentemente citada como uma forma de calcular o posicionamento de pessoas e objetos em ambientes *indoor* [29,30,31], onde não existe sinal GPS ou fraca cobertura de redes celulares. Os sistemas de navegação inerciais (INS - *Inertial Navigation Systems*) usam acelerómetros e giroscópios, e podem também usar magnetómetros, para calcular continuamente a posição e o movimento [30]. A fusão de sensores pode incluir ainda outros sensores, como o *Bluetooth*, Wi-Fi, antena rádio e GPS, ou seja todos os sensores correspondentes à técnicas descritas anteriormente, aumentando assim a precisão e a exatidão do cálculo da posição geográfica. Para navegação pedestre, o *focos*

desta dissertação, é frequentemente usada a técnica de fusão dos sensores magnetómetro, giroscópio e GPS [39,40].

Em aplicações baseadas em localização, o acelerómetro dos *Smartphones* pode ser usado para determinar padrões de movimento [53,55], como correr, saltar ou andar. Um exemplo é o sistema UPTIME [52] que recorre a um módulo de detecção de passos para calcular a distância percorrida num percurso pedestre pré-definido.

Um exemplo da aplicação da fusão de sensores é, se o trilho a ser percorrido for conhecido, com base na velocidade a que se desloca o objeto que contém os sensores, na orientação do movimento determinada através da bússola, e no ponto geográfico lido pelo última vez no GPS, calcular a posição atual.

Esta técnica pode ser usada para garantir atualização da posição após perder o sinal GPS, ou para poupar bateria. No caso de algumas aplicações de apoio à navegação automóvel, quando o veículo entra num túnel e se perde o sinal GPS, a aplicação continua a atualizar a posição no mapa com um elevado grau de certeza. O sistema RAPS [3] usa uma estratégia de fusão de sensores para reduzir o nº de consultas GPS, e aumentar assim a vida da bateria do *Smartphone*. Outro exemplo que tira partido da fusão de sensores é, se o equipamento estiver parado (não existir detecção de movimento ou deslocação), não solicitar atualização de posição a outros sensores. A fusão de sensores nos *Smartphones* é aplicada das mais variadas formas, em sistemas de navegação, realidade aumentada, aplicações de reconhecimento de gestos e jogos baseados em movimentos.

A maior dificuldade em usar estes sensores do *Smartphone* é a precisão, ou seja os erros introduzidos devido ao tipo de movimentação do telefone, problemas de calibração ou interferência externa. Os magnetómetros podem ser afetados por interferência magnética, mais comuns em ambientes *indoor*, devido a ligações elétricas ou mobiliário de metal. Os acelerómetros podem necessitar de calibração, para garantir que a aceleração da gravidade medida é de $9,81\text{m/s}^2$, caso contrário o cálculo da aceleração linear vai ser incorreto [54]. Os giroscópios tem muito menos ruído que os acelerómetros e não são afectados por fontes exteriores como o magnetómetro, mas como medem a velocidade angular do telefone e não a posição angular, apenas o movimento relativo do telefone pode ser derivado das leituras.

2.2 Eficiência Energética nos Serviços Baseados em Localização em Smartphones

Conforme já referido, o GPS é usado em detrimento do Wi-Fi, Torres de Células e *Bluetooth*, por ser muito mais preciso e mais exato. As aplicações móveis baseadas em localização e com apoio à navegação usam preferencialmente o GPS para, com base na posição do *Smartphone*, fornecer a posição e informação de contexto (p.ex., proximidade de pontos de interesse).

Uma das maiores contrariedade da opção GPS é o consumo de bateria, mais dispendioso a nível da energia elétrica do que as outras tecnologias menos precisas e menos exatas. Uma estratégia possível é baixar o grau de exigência da precisão ou exatidão, usando tecnologias complementares ao GPS, como Wi-Fi (WPS), *Bluetooth* ou Torres de Células. O nível de exatidão ou precisão de uma aplicação baseada em localização (LBA -

Location-Based Application) pode variar ao longo do tempo, e à medida que o utilizador se desloca, ou seja, não necessitamos de usar sempre o GPS para determinar a posição atual. Outra das suposições que pode ser usada é a de que nem sempre é necessário calcular a localização, o utilizador pode estar estacionário e a usar uma funcionalidade que não precise de atualizar a posição. Assim a aplicação pode inibir-se de ativar o GPS e poupar energia. Uma forma de minimizar o nº de consultas GPS é usar o acelerómetro do *Smartphone* para estimar a velocidade a que o utilizador se desloca, ou determinar se o utilizador está parado. Com base nestas observações, em Liu *et al* [4] é proposta uma estratégia de eficiência energética para aplicações móveis com serviços de localização. Esta solução pretende minimizar o consumo de energia recorrendo a estimativas dinâmicas da próxima posição, por amostragem da velocidade, e por seleção do método mais eficiente a nível energético.

Existem várias abordagens possíveis para resolver o problema dos consumos do sensor de GPS. Na secção seguinte é apresentado o RAPS [3] que apresenta várias estratégias para melhorar a eficiência energética dos *Smartphones*, usando quase todas as técnicas de localização descritas na secção 2.1, inclusive a fusão de sensores, tirando partido do acelerómetro do Nokia N95. Na secção 2.2.2 é apresentado o EnTracked [7], que usa como estratégia para minimizar os consumos de bateria agendamentos da atualização da posição.

2.2.1 RAPS

Um dos sistemas desenvolvidos para contornar o problema dos altos consumos da interface GPS nos *Smartphones* designa-se RAPS [3], *Rate-Adaptive Positioning System for Smartphone Application*, projeto levado a cabo pelo Laboratório de Redes Embebidas, do Departamento de Ciência da Computação da *University of Southern California*. Além do objetivo principal de reduzir os consumos de bateria dos *Smartphones*, este protótipo propõe-se resolver o problema da precisão da posição GPS em meios urbanos.

Esquema	Histórico	Acelerómetro	Celltower-RSS-ID Blacklist	Bluetooth
RAPS	✓	✓	✓	✓
RAPS-B	✓	✓	✓	X
RAPS-BC	✓	✓	X	X
RAPS-BCA	✓	X	X	X
<i>Always-on</i>	GPS periódico com intervalos de 20 segundos			
<i>Periodic</i>	GPS periódico com intervalos de 180 segundos			

Tabela 3. Seis esquemas diferentes para avaliação do RAPS¹³

O RAPS tem 4 componentes principais: 1) Detecção do movimento do utilizador usando o acelerómetro do Smartphone; 2) Estimativa de velocidade e de incerteza de posição através de dados históricos espaço-tempo dos movimentos do utilizador; 3) Detecção de indisponibilidade de sinal GPS através de informação RSS

¹³ Adaptado de Paek *et al*[3]

(*Received Signal Strength*) enviada pelas torres GSM (*Celltower-RSS Blacklist*; 4) Redução da incerteza da posição através de Bluetooth.

Para avaliar a performance do RAPS foram usados 6 *Smartphones* Nokia N95-3, munidos com GPS, acelerómetro, *Bluetooth*, Wi-Fi e antena 3G/EDGE, cada um com a sua própria estratégia de localização configurada. As várias opções são apresentadas na Tabela 3.

O *Smartphone* designado RAPS tem todos os componentes habilitados, o RAPS-B tem o *Bluetooth* desabilitado, o RAPS-BC tem também desabilitado o *celltower-RSS Blacklist* e o RAPS-BCA apenas usa o histórico espaço-tempo. O *Smartphone* com o esquema *Always-on* corresponde à situação em que o GPS está continuamente ligado (leituras de 20 em 20 segundos) e o no esquema *Periodic* as leituras GPS são recolhidas de 180 em 180 segundos.

Todos os telefones tiram partido do GPS para determinação da posição, mas com intervalos de tempo diferentes. Para cada um dos componentes do RAPS, existe uma decisão de quando deve ser ativado o GPS para calcular a posição e minimizar o erro da posição. Relativamente à exatidão da posição geográfica, o RAPS ativa o GPS quando é estimado que a exatidão seja superior a 100 metros. A posição pode não ser precisa, uma vez que quando é corrigida a posição com base na leitura GPS, o utilizador pode já ter-se deslocado. Seja como for, este é o *trade-off* entre poupar energia e obter a posição com maior exatidão.

O esquema *Always-on* ativa o GPS de 20 em 20 segundos, no RAPS é ativado em média com uma frequência de 630,9 segundos, 588,5 segundos para o RAPS-B, o RAPS-BC usa em média intervalos de 259,5 segundos e o RAPS-BCA 134,4 segundos.

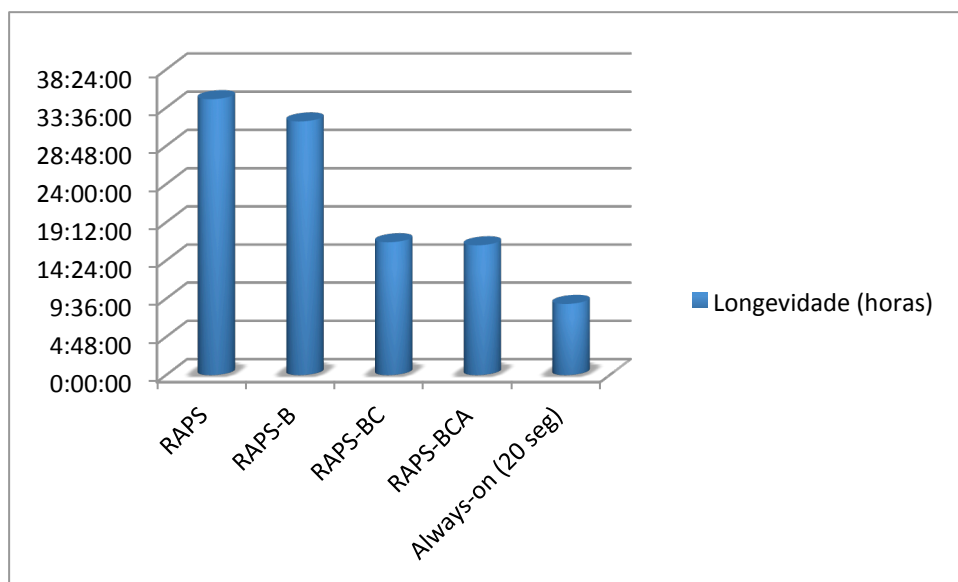


Figura 5. Longevidade da bateria (horas:minutos:segundos) de cada esquema de teste do RAPS¹⁴

¹⁴ Adaptado de Paek et al[3]

Cada telefone foi atualizado com informação espaço-tempo recolhida previamente, bem como informação das estações GSM sem cobertura GPS, e a bateria completamente carregada. Todos os telefones foram introduzidos num saco e carregados por um dos autores do artigo durante dois dias completos, dentro e nas proximidades do *campus* da USC, e efetuando os percursos normais da rotina diária, como deslocação entre casa e universidade, ir ao restaurante, ir ao supermercado, ir para a sala de aulas, laboratório, etc. Segundo os autores, não existiram repetições de percursos forçadas ou movimentos artificiais.

A experiência foi dada como concluída para cada terminal quando a bateria ficou abaixo dos 14%, e de seguida a aplicação foi explicitamente terminada. A experiência durou 36 horas para o esquema energeticamente mais eficiente e 9 horas para o que consumiu mais bateria.

Na Figura 5 são apresentados os resultados relativos à eficiência energética de cada uma das estratégias. O RAPS (com todos os mecanismos de localização ativos) é o mais eficiente, e o *always-on*, como se esperava, o menos eficiente. A nível de consumos de energia, com base na vida da bateria de cada um dos terminais e características de cada um dos sensores (*Bluetooth*, GPS e acelerómetro) foram estimados valores médios para cada um dos esquemas (Figura 6). A técnica *always-on* é a que consome mais bateria, os consumos do acelerómetro e do sensor *Bluetooth* são residuais nos esquemas RAPS, RAPS-B e RAPS-BC – continua a ser o GPS o sensor que contribui mais para o gasto de energia.

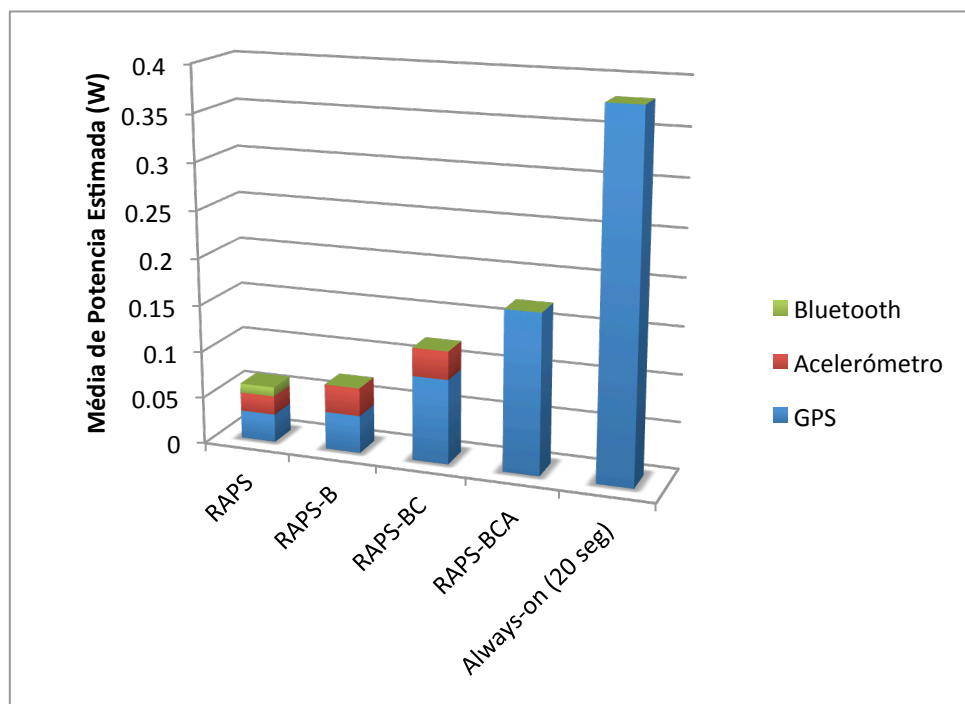


Figura 6. Consumo médio de energia estimado para cada esquema de teste do RAPS¹⁵

¹⁵ Adaptado de Paek et al[3]

As principais limitações deste sistema são a necessidade de inserir previamente no *Smartphone* dados das posições espaço-tempo e a *blacklist* das estações GSM, e os movimentos involuntários dos utilizadores que introduzem erro nas leituras do acelerómetro, baixando a precisão.

2.2.2 EnTracked

O sistema EnTracked (*Energy-Efficient Robust Position Tracking for Mobile Devices*) [7], desenvolvido pela Universidade de Aarhus (Dinamarca), baseia-se na estimativa e previsão de condições do sistema e mobilidade, e agenda atualizações de posições para reduzir o consumo de energia e otimizar a robustez. O EnTracked corre num *Smartphone* muito semelhante ao usado no RAPS [3], no Nokia N95 8GB, e o cenário usado para avaliar a solução envolve também percursos pedestres, com a diferença de servir para *tracking* (as atualizações da posição são enviadas para um servidor central via Internet). Para medir os consumos da bateria é usada a aplicação *Nokia Tool Profiler*¹⁶.

O objetivo do EnTracked é poupar energia da bateria, sem perder a exatidão. Os sensores usados nos algoritmos são o GPS e o acelerómetro. O acelerómetro serve para determinar se o utilizador está estacionário, caso esteja é desligada a atualização GPS. Para o cenário de validação proposto é considerado que um utilizador desloca-se a uma velocidade máxima de 10m/s num trilho conhecido pré-definido. É tido em conta o consumo de bateria da antena do terminal no envio da posição para o servidor.

Na Figura 7 é apresentada a lógica do cliente EnTracked que corre no *Smartphone* N95. O Servidor EnTracked envia um pedido de *tracking* ao *Smartphone*, indicando o requisito de exatidão (p.ex., 100 metros). Em (1) o cliente obtém a posição via GPS, e reporta-a de seguida ao servidor (2). Com base nas leituras do acelerómetro detecta se o *Smartphone* está em movimento ou parado (3). Se estiver em movimento, é calculada a velocidade através da próxima posição GPS (4). Com base na incerteza da exatidão do GPS, em (5) é calculado o próximo instante de tempo para nova leitura GPS. Em (6) são efetuados cálculos para saber quando desligar ou ligar o GPS e o rádio, tendo em conta os atrasos que se obtém em ligar ou desligar os sensores. Com os resultados dos cálculos obtidos no passo anterior, em (7) são agendadas ligações ou desligamentos do GPS ou do rádio. Quando se atinge o instante de tempo agendado para ligar o GPS, é reiniciado o processo.

Foram usados três esquemas energéticos para avaliar a solução: 1) o esquema EnTracked que usa o acelerómetro e efetua agendamentos de leituras GPS com base na incerteza, ou seja, todas as funcionalidades ativas; 2) o esquema EnTracked(β) que não usa o acelerómetro e 3) O esquema EnTracked(α) sem uso de acelerómetro e com uma só leitura GPS para estimar a velocidade. Os esquemas foram testados com requisitos de exatidão de 100 e 200 metros.

O esquema mais eficiente energeticamente para a exatidão de 100 metros foi o EnTracked (todas as funcionalidades ativas), e para 200 metros o EnTracked(β). Comparando com esquemas de atualizações de GPS periódicas recolhidas em laboratório (de 10 em 10 segundos para 100 metros e de 20 em 20 segundos para 200 metros), o EnTracked poupou em média 62,3% de bateria para a exatidão de 100 metros e o EnTracked(β)

¹⁶http://www.developer.nokia.com/Resources/Tools_and_downloads/Other/Nokia_Energy_Profiler/Quick_start.xhtml
(2013/01/11)

75,1% para a exatidão de 200 metros. A nível de robustez, o melhor foi o EnTracked, com uma média de erro de 24,8 metros para a exatidão de 100 metros e 24,8 metros para a exatidão de 200 metros.

Para aplicações de localização usadas em *tracking* ou redes sociais, uma exatidão entre 100 e 200 metros é aceitável, mas para outras aplicações como navegação pode ser inadequada. O requisito de 25 metros foi apenas testado em laboratório, e em emuladores, com consultas GPS periódicas. Os valores de leituras periódicas para os 100 metros (10 segundos) e 200 metros (20 segundos), foram também retirados de emuladores de Nokia N95.

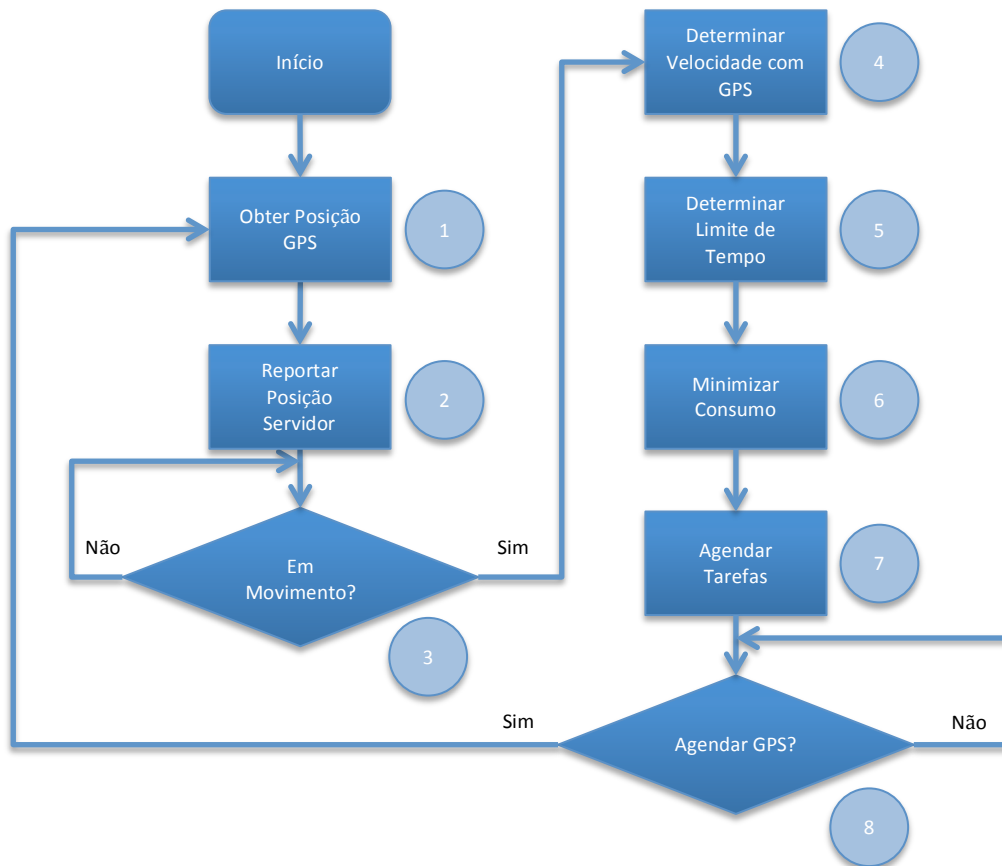


Figura 7. Fluxograma da lógica do cliente EnTracked¹⁷

Por forma a avaliar melhor os esquemas, deveriam ter sido repetidos os testes das leituras GPS periódicas em terminais reais, e testado nos três esquemas o comportamento da robustez e da eficiência energética para uma exatidão de 25 metros.

2.2.3 Comparação das Estratégias de Poupança de Energia do RAPS e do EnTracked

Na Tabela 4 é apresentada a comparação entre os sistemas analisados. Tanto o RAPS como o EnTracked usam a fusão de sensores como estratégia para minimizar os consumos de bateria dos *Smartphones*. Usam sempre o GPS para aumentar o grau de certeza da posição, mas recorrem a leituras menos periódicas para poupar

¹⁷ Adaptado de *Kjaergaard et al*[7]

energia. O sensor acelerómetro é utilizado nos dois sistemas para determinar se o *Smartphone* está parado e evitar assim leituras desnecessárias para atualização da posição. O RAPS tira um maior partido da fusão de sensores, usa além do acelerómetro as tecnologias GSM e *Bluetooth*. O *Bluetooth* diminui a incerteza na posição através da consulta da posição de outros *Smartphones* por esta mesma interface. Além do *Bluetooth*, usa duas técnicas de análise de cenários, a *Blacklist* de Torres de Células (para saber quais as localizações com fraca cobertura GPS e evitar leituras desnecessárias) e o histórico espaço-tempo (para reduzir a incerteza da posição). O RAPS é mais eficiente energeticamente do que o EnTracked, aparentemente devido às técnicas de análise de cenários referidas (na experiência relatada o *Bluetooth* contribui pouco para a poupança energética).

	GPS	Histórico Espaço-Tempo	Cell Tower ID BlackList	Bluetooth	Acelerómetro	Poupança de Energia
RAPS	✓	✓	✓	✓	✓	~85%
EnTracked	✓	X	X	X	✓	~70%

Tabela 4. Estratégias de Eficiência Energética do RAPS e do EnTracked

Ambos os sistemas comprovam que quanto mais periódicas forem as leituras GPS, menos tempo irá durar a bateria do *Smartphone*. A estratégia de consultar o acelerómetro para determinar se o *Smartphone* está parado e poupar leituras GPS parece ser eficaz. A estratégia no RAPS de usar o *Bluetooth* para evitar leituras GPS é interessante para aplicações móveis que estejam em uso por vários *Smartphones* em simultâneo no mesmo espaço geográfico. Usar técnicas de análise de cenários para reduzir a incerteza parece ser eficaz na experiência do RAPS, mas tem a desvantagem de ter de recolher e carregar previamente os dados históricos na aplicação. Conforme a origem ou a natureza dos dados históricos estes podem ser muito difíceis de recolher e manter.

2.3 Web Services em Aplicações Móveis com Serviços de Localização

Uma das maiores vantagens dos *Smartphones* é terem a capacidade de estar quase permanentemente ligados à Internet. Com o crescimento da banda larga móvel é possível ter cobertura Internet em quase todo o lado, e garantir sincronização de dados entre as aplicações móveis instaladas no equipamento e os servidores acessíveis via Internet.

Atualmente é possível aceder a quase todos os serviços Internet através de um *Smartphone*, o maior constrangimento é talvez o tamanho do ecrã e a falta de adequação de alguns sites aos *browsers* disponibilizados nas plataformas móveis. Os *sites* mais populares disponibilizam versões *mobile* das páginas para minimizar os problemas de *browsing*, e os próprios *browsers* permitem efetuar ações de *zoom* sobre as

páginas para permitir uma melhor visualização dos conteúdos. Alternativamente, alguns gigantes como o Facebook¹⁸, o YouTube¹⁹ e o LinkedIn²⁰ disponibilizam aplicações móveis no *marketplace* da Microsoft²¹, *Google Play*²² e *iTunes*²³ para melhorar a experiência de navegação dos conteúdos, evitando assim ter uma página customizada no site para *browsers* móveis, e fornecendo uma melhor experiência de utilização.

Além de *sites*, os *Smartphones* permitem sincronização com servidores de correio electrónico, servidores de mapas (apresentados na secção seguinte), servidores de notícias e serviços de informação (meteorologia, *feeds* de notícias, etc.), clientes de *streaming* para áudio e vídeo, com aplicações instaladas de origem no dispositivo.

Os SDKs das plataformas móveis permitem desenvolver aplicações que consomem *Web Services*, da mesma forma que as aplicações usadas em computadores pessoais, ou entre servidores. Os *Web Services*, baseados em mensagens XML (*Extensible Markup Language*) ou JSON (*JavaScript Object Notation*), protocolos SOAP (*Simple Object Access Protocol*), REST (*Representational state transfer*), usando por exemplo o transporte HTTP (*Hypertext Transfer Protocol*) - no caso do REST, só funciona via HTTP - têm ganho popularidade devido à heterogeneidade da Internet e à simplicidade na troca de mensagens, facilitando assim a implementação de comunicações e sincronizações cliente/servidor. Apesar de baseadas em mensagens de texto, é possível introduzir segurança nos *Web Services* através de SSL (*Secure Socket Layer*) no HTTP, e usando os mecanismos de segurança do SOAP.

O SOAP é um protocolo mais pesado que o REST. O REST é frequentemente escolhido para o desenvolvimento de consumo de *Web Services* em aplicações móveis por ser mais leve e mais intuitivo. As desvantagens do REST em relação ao SOAP são a falta de suporte para segurança e a obrigatoriedade do uso do HTTP como protocolo de transporte. Na maior parte dos casos os serviços são consumidos por aplicações que não foram desenvolvidas pelos criadores dos *Web Services*, e a escolha será determinada pelo servidor dos *Web Services*, se implementa REST ou SOAP, e se as mensagens são XML ou JSON.

Para serviços baseados em localização, a possibilidade de consumir *Web Services* que permitam obter informação de contexto sobre a posição em que o *Smartphone* se encontra, evita que a aplicação tenha de persistir todos os dados sobre pontos de interesse. Algumas aplicações móveis baseadas em localização e conscientes de contexto são discutidas em Pashten *et al* [34] e Schwinger *et al* [35]. Na secção seguinte, serão apresentados as *Google Maps APIs* e as *Bing Maps APIs*, que permitem consumir respectivamente os serviços *Google Maps* e *Bing Maps* em qualquer tipo de aplicação. Para aplicações em que seja um requisito controlar o percurso de um utilizador, por exemplo em aplicações de gestão de frota automóvel, podem ser utilizados *Web Services* que atualizem periodicamente e centralmente a posição do *Smartphone*.

¹⁸ <http://www.facebook.com/> (2013/01/11)

¹⁹ <http://www.youtube.com/> (2013/01/11)

²⁰ <http://www.linkedin.com/> (2013/01/11)

²¹ <http://www.windowsphone.com/marketplace/> (2013/01/11)

²² <https://play.google.com/store/> (2013/01/11)

²³ <http://www.apple.com/itunes/> (2013/01/11)

Além de consumidores, os *Smartphones* podem ser também servidores de *Web Services*, conforme avaliado em Berger *et al* [33]. Disponibilizar serviços a terceiros a partir do *Smartphone* pode trazer uma janela de oportunidades para as mais variadas aplicações, incluindo serviços de localização, como por exemplo disponibilizar vídeo *streaming* via HTTP para a internet ao vivo através da câmara do *Smartphone*, ou fornecer a localização do terminal por consulta direta, sem necessidade de um servidor de localização centralizado.

2.4 Mapas em Aplicações Móveis

Um mapa é a representação gráfica ou modelo de escala de conceitos espaciais. É um meio para transportar informação geográfica. Os mapas são um meio universal para comunicação, facilmente compreendidos e apreciados pela maioria das pessoas, seja qual for a sua língua ou cultura [8].

Os dispositivos com GPS incorporam mapas em formato electrónico, persistidos em suportes físicos presentes no próprio dispositivo (cartões SD, p.ex.) ou carregados via Internet em tempo-real. Os dispositivos usados para navegação automóvel, como os da Garmin²⁴, TomTom²⁵ ou NDrive²⁶ usam os mapas persistidos em cartões de memória, e as atualizações dos mesmos são efetuados através de um computador com acesso internet. A aplicação *Google Maps* utiliza a ligação de dados do dispositivo móvel para carregar os mapas em tempo-real.

Nas subsecções seguintes são apresentadas as diferenças entre mapas *offline* e *online*, e por fim são descritos os serviços *Google Maps APIs* e *Bing Maps APIs*.

2.4.1 Mapas Offline versus Mapas Online

Os mapas carregados em suporte físico são frequentemente usados em dispositivos GPS, e em várias aplicações de navegação desenvolvidas para *Tablets* e *Smartphones*. Exemplos de aplicações móveis que usam mapas persistidos em *Smartphones* são a *NDrive*, a *TomTom*, e a *Nokia Maps*²⁷ (usada em equipamentos Nokia com o Symbian OS). Existem milhares de aplicações de navegação para os *Smartphones*, umas transversais a todas as plataformas (Symbian, BlackBerry, Android, iPhone e Microsoft), outras específicas para cada sistema operativo.

A grande vantagem dos mapas *offline* é não necessitarem de ligação internet.

As desvantagens são: *i)* para aplicações de navegação automóvel, a necessidade de atualizar frequentemente os mapas, porque são criados novos troços de estradas e alterados sentidos de circulação diariamente; *ii)* na maior parte das aplicações, ocupação de espaço no suporte físico, é necessário carregar o mapa da cidade, de todo o país ou continente no dispositivo móvel, mesmo que só necessitemos de uma área delimitada; *iii)* Licenciamento. Cada vez que se necessita de atualizar um mapa, é necessário comprar a licença.

²⁴ <http://www.garmin.com/> (2013/01/11)

²⁵ <http://www.tomtom.com/> (2013/01/11)

²⁶ <http://www.ndrive.com/> (2013/01/11)

²⁷ <http://maps.nokia.com/> (2013/01/11)

Os mapas *online* têm como maior desvantagem, necessitar de ligação Internet ativa para poder funcionar no *Smartphone*. Inerentes a esta desvantagem, estão as dificuldades óbvias em usar a aplicação em zonas sem cobertura Internet, e os custos com transferências de dados, ainda mais elevados se o telemóvel estiver em *Roaming*. Como vantagens, têm-se as atualizações síncronas com o servidor de mapas, e pouca ocupação dos suportes de memória físicos. Algumas aplicações baseadas em mapas *online* usam técnicas de *caching* para as zonas circundantes à posição GPS, por forma a manter o funcionamento correto em caso de perda de conectividade Internet. Relacionado com os mapas *online* está o conceito de A-GPS (*Assisted-GPS*), em que a aplicação móvel usa a Internet para recolher informação necessária para a localização, como mapa e pontos de interesse. Exemplo de aplicações A-GPS são, a *Google Maps* e a *Google Maps Navigation*²⁸ para Android ou iPhone.

2.4.2 Google Maps APIs

O *Google Maps* é o serviço de mapas e informação geográfica da Google. Com o *Google Maps* é possível recolher *online* informação sobre localização de pontos de interesse, informações de contacto, direções detalhadas entre uma localização origem e uma localização destino, ter vistas satélite da qualquer localização, operações de *zoom*, *pan* e *tilt* sobre os mapas, vistas 3D com o *Earth view*²⁹ e o *Street view* [21] de alguns locais (maior cobertura dos EUA e Canadá), e ainda outras funcionalidades como customização e partilha de mapas.

Uma das mais interessantes funcionalidades para a comunidade de desenvolvimento de software, é a disponibilização dos *Google Maps API Webservice*³⁰, que podem ser invocados via HTTP, usando processamento de mensagens XML ou JSON. As *Google Maps APIs*, estão divididas em: *i) Directions API* (cálculo de direções entre localizações); *ii) Distance Matrix API* (cálculo de tempo e distancia para uma matriz de origens e destinos); *iii) Elevation API* (dada a latitude e longitude devolve a elevação); *iv) Geocoding API* (conversão de moradas para latitude e longitude); *v) Places API* (retorna informação sobre localizações geográficas e pontos de interesse). A disponibilização destes *Web Services* potenciou o desenvolvimento de sistemas de informação geográfica (p.ex. *GmapGIS*³¹), ou outras aplicações que tiram partido de serviços de localização geográfica³². Uma análise detalhada das APIs do *Google Maps* pode ser consultada no site *Georelated.com* [45].

2.4.3 Bing Maps APIs

O *Bing Maps* é o serviço de mapas e informação geográfica da Microsoft. Oferece todas as funcionalidades do serviço *Google Maps*, sendo uma das possíveis alternativas, inclusive para Android. O *Bing Maps* oferece as mesmas funcionalidades que o *Google Maps*, tais como operações de *zoom*, *pan* e *tilt*, criar *overlays* com primitivas geométricas, e outras adicionais como seleção de tipo de mapa, ou suporte de *OGC WMS Services*³³.

²⁸ <http://www.google.com/mobile/navigation/> (2013/01/11)

²⁹ <http://maps.google.com/earthview/> (2013/01/11)

³⁰ <https://developers.google.com/maps/documentation/webservices/> (2013/01/11)

³¹ <http://www.gmapgis.com/> (2013/01/11)

³² <http://googlemapsmania.blogspot.pt/> (2013/01/11)

³³ <http://www.opengeospatial.org/standards/wms> (2013/01/11)

Usam a vista *Bird's Eye View*, preferida por muitos utilizadores em detrimento da *Satellite View* do *Google Maps*, por ser mais detalhada. Para interagir com o serviço, são fornecidas muitas interfaces, tais como *Javascript*, *Silverlight Web RIA Controls*³⁴, e *Web Services* REST e SOAP. Estão disponíveis SDKs para as plataformas Windows Phone, Android [46] e iOS. Neste aspecto o *Bing Maps* ganha ao *Google Maps*, por disponibilizar uma maior variedade de interfaces.

As *Bing Maps APIs* fornecem nos *Web Services* SOAP³⁵, *Geocode Service* (procura de posição geográfica com base em atributos como endereço, nome, e vice-versa), *Imagery Service* (imagens da localização), *Route Service* (cálculo de rotas) e *Search Service* (procura de localizações por palavras chave). Usando as outras interfaces, como os *Bing Maps REST Services*³⁶, obtêm-se as mesmas funcionalidades. Uma análise das *Bing Maps APIs*³⁷ pode ser consultada no site Georelated.com [44].

³⁴ <http://www.silverlight.net/> (2013/01/11)

³⁵ <http://msdn.microsoft.com/en-us/library/cc981067.aspx> (2013/01/11)

³⁶ <http://msdn.microsoft.com/en-us/library/ff701702> (2013/01/11)

³⁷ <http://www.microsoft.com/maps/developers/> (2013/01/11)

3 Suporte no Android

Após a apresentação do estado da arte em técnicas e tecnologias de localização, sensores, estratégias de poupança de bateria, mapas de apoio à navegação no âmbito das aplicações móveis, e *Web Services* em aplicações móveis, e tendo em conta os objectivos desta dissertação, é pertinente analisar a sua existência e aplicabilidade na plataforma Android, na perspectiva do programador de aplicações móveis.

Na secção 3.1 é introduzida sumariamente a plataforma Android. De seguida, na secção 3.2, são apresentados os principais conceitos de programação em Android. Na secção 3.3 é descrito como podem ser utilizadas as técnicas e tecnologias de localização e fusão de sensores nesta plataforma. Na secção 3.4 são apresentadas algumas estratégias para poupar bateria. Em 3.5, é explicado como interagir com mapas via SDK do Android. Por fim, na secção 3.6 são descritas os métodos usados para consumir *Web Services* na plataforma Android.

3.1 Plataforma Android

O Android é uma plataforma aberta (*open source*) de desenvolvimento de aplicações para dispositivos móveis, suportada pela Google [5]. Inclui um sistema operativo baseado no *Kernel Linux*, um *Middleware* e um conjunto de aplicações nativas. O Android SDK fornece ferramentas e APIs necessárias para desenvolver aplicações na plataforma Android, usando a linguagem Java.

Na Figura 8 é apresentada a arquitetura da plataforma Android. A primeira camada corresponde ao conjunto de aplicações disponibilizadas de origem e desenvolvidas em Java, como o cliente de e-mail, o programa de envio e recepção de SMS, a aplicação calendário, o *Google Maps*, *browser*, contactos, e outras.

A segunda camada disponibiliza a *framework* de aplicações, onde o programador dispõe de acesso a funções específicas do hardware do equipamento onde corre o Android (GPS, câmara, etc.), de informação de localização, correr processos em *background*, adicionar notificações, etc. A arquitetura aplicacional do Android é desenhada de forma a facilitar a reutilização de código, qualquer aplicação pode publicar as suas capacidades, e outras aplicações podem usar essas mesmas capacidades. Por exemplo, se uma aplicação necessitar de tirar uma fotografia com a câmara do dispositivo, deverá de existir uma outra aplicação que já o faça, e pode ser usada sem que seja preciso desenvolver o código de raiz. Não é necessário incorporar ou ligar o código da aplicação que tira a fotografia, basta iniciar a atividade da aplicação que tira fotos. Também é possível redefinir componentes com este mecanismo.

Na camada seguinte residem as bibliotecas (C / C++), expostas ao programador pela *framework* de aplicações. A Android *runtime* contém as bibliotecas core da linguagem Java. É onde reside também a máquina virtual Java do Android (*Dalvik Virtual Machine*).

A última camada corresponde ao *Kernel Linux*, que trata de funcionalidades nucleares como a gestão de memória, gestão de processos e segurança. É onde reside o modelo de controladores para interação com o hardware (Wi-Fi, Áudio, Gestão de energia, etc.).

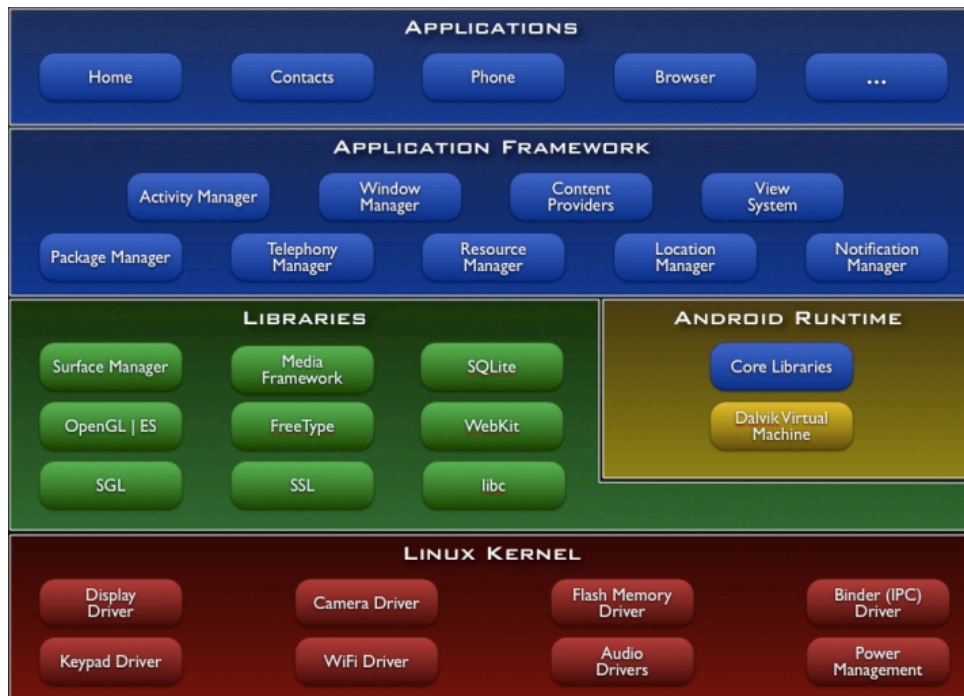


Figura 8. Arquitetura Android³⁸

O Android corre atualmente em milhares de *Smartphones* e *Tablets*, e começa a aparecer em outros dispositivos como televisores e *media centres*. Com o aparecimento do Android, potenciou-se o aparecimento de novos serviços baseados em localização (*Location-Based Services*) em *Smartphones*, conforme detalhado em Kushwaha *et al* [6], primeiro por ser uma plataforma aberta e facilmente adotada por uma ampla comunidade de programadores, depois por disponibilizar uma série de classes, bibliotecas e APIs que permitem interagir com o hardware dos *Smartphones* (GPS, Wi-Fi, antena rádio, acelerómetro, magnetómetro, etc.) e tirar partido de alguns serviços da *cloud* da Google como as *Google Maps APIs*³⁹.

3.2 Desenvolvimento de Aplicações em Android

Nesta secção são descritos sumariamente os conceitos mais importantes relativos ao desenvolvimento de aplicações em Android. Os detalhes presentes nesta secção podem ser consultados no tutorial *Android Development Tutorial*⁴⁰, no site *Vogella.com* [57], e no site *Android Developers* [5].

3.2.1 Segurança e Permissões

³⁸ Fonte: <http://developer.android.com/images/system-architecture.jpg> (2013/01/11)

³⁹ <http://maps.google.com/> (2013/01/11)

⁴⁰ <http://www.vogella.com/articles/Android/article.html> (2013/01/11)

Durante a instalação de uma aplicação num dispositivo Android, o sistema operativo vai criar identificadores únicos para o utilizador e o grupo de utilizadores da aplicação. Cada aplicação corre no seu próprio processo, ou seja, está isolada das outras aplicações. Caso seja necessário existir troca de dados entre aplicações, é necessário usar as classes `Service` ou `ContentProvider`⁴¹. O local onde são definidas as permissões da aplicação é no ficheiro de configuração `AndroidManifest.XML`⁴², por exemplo, se a aplicação tem permissões para aceder à Internet, ou permissões para escrever no cartão SD do *Smartphone* (*tag* `<uses-permission>`). As aplicações disponíveis no *Google Play*, que necessitem de permissões especiais (p.ex., acesso à Internet), irão pedir ao utilizador a autorização de acesso no ato das instalação, se o utilizador não autorizar a aplicação não será instalada. A aplicação para ser instalada no dispositivo necessita de ser assinada digitalmente pelo programador⁴³.

3.2.2 Interface (Activity, View, ViewGroup e Fragment)

Uma aplicação Android é composta por uma ou mais atividades (classe `Activity`⁴⁴). Uma `Activity` corresponde à representação gráfica de uma aplicação em Android. Todas as classes `Activity` da aplicação devem ser declaradas (*tag* `<activity>`) no ficheiro `AndroidManifest.XML`. Todas as classes que herdam de `Activity` terão de implementar o método `onCreate(bundle)` para inicializar a atividade. Neste método deve ser invocado o método `setContentView(int)`, que recebe o *resource id* do *layout* correspondente à sua interface, e associar os *widgets*⁴⁵ da interface a objetos da classe através do método `findViewById(int)`.

As atividades em Android são geridas sob a forma de pilha (*stack*). Quando uma atividade é inicializada, passa para o topo da pilha e passa a ser a atividade que está a correr (*running Activity*). A atividade anterior está na posição imediatamente anterior da pilha, e voltará para o topo quando a atividade atualmente a correr terminar a sua execução (*exit*).

Uma atividade em Android tem essencialmente quatro estados: 1) quando está visível no écran, está ativa ou *running*; 2) Se uma atividade está visível mas não está focada, está *paused*. Uma atividade no estado *paused* continua “viva”, mas por problemas de falta de recursos de memória no dispositivo Android pode ser reciclada pelo sistema, e destruída; 3) Se uma atividade está completamente obscurecida por outra atividade (não visível pelo utilizador), está no estado *stopped*. Será destruída pelo sistema quando for necessária memória para outra atividade; 4) Se a atividade está *paused* ou *stopped*, o sistema pode descartar a atividade da memória. Quando a atividade volta a ser apresentada ao utilizador, tem de ser completamente inicializada e restaurada para voltar ao estado anterior.

Na Figura 9 é apresentado o ciclo de vida de uma atividade no Android. O ciclo de vida completo (*entire lifetime*) ocorre entre a primeira vez que o método `onCreate(bundle)` é invocado, e a invocação de

⁴¹ <http://developer.android.com/reference/android/content/ContentProvider.html> (2013/01/11)

⁴² <http://developer.android.com/guide/topics/manifest/manifest-intro.html> (2013/01/11)

⁴³ <http://developer.android.com/tools/publishing/app-signing.html> (2013/01/11)

⁴⁴ <http://developer.android.com/reference/android/app/Activity.html> (2013/01/11)

⁴⁵ <http://developer.android.com/reference/android/widget/package-summary.html> (2013/01/11)

`onDestroy()`. O ciclo de vida visível (*visible lifetime*), ocorre entre a invocação dos métodos `onStart()` e `onStop()`. O ciclo de vida de *foreground* (visibilidade da atividade no ecrã) ocorre entre os métodos `onResume()` e `onPause()`.

As atividades usam vistas (classes `View` e `ViewGroup`) e fragmentos (classe `Fragment`) para criar as interfaces.

A classe `Fragment`⁴⁶ só é disponibilizada a partir do Android 3.0 (API level 11), e permite representar um subconjunto de uma interface de utilizador, ou seja, é uma secção de uma `Activity`. Os fragmentos são úteis para portar as interfaces das aplicações Android para dispositivos com ecrãs maiores, como os dos *Tablets*.

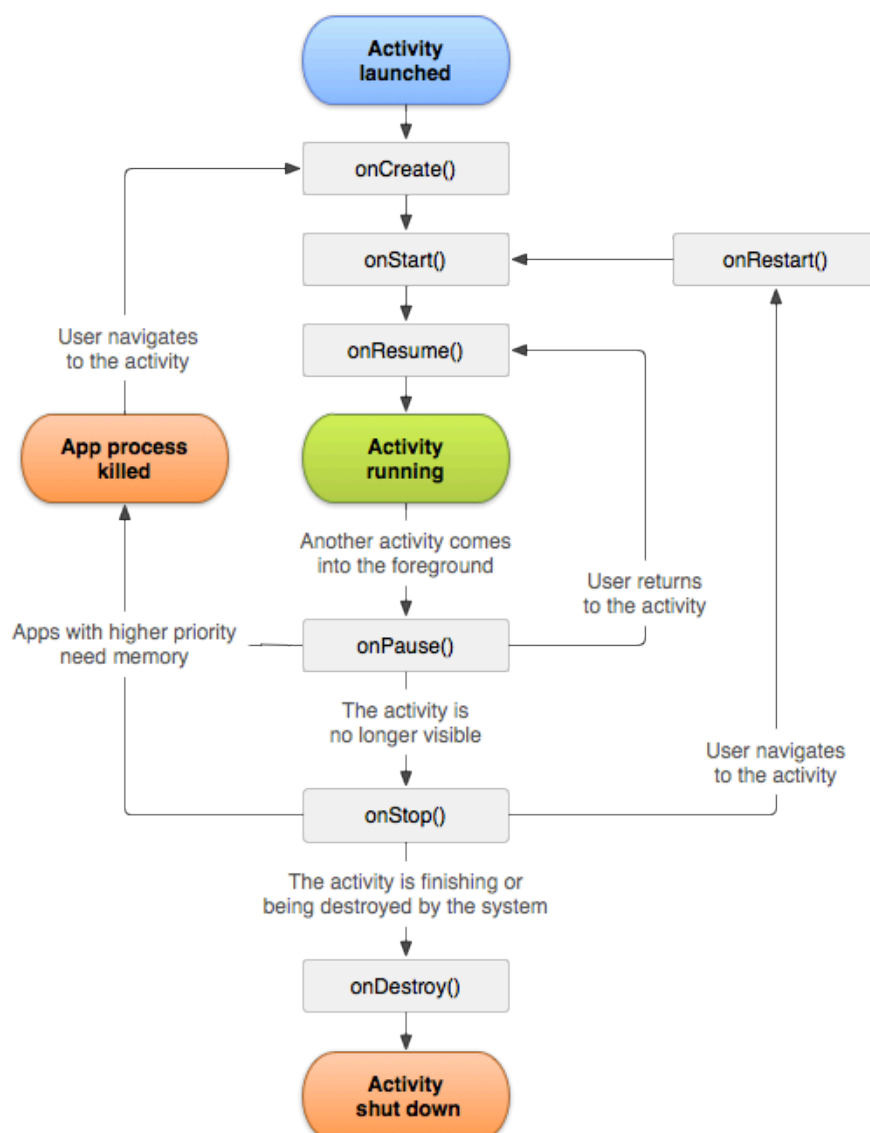


Figura 9. Ciclo de Vida de uma `Activity` na plataforma Android⁴⁷

⁴⁶ <http://developer.android.com/guide/components/fragments.html> (2013/01/11)

⁴⁷ Fonte: <http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle> (2013/01/11)

A classe `View`⁴⁸ representa uma área rectangular do écran, e é responsável por desenhar e tratar de eventos de todos os componentes de uma interface em Android (*widgets*), como por exemplo, caixas de texto (`TextView`⁴⁹), botões (`Button`⁵⁰) ou imagens (`ImageView`⁵¹). A classe `ViewGroup`⁵² é uma `View` especial que permite conter outras `View`, é útil para implementar *layouts* mais complexos, mas tem problemas de performance.

3.2.3 Intent

A classe `Intent`⁵³ representa mensagens assíncronas que permitem solicitar funcionalidades de outras componentes do sistema Android, entre atividades, ou entre uma atividade e um serviço. Pode ser usado para lançar uma nova atividade a partir da atividade atual (método `startActivity`), ou para enviar um `broadcastIntent` para uma série de serviços ou atividades (que tenham um `BroadcastReceiver`⁵⁴), ou lançar (`startService`) ou associar-se (`bindService`) a um serviço a correr em *background* no Android. A definição de serviço é apresentada na subsecção seguinte.

Os `Intent` podem ser explícitos (é explicitado qual a componente que deve ser chamada, por exemplo, uma atividade Android) ou implícitos (não é especificado qual o componente Android a ser chamado, por exemplo, abrir um URL).

Um `Intent` permite também passar dados entre atividades. A componente que cria o `Intent` pode passar dados sob a forma de (chave, valor) para o componente que é chamado, através do método `putExtra()`. A chave é sempre uma `String`, o valor pode ser um tipo de dados primitivo (`int`, `boolean`, `float`, etc.), ou `String`, `Bundle`, `Parcelable` e `Serializable`. A componente chamada pode aceder aos dados passados através do método `Bundle extras = getIntent().getExtras()`.

3.2.4 Processos, Threads e Serviços

Processos, *Threads*⁵⁵ e Serviços representam entidades diferentes no Android.

Um serviço (`Service`⁵⁶) Android é uma componente da aplicação que efetua tarefas em *background* sem interagir com o utilizador, ou que fornece funcionalidades para uso de outras aplicações. Um serviço não é um processo, corre no mesmo processo da aplicação. Um serviço não é uma *thread*, não corre fora da *thread* de execução.

Quando um componente de uma aplicação é iniciado, e a aplicação não tem qualquer componente a correr, o sistema operativo Android inicia um novo processo e uma única *thread* de execução. Por defeito, todos os

⁴⁸ <http://developer.android.com/reference/android/view/View.html> (2013/01/11)

⁴⁹ <http://developer.android.com/reference/android/widget/TextView.html> (2013/01/11)

⁵⁰ <http://developer.android.com/reference/android/widget/Button.html> (2013/01/11)

⁵¹ <http://developer.android.com/reference/android/widget/ImageView.html> (2013/01/11)

⁵² <http://developer.android.com/reference/android/view/ViewGroup.html> (2013/01/11)

⁵³ <http://developer.android.com/reference/android/content/Intent.html> (2013/01/11)

⁵⁴ <http://developer.android.com/reference/android/content/BroadcastReceiver.html> (2013/01/11)

⁵⁵ <http://developer.android.com/guide/components/processes-and-threads.html> (2013/01/11)

⁵⁶ <http://developer.android.com/reference/android/app/Service.html> (2013/01/11)

componentes de uma aplicação correm no mesmo processo e na mesma *thread*. Sempre que outro componente é iniciado, e o processo da aplicação já existe, então o componente corre dentro desse mesmo processo e na mesma *thread* de execução. Contudo, se for necessário associar componentes da aplicação a processos diferentes é possível fazê-lo no ficheiro `AndroidManifest.xml`, mas não aconselhável.

Os processos, como as atividades, também têm um ciclo de vida. O Android tenta manter o processo em memória o maior tempo possível, mas acaba por remover processos antigos ou menos importantes para obter memória para um novo processo. Para decidir que processos manter e descartar, o Android usa uma hierarquia de importância dos processos, com uma classificação de cinco níveis, sendo o nível mais importante o primeiro: 1) *foreground process* (processo necessário ao que o utilizador está atualmente a executar); 2) *visible process* (processo sem componentes visíveis, mas que pode afectar o que o utilizador vê no ecrã); 3) *service process* (um processo que está a correr um serviço); 4) *background process* (um processo de uma atividade que invocou o método `onStop()`); 5) *empty process* (um processo sem componentes de aplicação ativos).

Conforme já referido, quando a aplicação é lançada, o Android cria uma *thread* de execução, também designada de *main thread*. Esta *thread* é responsável por tratar dos eventos relacionados com os *widgets*, incluindo os eventos de desenho (*drawing events*) dos vários componentes da interface. Esta *thread* é também designada por *User Interface Thread*.

O Android fornece mecanismos para lançar novas *threads*. Podem ser lançadas *threads* designadas *Worker Threads* ou *Background Threads* (`new Thread(new Runnable() { ... }).start()`), que podem executar ações que não são instantâneas, mas que não podem aceder ao *Android UI Toolkit* (do uso exclusivo da *main thread*). A classe `AsyncTask`⁵⁷ permite efetuar operações em background e publicar os resultados na *main thread*.

O Android fornece um mecanismo de comunicação entre processos (IPC), usando chamadas de procedimentos remotos (RPC). Para implementar IPC, é necessário a aplicação associar-se ao um serviço (`bindService()`).

3.2.5 SDK, AVD, ADT e DVM

O *Android Software Development Kit* (SDK) contém as ferramentas necessárias para desenvolver, compilar e criar os *packages* de aplicações Android. A maioria destas ferramentas são baseadas em linha de comando. O Android SDK fornece um emulador, que permite que as aplicações sejam testadas sem ser necessário usar um dispositivo Android. Através do SDK podem ser criados vários *Android Virtual Devices* (AVD).

A Google fornece as *Android Development Tools* (ADT), para permitir desenvolver aplicações Android no Eclipse. As ADT são *plugins* que acrescentam ao IDE do Eclipse as funcionalidades do SDK. Com as ADT é possível criar, compilar, depurar e instalar aplicações Android a partir do Eclipse. Também permite criar e editar através do IDE, AVDs. As ADTs fornecem também ao Eclipse editores de recursos de aplicações Android, como p.ex., editor de *Layouts* ou o editor do `AndroidManifest.xml`.

⁵⁷ <http://developer.android.com/reference/android/os/AsyncTask.html> (2013/01/11)

A *Dalvik Virtual Machine* (DVM), é a máquina virtual Java usada pelo Android para correr as aplicações Java. A DVM usa um *bytecode* diferente da Java Virtual Machine, ou seja, os ficheiros java não podem ser corridos diretamente em Android sem serem primeiro convertidos para o formato *Dalvik bytecode*.

3.3 Técnicas e Tecnologias de Localização e Fusão de Sensores no Android

Todas as técnicas, tecnologias de localização e sensores descritos na secção 2.1 estão disponíveis na maioria dos *Smartphones* Android. A camada *Application Framework* apresentada na Figura 8 da secção 3.1, disponibiliza ao programador classes e métodos para interagir com estes sensores via SDK, e desenvolver assim técnicas de localização customizadas. Os detalhes referidos nesta subsecção podem ser consultados no *site Android Developers* [5].

O Android disponibiliza a classe `Location`⁵⁸, que representa uma localização captada em um determinado instante. Uma instância da classe `Location` contém a latitude, a longitude, o instante de tempo da atualização (UTC *timestamp*), e opcionalmente pode conter a altitude, a velocidade, e *bearing* (medida em graus entre o caminho percorrido e a direcção da bússola *East-of-true-North* para o destino). A classe `LocationManager`⁵⁹, permite aceder aos serviços de localização do sistema, e a classe `LocationListener`⁶⁰, usada para receber notificações da classe `LocationManager` quando a localização é alterada. O `LocationListener` pode usar os *providers* *i)GPS*, *ii)NETWORK*, que inclui as técnicas WPS (ou Wi-Fi MAC-ID) e Cell-ID, ou *iii)PASSIVE*. É possível criar vários objetos da classe `LocationListener` e associá-los ao objecto `LocationManager`. Por exemplo, se for criado um *listener* para o *provider GPS* e um *listener* para o *provider NETWORK*, o objecto `LocationManager` irá receber notificações cada vez que o GPS detecta uma nova posição, ou quando o *Smartphone* se associa a outro ponto de acesso Wi-Fi, ou quando detecta outra *Cell-ID*. Para cada `LocationListener` é definido na sua instanciação qual o intervalo de tempo de atualizações da posição (milissegundos) e qual a exatidão pretendida (metros). A classe `Criteria`⁶¹ permite indicar o critério de seleção de um *provider*. Os critérios podem ser instanciados tendo como *input* requisitos de exatidão, eficiência energética, e capacidade de reportar altitude, velocidade e *bearing*.

A primeira limitação que se pode retirar da *framework* é a impossibilidade de escolher qual o tipo de *provider* de rede, se só o WPS ou só o *Cell-Tower-ID*. A segunda limitação é a necessidade de definir a periodicidade das consultas aos sensores. Para estratégias de poupança de bateria em que se pretendam evitar consultas GPS, WPS ou *Cell-ID* periódicas, o *workaround* consiste em programar *timers* para agendar leituras da posição geográfica (usando as classes `Timer`⁶² e `TimerTask`⁶³).

⁵⁸ <http://developer.android.com/reference/android/location/Location.html> (2013/01/11)

⁵⁹ <http://developer.android.com/reference/android/location/LocationManager.html> (2013/01/11)

⁶⁰ <http://developer.android.com/reference/android/location/LocationListener.html> (2013/01/11)

⁶¹ <http://developer.android.com/reference/android/location/Criteria.html> (2013/01/11)

⁶² <http://developer.android.com/reference/java/util/Timer.html> (2013/01/11)

⁶³ <http://developer.android.com/reference/java/util/TimerTask.html> (2013/01/11)

Sensor	Tipo	Descrição	Aplicação	Versão Android		
				4.0	2.3	2.2
TYPE_ACCELEROMETER	Hardware	Mede a força de aceleração em m/s ² aplicada ao <i>Smartphone</i> nos três eixos (x,y,z), incluindo a força de gravidade.	Detecção de movimento	✓	✓	✓
THE_AMBIENT_TEMPERATURE	Hardware	Mede a temperatura ambiente em graus Celsius.	Monitorização de temperatura	✓	X	X
TYPE_GRAVITY	Software ou Hardware	Mede a força da gravidade em m/s ² aplicada ao <i>Smartphone</i> nos três eixos (x,y,z).	Detecção de movimento	✓	✓	X
TYPE_GYROSCOPE	Hardware	Mede o rácio de rotação em rad/s em redor dos três eixos (x,y,z)	Detecção de rotação	✓	✓	X
TYPE_LIGHT	Hardware	Mede o nível de luz ambiente em LUX ⁶⁴	Controlar o brilho do ecrã	✓	✓	✓
TYPE_LINEAR_ACCELERATION	Software ou Hardware	Mede a força de aceleração em m/s ² aplicada ao <i>Smartphone</i> nos três eixos (x,y,z).	Monitorar a aceleração em relação a um eixo.	✓	✓	X
TYPE_MAGNETIC_FIELD	Hardware	Mede o campo magnético ambiente nos três eixos (x,y,z) em μ T.	Bússola	✓	✓	✓
TYPE_ORIENTATION	Software	Mede os graus de rotação do equipamento em redor dos três eixos (x,y,z).	Determinar posição do equipamento	✓	✓	✓
TYPE_PRESSURE	Hardware	Mede a pressão do meio ambiente em hPa ou mbar.	Monitorar alterações de pressão	✓	✓	X
TYPE_PROXIMITY	Hardware	Mede a proximidade de um objeto em relação ao ecrã do <i>Smartphone</i> em cm.	Posição do telefone durante uma chamada	✓	✓	✓
TYPE_RELATIVE_HUMIDITY	Hardware	Mede a humidade relativa do ambiente (%).	Monitorar humidade	✓	X	X
TYPE_ROTATION_VECTOR	Software ou Hardware	Mede a orientação do <i>Smartphone</i> fornecendo o vector de rotação do equipamento.	Detecção de movimento e de rotação do <i>Smartphone</i>	✓	✓	X
TYPE_TEMPERATURE	Hardware	Mede a temperatura do equipamento em graus Celcius.	Monitorar temperatura	✓	✓	✓

Tabela 5. Sensores suportados pela Plataforma Android⁶⁵

⁶⁴ <http://www.thefreedictionary.com/lux> (2011/01/13)

Desta forma, e se quisermos usar várias técnicas de localização em simultâneo, a bateria irá ser consumida rapidamente se utilizarmos intervalos de tempo curtos para acerto da posição. Para minimizar esta limitação, o Android permite implementar um *listener* correspondente ao *provider* `PASSIVE`. Este *provider* recebe atualizações de posição de outras aplicações ou serviços, sem pedir explicitamente a correção da localização, ou seja, passivamente. Poderá ser útil se tivermos outra aplicação a correr no *Smartphone* (p.ex., *Google Latitude*), que já implemente um `LocationListener` para o *provider* `GPS`, e evitando ter assim dois processos a solicitarem leituras GPS em instantes diferentes.

Se usarmos os *Google Maps*, descritos mais à frente na secção 3.6, podemos usar a classe `MyLocationOverlay`⁶⁶. Esta classe já implementa *location listeners* (`GPS` e `NETWORK`), e abstrai o programador da implementação descrita nos três parágrafos anteriores. Ao escolher o uso dos *Google Maps* e da classe `MyLocationOverlay`, não faz sentido instanciar novos *listeners* de localização, porque estaríamos a usar o mesmo *listener* em duplicado. A desvantagem em usar esta classe é não poder definir a periodicidade das leituras. A vantagem, é passar para o Android a complexidade de gestão da localização (p.ex., passar a receber *updates* do *provider* `NETWORK` quando se perde o sinal GPS de forma automática).

Relativamente ao *Bluetooth*, a versão 2.1 do protocolo foi introduzida na versão Android 2.0, aumentando a segurança e a velocidade, e baixando os consumos de bateria no uso deste componente de *hardware*. Através das *Bluetooth APIs* disponibilizadas pelo Android, é possível *i)* Procurar por dispositivos *Bluetooth*; *ii)* Consultar o sensor *Bluetooth* (classe `BluetoothAdapter`⁶⁷) para determinar que dispositivos estão emparelhados; *iii)* Estabelecer canais `RFCOMM` (*Radio Frequency Communication*); *iv)* Ligar a outros dispositivos por *Service Discovery*; *v)* Transferir dados por *Bluetooth*; *vi)* Gerir várias ligações em simultâneo. Desta forma, é possível através de *Bluetooth*, determinar a posição através de outros dispositivos que correm a mesma aplicação e comuniquem esta informação entre si.

A nível dos sensores, a plataforma Android disponibiliza a classe `Sensor`⁶⁸, que representa os sensores do *Smartphone*. Além do giroscópio (`TYPE_GYROSCOPE`), acelerómetro (`TYPE_ACCELEROMETER`) e magnetómetro (`TYPE_ORIENTATION`), existem outros sensores disponíveis, como o sensor de temperatura ou o sensor de proximidade, este último útil para detectar toques indevidos nos ecrãs *touchscreen*, por exemplo, toque da orelha no ecrã quando se está em conversação. Na Tabela 5 são apresentados os sensores disponíveis nas versões 4.0 (API Level 15), 2.3 (API Level 9) e 2.2 (API Level 8) do Android.

A classe `Sensor` disponibiliza o método `getSensorList(int)` que retorna a lista de sensores disponíveis no equipamento. A classe `SensorManager`⁶⁹ permite ao programador aceder aos sensores do equipamento, a classe `SensorEventListener`⁷⁰ recebe notificações da classe `SensorManager`

⁶⁵ Adaptado de http://developer.android.com/guide/topics/sensors/sensors_overview.html (2013/01/01)

⁶⁶ <https://developers.google.com/maps/documentation/android/reference/com/google/android/maps/MyLocationOverlay> (2013/01/11)

⁶⁷ <http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html> (2013/01/11)

⁶⁸ <http://developer.android.com/reference/android/hardware/Sensor.html> (2013/01/11)

⁶⁹ <http://developer.android.com/reference/android/hardware/SensorManager.html> (2013/01/11)

⁷⁰ <http://developer.android.com/reference/android/hardware/SensorEventListener.html> (2013/01/11)

quando os valores do sensor são alterados, e a classe `SensorEvent`⁷¹ representa os eventos dos sensores. A nível dos sensores de movimento dos *Smartphones* já referidos nesta secção – acelerómetro, giroscópio e magnetómetro – a partir versão 2.3 do sistema operativo Android é incluída na API dos sensores, fusão de sensores. O sensor `TYPE_LINEAR_ACCELERATION` corresponde à fusão dos sensores `TYPE_ACCELEROMETER` e `TYPE_GYROSCOPE`. O sensor `TYPE_GRAVITY` usa também o sensor `TYPE_ACCELEROMETER` e um filtro *Butterworth*⁷² para calcular a força da gravidade.

Conforme já referido, pode ser necessário calibrar o acelerómetro nos *Smartphones*. No Android, podem ser usadas aplicações disponíveis no *Google Play*, como por exemplo a aplicação *Calibrate the app*⁷³, ou aceder ao menu de serviço do *Smartphone* (no LGE P-500, digitar no teclado de telefone 3845#*500#) e escolher a opção de calibrar acelerómetro (*Device Settings->Calibrate Sensor->Accelerometer*).

3.4 Eficiência Energética nos Serviços baseados em Localização na Plataforma Android

Conforme já referido no capítulo anterior, o GPS é o sensor que consome mais energia (pode drenar a bateria de um *Smartphone* em cerca de 12 horas ou menos), e as formas mais comuns de poupar energia são usar outras tecnologias alternativas ao GPS (menos precisas) para acertar a posição, e aumentar o intervalo de tempo das atualizações GPS. Por outro lado, se tivermos em simultâneo vários sensores ativos, a bateria irá ser consumida rapidamente.

A classe `Criteria` apresentada na secção anterior, permite definir critérios relacionados com o consumo de bateria. Podemos definir como critério para receber atualização da localização no `LocationManager`, `setPowerRequirement (Criteria.POWER_LOW)`. Se fizermos isso, e não existirem outros critérios como a exatidão (`Criteria.ACURRACY_FINE`), o método `getBestProvider` irá devolver o `provider NETWORK`. As técnicas de poupança de bateria passam por avaliar quais os requisitos de exatidão versus os de consumo de bateria. Se a exatidão necessária for entre 20 e 1000 metros, podemos usar as triangulações WPS e Torres de Células (`provider NETWORK`), e poupar assim energia. Se necessitarmos de melhor exatidão e usar o GPS, o critério de consumo de energia não pode ser exigente.

Para além da criação de critérios e seleção automática de `providers` com base nesses mesmos critérios, a *framework* Android não fornece quaisquer mecanismos dinâmicos de eficiência energética nas técnicas de localização.

As estratégias de poupança de energia no Android passam por desativar e ativar atualizações de `providers` ou `listeners` de sensores, e dos mecanismos de economia de bateria dos *Smartphones* Android.

Uma boa prática para poupar energia, é tirar partido do ciclo de vida das atividades do Android para desligar `listeners` dos sensores. O ciclo de vida de uma `Activity` está representado na Figura 9 na secção 3.2. Os métodos `OnResume()` e `OnPause()` devem respetivamente ativar e desativar `listeners`, desativando sensores que não vão ser usados em outras atividades.

⁷¹ <http://developer.android.com/reference/android/hardware/SensorEvent.html> (2013/01/11)

⁷² <http://www.websters-online-dictionary.org/definition/Butterworth+filter> (2013/01/11)

⁷³ <http://luciddreamingapp.com/android/help-how-to/calibrate-the-app/> (2013/01/11)

Outra boa prática no desenvolvimento de aplicações de localização, é desligar atualizações de um *provider* que fornece informação que não serve os requisitos da aplicação. Com base nos métodos `getAccuracy()` e `hasAccuracy()` disponibilizados pela classe `Location`, se o nosso requisito de exatidão for de 30 metros e a exatidão de localização retornada for de 100 metros, podemos desativar as atualizações do *provider* que forneceu essa localização, e poupar assim energia. Esta ação deve ser tomada quando se recebe a atualização da localização no método `onLocationChanged()` do `LocationListener`.

Uma das estratégias mais usadas para reduzir os consumos de energia nas aplicações com serviços de localização na plataforma Android é a fusão de sensores. A estratégia pode ser usada para, conforme as condições, ativar ou desativar *providers* de localização, e recolher informação de localização de outros sensores que consomem menos energia. Em Zhuang *et al* [32], além da descrição das limitações da plataforma Android relativas à consciência energética das técnicas de localização, é apresentada uma proposta de *framework* para o Android, que permite tornar aplicações baseadas em localização nos *Smartphones* mais eficientes a nível de consumo de bateria. No protótipo apresentado neste artigo [32], são usados os sensores acelerómetro e magnetómetro para determinar se o *Smartphone* está parado. Se estiver, desliga os *listeners* de localização, reduzindo os custos energéticos.

Outra estratégia é não usar os *listeners* de localização, ou seja, não usar as tecnologias GPS, WIFI e GSM. A aplicação ANDROID UPTIME [52], que usa uma técnica usada em navegação marítima (*Dead Reckoning*), que consiste em calcular a posição atual com base na posição anterior, a distância percorrida, e o curso do movimento, tira apenas partido do acelerómetro e da bússola. Apenas é necessário ter a posição inicial, retirada do GPS, WIFI ou GSM. O acelerómetro é responsável por detectar passos, e também por estimar a distância de cada passo. A bússola serve para determinar a orientação dos passos.

3.5 Web Services na Plataforma Android

Conforme descrito na secção 2.3, a ligação Internet nos *Smartphones* potencia o uso de *Web Services* para comunicar com servidores, ou permitir ao próprio *Smartphone* disponibilizar serviços. Para interagir com o serviço *Google Maps* na plataforma Android, não é necessário invocar diretamente os *Web Services* das *Google Maps APIs*, as invocações dos serviços são encapsuladas nas classes disponibilizadas pela *framework*, e descritas na secção anterior.

Para aplicações que necessitem de consultar um servidor para recolher informação, por exemplo, consulta de pontos de interesse numa determinada posição, ou que necessitem de fornecer dados a um servidor, por exemplo comunicar a posição atual, a forma mais simples de implementação e também a mais usada é via HTTP e com troca de mensagens XML ou JSON.

A forma mais simples e estruturada de receber informação, é através de POST ou GET de mensagens XML ou JSON. O Android disponibiliza as classes `DefaultHttpClient`, `HttpPost`, `HttpGet`, `HttpResponse` e `HttpEntity` para efetuar o pedido HTTP, e guardar em memória as respostas HTTP. Uma vez em memória, a mensagem pode ser processada usando o DOM (*Domain Object Model*) ou um *parser*

SAX (*Simple API for XML*), se a mensagem for XML. A escolha entre um e outro será definida primeiro por uma questão de preferência, segundo por questões de memória do dispositivo. O DOM necessita de mais memória do que o SAX, para mensagens muito extensas deve ser usado um *parser* SAX. As bibliotecas são as mesmas disponibilizadas para a linguagem JAVA, no JRE (*Java Runtime Environment*) Standard. Faltam contudo na plataforma Android outras bibliotecas para *parsing* de XML como a *StAX (Streaming API for XML)* e a *Java XML Binding API*. Para processar mensagens JSON, a plataforma Android disponibiliza as bibliotecas `json.org`, as mesmas usadas no JRE Standard.

Se for necessária a implementação de um cliente SOAP numa aplicação móvel para Android, não existe nenhuma biblioteca disponível de origem, ou seja, o protocolo tem de ser todo implementado manualmente usando as classes e métodos disponíveis para o HTTP. Para resolver este problema, existe disponível uma biblioteca *open source* designada `ksoap2-android`⁷⁴, que permite implementar rapidamente um cliente SOAP em Android.

Os clientes REST são implementados com maior simplicidade do que os clientes SOAP, e são também menos pesados para o *Smartphone*. Para aplicações Android que necessitem de processamento rápido, e não existam requisitos de segurança, deve ser usado o REST em detrimento do SOAP.

3.6 Mapas na Plataforma Android

Desde a versão 2.0 do Android, que a aplicação *Google Maps* é disponibilizada grátis nos *Smartphones* com este sistema operativo. A plataforma Android disponibiliza no SDK um pacote Java (`maps.jar`) que permite utilizar as *Google Maps APIs* (apresentadas na secção 2.4) no desenvolvimento de aplicações móveis.

Para usar as *Google Maps API*, é necessário obter a *Maps API Key*⁷⁵. Para a versão 1 das *Google Maps API's*, uma vez obtida a chave, esta deve ser copiada para o atributo XML `apiKey` da `MapView`, presente no ficheiro XML de *Layout* onde vai ser apresentado o mapa. O próximo passo é criar uma classe Java que herde de `MapActivity`, e que relacione a `MapView` XML com uma instância da classe `MapView`. A `MapView` corresponde à visualização dos *Google Maps*, e disponibiliza vários métodos de interação como, habilitar os controlos de *zoom*, habilitar a vista satélite, adicionar *overlays* sobre o mapa (classes `Overlay`, `ItemizedOverlay` e `OverlayItem`), que podem corresponder a trilhos, direções, pontos de interesse, ou outro qualquer elemento gráfico que se deseje apresentar, ou centrar o mapa num ponto geográfico, entre outras funcionalidades. Um tipo de *overlay* especial é obtido através da instância da classe `MyLocationOverlay` (já descrita na secção 3.3), que permite apresentar de uma forma expedita a localização atual do *Smartphone* (implementando *listeners* de localização), e mostrar também uma bússola sobre o mapa (apresentando a sua orientação através da implementação da classe `SensorListener`, e tirando partido do magnetómetro). Caso não se opte pela classe `MyLocationOverlay`, será necessário

⁷⁴ <http://code.google.com/p/ksoap2-android/> (2013/01/11)

⁷⁵ <https://developers.google.com/maps/documentation/android/mapkey> (2013/01/11)

implementar os *location listeners* (GPS, NETWORK e/ou PASSIVE), bem como o *OverLay* que contenha o ponto de localização do dispositivo.

A maior limitação da aplicação *Google Maps* e das *Google Maps APIs* na plataforma Android, é a necessidade de ter no dispositivo uma ligação Internet ativa, induzindo assim custos com telecomunicações, mais elevados quando em *Roaming*, ou impossibilidade de uso quando não existe qualquer cobertura Internet (Wi-Fi, GPRS, 3G ou 4G).

Em Julho de 2011, a Google introduziu a opção “*Labs*” na versão 5.8.0 da aplicação *Google Maps*, que entre outras funcionalidades de teste (laboratório), disponibiliza a possibilidade de efetuar o pré-carregamento de uma área de 10 milhas quadradas (~16 Km²), permitindo assim utilizar a aplicação numa porção de mapa, sem ligação de dados ativa. Contudo, devido a restrições de licenciamento da própria Google, os mapas não podem ser persistidos em memória através do SDK do Android, mesmo após da inclusão desta nova funcionalidade.

As alternativas atuais mais usadas para tornar esta dificuldade são o OSMDROID [9], que usa os mapas grátis *OpenStreetMap* [10] e o SDK do ESRI ArcGis [11]. Existe também a possibilidade de usar o *Bing Maps*, através do *Bing Maps Android SDK* [46]. Os mapas são criados através da aplicação *Mobile Atlas Creator*⁷⁶ num PC ou Mac, e depois podem ser persistidos no cartão SD do *Smartphone* sob a forma de ficheiros, ou em BD SQLite. Além do formato OSMDroid, a aplicação permite gerar outros formatos como os das aplicações Android *LOCUS*⁷⁷, *RMAPS*⁷⁸ e *ORUXMAPS*⁷⁹, e para outras plataformas móveis (iPhone e Windows Phone).

A alternativa mais interessante é a da ESRI, uma vez que dispõe de vistas de satélite, e com os mapas mais atualizados que as restantes. Os mapas disponibilizados pelo serviço *OpenStreetMap* são menos ricos que os da *Google* porque não têm vistas satélite, são mapas de estradas. O *Bing Maps Android SDK*, por não ser mantido diretamente pela Microsoft, tem menos suporte que as duas opções anteriores, pois depende de terceiros a integração entre Android e as *Bing Maps APIs*.

A versão 1.1 do SDK ArcGis para Android, disponibilizada em Abril de 2012, permite usar mapas pré-carregados no cartão SD do telemóvel ou *Tablet*, com o formato *compact cache*, que pode ser gerado e exportado a partir do ESRI ArcGISServer. Um exemplo de navegação em modo *offline* está disponibilizado na *sample* do SDK designada “*LocalTileLayer*”. Além disso, o SDK ArcGis permite usar outros servidores de mapas além dos da ESRI, como os serviços de mapas *Google Maps* ou os *Bing Maps* da Microsoft.

⁷⁶ <http://mobac.sourceforge.net/> (2013/01/11)

⁷⁷ <https://play.google.com/store/apps/details?id=menion.android.locus&hl=en> (2013/01/11)

⁷⁸ <https://play.google.com/store/apps/details?id=com.robert.maps&hl=en> (2013/01/11)

⁷⁹ <https://play.google.com/store/apps/details?id=com.orux.oruxmaps> (2013/01/11)

4 mtAndroid – Arquitetura, Desenho e Implementação

Neste capítulo é apresentada a forma mais adequada de implementar as técnicas e tecnologias de localização e fusão de sensores, estratégias de eficiência energética, mapas e *web services* em *Smartphones* Android, tendo em conta os objectivos desta dissertação.

Para este efeito foi desenvolvido um protótipo de uma aplicação móvel, designado por **mtAndroid**, usando o Android SDK (Google APIs Levels 8, 9 e 15).

Na secção 4.1 é descrito o problema específico de localização que o protótipo se propõe solucionar. De seguida, em 4.2, é apresentada a arquitetura e tecnologias envolvidas. Na secção 4.3 é apresentado o modelo de domínio, e na secção 4.4 o formato dos dados. A proposta de implementação dos mapas é descrita na secção 4.5 e, finalmente, em 4.6 são apresentadas as técnicas de localização e estratégias de eficiência energética a implementar no protótipo.

4.1 Requisitos Gerais

Conforme referido no Capítulo 1, considerou-se neste trabalho como exemplo de referência de uma aplicação móvel com serviços de localização e apoio à navegação em percursos pedestres *outdoor* sobre mapas a aplicação *SIQuant MobileTrails* [62, 64], desenvolvida para a plataforma Windows Mobile. A aplicação *MobileTrails* permite apoiar e complementar a experiência das visitas turísticas, em percursos pedestres, a parques naturais, e a centros urbanos ou históricos das cidades.



Figura 10. Protótipo *mtAndroid*

Para avaliar os desafios no desenvolvimento de aplicações móveis com serviços de localização e de navegação pedestre em trilhos *outdoor* sobre mapas, desenvolveu-se uma aplicação que introduz pontos de melhoria relativamente à aplicação *SIQuant MobileTrails*, mas que a considera como modelo de referência. Esta aplicação protótipo de avaliação designa-se **mtAndroid**.

O cenário a ser validado pelo protótipo é genericamente o seguinte:

“Aplicação móvel que apoia um turista numa caminhada, que percorre um trilho outdoor pré-definido, usando mapas digitais que apresentam os trilhos delineados, com indicação da posição geográfica atual, notificações de afastamento do trilho, notificações de proximidade de pontos de interesse, e que disponibiliza a partir do mapa informação adicional sobre os trilhos e pontos de interesse, espécies e eventos.”

O **mtAndroid** deve satisfazer os seguintes principais requisitos:

- Importar e atualizar via Internet a base de dados com os conteúdos necessários ao correto funcionamento da aplicação no terminal Android;
- Detetar a posição geográfica de cada utilizador a cada momento através de várias tecnologias de localização existentes no Android e apresenta-la no écran do dispositivo;
- Informar o utilizador sempre que se afasta do trilho em 100 metros;
- Informar o utilizador da proximidade de um ponto de interesse a 30 metros;
- Fornecer informação sobre espécies (reino animal e reino vegetal) e eventos (históricos, culturais), e relacioná-los com pontos de interesse;
- Usar estratégias de poupança de energia (minimizar os consumos de bateria principalmente devido ao uso do GPS);
- Garantir uma exatidão de 10 metros no posicionamento.

4.2 Arquitetura



Figura 11. Arquitetura do protótipo *mtAndroid*

Na Figura 11 é apresentada a arquitetura de camadas do protótipo *mtAndroid*. A camada de apresentação contém os *layouts* da aplicação. A camada seguinte contém a lógica de negócio, onde são implementadas as funcionalidades, e são garantidas as interfaces com o serviço *Google Maps* e o servidor da aplicação. A terceira camada implementa o modelo de domínio descrito na secção 4.3, e a última camada corresponde à base de dados da aplicação.

No arranque, a aplicação verifica se existem atualizações dos conteúdos. Os conteúdos estão alojados num servidor HTTP na Internet. A primeira vez que a aplicação corre, irá carregar todos os conteúdos disponíveis. Os dados mais relevantes para o correto funcionamento da aplicação são atributos dos pontos de interesse e dos trilhos, nomeadamente a informação geográfica dos mesmos.

Outros dados complementares são os relativos às espécies e aos eventos. Para desenvolvimento do protótipo, considerou-se um modelo de domínio apresentado na secção 4.3. Na secção 4.4 é descrito o formato dos conteúdos. Os conteúdos são disponibilizados em ficheiros XML, que referenciam também as imagens (ícones e fotografias), em formato texto, codificadas em BASE64. Todas as imagens foram codificadas usando um codificador disponível na Internet⁸⁰. A aplicação processa os ficheiros através de um DOM *parser*, e importa-os para uma base de dados SQLite (motor de base de dados nativo da plataforma Android). A primeira vez que se corre a aplicação, é também criada a base de dados.

⁸⁰ <http://www.motobit.com/util/base64-decoder-encoder.asp> (2013/01/11)

A persistência dos mapas usando a solução ArcGISServer citada no capítulo 3, secção 3.6, exige um servidor devidamente licenciado para o efeito. As soluções disponibilizadas *online* sem custos de licenciamento são menos interessantes porque não disponibilizam vistas de satélite, apenas estradas. Por estas razões, no âmbito desta dissertação, tomou-se como decisão de implementação do protótipo para validar a solução, o uso da API da Google para interagir com os mapas, que necessita da ligação Internet ativa. Quando não existe Internet, é colocada uma imagem do mapa gravada previamente. A forma como vão ser usados os mapas está detalhada na secção 4.5.

Para garantir exatidão da posição geográfica e acesso ao serviço *Google Maps* é necessário ter, respectivamente, o GPS e a ligação Internet (Wi-Fi/GPRS/3G/HSPDA/4G) ativos. Devem ser utilizados mecanismos que, caso não exista sinal GPS, garantam o cálculo da localização através do operador de rede móvel ou redes Wi-Fi. As técnicas de localização a serem usadas na aplicação serão descritas na secção 4.6.

4.3 Modelo de Domínio

Na Figura 12 é apresentado o diagrama UML [63] do modelo de domínio para os trilhos, pontos de interesse, categorias, *tags*, espécies e eventos.

Um ponto de interesse (POI) é caracterizado por um identificador (p. ex., *id="ist.utl.pt"*), por um nome legível (p. ex., *name="Instituto Superior Técnico"*), por uma descrição (p.ex., *description="Campus da Alameda do Instituto Superior Técnico da Universidade Técnica de Lisboa"*), pela latitude e pela longitude, e por uma imagem. Um POI pode estar relacionado com vários trilhos. Um trilho (Trail), é caracterizado por um identificador (p.ex., *id="trilhoISTAlamedaFaculdadeCiencias"*), um nome legível (p.ex., *name="Trilho IST-> FC"*), uma descrição (*description="Trilho entre o Instituto Superior Técnico e a Faculdade de Ciências"*), uma imagem, a descrição do ponto inicial, a descrição do ponto final, o total de Km do trilho, e o total de horas previstas que demora a percorrer, e um mapa, que corresponde a uma imagem do mapa a ser usada quando os *Google Maps* não estão disponíveis por falta de ligação Internet. Um trilho tem dois ou mais pontos geográficos (GeoPointTrail), ordenados (atributo *order*). A ordem irá definir como é desenhado o trilho. Os pontos de interesse podem estar relacionados com trilhos, e trilhos com pontos de interesse. A um trilho podem estar associados percursos (Route), caracterizados por o instante de tempo de início do percurso, o instante de tempo atual do percurso, o nº de horas passado desde o início, a distância percorrida em Quilómetros até ao instante atual, e a localização atual (latitude e longitude). A um percurso pertencem um ou mais pontos geográficos ordenados (GeoPointRoute).

Pontos de interesse e trilhos pertencem a uma só categoria (entidade POICategory para os POIs e entidade TrailCategory para os trilhos). As categorias dos POIs podem pertencer a várias categorias (POICategory). As categorias (classe abstracta *Category*) são caracterizadas por um identificador, um nome legível (p.ex. *Name="Escolas Ensino Superior"*) e por uma descrição (p.ex. *Description=" Esc. Ens. Superior"*), e um ícone.

Pontos de interesse e trilhos podem ser caracterizados por TAGs genéricas, pares (nome, valor), por exemplo, *name="morada"* e *value="Avenida Rovisco Pais"*. As entidades TagPOI e TagTrail servem para inserir

informação variada sobre os pontos de interesse e os trilhos, respectivamente, e disponibilizar mais informação para as entidades POI e Trail, mas não obrigatórias para o correto funcionamento da aplicação.

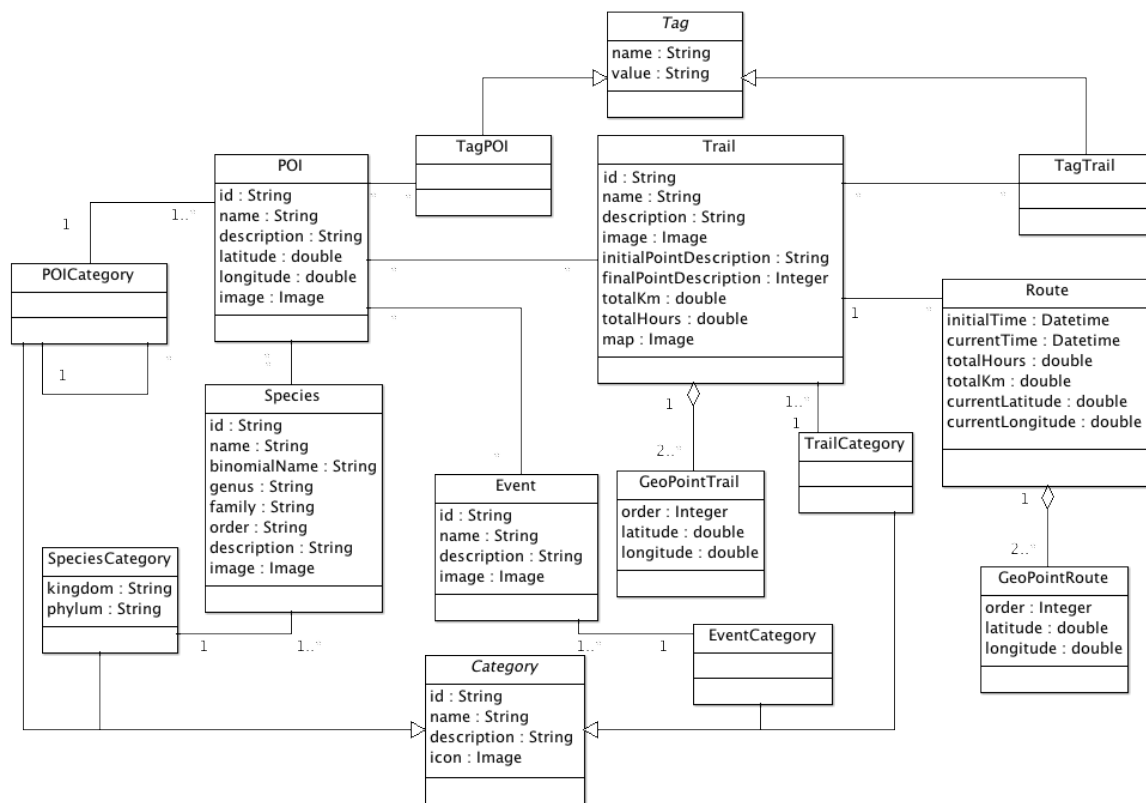


Figura 12. Diagrama UML do Modelo de Domínio

Aos pontos de interesse podem estar associadas uma ou mais espécies. Cada espécie pode estar associada a vários pontos de interesse. Uma espécie é caracterizada por um identificador, um nome legível, um nome binominal (nome científico da espécie), um género, uma família, uma ordem, uma descrição e uma imagem. Uma espécie pertence a uma categoria (SpeciesCategory) caracterizada por um reino (animal, vegetal) e um filo. As categorias das espécies são categorias (Category).

Os pontos de interesse podem estar associados a um ou mais eventos. Cada evento pode estar associado a um ou mais pontos de interesse. Um evento é caracterizado por um identificador, um nome legível, um tipo (histórico, cultural, etc.) uma descrição, e uma imagem. Um ou mais eventos pertencem a uma categoria.

4.4 Conteúdos

Os conteúdos da aplicação estão disponibilizados on-line, e sob a forma de ficheiros XML. Nas subsecções seguintes são descritos os ficheiros XML, usados para fornecer os conteúdos ao protótipo da aplicação. Todos os ficheiros estão no formato `encoding="ISO-8859-1"` para suportar os caracteres especiais da língua portuguesa, p.ex., "ç", "ã", etc. Os ficheiros XML são importados da Internet por um DOM XML Parser, e pela

seguinte ordem: categorias das espécies, lista de espécies, categorias dos eventos, lista de eventos, categorias de pontos de interesse, categorias dos trilhos, lista de trilhos e lista de pontos de interesse.

4.4.1 Categorias

Todas as categorias são representadas em ficheiros independentes. As categorias dos POIs estão presentes no ficheiro `CategoriesPOIs.xml`, as categorias dos trilhos no ficheiro `CategoriesTrails.xml`, as categorias das espécies no ficheiro `CategoriesSpecies.xml` e finalmente as categorias dos eventos no ficheiro `CategoriesEvents.xml`.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<categories>
  <category>
    <id>estacoesCp</id>
    <name>Estações de Caminhos de Ferro</name>
    <description>Estações de Caminhos de Ferro de Portugal</description>
    <icon>iVBOR...ggg==</icon>
  </category>
  <category>
    <id>praias</id>
    <name>Praias</name>
    <description>Praias de Portugal</description>
    <icon>R0lG...ADs=</icon>
  </category>
</categories>
```

Figura 13. Exemplo do ficheiro `CategoriesPOIs.xml`

Na Figura 13 é apresentado um exemplo do ficheiro XML que contém as categorias dos POIs. Todos os ficheiros XML têm o elemento `<categories>` que representa a lista de categorias. Cada categoria é representada pelo elemento `<category>` que contém sempre os elementos `<id>`, `<name>`, `<description>` e `<icon>`. O ícone é representado em BASE64, na Figura 13 foi omitido o conteúdo de `<icon>` para facilitar a visualização. Adicionalmente, para o caso das espécies, existem também as TAGs `<kingdom>` e `<phylum>`. Para representar a relação recursiva na classe `POICategory` (uma categoria POI pode pertencer a várias categorias POI), existe para as categorias dos POIs o elemento `<mainCategories>`, que irá conter a lista de identificadores das categorias mãe. Por exemplo, o código XML `<mainCategories><mainCategory><id>category1</id><mainCategory><mainCategory><id>category2</id></mainCategory></mainCategories>` indica que `category1` e `category2` são categorias às quais a categoria pertence.

Todos os elementos atrás descritos correspondem aos atributos definidos para as categorias (`Category`, `SpeciesCategory`, `POICategory`, `TrailCategory`, `EventCategory`) no modelo de domínio representado na Figura 12.

4.4.2 Espécies e Eventos.

Da mesma forma, existirão ficheiros XML para representar espécies e eventos (`SpeciesLists.xml` e `Events.xml`).

À semelhança dos ícones das categorias, as imagens das espécies e dos eventos vão ser codificados em BASE64, os restantes atributos (*id*, *name*, *description*, *binomialName*, *genus*, *family*, *order*, *image*) corresponderão a *tags* XML do elemento `<species>`. Devido ao plural de *species* ser a mesma palavra, a lista de espécies é representada por `<speciesList>`. Para os eventos o elemento que representa a lista é o `<events>`, que contém um ou mais elementos `<event>`, com as *tags* *id*, *name*, *description* e *image*. As imagens das espécies e dos eventos são representadas em BASE64.

4.4.3 Trilhos e Pontos de Interesse

Apesar de a W3C ter em estado *draft* uma especificação [12,13,14] que permite representar POIs e trilhos em XML, ainda não existe uma norma para a representação de trilhos e pontos de interesse. Na representação da W3C um trilho é considerado um ponto de interesse. Neste formato, os trilhos podem ser representados com o elemento `<Line>` presente na tag `<Location>`, que contém o elemento `<posList>` com todos os pontos geográficos (pares latitude e longitude) do trilho.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<pois>
  <poi>
    <id>CTTSaoPedroEstoril</id>
    <name>CTT São Pedro do Estoril</name>
    <latitude>38.696397</latitude>
    <longitude>-9.372962</longitude>
    <description>Estação de Correios de São Pedro do Estoril.</description>
    <category><id>estacoesCtt</id></category>
    <tags><tag><name>horário</name><value>09:00-18:00</value></tag>
    </tags>
    <events>
      <event>
        <id>joysticksArrival</id>
      </event>
    </events>
    <speciesList>
      <species>
        <id>melro</id>
      </species>
    </speciesList>
    <trails>
      <trail>
        <id>trilhoSaoPedroEstorilCTTPraia</id>
      </trail>
    </trails>
    <image>/9j/... /9k=</image>
  </poi>
</pois>
```

Figura 14. Exemplo do ficheiro POIs.xml

Embora o formato permita disponibilizar informação de pontos de interesse de uma forma normalizada, e facilitar a troca de POIs com outras aplicações, não fazia sentido estar a adaptar o modelo de domínio ao formato da W3C, porque iria contra o paradigma da modulação orientada a objetos. Como tal, foi definido o formato exemplificado na Figura 14 e na Figura 15 para representar os POIs e os trilhos.

Na Figura 14, é apresentado um exemplo do ficheiro POIs.xml, que contém apenas um POI. A lista de POIs é representada pelo elemento <pois>. Cada ponto de interesse é representado pelo elemento <poi>, caracterizado pelas tags *id*, *name*, *latitude*, *longitude*, *description* e *image*. As imagens são representadas no formato BASE64.

O elemento <category> indica que o POI pertence à categoria com *id=estacoesCtt*. O elemento <tags> contém a lista de tags que caracterizam o POI, neste caso só uma <tag> com *name="horário"* e *value="09:00-18:00"*. Além das tags, o POI da figura contém listas de espécies, eventos e trilhos relacionados, cada uma só com um elemento para facilitar a visualização. A lista de categorias dos POIs (ficheiro CategoriesPOIs.xml) terá de ser carregada antes deste XML.

Por decisão de implementação, optou-se por definir que as relações entre a classe POI e as restantes classes (Trail, Event, Species) é representada apenas no POI, ou seja, é no XML que contém a lista de pontos de interesse que são inseridas as ligações dos POIs aos trilhos, espécies e eventos. No exemplo apresentado o POI *id= CTTSaoPedroEstoril* está relacionado com o trilho *id=trilhoSaoPedroEstorilCTTPraia*, com o evento *id=joysticksArrival* e com a espécie *id=melro*.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<trails>

  <trail>
    <id>trilhoSaoPedroEstorilCTTPraia</id>
    <name>Sao Pedro do Estoril, CTT -> Praia</name>
    <description>Trilho que se inicia nos CTT de São Pedro do Estoril e que termina
na praia de São Pedro do Estoril
    </description>
    <initialPointDescription>CTT se São Pedro do Estoril</initialPointDescription>
    <finalPointDescription>Praia de São Pedro do Estoril</finalPointDescription>
    <totalKm>2.1</totalKm>
    <totalHours>0.2</totalHours>
    <category>
      <id>trilhosCascais</id>
    </category>
    <geoPoints>38.696397 -9.372962 38.696325 -9.37225 38.693776 -9.369476</geoPoints>
    <image>
      /9j/...

    <image>

  </trail>
</trails>
```

Figura 15. Exemplo do ficheiro Trails.xml

Na Figura 15 é representado o trilho relacionado com o ponto de interesse da Figura 14. Este ficheiro é carregado antes do anterior para que o POI possa referenciar um trilho existente.

A tag `<category>` identifica a categoria do trilho (`id=trilhosCascais`), que terá de ser carregada anteriormente no ficheiro `CategoriesTrails.xml`.

O elemento `<geoPoints>` contém a lista de pares longitude e latitude correspondentes à classe `GeoPointTrail`, separados por espaços. Este elemento representa os pontos que servem para delinear o trilho. O atributo `order` será dado de forma sequencial, neste caso a lista de `GeoPointTrail` será: (`latitude=38.696397` , `longitude=-9.372962` , `order=1`); (`latitude=38.696325` , `longitude=-9.37225` , `order=2`); (`latitude=38.693776` , `longitude=-9.369476` , `order=3`).

Todos os restantes elementos XML correspondem aos atributos da classe `Trail` apresentados no modelo de domínio. A imagem do trilho é codificada em BASE64.

4.5 Mapas e Representação de Trilhos e Pontos de Interesse

Os mapas propostos para a solução do problema são os disponibilizados pela Google. O SDK do Android fornece através das *Google Maps APIs*, a possibilidade de interagir com o serviço *Google Maps*, e utilizar uma série de mecanismos nativos disponíveis, como funcionalidades de *zoom*, centrar mapa num ponto geográfico, e acrescentar camadas (*Overlays*) aos mapas, tais como pontos de interesse e trilhos.

No mapa vai ser apresentada a vista satélite do *Google Maps*, e vai-se permitir usar os mecanismos de *zoom* nativos da `MapActivity`.

O protótipo apresenta os mapas na opção “Percurso” do menu inicial. É calculado o nível de *zoom* e qual o ponto geográfico do centro do trilho, por forma a visualizar o trilho completo. O ponto central é determinado calculando quais os pares (latitude, longitude) máximo e mínimo, entre todos os pontos do trilho.

Os pontos de interesse são representados pelos ícones das categorias importados para a base de dados da aplicação. Ao clicar sobre o ícone no mapa é apresentada a sua descrição, e restantes informações presentes na base de dados. Para apresentar os ícones no mapa, são usadas as classes `Overlay`, `ItemizedOverlay` e `OverlayItem` apresentadas na secção 3.6. O *overlay* que contém os *itemizedOverlays* é de seguida adicionado à `mapView`.

Para representar o trilho é usada a classe `Overlay`, os pontos são desenhados através do método `drawOval` e o trilho pelo método `drawPath`, ambos da classe `Canvas`, sobre o mesmo `Overlay`, que é depois adicionado à `MapView`.

Na apresentação do mapa para navegação é utilizada a classe `MyLocationOverlay` descrita na secção 3.6, que permite desenhar no ecrã a bússola e a localização atual do *Smartphone*.

Para que o mapa seja apresentado no ecrã, é necessário ter a ligação internet ativa. Caso não exista ligação internet, os trilhos e os pontos de interesse serão mesmo assim desenhados, bem como a indicação da posição do utilizador, e é possível usar as tecnologias de localização descritas na secção seguinte. A experiência visual será contudo mais pobre. Para melhorar a experiência visual, é apresentada uma imagem do mapa do trilho quando não existe Internet, mas sem acesso aos controlos de interação com os *Google Maps* (Zoom, Pan, etc.).

Para obter as imagens dos mapas dos trilhos, foi adicionada a funcionalidade de gravar mapas dos trilhos quando se tem uma ligação de dados ativa. A função de gravar é apresentada nos Percursos, grava o mapa sem *Overlays* para o cartão SD e atualiza a base de dados SQLite com esse mesmo mapa. As imagens gravadas no cartão SD podem ser depois convertidas para BASE64, e copiadas para o ficheiro XML dos trilhos, por forma a existir um mapa por defeito do trilho. Conforme o tamanho do écran, o mapa gravado pode ficar com resolução diferente. Nos dois *Smartphones* usados para testar os últimos protótipos (Sony Ericsson Xperia X10 e Samsung Galaxy Note N-7000) os mapas gravados apresentaram imagens com resolução diferente. Por isto manteve-se esta funcionalidade que pode ser útil para outros modelos com tamanho de écran diferente.

Sempre que a aplicação detetar que não existe Internet no menu de navegação, dará um aviso sobre a indisponibilidade do serviço de dados, mas continuará a funcionar, e adicionará um *Overlay* que contém a imagem do mapa à *MapView*.

4.6 Técnicas e Tecnologias de Localização com Eficiência Energética

Conforme referido no capítulo 3 (secções 3.2 e 3.3), com base nas tecnologias, sensores, estratégias e boas práticas enunciadas é possível implementar algoritmos de localização energeticamente eficientes. Contudo, se quisermos tirar partido da *framework* Android e dos *Google Maps* para implementar de uma forma transparente a visualização da posição em mapas, ou seja, usar as classes já disponibilizadas pelas *Google Maps* APIs, será mais difícil personalizar e adaptar as técnicas pré-definidas.

A primeira abordagem foi instanciar três *LocationListener* (GPS, NETWORK, PASSIVE), e um *LocationManager*, conforme apresentado no capítulo 3 (secção 3.3), em conjunto com a classe *MyLocationOverlay*, para apresentar a posição nos *Google Maps*. A tecnologia de localização preferida para determinar a localização era o GPS. Os valores da frequência das atualizações e da exatidão eram definidos após testes em trilhos, para determinar quais os *thresholds*, que garantiam o melhor rácio exatidão/consumo bateria. Se o sinal GPS se perdia, devido a condições atmosféricas ou a obstáculos físicos, o *listener* era desativado, e a posição continuava a ser atualizada pelo *provider* NETWORK.

O problema em usar esta abordagem é já existirem *listeners* de localização a correr no objeto da classe *MyLocationOverlay*. Estava-se a usar assim o dobro do consumo de energia para obter leituras do GPS. A maior parte dos exemplos disponibilizados *on-line* com a classe *MyLocationOverlay* usam esta técnica, o que é muito pouco eficiente a nível de consumo de bateria. Além disso, achou-se interessante avaliar o consumo de energia da aplicação recorrendo apenas à classe *MyLocationOverlay*. Na subsecção 4.6.2 é apresentada a forma de avaliação.

Dado que a aplicação *mtAndroid* irá ser utilizada em ambiente *outdoor*, em caminhadas pedestres com pouca cobertura Wi-Fi, o *provider* NETWORK poderá não garantir um cálculo preciso da posição geográfica, como se pretende neste tipo de aplicações. Seja como for, caso o GPS se torne indisponível, a classe *MyLocationOverlay* irá receber atualizações de posição do *provider* NETWORK. Nos testes realizados, o

provider NETWORK parece apenas ser utilizado no cálculo da posição inicial (método `runOnFirstFix()` da classe `MyLocationOverlay`), a preferência é sempre pelo *provider* GPS quando este está disponível.

Sempre que a atividade está em pausa (`onPause()`), parada (`onStop()`), ou é destruída (`onDestroy()`) são desligados os *listeners* de localização (através do método `disableMyLocation()` da classe `MyLocationOverlay`), e desligadas as atualizações do sensor magnetómetro (através do método `disableCompass()` da classe `MyLocationOverlay`). Estes dois métodos permitem evitar consultas desnecessárias, e poupar assim bateria. Desta forma, só no ecrã da aplicação correspondente à `MapActivity` é que estão ativos os *listeners* de localização e as leituras do magnetómetro.

Sempre que os *listeners* de localização detetam uma alteração da localização (método `onLocationChanged(Location)` da classe `MyLocationOverlay`), são verificadas duas condições: 1) Sempre que existe uma alteração de posição geográfica irá ser verificado se existe algum ponto de interesse perto da localização atual (< 30 metros). Caso exista, é usado o vibrador do *Smartphone* (classe `Vibrator`⁸¹), para vibrar durante 1 segundo, e lançado para o ecrã um *pop-up* (`Toast`⁸²) a indicar qual o ponto de interesse mais próximo e qual a distância deste em metros; 2) Sempre que o utilizador está afastado do trilho (>100 metros), será enviado um *Toast* e uma vibração de 1 segundo a avisar que se está a afastar do trilho. Sempre que existe alteração da localização, e o percurso foi iniciado, é atualizado o estado do percurso (latitude e longitude atuais, velocidade média, distância percorrida).

Para calcular a distância entre pontos GPS foi usado um método próprio (método `gps2m` da classe `pt.utl.ist.meic.tese.mtAndroid.util.GPSDistanceMeters`), em detrimento dos métodos `distanceTo()`⁸³ ou `distanceBetween()`⁸⁴ disponíveis na classe `Location`. É relatado em alguns fóruns da Internet [43] sobre programação em Android que estes métodos retornam por vezes leituras erróneas (com erros que podem ir até 100 metros). A solução passa por calcular a distância num método próprio. O método implementado na aplicação *mtAndroid* calcula a distância com base na fórmula de *Haversine*⁸⁵:

$$\begin{aligned} a &= \sin^2(\Delta\phi/2) + \cos(\phi_1).\cos(\phi_2).\sin^2(\Delta\lambda/2) \\ c &= 2.\text{atan2}(\sqrt{a}, \sqrt{1-a}) \\ d &= R.c \end{aligned}$$

A constante R representa o raio da Terra (6371 Km). A variável d é a distância (nas unidades de R). As variáveis ϕ e λ correspondem respectivamente à latitude e à longitude, ambas em radianos.

⁸¹ <http://developer.android.com/reference/android/os/Vibrator.html> (2013/01/11)

⁸² <http://developer.android.com/guide/topics/ui/notifiers/toasts.html> (2013/01/11)

⁸³ [http://developer.android.com/reference/android/location/Location.html#distanceTo\(android.location.Location\)](http://developer.android.com/reference/android/location/Location.html#distanceTo(android.location.Location)) (2013/01/11)

⁸⁴ [\(2013/01/11\)](http://developer.android.com/reference/android/location/Location.html#distanceBetween(double,%20double,%20double,%20double,%20float))

⁸⁵ <http://www.movable-type.co.uk/scripts/latlong.html> (2013/01/11)

Outras formas de calcular a distância entre pontos (latitude, longitude) estão disponíveis no site *Movable Type Ltd*⁸⁶.

4.6.1 Acelerómetro e fusão de sensores

O sensor mais referido pela literatura estudada nesta dissertação sobre tecnologias de localização, que permite minimizar as leituras do GPS e poupar assim bateria, é o acelerómetro. A versão 2.3 do Android introduz o sensor `TYPE_LINEAR_ACCELERATION` que é a fusão do acelerómetro com o giroscópio, simplificando a matemática e processamentos necessários para obter a aceleração linear do dispositivo (aceleração absoluta menos a aceleração da gravidade) nos três eixos referenciais (x, y e z). Este sensor permite calcular a velocidade a que o *Smartphone* se desloca através da integração da aceleração linear, e efectuado nova integração é possível calcular o deslocamento, conforme detalhado em Seifert *et al* [54]. Infelizmente os erros introduzidos em cada integração são enormes, porque nem a velocidade nem a aceleração são constantes.

Os acelerómetros nos Smartphones têm na realidade muitas limitações. O maior problema do acelerómetro é captar qualquer tipo de movimento, sem saber que tipo de movimento se trata. Dito de outra forma, o facto do acelerómetro detectar movimento, não significa que o utilizador do *Smartphone* esteja a andar ou a correr, basta deslocar o dispositivo no ar um milímetro para detectar a existência de movimento. Para detectar que o *Smartphone* não está em movimento é necessário que o equipamento não se desloque em qualquer direção ou ângulo, caso contrário existirá variação nas leituras.

Só em condições muito controladas é possível reduzir os erros nas leituras destes sensores por forma a calcular a posição do dispositivo, como em sistemas inerciais [30] em que os sensores não têm os mesmos graus de liberdade de movimento de um *Smartphone*, ou navegação automóvel com o *Smartphone* fixo num suporte. Se o *Smartphone* estivesse estático a nível de rotação (por exemplo, preso ao ombro do utilizador), e o pedestre tivesse movimentos lineares e constantes durante o percurso seria minimizado o número de erros nas leituras aos sensores de movimento. Para a aplicação específica que se pretende implementar, não é amigável para o utilizador ter de fixar o *Smartphone* algures no corpo, porque pretende-se que o utilizador visualize os mapas e a informação disponível na aplicação com toda a liberdade de movimentos.

Usando a técnica de análise de cenários, com valores históricos da aceleração linear, é possível determinar padrões de movimentos [53,55], por exemplo, se o utilizador está parado, ou a andar, ou a correr. O problema do uso desta técnica é a especificidade de cada utilizador no que concerne ao seu padrão de movimentação. Cada utilizador teria de calibrar a aplicação em função do seu padrão de movimentos, e mesmo assim seria difícil de determinar se o utilizador está parado ou a andar (eixo x), porque o *Smartphone* não está fixo nos restantes eixos (y e z), e pode estar a variar valores em x sem que o utilizador esteja a deslocar-se (por exemplo, abanando o dispositivo).

⁸⁶ <http://www.movable-type.co.uk/scripts/latlong.html> (2013/01/11)

Outra limitação, é a necessidade de calibrar o acelerómetro antes de iniciar a aplicação. Para o utilizador da aplicação *mtAndroid* a configuração dos sensores deve ser transparente, como na aplicação *SIQuant MobileTrails*.

Não foi implementada fusão de sensores usando o acelerómetro, giroscópio e magnetómetro, por nesta aplicação específica existir muito ruído nos movimentos do utilizador e do *Smartphone* ou *Tablet*, e as leituras periódicas dos sensores consumirem mesmo assim bateria significativa.

4.6.2 Estratégias de poupança de bateria

Conforme já referido no início desta secção, a única funcionalidade implementada na aplicação *mtAndroid* para melhorar a eficiência energética é ativar (`enableMyLocation()`) ou desativar (`disableMyLocation()`) as atualizações de posição de acordo com o ciclo de vida das atividades em Android (quando o *Smartphone* entra em *onPause*, *onStop* ou *onDestroy* desativa os *listeners* de localização, e ativa-os quando entra em *onCreate* e *onResume*). Não se implementou outros mecanismos pois, a classe `MyLocationOverlay`: 1) não permite alterar os parâmetros dos *listeners*, como o intervalo de atualização da localização ou a exatidão da posição, ou 2) desativar *location listeners* na aplicação quando estes não garantem a exatidão exigida (por exemplo, desligando o provider NETWORK quando a exatidão é superior a 100 metros).

Para efeitos de avaliação da eficiência energética da classe `MyLocationOverlay` foi implementada uma `Activity` (`TrailMapActivityGPSOnly.java`) que invés de usar a classe `MyLocationOverlay`, implementa um *listener* próprio para o `GPS_PROVIDER` (com possibilidade de se alterar o intervalo de atualização de leituras GPS e alterar o valor exatidão). Para trocar entre as duas atividades (`TrailMapActivity.java` ou `TrailMapActivityGPSOnly.java`), basta alterar o valor do atributo booleano `GPSOnly` da classe correspondente à atividade principal (`mtAndroidActivity.java`).

O objetivo é verificar se as funções e algoritmos de localização encapsulados pela classe `MyLocationOverlay` (não existe o código fonte disponível *on-line*) comparam bem com uma classe customizada, que usa apenas o GPS, e com várias parametrizações. A avaliação da eficiência energética de ambas as *Activities* é apresentada no Capítulo 5, na secção de avaliação de requisitos não-funcionais.

Outra estratégia passa por ativar a função de economia de bateria disponível no menu de definições do Android, que atinge a poupança reduzindo a frequência do(s) CPU(s) do *Smartphone*, diminuindo a intensidade do brilho do écran, desativando sensores quando não são precisos (p.ex., desativar o Wi-Fi quando não existem pontos de acesso nas proximidades), e desativando mecanismos de atualização de dados (p.ex., sincronismo dos e-mails).

As tecnologias de localização não são as únicas componentes responsáveis por drenar a bateria de um *Smartphone*. O primeiro passo para poupar bateria deve passar por uma seleção criteriosa das aplicações que correm no dispositivo. Em Murmura *et al* [58] é referido que muitas das aplicações grátis disponíveis no

Google Play despendem 75% do total da bateria consumida pela aplicação em publicidade. Muitas aplicações correm em *background* sem que o utilizador se aperceba disso, e consomem bastante energia. Usando, por exemplo, a aplicação colaborativa de diagnóstico energético *Carat*⁸⁷ (desenvolvida pela Universidade de Berkeley) é possível ajudar o utilizador a determinar quais são as aplicações que consomem mais energia, e se é aceitável manter o seu funcionamento ou se é preferível desinstalar a aplicação.

4.7 Dificuldades no Desenvolvimento em Android

Além das questões relacionadas com as opções de implementação, é pertinente referir outras questões relacionadas com o desenvolvimento do protótipo, nomeadamente as dificuldades sentidas durante os testes, e algumas dificuldades com a gestão de memória no ciclo de vida das atividades.

As principais dificuldades sentidas no desenvolvimento dos protótipos foram na fase de testes e de depuração de erros dos casos de uso relacionados com técnicas de localização (GPS), na validação em dispositivos diferentes, e na passagem de variáveis entre atividades.

4.7.1 Fase de Testes e Depuração de Erros

Para testar os protótipos por forma a obter leituras reais, é necessário fazê-lo no terreno. Esta é uma dificuldade para qualquer aplicação a ser usada numa plataforma móvel que necessite de serviços de localização *outdoor*. Para tal é necessário instalar a aplicação no dispositivo e efetuar os testes num ambiente real, sem recorrer a simuladores. As principais vantagens são, ser mais rápido correr as aplicações diretamente no dispositivo quando comparado com o emulador, usar coordenadas reais lidas do GPS sem ter de as inserir manualmente ou com um simulador.

A grande desvantagem é a depuração de erros, que implica abandonar o terreno e voltar para o laboratório para efetuar as correções, obrigando a muitas deslocações físicas entre o desenvolvimento e os testes.

Outra desvantagem é a necessidade de ter de obter sinal de três satélites antes de começar os testes, o que pode demorar algum tempo. O tempo varia de acordo com a qualidade do leitor GPS do dispositivo e com a cobertura de sinal do terreno selecionado.

Os derradeiros testes (avaliação da eficiência energética) foram efetuados com o Samsung Note GT N-7000, e o maior constrangimento foi o consumo de bateria. O CPU poderoso e o enorme ecrã gastam a bateria rapidamente, após três ou quatro caminhadas num percurso de quinze minutos era necessário desligar e carregar a bateria para depois continuar os testes.

⁸⁷ <http://carat.cs.berkeley.edu/> (2013/01/11)

4.7.2 Validação em Dispositivos Diferentes

Existe uma enorme heterogeneidade de dispositivos Android. Ao contrário dos *Smartphones* da Apple, que existem em poucos modelos e controlados pelo mesmo fornecedor, os dispositivos Android são provenientes de vários fornecedores. Os equipamentos testados para os protótipos são LGE, Sony Ericsson e Samsung, cada um com a sua versão de Android e com tamanhos de écran diferentes. Existem muitos outros modelos e fornecedores (ZTE, Huawei, HTC só para mencionar alguns).

A performance difere bastante entre os vários modelos (o Samsung Galaxy Note é muito mais rápido que o Sony Ericsson Xperia X10, e este último muito mais rápido que o LG P-500).

O desenho das interfaces das aplicações móveis tem de ter em conta a sua adaptabilidade a diferentes tipos de écran. O design dos *Layouts* Android é assim mais difícil e limitado do que o design de uma interface para uma aplicação para PC ou Mac, uma vez que é necessário localizar os elementos da interface (botões, tabelas, *listViews*, *mapViews*, etc.) de uma forma relativa. Uma interface que parece perfeita no LG P-500, com tamanho fixo, pode ficar com um aspecto pouco agradável no Samsung Galaxy Note que tem um écran sobejamente maior. No desenvolvimento dos protótipos, além do alinhamento dos vários elementos das interfaces, surgiram dificuldades em alinhar os mapas *offline*. A resolução dos *Google Maps* varia com a resolução e tamanho do écran do *Smartphone*.

As versões de sistema operativo variam muito entre dispositivos. Os equipamentos usados para testar a aplicação *mtAndroid* têm todos versões diferentes entre si. Alguns protótipos desenvolvidos para as versões 2.2 (LG-P500, APIs Level 8) e 2.3 (SE Xperia X10, APIs Level 9) não funcionaram na versão 4.0 (Samsung Galaxy Note, APIs Level 15). Por exemplo, o processamento dos ficheiros XML teve de ser alterado para funcionar no Samsung Galaxy Note (usar a classe `XPathFactory` em detrimento da classe `DocumentBuilder`). Esta alteração impede que *Smartphones* mais antigos (caso do primeiro equipamento usado no desenvolvimento, o HTC Magic, versão Android 1.6, APIs Level 4) consigam correr a aplicação *mtAndroid*.

Por último, outra dificuldade pretende-se com os fabricantes poderem optar por incluir ou não alguns sensores. Por exemplo, alguns equipamentos Android não estão munidos de giroscópio, o que implica obter os mesmos valores nos sensores `TYPE_LINEAR_ACCELERATION` e `TYPE_ACCELEROMETER`. Caso a lógica de uma aplicação Android use estes dois sensores de acordo com a sua definição, irá funcionar incorretamente nos *Smartphones* sem giroscópio.

4.7.3 Passagem de Variáveis entre Atividades (Android Activity)

A forma mais usual de passar valores entre atividades Android é usar a classe `Intent`, embora para estruturas pesadas como listas complexas de objetos (p.ex., `ArrayList` de Objetos com muitos atributos, incluindo imagens e mais listas de objetos), a passagem de parâmetros seja extremamente lenta, e com limites de memória variáveis conforme o *Smartphone*. Esta foi a primeira abordagem usada, mas descartada quando as estruturas começaram a ficar mais complexas.

A opção de implementação foi criar classes com atributos estáticos, que estão associados à atividade principal. As estruturas estáticas são limpas sempre que a Activity que as instanciou acaba o ciclo de vida. Desta forma, enquanto a atividade principal estiver a correr não se irão perder as variáveis que devem ser mantidas em todos os menus da aplicação. O mesmo mecanismo foi usado para tornar mais rápido o acesso a listas de elementos (espécies, eventos, pontos de interesse, trilhos), mas neste caso as estruturas estáticas são associadas a outras atividades que não a principal, não ocupando recursos na memória do dispositivo quando se volta ao menu principal. Outras alternativas são persistir valores em base de dados ou usar a interface `SharedPreferences.Editor`⁸⁸, esta última mais associada a variáveis globais.

⁸⁸ <http://developer.android.com/reference/android/content/SharedPreferences.Editor.html> (2013/01/11)

5 Avaliação

Neste capítulo é apresentada a avaliação do protótipo da aplicação *mtAndroid*, compilado para as Google API's Level 15 (Android 4.0.3), e executado no Samsung Galaxy Note GT N-7000. A Aplicação foi igualmente validada com sucesso nos outros equipamentos usados para testes, no LGE P-500 (Android 2.2) e no Sony Ericsson Xperia X10 (Android 2.3). De seguida é descrita a interface e a validação dos casos de uso implementados, para satisfazer os requisitos iniciais.

5.1 Avaliação de Requisitos Funcionais

5.1.1 Inicialização

Nesta secção é descrito o arranque da aplicação *mtAndroid*. A primeira vez que a aplicação corre é necessário ter uma ligação de dados ativa para importar os conteúdos XML disponíveis on-line, e carrega-los em base de dados. Caso não exista ligação Internet no *Smartphone*, aparece a *Dialog Box* de aviso "Sem Ligação Internet". Caso já tenha corrido a primeira vez, não há necessidade de ter a ligação Internet pois a BD já foi criada e atualizada no dispositivo. Após o carregamento dos dados, é apresentado o menu principal da aplicação.

Na Figura 16 são apresentados os ecrãs que correspondem à fase de arranque da aplicação *mtAndroid*.

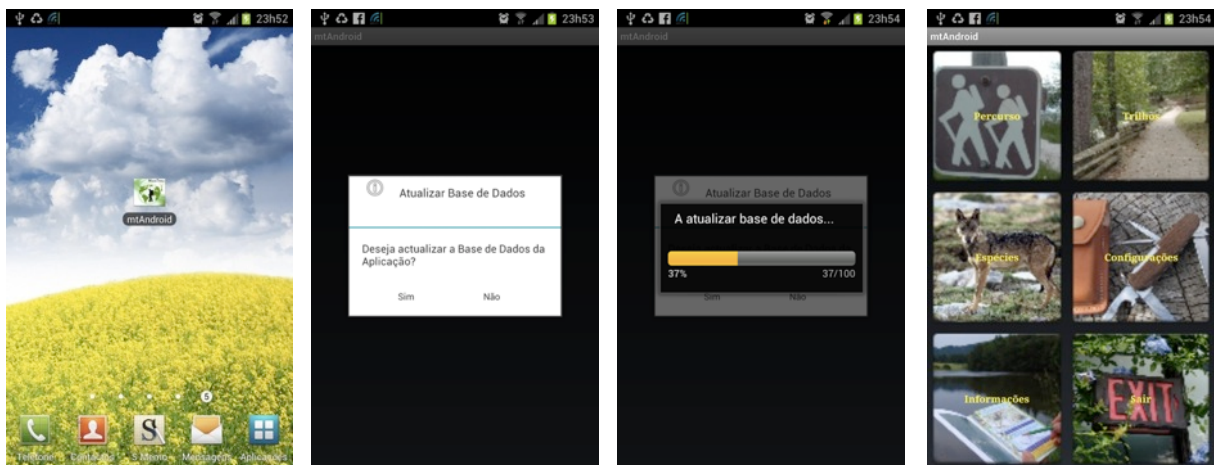


Figura 16. Arranque da Aplicação *mtAndroid*

Após esta fase, é satisfeito o requisito de comunicar com um servidor HTTP remoto para consultar e atualizar dados sobre trilhos, espécies, eventos e pontos de interesse. Todos os dados importados dos ficheiros XML para a BD são acessíveis na aplicação e visualizados conforme descrito na secção seguinte.

5.1.2 Visualização da Informação de Espécies, Eventos, POIs e Trilhos

Nesta secção é apresentada a interface da aplicação *mtAndroid* na componente de visualização da informação disponível. O menu principal apresenta seis botões, que são (da esquerda para a direita e de cima para baixo): Percurso, Trilhos, Espécies, Configurações, Informações (POIs e Eventos) e Sair. O menu principal usa um *layout*

tabular (*TableLayout*) de forma a preencher todo o ecrã seja qual for o tamanho do visor do *Smartphone*. Apesar de não ser um requisito inicial explícito, optou-se durante o desenvolvimento por garantir que a interface fosse adaptável a qualquer tipo de ecrã, para poder correr em vários *Smartphones*.

Os botões “*Trilhos*” e “*Espécies*” contém respectivamente os dados dos trilhos e das espécies. O botão “*Informações*” contém os dados dos pontos de interesse e eventos. O ecrã seguinte ao clique de um destes botões é o ecrã das categorias (Categorias de Trilhos e Espécies nos dois primeiros casos, e Categorias de POIs e Eventos no caso do botão “*Informação*”). Na Figura 17 é apresentado um exemplo de visualização dos POIs e eventos. O segundo ecrã corresponde às categorias, e o terceiro ecrã à lista de todos os pontos de interesse disponíveis por se ter clicado na categoria “*Pontos de Interesse*”. O mesmo comportamento existe nas espécies e nos trilhos. Depois de se escolher o elemento da lista (neste caso, lista de POIs) é apresentado um ecrã com o detalhe da entidade.



Figura 17. Visualização de Pontos de Interesse e Eventos

A interface desenhada para percorrer listas de elementos, como os Trilhos, Pontos de Interesse, Espécies e Eventos também é similar. Na Figura 18 é ilustrado o exemplo da interface das Espécies, com os seus atributos específicos, mas que retrata o comportamento de todas as restantes listas. No detalhe da espécie, evento, trilho ou POI, ao clicar na imagem é lançada uma nova *Activity* mas com o *Zoom* da imagem do elemento. Tanto no detalhe como no *Zoom* da imagem o menu de botões é igual conforme apresentado nos *screenshots* da Figura 18. O símbolo *home* faz a aplicação voltar ao menu principal (último ecrã da Figura 16). O símbolo *back* retorna a aplicação ao ecrã anterior, e os símbolos *previous* e *next* permitem percorrer a lista dos elementos selecionados (no caso do exemplo da Figura 18, a lista de espécies). Do lado direito dos botões *previous* e *next* aparece a indicação do número de ordem do elemento da lista selecionado, e do total de elementos da lista (no caso da lista de espécies da Figura 18, o elemento selecionado é o primeiro de uma lista de oito).

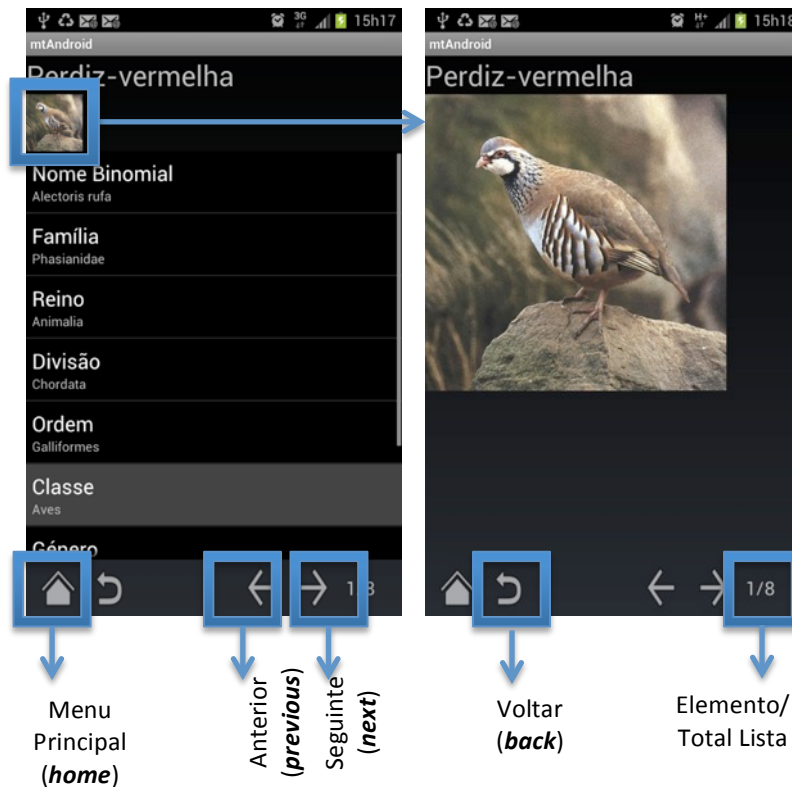


Figura 18. Interface Espécies

Quando se está a visualizar o detalhe de um dos elementos, é possível verificar as entidades relacionadas. No caso dos POIs, é possível ver as espécies associadas, os eventos associados e os trilhos associados. No caso dos trilhos, das espécies e dos eventos, é possível ver quais os POIs associados. Ao clicar nas associações é aberta a *Activity* que corresponde ao detalhe do primeiro elemento da lista de elementos associados. Por exemplo, o trilho “São Pedro do Estoril, CTT -> Praia” tem associados três pontos de interesse: CTT São Pedro do Estoril, Estação de São Pedro de Estoril e Praia de São Pedro do Estoril. Ao clicar no campo “Pontos de Interesse Associados”, é aberto o detalhe do POI “CTT São Pedro do Estoril”, mas com a possibilidade de visualizar os restantes POIs através dos botões *previous* e *next*. Estas associações refletem as associações do modelo de domínio apresentado no Capítulo 4.

5.1.3 Mapas, Trilhos e Percursos

O botão “*Percurso*” serve para visualizar o mapa de um trilho, visualizar a localização do *Smartphone* no mapa, e iniciar efetivamente o percurso no trilho selecionado. O primeiro requisito para iniciar o percurso é ter um trilho selecionado. Se clicarmos no botão “*Percurso*” sem um trilho selecionado, aparecerá a mensagem de aviso “*Trilho Não Selecionado*”, e a opção de saltar para a opção “*Trilhos*”, resultado atingido também se clicarmos no botão com o mesmo nome. Na Figura 19 é apresentado o fluxo de visualização e seleção de trilhos, o primeiro écran que aparece a seguir à mensagem de aviso é o da lista de categorias de trilhos (neste caso com duas categorias, “*Trilhos de Cascais*” e “*Trilhos de Lisboa*”). Selecionado uma das categorias, são

apresentados de seguida os trilhos que lhe pertencem (no exemplo da figura, selecionada a categoria “Trilhos de Cascais”, que contém apenas o trilho “São Pedro do Estoril CTT -> Praia”). Se clicarmos em “Trilhos” aparecerá a lista de todos os trilhos, seja qual for a sua categoria. Após escolha do trilho desejado, aparece a descrição do trilho e a opção “Selecionar Trilho”. Neste écran aparecem todos os atributos definidos no modelo de domínio apresentado no Capítulo 4, como a imagem, o ponto inicial, ponto final, distância, duração, descrição, bem como os pontos de interesse relacionados. Pode-se saltar para a lista de POIs do trilho a partir deste écran clicando na linha de “Pontos de Interesse Associados”, para a categoria do trilho, visualizar a imagem do trilho em tamanho maior ao clicar nesta.

Após selecionar o trilho, pode-se voltar ao menu principal (clicando no botão *home*, ou várias vezes nos botões *back* do *Smartphone* ou do menu da aplicação) e entrar em “Percursos”.

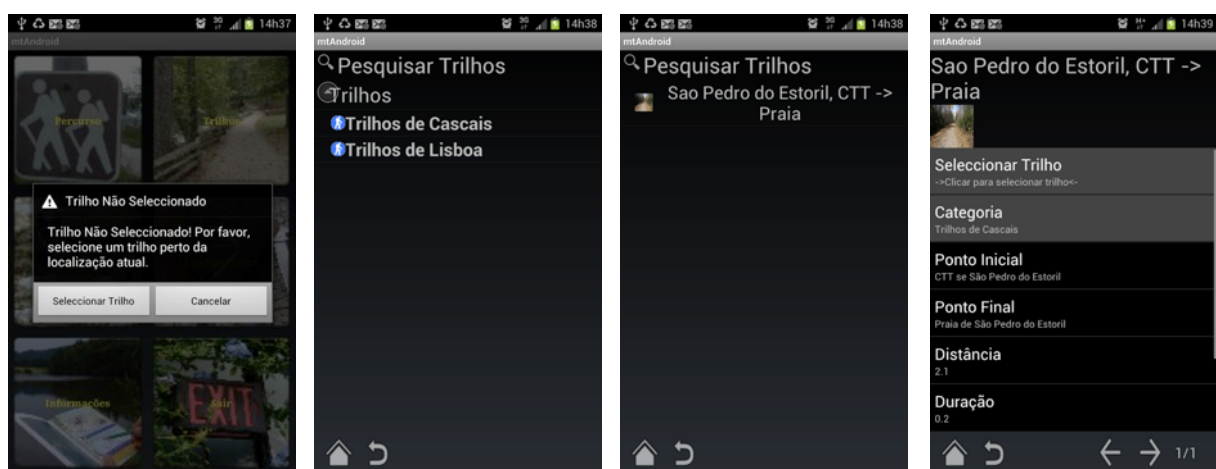


Figura 19. Seleção de Trilho

Uma vez clicado o botão “Percursos”, a aplicação irá verificar duas situações. Primeiro, se tem ligação Internet ativa. Caso não tenha dará o aviso com a *Dialog Box* “Sem Ligação Internet” (o mesmo aviso que aparece no arranque da aplicação caso essa condição se verifique). Caso a Internet esteja desligada na opção “Percursos”, irão existir dois constrangimentos, o primeiro é não ser possível usar o *provider* NETWORK_PROVIDER na determinação da localização do *Smartphone*, o que implica ficar sem função de localização se o GPS_PROVIDER ficar indisponível, o segundo é não poder usar os *Google Maps*, perdendo-se a interação com o mapa (*Zoom, Pan, Tilt, etc.*). A segunda situação é verificar se o GPS está ligado. Caso não esteja, é apresentado novo aviso e dada a opção para abrir o menu de ativação do GPS no *Smartphone*, conforme ilustrado na Figura 20. Apesar do menu de seleção variar de *Smartphone* para *Smartphone*, a invocação da *Activity* de sistema Settings.ACTION_LOCATION_SOURCE_SETTINGS funcionou em todos os equipamentos testados.

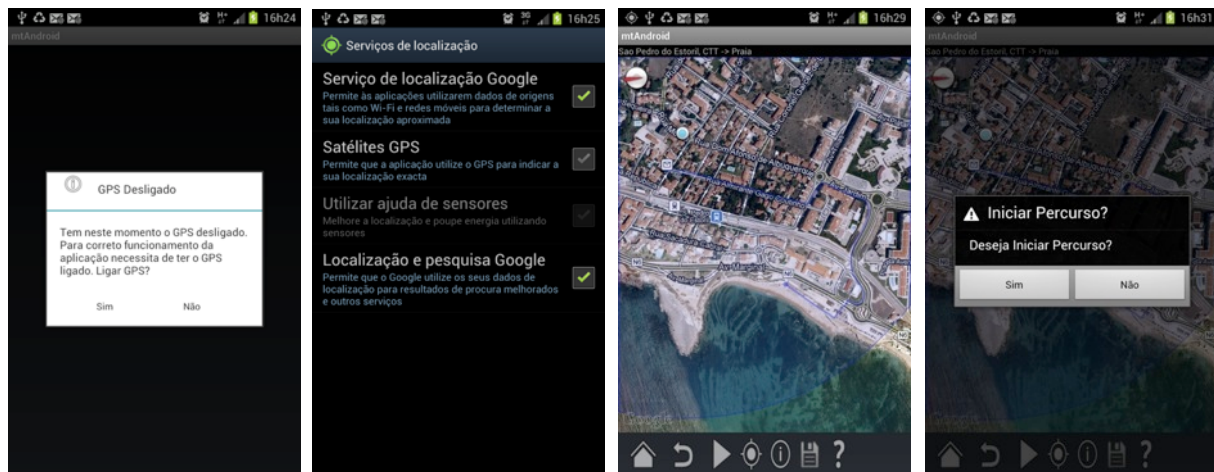


Figura 20. Iniciar Percurso

Após estas validações, o utilizador pode iniciar o percurso, carregando no ícone “Play”, e aparece a *Dialog Box* a perguntar “*Deseja Iniciar Percurso?*”, conforme ilustrado na Figura 20. A seguir será explicado o significado de cada um dos botões da interface do percurso.

Para validação das funções de localização e das Google API’s em Android, a opção onde se utilizam todas as funcionalidades relativas a interação com GPS, tecnologias de localização baseadas em rede (Redes Móveis e Wi-Fi), e interação com mapas, é a opção “*Percurso*”.

Na Figura 21 é descrita a interface do *layout* dos percursos. O mapa do percurso e todos os *Overlays* estão contidos na *mapView*, que corresponde a toda a área que contém o mapa. O mapa da figura corresponde ao trilho “*São Pedro do Estoril CTT-> Praia*”, já referido no Capítulo 4.

O primeiro *overlay* é o próprio mapa, que corresponde aos *Google Maps* se existir ligação internet, ou a uma *snapshot* dos *Google Maps* se a ligação de dados estiver desativada. O mapa apresenta o nível de zoom ideal para apresentar o trilho completo e centrado na *mapView*.

O segundo *overlay* corresponde à classe *MyLocationOverlay*. No canto superior esquerdo é apresentada a bússola (*Compass*) disponibilizada pela classe *MyLocationOverlay*. A localização do *Smartphone* (*myLocation*) é disponibilizada pela mesma classe, e é dada pelo GPS ou pelo *NETWORK provider*.

O terceiro *overlay* corresponde à visualização do trilho, a linha azul desenhada sobre o mapa presente na Figura 21, construída a partir dos pontos com as coordenadas GPS do trilho (existentes em BD e previamente importados do ficheiro XML *Trails.xml*).

Por último é adicionado o quarto *overlay* correspondente aos pontos de interesse (POI). Cada POI corresponde a um *ItemizedOverlay*. No trilho da Figura 21 podem ser visualizados dois POIs (CTT São Pedro do Estoril e Estação de São Pedro do Estoril), cuja referência ao trilho advém da relação entre trilhos e POIs implementada de acordo com o modelo de domínio, relação importada do ficheiro *POIs.xml*. O ícone apresentado no mapa corresponde ao ícone da categoria do POI. Se clicarmos no ícone, aparecerá uma *Dialog Box*, com a indicação do nome e da descrição do POI, com o botões “*Ver Detalhe*” e “*Fechar*” (fechar *Dialog Box*).

Box). Se clicarmos em “Ver Detalhe”, é apresentado o detalhe do POI (categoria, latitude, longitude, descrição, espécies associadas, eventos associados e trilhos associados).

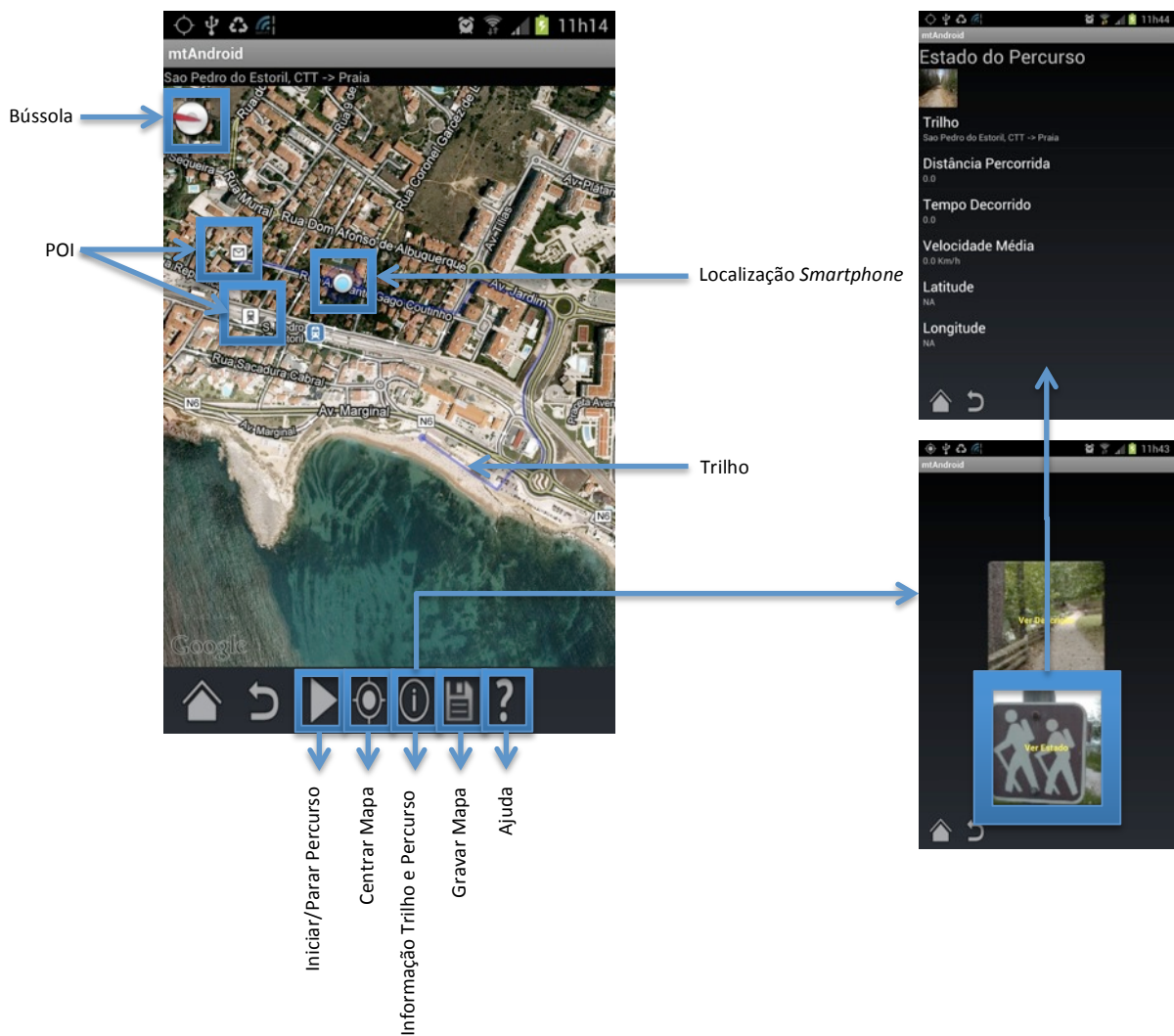


Figura 21. Percursos

Além dos botões “Home” e “Back”, existem cinco botões específicos deste *layout*. O botão “Iniciar/Parar” percurso serve para iniciar/parar o percurso, por forma a obter estatísticas sobre o percurso (distância percorrida em Km, duração do percurso, velocidade média e latitude e longitude atuais). Esta informação pode ser obtida enquanto o percurso estiver a decorrer, clicando no botão “Informação Trilho e Percurso”, e no menu seguinte clicando na opção “Ver Estado”, conforme exemplificado na figura. A outra opção (botão “Ver Descrição”) abre a *Activity* correspondente ao detalhe do trilho atual, ou seja, o último *layout* da Figura 19. A informação do percurso é atualizada cada vez que a localização é alterada (evento `onLocationChanged`). O botão “Centrar Mapa” serve para apresentar o mapa na posição inicial (centrado e com o trilho completamente visível). O botão “Gravar Mapa” serve para tirar uma fotografia do mapa atual, útil para adaptar o mapa disponível no XML ao ecrã do *Smartphone*. Ao carregar neste botão a aplicação cria um

Bitmap a partir da *drawing cache* da *MapView*. Depois, grava a imagem para ficheiro um ficheiro PNG com o nome igual ao identificador do trilho para a raiz do cartão SD do *Smartphone*, e atualiza o mapa do trilho na base de dados. O ficheiro gerado pode depois ser utilizado para atualizar a *tag <map>* do ficheiro XML do trilho, após conversão do ficheiro para BASE64. Esta funcionalidade é apenas incluída para facilitar os testes no protótipo, foi a forma escolhida para obter mapas *offline* de trilhos de dispositivos diferentes (diferentes resoluções no *Google Maps*). O botão de ajuda servirá para descrever a interface deste *layout* (não implementado).

Durante o percurso, foram validadas com sucesso as notificações de proximidade de pontos de interesse (30 metros) e afastamento do trilho (100 metros).

5.2 Avaliação de Requisitos Não-Funcionais

5.2.1 Poupança de Energia

Conforme referido na Secção 4.6, as estratégias de poupança de energia foram limitadas, uma vez que com a classe `MyLocationOverlay` apenas é possível poupar bateria através dos métodos `disableMyLocation()` e `disableMyCompass()`. Contudo é interessante comparar o consumo de bateria despendido por esta classe com outra que use um *location listener* para o *provider* GPS, para determinar se a classe `MyLocationOverlay` usa algum mecanismo que poupe energia.

O modo de economia de bateria nativo do Android, também referido na secção 4.6, está fora do âmbito desta avaliação. Pode ser conseguido ativando no menu de definições do Android as opções “Economia de energia do sistema” (*System power saving mode*) ou “Personalizar economia de energia” (*Custom power saving mode*).

Um dos temas não abordados nesta dissertação é a dificuldade em medir a energia consumida por aplicação em *Smartphones* [58]. Uma das aplicações mais usadas para medir o consumo de energia de aplicações em Android é a *app PowerTutor*⁸⁹, desenvolvida pela Universidade de Michigan, apresentada e detalhada em Zhang *et al* [59]. A aplicação *PowerTutor* foi a forma escolhida para avaliar a eficiência energética da aplicação *mtAndroid*.

Na Figura 22 é apresentado um exemplo das medidas recolhidas pela aplicação *PowerTutor*, para o esquema *mtAndroid* que usa só o GPS (sem a classe `MyLocationOverlay`), com leituras efetuadas em intervalos de 5 segundos e com uma exatidão de 10 metros. Os primeiros três ecrãs correspondem às leituras da aplicação *mtAndroid* (opção do menu principal da *PowerTutor* “*Application Viewer*”), o último écran corresponde às leituras de sistema (opção do menu principal da *PowerTutor* “*System Viewer*”) que é onde é possível ler os consumos do GPS (para todas as aplicações que usam o GPS).

⁸⁹ <http://www.powertutor.org/> (2013/01/11)

O trilho escolhido para a avaliação foi o “São Pedro do Estoril CTT-> Praia” (linha azul desenhada no mapa da Figura 21). Foi escolhido um troço do percurso sem proximidade de pontos de interesse, e não houve afastamento do trilho, para garantir que não existiam notificações e consumos adicionais.

O troço do trilho foi percorrido durante seis minutos para avaliar os consumos médios da aplicação *mtAndroid*, e durante dois minutos para avaliar os consumos GPS, a uma velocidade de média de 5 Km/h.

Em todos os esquemas se utilizou a ligação de dados ativa (HSPDA/3G), Wi-Fi desligado, e mantido o ecrã ligado (sem apagar o ecrã para poupança de energia, *flag* `FLAG_KEEP_SCREEN_ON` adicionada no *layout* que contém a *mapView*). Todas as restantes aplicações que usam o GPS foram desligadas (serviço de localização do Android desativado), para garantir que os consumos de energia do GPS eram exclusivos da aplicação *mtAndroid*.

Os valores utilizados na comparação entre os esquemas energéticos são o consumo médio dos últimos cinco minutos para a aplicação *mtAndroid* (máximo de histórico por aplicação) e a energia despendida pelo GPS/minuto, com dados do último minuto (máximo de histórico do sistema).

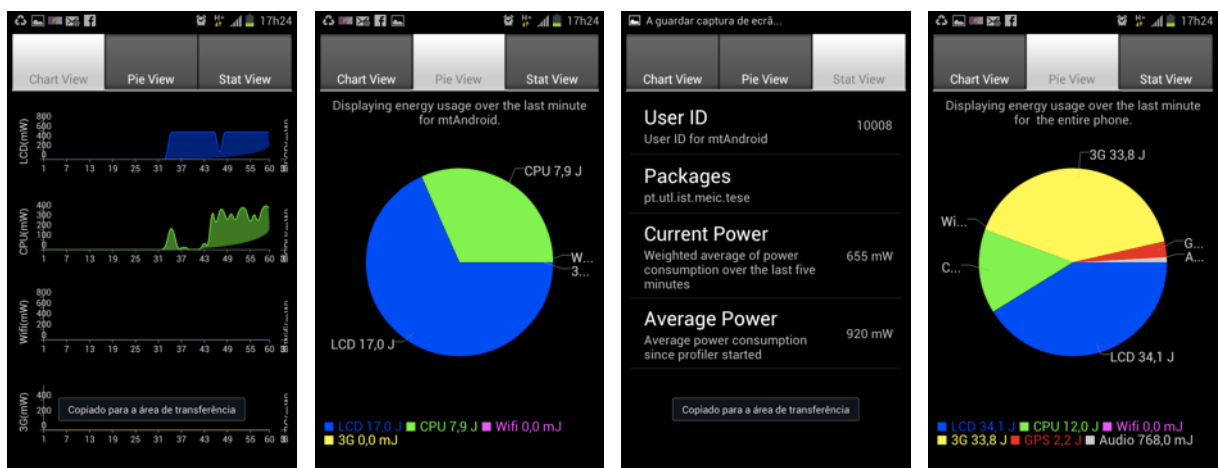


Figura 22. Medidas PowerTutor da Aplicação *mtAndroid* (GPS, 5 seg., 10m)

O primeiro passo da avaliação corresponde a iniciar a aplicação *PowerTutor* (*Start Profiler*). De seguida é iniciada a aplicação *mtAndroid* para o cenário de percurso (opção “Trilhos”->”Trilhos de Cascais”->”Sao Pedro do Estoril, CTT -> Praia”, “Selecionar Trilho”, home, “Percurso”). É esperado que o GPS fique a funcionar (o ícone do GPS deixa de piscar, significando que adquiriu sinal e a posição fica corrigida). A partir daqui o trilho é percorrido a pé durante cerca de seis minutos para garantir que a aplicação *PowerTutor* recolhe dados para a aplicação *mtAndroid* na função de “Percurso” durante 5 minutos (máximo de histórico das leituras da aplicação *PowerTutor*). Passados os seis minutos de caminhada, é fechada a aplicação *mtAndroid* (*home, exit*), aberta a aplicação *PowerTutor*, e recolhidos os valores *Average Power* na *tab Stat View*.

Depois é repetido o mesmo procedimento para recolher a energia despendida pelo GPS (*System Viewer* > *Pie View*), usando uma duração de dois minutos durante o percurso, para garantir que se tem os dados do último minuto.

Teste	Opções		Resultados		
	Intervalo GPS (segundos)	Exatidão GPS (metros)	Consumo médio (mW)	Consumo GPS (mW)	Consumo Total (mW)
T1	20	10	980	160	1140
T2	20	0	900	177	1077
T3	10	10	954	133	1087
T4	10	0	928	120	1048
T5	5	10	992	216	1208
T6	5	0	996	216	1212
T7	2,5	10	957	210	1167
T8	2,5	0	1011	216	1227
T9	0	0	979	423	1402
T10	0	0	844	423	1267

Tabela 6. Testes aos Consumos Médios de Energia da Aplicação *mtAndroid*

Foram realizados dez testes, conforme apresentado na Tabela 6: *T1*) exatidão de 10 metros e leituras de 20 em 20 segundos; *T2*) exatidão de 0 metros e leituras de 20 em 20 segundos; *T3*) exatidão de 10 metros e leituras a cada 10 segundos; *T4*) exatidão de 0 metros e leituras de 10 em 10 segundos; *T5*) exatidão de 10 metros, leituras a cada 5 segundos; *T6*) exatidão de 0 metros, leituras de 5 em 5 segundos; *T7*) exatidão de 10 metros, leituras a cada 2,5 segundos; *T8*) exatidão de 0 metros, leituras de 2,5 em 2,5 segundos; *T9*) exatidão de 0 metros e leituras GPS permanentes (0 segundos) e *T10*) usada a classe `MyLocationOverlay` (GPS e NETWORK, 0 metros de exatidão e leituras permanentes). A aplicação *PowerTutor* fornece os valores dos consumos de energia do GPS em Joules, para converter para mW basta apenas multiplicar por um milhar e dividir pelo tempo (60 segundos), unidades apresentadas na tabela de avaliação.

Repetiu-se várias vezes os testes. Os valores de consumo de bateria médios para os últimos cinco minutos (sem contabilizar o consumo do GPS) variaram quase sempre entre os 900mW e os 1100mW para os testes T1 a T9. O teste T10 apresentou sempre valores abaixo dos 900mW. A exatidão da aplicação *PowerTutor*, pela experiência retirada durante os testes, ronda os 80mW. Os valores recolhidos durante a última avaliação são apresentados na Tabela 6.

Tendo em conta a exatidão das leituras da aplicação PowerTutor (80mW), não se consegue inferir muitas conclusões a nível do consumo médio, contudo observou-se que o teste T10 tem sempre melhores resultados que o T9 (T10 teve sempre resultados abaixo dos 900mW), porque a classe `MyLocationOverlay` necessita de menos CPU e LCD para desenhar a localização. Uma das possibilidades para existir esta diferença é a classe usar primitivas gráficas mais perto do *kernel*, ganhando eficiência energética em relação às classes desenvolvidas de raiz que usam as primitivas gráficas da linguagem Java, e que exigem mais CPU. Não é possível confirmar esta teoria devido ao código fonte da classe `MyLocationOverlay` não ser disponibilizado pela Google.

Quanto aos consumos do GPS, o resultado esperado na avaliação seria ter mais energia despendida nos testes T9 e T10, onde o GPS está continuamente ligado, e ter menos energia despendida à medida que se aumentava o intervalo de tempo entre as leituras. Todavia, observou-se para os testes T1 e T2 um consumo superior aos testes T3 e T4. Para a periodicidade de 20 segundos (T1 e T2), o GPS parece perder o sinal (mesmo comportamento quando adquire sinal da 1ª vez), demora mais tempo a efetuar a leitura da posição, e consome mais energia na componente GPS. Para os restantes testes (de T3 a T10) a leitura GPS é praticamente instantânea.

Foi constatado que a diferença entre 0 e 10 metros de exatidão é claramente visível no mapa durante o percurso, e conforme se pode observar na tabela, este parâmetro não influi muito no consumo energético. Obviamente que o facto da exatidão rondar os 80mW ajuda a que se obtenham valores ligeiramente diferentes para parâmetros similares, mas é conclusivo que a diferença de exatidão entre 0 e 10 metros não afeta significativamente os consumos do GPS. Por exemplo, T4 teve melhores resultados que T3, apesar de T3 usar uma exatidão de 10 metros e T4 de 0 metros. Para garantir que a posição está sob o trilho deve-se usar uma exatidão de 0 metros, o mais próximo possível da posição real. Não foram efetuados testes com valores de exatidão superiores a 10 metros, por serem pouco adequados a percursos pedestres.

O teste mais eficiente energeticamente durante a última avaliação é o T4 (leituras GPS de dez em dez segundos, exatidão de 0 metros). A energia ganha a nível de consumo do CPU e do LCD pela implementação da localização com recurso à classe `MyLocationOverlay`, teste T10, não compensa o consumo desperdiçado pelo GPS em permanente atualização. A atualização da posição de dez em dez segundos durante um percurso pedestre a cerca de 5 Km/h é bastante aceitável, conforme se verificou durante esta avaliação.

Para o caso prático da aplicação *mtAndroid*, a escolha da parametrização deve recair sobre a usada no teste T4, que garante boa precisão e boa exatidão da localização para percursos pedestres, e consome menos energia que os testes T5-T10, por usar menos vezes o GPS (leituras de 10 em 10 segundos). Para leituras de 20 em 20 segundos, o GPS consome mais energia por necessitar de readquirir sinal satélite.

5.2.2 Escalabilidade

A escalabilidade não foi um requisito inicial, mas para trabalho futuro é importante identificar este ponto de melhoria. O protótipo necessita de ser adaptado para comportar um maior nº de POIs e Trilhos,

nomeadamente melhorar o algoritmo de importação dos ficheiros XML (usar um SAX Parser invés do DOM Parser), e alterar o modo de acesso às listas de objetos consultando mais vezes a base de dados SQLite invés de carregar os objetos para memória.

O limite de memória por aplicação nos *Smartphones* Android de primeira geração é de 16MB, de 24MB nos equipamentos de segunda geração, e de 32MB para os mais recentes. Para *Smartphones* topo de gama mais recentes (caso do *Smartphone* utilizado nos testes, Samsung Galaxy GT N-7000), os valores podem ir até aos 64MB. Este valor pode ser consultado através do método `Runtime.getRuntime().maxMemory()`.

Para tornar a aplicação mais escalável, é primordial usar o menos possível a memória do equipamento, carregando apenas o necessário para cada ecrã ou atividade. Para este efeito será necessário alterar o modelo de domínio para consultar mais vezes a base de dados SQLite, carregando apenas os atributos necessários em cada instante, invés de carregar para memória as listas de POIs, Trilhos, Espécies, etc., com todos os atributos. A desvantagem desta alteração é perder o encapsulamento entre o modelo de domínio e a base de dados (uma das grandes vantagens na programação orientada a objetos), com consequente aumento da ilegibilidade do código fonte. Outra otimização pode ser feita a nível dos dados. Enquanto que para o mapa *offline* deve ser mantida a resolução da imagem, as fotografias das espécies, eventos e POIs podem ter uma resolução inferior, ocupando assim menos espaço em memória na aplicação.

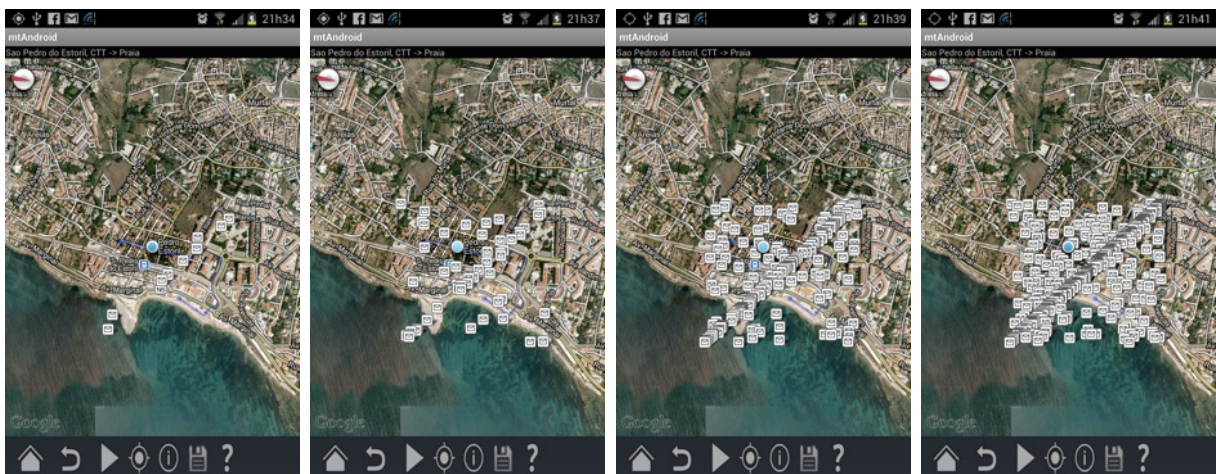


Figura 23. Testes com 10, 50, 200 e 400 POIs sobre o Mapa no protótipo *mtAndroid*

Um parâmetro a ter em conta em termos de escalabilidade é o nº de *markers* que podem aparecer no mapa. Na aplicação *mtAndroid*, o mapa corresponde a um *Overlay*, o trilho a outro, e para cada POI é criado um novo *ItemizedOverlay* que pertence ao *Overlay* dos POIs.

Para minimizar o problema da performance no desenho do mapa quando se tem muitos *Overlays*, pode ser usada uma extensão da *Google Maps API*, a *mapview-overlay-manager*⁹⁰, que usa o padrão de desenho de software *Lazy Loading* [60] para melhorar a performance da apresentação dos elementos na *mapView*.

⁹⁰ <http://code.google.com/p/mapview-overlay-manager/> (2013/01/11)

Tempo de Visualização Nº de POIs	Tempo de Visualização no Mapa (segundos)	Tempo de Apresentação DialogBox POI (segundos)
50	0	0
100	0	0
200	0	0
300	3	0
400	9	0
500	19	0
600	25	0
700	40	0
800	67	0
900	120	0

Tabela 7. Tempo de Visualização do Mapa e Tempo de apresentação da *Dialog Box* de um POI

Para testar a escalabilidade da aplicação foi alterado o código do protótipo para gerar vários POIs com coordenadas GPS aleatórias, dentro do perímetro rectangular que envolve os limites do trilho. O POI utilizado é sempre o mesmo (*cttSaoPedroEstoril*), mas com latitude e longitude diferentes. Na Figura 23 são apresentados exemplos de geração de 10, 50, 200 e 400 POIs sobre o trilho de São Pedro do Estoril.

Pretende-se com este teste de escalabilidade: 1) quanto tempo a aplicação demora a apresentar o mapa, conforme o nº de POIs; 2) quanto tempo a aplicação demora a apresentar a *Dialog Box* que permite ver a informação do POI, após carregar no ícone respectivo (método *OnTap* da classe *ItemizedOverlay*), conforme o nº de POIs.

Na Tabela 7 são apresentados os testes de escalabilidade (nº de POIs e respetivos tempos de espera na visualização no mapa e tempo de espera na apresentação da *Dialog Box* do POI). O tempo de apresentação da *Dialog Box* não é afetado com o aumento de POIs, ou seja, após todos os POIs serem apresentados sobre o mapa, se clicarmos em um dos ícones que representa um POI, a *Dialog Box* que permite visualizar a informação relacionada com o POI aparece instantaneamente, trate-se de 50 ou 900 pontos de interesse. Todavia, o tempo de apresentação do mapa começa a aumentar a partir dos 300 POIs (3 segundos), e demora 120 segundos a apresentar o mapa com 900 POIs. As operações de *zoom* também ficam mais lentas à medida que se aumenta o nº de pontos de interesse.

6 Conclusões e Trabalho Futuro

6.1 Conclusões

A plataforma Android disponibiliza através das *Google Maps APIs* uma série de funcionalidades que facilitam o desenvolvimento de aplicações móveis baseadas em serviços de localização e de navegação sobre mapas, sem que se tenha de recorrer a fornecedores de sistemas de informação geográfica *third-party*, adquirir mapas ou outros *toolkits* licenciados.

As classes disponibilizadas para interagir sobre os mapas, os serviços de localização e os sensores garantem um bom encapsulamento dos *drivers* e do *hardware* dos sensores do *Smartphone* e estão bem documentadas no site *Android Developers* [5]. O Android implementa técnicas de localização usando redes de dados (WPS e Torres de Células) e o GPS. Uma das limitações do Android é não permitir distinguir a técnica WPS da técnica de Torres de Células, a técnica de localização é implementada pela *framework* como `NETWORK`, não sendo possível optar por cada uma das técnicas individualmente.

A tecnologia de localização mais exata e precisa é o GPS. O GPS garante a privacidade do utilizador (só ele sabe a sua localização). Todavia, uma das limitações do GPS é o consumo de energia. Para minimizar este problema pode ser implementada fusão de sensores, que permite usar o GPS em conjunto com outras técnicas de localização (WPS, Torres de Células), e/ou com leituras de outros componentes do *Smartphone* (*Bluetooth*, acelerómetro, giroscópio, magnetómetro). A fusão de sensores permite reduzir o número de leituras GPS, tornar a aplicação mais eficiente a nível energético, e aumentar a exatidão e precisão da posição. Outra limitação do GPS, é a falta de cobertura em ambientes *indoor*, ou em ambientes urbanos com prédios altos. A solução neste caso também passa pela técnica de fusão de sensores. A *framework* Android disponibiliza classes e métodos para interagir com todos os sensores, permitindo implementar a técnica de fusão de sensores mais adequada à aplicação. As técnicas de fusão de sensores para melhorar a exatidão, precisão e a eficiência energética têm de ser implementadas. Existem já a partir da versão Android 2.3 (API Level 9), sensores virtuais (resultantes da fusão de sensores existentes) que facilitam a monitorização dos movimentos dos *Smartphones*.

A maior parte da literatura consultada sobre as problemáticas dos consumos de energia elevados do GPS e a localização *indoor* em *Smartphones*, fornece como solução a técnica de fusão de sensores, recorrendo sempre ao acelerómetro. Todavia, na prática, o uso do acelerómetro só faz sentido se o *Smartphone* estiver fixo (sem liberdade de movimentos), e se estiver calibrado para o padrão de movimentos do utilizador da aplicação.

O problema do consumo de energia dos *Smartphones* não é exclusivo das técnicas de localização, como verificado nos testes realizados. O écran e o CPU dos *Smartphones* consomem mais energia do que o GPS. As aplicações que correm em *background* para atualização de dados via Internet (e-mail, Facebook) consomem também mais energia do que o carregamento do mapa *online* da Google na *mapView*.

Atualmente, para se tirar partido das funcionalidades de um *Smartphone*, é fundamental que se tenha uma ligação de dados ativa. Os *Smartphones* usam aplicações que consomem *Web Services*, para tirar partido de

serviços disponíveis na Internet, como repositórios de informação (evitando ocupação nos suportes de armazenamento dos telefones), ou para enviar informação. Por isto, conclui-se que o uso do serviço *Google Maps* é a melhor opção para o desenvolvimento destas aplicações na plataforma Android, porque disponibiliza os mapas *on-the-fly*, sempre com as últimas atualizações, e sem ocupar recursos de armazenamento no *Smartphone*. Os custos dos dados estão a decrescer, mas continuam a ser uma forte barreira ao uso dos mapas *online*. A maior limitação no uso das *Google Maps APIs* no SDK do Android é a inexistência de um mecanismo de *cache*, que permita pré-carregar uma porção de mapa para uma área específica. Existem áreas sem cobertura Internet, e assim será impossível visualizar o mapa quando o *Smartphone* está posicionado nessas localizações. Usando a funcionalidade *Google Labs* dos *Smartphones* Android é possível fazê-lo para a aplicação nativa *Google Maps*, mas não através das *Google Maps APIs*, por restrições de licenciamento. Dada a frequência com que esta funcionalidade é solicitada em fóruns da internet especializados em desenvolvimento na plataforma Android, é possível que venha a ser disponibilizada brevemente numa nova versão do SDK.

Além dos custos associados à transmissão de dados, um dos problemas em ter a Internet ativa durante a utilização de aplicações com serviços de localização é a privacidade. Uma aplicação mal intencionada pode fazer *tracking* de um utilizador sem que ele se aperceba e guardar numa base de dados remota todos os seus movimentos, sem sua autorização. Da mesma forma, as localizações baseadas em triangulação de rede, WPS e Torres de Células, podem pôr em causa o anonimato do utilizador. A técnica GPS assegura privacidade, já que só o receptor GPS conhece a posição do utilizador.

Outro custo tido com a ligação de dados ativa, é o consumo de bateria. Quase todas as aplicações Android têm de providenciar por alguma forma atualizações via internet, seja para instalar uma nova versão da aplicação ou simplesmente para sincronizar dados. Quantas mais atualizações, mais energia irá ser despendida no *Smartphone*.

Para validar as funcionalidades existentes na plataforma Android para localização e interação com mapas, propôs-se um cenário de navegação pedestre, *outdoor*, em trilhos pré-definidos. O protótipo (*mtAndroid*) desenvolvido permitiu testar e validar algumas das tecnologias de localização disponibilizadas pela plataforma Android, como o GPS, o WPS e as Torres de células. Tendo em conta os requisitos da aplicação, apenas o GPS fornece a exatidão e a precisão exigidas. Para minimizar os consumos de energia, a solução passa por reduzir a frequência de consultas GPS, e desligar as atualizações quando se sai da atividade que contém o mapa.

A avaliação do protótipo permitiu concluir que, para *Smartphones* mais recentes, o consumo de bateria é mais problemático devido: (1) aos consumos de energia do écran (cada vez maiores e com maior resolução), (2) aos consumos de energia do CPU (cada vez mais potentes), e (3) aos consumos de energia despendidos na sincronização e atualização dos dados de todas as aplicações ligadas à *cloud*. Em termos de avaliação energética, a Google deveria disponibilizar uma ferramenta oficial para ajudar os programadores a medir os consumos de energia das aplicações (semelhante ao *Nokia Tool Profiler*), já que a aplicação *PowerTutor* teve a sua última atualização em Outubro de 2010, e não funciona em todos os *Smartphones* Android.

6.2 Trabalho Futuro

Para melhorar a gestão dos dados da aplicação *mtAndroid*, deve ser desenvolvida uma Aplicação *Web* de introdução e atualização de dados (atributos e imagens), com geração dos ficheiros XML . A aplicação deve permitir carregar as imagens relativas às entidades da aplicação (POIs, Eventos, Espécies, Trilhos, Categorias), e codifica-las no formato BASE64.

Um outro melhoramento passa por criar Mapas *offline* personalizados para dispositivos diferentes. A aplicação carregaria o mapa disponível para o tamanho de écran correspondente ao modelo do *Smartphone*.

Para ser possível utilizar o sensor acelerómetro e poupar bateria, desligando as leituras GPS quando o utilizador não está em movimento, é necessário detetar padrões de movimento. Um trabalho futuro interessante é incluir um módulo de aprendizagem de padrões de movimento por utilizador, para com base em dados históricos (técnica de análise de cenários), determinar se o utilizador está a andar ou parado.

É possível tirar partido do sensor *Bluetooth* para comunicar com outros *Smartphones* nas proximidades que também correm a aplicação *mtAndroid*. O *Bluetooth* podia avisar o utilizador que existe um outro utilizador próximo (até 10 metros), ou pedir as coordenadas de localização por esta via, poupando uma consulta GPS.

Para garantir escalabilidade, é necessário reprogramar o código fonte do protótipo para usar mais vezes a base de dados, e manter o mínimo de informação possível dos objetos do modelo de domínio em memória.

Por último, deve ser incluída a base de dados na instalação da aplicação, sem ser necessário importar os conteúdos via Internet a primeira vez que se corre a aplicação.

Referências

- [1] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing; IEEE Comput, 34(8), 57-66. (2001)
- [2] B. W. Parkinson and J. J. Spilker. Global Positioning System: Theory and Practice, volume I. American Institute of Aeronautics and Astronautics, Inc., Washington, DC. (1996)
- [3] J. Paek, J. Kim and R. Govindan, Energy-Efficient Rate-Adaptive GPS-based Positioning for Smartphones, in proceedings of 8th International Conference on Mobile Systems, Application, and Services (MobiSys'10). (2010)
- [4] H. Liu, F. Xia, Z. Yang, and Y. Cao, An Energy-Efficient Localization Strategy for Smartphones, Computer Science and Information Systems, Vol. 8, No. 4, 1117-1128. (2011)
- [5] Android Developers, website: <http://developer.android.com/> (2013/01/11)
- [6] A. Kushwaha and V. Kushwaha, Location Based Services using Android Mobile Operating System, IJAET. ISSN: 2231-1963. 14. Vol. 1, Issue 1, pp.14-20. (2011)
- [7] M. B. Kjaergaard, J. Langdal, T. Godsk, and T. Toftkjaer. Entracked: energy-efficient robust position tracking for mobile devices, in Proceedings of 7th International Conference on Mobile Systems, Applications, and Services (MobiSys'09), (2009).
- [8] Merriam, D.F., Kansas 19th century geologic maps. Kansas Academy of Science, Transactions 99, p. 95-114. (1996)
- [9] OpenStreetMap – Tools for Android [online], website: <http://code.google.com/p/osmdroid/>, (2013/01/11)
- [10] OpenStreetMap – The Free Wiki World Map [online]. Available: <http://www.openstreetmap.org/>, (2013/01/11)
- [11] ArcGIS Ressource Center – ArcGIS for Android [online], Available: <http://resources.arcgis.com/content/arcgis-android/about>, (2013/01/11)
- [12] W3C – Points of Interest Working Group [online], Available: <http://www.w3.org/2010/POI/>, (2013/01/11)
- [13] Singh, R, rajsingh.org blog, the geoweb, interoperability, OGC, and random rants [online], Available: <http://www.ajsingh.org/poiwg/poi.xsd>, (2013/01/11)
- [14] Singh, R., Establishing a Global Data Sharing Framework for Place Names [Powerpoint Slides], retrieved from the University of Harvard website (2013/01/11)
- [15] Jones, K., Liu, L., Alizadeh-Shabdiz, F., Improving Wireless Positioning with Look-ahead Map-Matching, in Proceedings of the 2007 Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking&Services (MobiQuitous), p.1-8, August 06-10, (2007)
- [16] Cheng, Y. C., Chawathe, Y., Lamarca, A., Krumm, J., Acuracy Characterization for Metropolitan-scale Wi-Fi Localization, In Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005), p. 233-45 (2005)
- [17] Ann Cavoukian, Kim Cameron, Wi-Fi Positioning Systems: Beware of Unintended Consequences, retrieved from Information and Privacy Commissioner of Ontario-Canada, website: <http://www.ipc.on.ca/images/Resources/wi-fi.pdf> (2013/01/11)
- [18] Byers, S., Kormann, D., 802.11B Access Point Mapping. In Communications of the ACM, May 2003, Vol.46, No. 5. (2003)
- [19] Schilit B. et. al. Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative. In Proceedings of The First ACM International Workshop on Wireless Mobile Applications and Services on WLAN, San Diego, CA. September. (2003)
- [20] Vaughan-Nichols, S.J., How Google - and Everyone else – gets Wi-Fi location data, retrieved from ZDNET [online], website:<http://www.zdnet.com/blog/networking/how-google-8211and-everyone-else-8211gets-wi-fi-location-data/1664>. (2013/01/11)
- [21] Street View – Google Maps [online], website: <http://maps.google.com/streetview>. (2013/01/11)

- [22] Alonistioti, A., Panagiotakis, S., Houssos, N., Kaloxylas, A., Issues for the provision of Location-dependent services over 3G networks, 3rd generation infrastructure and services conference (3GIS), Athens, Greece, July. (2001)
- [23] G. Mao, B. Fidan, and B. D. O. Anderson, Wireless sensor network localization techniques, *Computer Networks*, vol. 51, no. 10, pp. (2007)
- [24] Franceschini, F., A review of localization algorithms for distributed wireless sensor networks in manufacturing. *International journal of computer integrated manufacturing*, 22(7): p. 698-716. (2009)
- [25] Constandache, I., Gaonkar, S., Sayler, M., Choudhury, R., Cox, L., EnLoc: Energy-Aware Localization Using Mobile Phones. In *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services*, pages 2716 –2720. (2009)
- [26] Paek, J., Kim, K., Jatinder P. Singh, J., Govindan, R., Energy-Efficient Positioning for Smartphones using Cell-ID Sequence Matching, in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. (2011)
- [27] A. Kotanen, M. Hännikäinen, H. Leppäkoski, and T. D. Hämäläinen. Experiments on Local Positioning with Bluetooth. In *Proc. IEEE Int. Conf. Inf. Technol.: Comput. Commun.*, 297-303. (2003)
- [28] Wong, L., Potential Bluetooth Vulnerabilities in Smartphones, AISM, 2005, the School of Computer and Information Science (SCIS) & Edith Cowan University, Perth, Australia.
- [29] Woodman, O. and Harle, R., Pedestrian Localization for Indoor Environments, in *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp)*, Seoul, Korea, ACM. (2008)
- [30] Serra, A., Dessi, T., Carboni, D., Popescu, V., Atzori, L., Inertial Navigation Systems for User-Centric Indoor Applications, In *Proceedings of NEM Summit - Towards Future Media Internet Barcelona, Spain*. (2010)
- [31] Serra, A., Carboni, D., Marotto, V., Indoor pedestrian navigation system using a modern smartphone, In *Proceedings of MobileHCI2010*. ACM Press. Lisbon, Portugal. (2010)
- [32] Zhuang, Z., Kyu-Han, K., Singh, J., Improving Energy Efficiency of Location Sensing on Smartphones, In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. (2010)
- [33] Berger, S., McFaddin, S., Narayanaswami, C., Raghunath, M., Web Services on Mobile Devices - Implementation and Experience, *wmcsa*, pp.100, Fifth IEEE Workshop on Mobile Computing Systems & Applications. (2003)
- [34] Pashtan, A., Heuser, A., Sheuermann, P., Personal Service Areas for Mobile Web Applications. *IEEE Internet Computing* 8, 6. (2004)
- [35] Schwinger, W., Grin, C., Prill, B., and Retschitzegger, W. A light-weight framework for location-based services. In *Lecture Notes in Computer Science (Berlin, 2005)*, Springer, pp. 206_210. (2005)
- [36] Belchior, A., Rodrigues da Silva, A., eXcitingTrails/Events: Events for Turistic Scenarios, VI Congresso Ibero-americano de Telemática (CITA 2011). (2011)
- [37] Liu, C., Zhu, Q., Holroyd, K., Seng, E., Status and trends of mobile-health applications for iOS devices: A developer's perspective, *Journal of Systems and Software*, Volume 84, Issue 11, Pages 2022-2033. (2011)
- [38] Gartener, Forecast: Mobile Communications Devices by Open Operating System, Worldwide, 2008-2015, retrieved from Gartener [online], website: <http://www.gartner.com/resId=1619615> (2013/01/11)
- [39] Ladetto, Q., Gabaglio, V., Merminod, B., Combining Gyroscopes, Magnetic Compass and GPS for Pedestrian Navigation, *Proc. Int. Symposium on Kinematic Systems in Geodesy, Geomatics and Navigation (KIS 2001)*, pp 205-213. (2001)
- [40] Ladetto, Q., Merminod, B., Digital magnetic compass and gyroscope integration for pedestrian navigation, in *9th Saint Petersburg International Conference on Integrated Navigation Systems*, Saint Petersburg, Russia, May 27-29. (2002)
- [41] Kothari, N., Kannan, B., Dias, M.B., Robust Indoor Localization on a Commercial Smart-Phone, CMU-RI-TR-11-27. (2011)
- [42] Google Maps [online], website: <http://maps.google.com> (2013/01/11)
- [43] Stackoverflow [online], website: <http://stackoverflow.com> (2013/01/11)

- [44] Georelated.com, [online], Cloud Web Map API Services Reviewed - Bing Maps, website: http://www.georelated.com/2012/02/cloud-web-map-api-services-reviewed_19.html (2013/01/11)
- [45] Georelated.com, Cloud Web Map API Services Reviewed - Google Maps [online], website: <http://www.georelated.com/2012/02/cloud-web-map-api-services-reviewed.html> (2013/01/11)
- [46] Bing Maps Android SDK [online], website: <http://bingmapsandroidsdk.codeplex.com/> (2013/01/11)
- [47] Davis, R. E., Frey, R., Sarquis, M, and Sarquis, J. Holt, Rinehart and Winston , Modern Chemistry, Austin, TX, (2009)
- [48] Hightower, J. and Borriello, G., Location Sensing Techniques. Technical Report , Computer Science and Engineering, University of Washington (2001)
- [49] Maxim, P., Hettiarachchi, S., Spears, W., Spears, D., Hamman, J., Kunkel, T., Speiser, C., Trilateration localization for multi-robot teams. Sixth International Conference on Informatics in Control, Automation and Robotics, Special Session on Multi-Agent Robotic Systems (2008)
- [50] Constandache, I., Choudhury, R., Rhee, I., CompAcc: Using Mobile Phone Compasses and Accelerometers for Localization, INFOCOM, San Diego, USA (2010)
- [51] Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A., A Survey of Mobile Phone Sensing. In IEEE Communications Magazine (2010)
- [52] Alzantot, M., Youssef, M., UPTIME: Ubiquitous Pedestrian Tracking using Mobile Phones, IEEE WCNC (2012)
- [53] Das, S., Green, L., Perez, B., Murphy, M., Detecting User Activities using the Accelerometer on Android Smartphones (2010)
- [54] Seifert, K., Camacho, O. Implementing Positioning Algorithms Using Accelerometers. Application Note AN3397 Rev 0, 02/2007. Freescale Semiconductor (2007)
- [55] Kwapisz, J., Weiss, G., Moore, S., Activity recognition using cell phone accelerometers. SIGKDD Explor News1 12: 74–82 (2011)
- [56] Baskerville, R.L., Investigating Information Systems with action research. Communications of the Association for Information Systems (1999)
- [57] Vogella.com [online], website: <http://www.vogella.com> (2013/01/11)
- [58] R. Murmura, J. Medsger, A. Stavrou, J. M. Voas, Mobile Application and Device Power Usage Measurements, IEEE International Conference on Software Security and Reliability, Washington D.C., USA (2012)
- [59] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. M. Mao, L. Yang, Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In CODES+ISSS (2010)
- [60] Fowler, M., Patterns of Enterprise Application Architecture, Addison-Wesley, pp.200-214 (2003)
- [61] Geocaching [online], website: <http://www.geocaching.com> (2013/01/11)
- [62] MobileTrails [online], website: <http://www.siquant.pt/portal/MobileTrails@245.aspx> (2013/01/11)
- [63] Rodrigues da Silva, A., Videira, C., UML, Metodologias e Ferramentas CASE, 2ª Edição, Volume 1, Centro Atlântico Editora (2005)
- [64] Ribeiro, A., Rodrigues da Silva, A., Survey on Cross-Platforms and Languages for Mobile Apps, in Proceedings of QUATIC'2012 Conference, (2012)
- [65] Ferreira, J. C., Rodrigues da Silva, A., Afonso, J. L., EV-Cockpit - Mobile Personal Travel Assistance for Electric Vehicles, in Proceedings of AMAA 2011, 15th International Forum on Advanced Microsystems for Automotive Applications (2011)
- [66] Teixeira, H., MobileTrails: Aplicação Móvel com Localização Geográfica, MSc Thesis (MERC) IST/UTL (2009)