INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrónica e
Telecomunicações e de Computadores

Simulação de Plataformas para Dispositivos Móveis
Infraestrutura de Simulação Sensorial para *Android*

Jorge Miguel Alves Zeferino

(Licenciado)

Projeto para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientador: Doutor Luís Filipe Morgado

Júri:

Presidente: Mestre Pedro Alexandre Pereira
Vogais: Mestre Paulo Barros Pereira
Doutor Luís Filipe Morgado

Outubro de 2014

Índice

Lista de figuras

Agradecimentos

Agradeço ao Professor Doutor Luís Morgado, por ter assumido a orientação deste projeto, pelo estímulo, colaboração e apoio que me proporcionou, e pela disponibilidade que sempre mostrou para analisar alternativas e discutir soluções. A sua consideração e experiência foram fundamentais e ajudaram a encontrar os caminhos adequados para a concretização deste projeto.

Agradeço aos Professores Carlos Duarte e Paulo Pereira, pelas discussões, opiniões, sugestões e sabedoria que partilharam ao longo do meu percurso académico.

Agradeço aos colegas que conheci no ISEL e com que tive oportunidade de crescer e evoluir como pessoa e estudante, nomeadamente o Igor Cândido, o Nelson Matias e o João Martins.

Agradeço ao Nuno Fernandes, amigo de longa data, por todo o apoio.

Por último, mas mais importante, agradeço aos meus pais e irmão, Guilherme Zeferino, Maria Zeferino e João Zeferino pelo apoio incondicional e por me favorecerem acima de tudo.

Resumo

A crescente disponibilidade de dispositivos móveis, nomeadamente, *smartphones* e *tablets*, tem dinamizado o desenvolvimento de aplicações para as plataformas de operação desses dispositivos, como é o caso da plataforma *Android*. Normalmente, o desenvolvimento de aplicações é realizado numa plataforma hospedeira, *Microsoft Windows*, com base em simuladores da plataforma alvo. Estes simuladores são tipicamente fornecidos pelo próprio fabricante ou outros, apresentando diferentes tipos de restrições, quer em termos de recursos computacionais requeridos para o seu funcionamento, quer em termos de desempenho e funcionalidade. Existe diverso *hardware* que não se encontra disponibilizado ao nível do simulador, como é o caso do suporte de multitoque, sensores e *Global Positioning System* (GPS).

As soluções existentes para fazer a simulação sensorial caracterizam-se como não completas, não corretas ou dependentes de plataformas externas. Estas soluções apresentam um conjunto de funcionalidades reduzidas ou requerem versões específicas da plataforma *Android*, inibindo a sua utilização imediata.

O presente relatório apresenta a proposta de uma infraestrutura de simulação sensorial que prevê um comportamento dinâmico e independente da versão da plataforma *Android*, enfatizando assim o suporte de atualizações da plataforma e a adição de novos sensores.

Com base na proposta referida, descreve-se o projeto e a implementação de uma infraestrutura de simulação sensorial em tempo real para a plataforma *Android* que suporta a utilização da biblioteca de sensores usando os sensores de um dispositivo real.

Palavras-chave: *Android*, sensores, simulação, portabilidade, tempo real.

Abstract

The growing availability of mobile devices, including smartphones and tablets, has spurred the development of applications for the operating platforms of these devices such as the Android platform. Most of the mobile development is done in a host machine running for example Microsoft Windows with the platform running on the simulation target. These simulators are typically supplied by the manufacturer or others. All have different constraints in terms of computing resources required for its functioning in terms of performance and functionality. There is some hardware which is not available in the simulator as the case of multi-touch information, sensors and Global Positioning System (GPS).

Existing solutions to simulate sensors are characterized as incomplete, incorrect or dependent on external platforms. These solutions offers lack of functionality or require specific versions of the Android platform, inhibiting its immediate use.

This paper presents a proposal which provides a dynamic and Android version independent thus emphasizing the updates of the platform and the addition of new sensors.

Based on the proposal referred we describe the design and implementation of an infrastructure for real time sensor simulation for Android platform that supports the use of sensor library using the sensors of a real device.

Keywords: Android, sensors, simulation, portability, real time.

In all forms of strategy, it is necessary to maintain the combat stance in everyday life and to make your everyday stance your combat stance.

Miyamoto Musashi

Capítulo 1

Introdução

Os dispositivos móveis são, atualmente, utilizados por grande parte da população e cada vez mais existem aplicações capazes de realizar diferentes tipos de tarefas.

No âmbito do desenvolvimento de aplicações móveis é comum ser fornecido pelos proprietários da tecnologia um conjunto de ferramentas que permitem desenvolver e testar o código aplicacional. Agregado aos pacotes de desenvolvimento existe normalmente o simulador de um dispositivo móvel que executa o respetivo sistema operativo e aplicações.

O desenvolvimento de aplicações móveis é normalmente efetuado num ambiente de simulação que é executado num computador, tornando o processo de desenvolvimento mais eficiente, versátil e integrado, comparado com o desenvolvimento feito diretamente num dispositivo móvel. Usando o simulador é possível fazer alterações de versão de sistema, configurar opções relativas ao tamanho do ecrã e requisitos de *hardware* com facilidade. Os ambientes de simulação, têm no entanto, um conjunto de funcionalidades que não podem ser testadas dado que não existe *hardware* que dê esse suporte, como por exemplo a utilização de sensores, GPS, e multitoque.

O foco deste projeto centra-se na biblioteca de sensores da plataforma *Android*. Esta biblioteca disponibiliza um conjunto de operações relacionadas com os sensores. Os sensores são de tipos variados e podem, por exemplo, reportar dados relativos à pressão atmosférica, temperatura e iluminação presente no ambiente. A plataforma *Android*, originalmente, não fornece nenhuma forma integrada de utilizar os sensores e respetivas operações num simulador. Existem, no entanto, outros simuladores ou

aplicações concebidas por terceiros que permitem fazer a simulação de alguns sensores mais comuns, contudo estas soluções têm limitações.

A solução a implementar deve permitir que uma aplicação que use a infraestrutura de simulação sensorial possa ser desenvolvida e testada num simulador qualquer, usando os dados dos sensores de um dispositivo real. Esta solução propõe que o código desenvolvido durante a simulação seja idêntico ao da biblioteca de sensores da plataforma *Android*, permitindo assim transpor esse código para um dispositivo real sem que este sofra alterações significativas.

1.1 Objetivos e contribuição

Com o objetivo de se obter uma solução útil e inovadora para ser disponibilizada à comunidade, a infraestrutura de simulação sensorial deve obedecer às seguintes propriedades:

i) Portabilidade - O código realizado usando a infraestrutura de simulação sensorial deve poder ser executado diretamente na plataforma *Android* sem ter de sofrer alterações significativas de semântica;

ii) Atualização - A infraestrutura deve suportar futuras atualizações da *Application Programming Interface* (API) de sensores da plataforma *Android*;

iii) Independência - A infraestrutura deve poder ser executada em qualquer versão da plataforma *Android*, desde que esta seja superior à versão 2.3 (*Gingerbread*);

iv) Completude - Todas as funções essenciais e sensores devem poder ser utilizados, desde que estejam disponíveis no dispositivo real;

v) Tolerância a falhas – Em caso de falta de rede a solução deve ser capaz de transmitir essa informação ao programador;

vi) Desempenho - Deve ser possível fazer o uso da infraestrutura em tempo real sem que seja observado impacto visual significativo.

Este projeto contribui com uma infraestrutura portável que pode ser utilizada em qualquer versão da plataforma *Android* superior à 2.3 e em qualquer simulador que execute a plataforma *Android*. Esta característica é conseguida através do uso de reflexão e tirando partido de técnicas da linguagem *java* e da plataforma *Android*.

Do ponto de vista do programador, o uso da infraestrutura é idêntico ao uso da biblioteca de sensores da plataforma *Android*, existindo apenas ligeiras diferenças que serão explicadas mais à frente.

As principais características da solução são a portabilidade e a independência.

1.2 Metodologia de desenvolvimento

Como metodologia de desenvolvimento do projeto foi utilizado o processo *Scrum*. O processo *Scrum* é um processo ágil, proposto por *Ken Schwaber*, baseado numa abordagem iterativa e incremental de desenvolvimento de *software*. O processo é normalmente suportado por equipas de projeto compostas por poucos elementos [1]. No contexto deste projeto, dado que apenas existem dois elementos (aluno e o orientador) envolvidos, o processo sofreu ligeiras alterações. A Figura 1.1 apresenta os conceitos do processo *Scrum*, aplicados no contexto deste projeto.

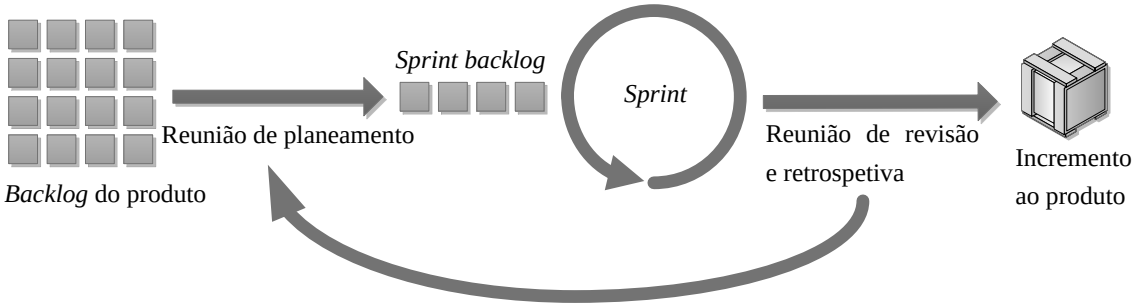

Figura 1.1: Processo *Scrum* adaptado ao contexto do projeto

O *Backlog* do produto representa uma lista com todos os requisitos do produto. A reunião de planeamento antecede o *Sprint Backlog* e tem como propósito identificar quais os requisitos que vão ser selecionados para dar seguimento ao mesmo.

O *Sprint Backlog* é uma lista com todas as tarefas que se pretendem realizar durante o próximo *Sprint*, de forma a implementar um determinado número de requisitos selecionados a partir do *Backlog* do produto.

Os *Sprints* são iterações que têm a duração de duas a quatro semanas. No final de cada *Sprint*, uma versão do produto com novas funcionalidades deve estar concluída. As

tarefas que ficaram por concluir retornam ao *Backlog* do produto. Antes do início de um novo *Sprint* é realizada uma reunião, a reunião de revisão e retrospetiva, na qual são apresentadas ao orientador as funcionalidades implementadas no último *Sprint*. O orientador é convidado a pronunciar-se sobre possíveis alterações, que serão consideradas em futuros *Sprints*. Por fim, é feita uma retrospetiva, ainda nesta reunião, que serve para a análise da aplicação do processo *Scrum*, promovendo a discussão de melhores práticas e formas de tornar o processo mais eficiente.

O processo é reiniciado, iniciando o seu novo ciclo na reunião de planeamento. São feitos tantos ciclos quanto os suficientes para terminar todos os requisitos do *Backlog* do produto.

1.2.1 Aplicação do processo *Scrum*

No início do desenvolvimento deste projeto, com o objetivo de adquirir informação para a criação do *Backlog* do produto, foi usada preferencialmente a informação disponível em diferentes artigos científicos relativos à computação móvel e sensorial. Foi também considerada a informação presente na página da internet da plataforma *Android*, *Android Developers* [2].

Uma vez adquirido conhecimento através da investigação inicial, foi iniciado um ciclo do processo *Scrum*, permitindo alcançar um protótipo minimalista. A partir desse protótipo iniciaram-se novos ciclos que permitiram efetuar as alterações necessárias para a obtenção de um protótipo melhorado. Cada um dos protótipos melhorados possuí novas funcionalidades que foram adicionadas com base na avaliação feita ao último protótipo.

O primeiro ciclo correspondeu à exploração da biblioteca de sensores da plataforma *Android*, ao estudo de soluções existentes e à definição dos requisitos da aplicação (entre Dezembro de 2013 e Janeiro de 2014).

O segundo ciclo consistiu na definição da aplicação móvel, dando ênfase ao mecanismo de comunicação (entre Fevereiro de 2014 e Abril de 2014). No terceiro ciclo (entre Abril de 2014 e Junho de 2014), foi desenvolvida a biblioteca de simulação sensorial e avaliada a solução existente. Neste ciclo foi também realizado o primeiro protótipo que incluía a interação entre todos os componentes da solução. O quarto ciclo (Julho de 2014) consistiu em analisar as estratégias implementadas e melhorar detalhes da implementação.

Um ciclo completo pode ser repartido em pequenos ciclos que são responsáveis por construir e analisar os resultados da implementação de pequenas funcionalidades.

1.3 Estrutura do documento

Este relatório de projeto de mestrado está organizado em duas partes distintas. Na primeira parte é feita uma apresentação do contexto do projeto, onde são descritos os principais temas envolvidos, as soluções encontradas e os conceitos fundamentais. Na segunda parte são apresentadas as contribuições específicas do projeto realizado, que dizem respeito à arquitetura e à solução proposta.

Para além deste capítulo de introdução, os temas abordados encontram-se organizados em mais cinco capítulos, cujo conteúdo é o seguinte:

Capítulo 2: Capítulo de enquadramento, onde são abordados os conceitos subjacentes ao estado do desenvolvimento de aplicações *Android,* é apresentada a biblioteca de sensores da plataforma *Android* e é apresentado o trabalho relacionado, onde são discutidas as diferentes abordagens encontradas na literatura.

Capítulo 3: Descreve a arquitetura subjacente à solução proposta, enfatizando as duas componentes fundamentais da solução e fazendo a sua decomposição.

Capítulo 4: Contém a descrição dos detalhes mais relevantes na implementação da solução da infraestrutura.

Capítulo 5: Apresenta uma comparação entre a utilização da infraestrutura e da biblioteca de sensores da plataforma *Android*, é explicado o modo de utilização da infraestrutura e é mencionado o desempenho obtido.

Capítulo 6: Capítulo onde são apresentadas as principais dificuldades sentidas e conclusões resultantes do projeto realizado, bem como possíveis caminhos para desenvolvimento futuro.

Capítulo 2

Enquadramento

Este capítulo começa por apresentar a plataforma *Android*, destacando a sua arquitetura e particularidades. São explicadas as características e o funcionamento da biblioteca de sensores da plataforma *Android*. O capítulo termina com a descrição das principais soluções existentes que tentam resolver o problema da simulação sensorial para a plataforma *Android*.

2.1 A plataforma *Android*

O *Android* é uma plataforma gratuita e de código aberto (*open-source*), que deriva do sistema operativo *Linux*, fundada em Outubro de 2003 por *Andy Rubin* e adquirida em 2005 pela empresa *Google*. Em 2007 foi anunciada a *Open Handset Alliance* (OHA), consórcio de empresas com o objetivo de desenvolver padrões abertos para dispositivos móveis, onde o sistema operativo é oficialmente tornado num *software* de código aberto. Em 2008, o *Android Software Development Kit* (SDK) 1.0 é lançado. Em 2009 assiste-se a um aumento dos dispositivos baseados em *Android* e ao lançamento de novas versões do sistema operativo. De seguida é apresentado um resumo do histórico de versões e respetiva API da plataforma *Android* até à data:

- *Android* 1.0 (API nível 1) - Setembro de 2008;
- *Android* 1.1 (API nível 2) - Fevereiro de 2009;
- *Android* 1.5 (API nível 3) *Cupcake* - Abril de 2009;

- *Android* 1.6 (API nível 4) *Donut* - Setembro de 2009;
- *Android* 2.0/2.1 (API nível 5, 6, 7) *Eclair* - Outubro de 2009 / Janeiro de 2010;
- *Android* 2.2.x (API nível 8) *Froyo* - Maio de 2010;
- *Android* 2.3.x (API nível 9, 10) *Gingerbread* - Dezembro de 2010;
- *Android* 3.x (API nível 11, 12, 13) *Honeycomb* - Fevereiro de 2011;
- *Android* 4.0.x (API nível 14, 15) *Ice Cream Sandwich* - Outubro de 2011;
- *Android* 4.1/4.2/4.3 (API nível 16, 17, 18) *Jelly Bean* - Junho de 2012;
- *Android* 4.4 (API nível 19, 20) *KitKat* - Outubro de 2013;
- *Android* 5.0 (API nível 21) *Lollipop* - Novembro de 2014.

Na página da internet [3] da plataforma existe uma área dedicada a estatísticas relacionadas com a utilização da mesma. O gráfico presente na Figura 2.1 representa os dispositivos que acederam à loja de aplicações da plataforma *Android* (*Android Market*) durante um período de sete dias até 9 de Setembro de 2014.

A versão do sistema operativo mais usada é a versão *Jelly Bean* com cerca de 53,8%, em segundo lugar a versão *KitKat* com 24.5% e em terceiro lugar a versão *Gingerbread* com cerca de 11,4%. Verifica-se a natural tendência para cada vez mais os dispositivos estarem munidos de um sistema operativo mais recente, ainda que a versão *Jelly Bean* represente uma percentagem muito significativa.

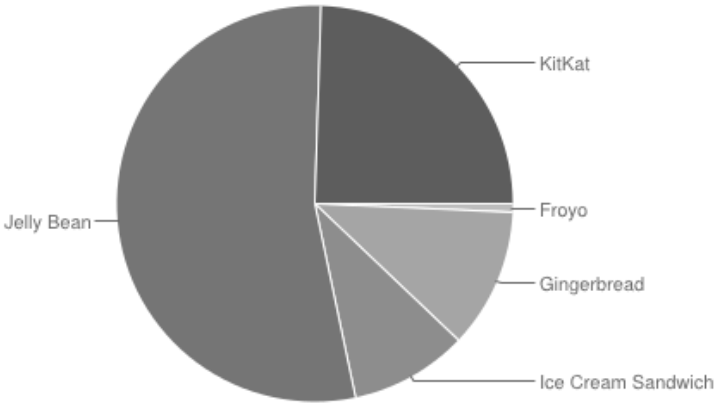

Figura 2.1: Percentagem do uso das versões da plataforma *Android* [3]

2.1.1 A arquitetura da plataforma *Android*

A arquitetura da plataforma *Android* está organizada em quatro camadas, como pode ser observado na Figura 2.2. Caracterizando as camadas no sentido de mais baixo nível para o mais alto nível tem-se a seguinte organização:

A camada de mais baixo nível (*Linux Kernel*), tem por base um *kernel Linux*, que foi adaptado pela *Google* para servir os dispositivos móveis. Estas adaptações relacionam-se com a gestão de energia, gestão de segurança, suporte de *hardware* e desempenho do sistema de ficheiros. Esta camada é também responsável por conter todos os controladores de dispositivos (*device drivers*), como o da câmara, teclado, ecrã, etc.

A camada *Libraries* inclui bibliotecas escritas em C e C++, que são expostas ao programador através da camada *Application Framework*, recorrendo a *bindings* do *Java Native Interface* (JNI). Estas bibliotecas fornecem a grande maioria das funcionalidades relacionadas com o tratamento de dados e a sua apresentação, nomeadamente: *SQLite* - motor de base de dados, *OpenGL* - atualização do comportamento gráfico (*rendering*) *2D* e *3D*, *WebKit* - visualizador de *HyperText Markup Language* (HTML), etc.

A este nível existe o núcleo de bibliotecas *java* que oferecem a maior parte das funcionalidades do *Java Standard Edition* (*java* SE). É também a este nível que se encontra a máquina virtual *Dalvik* onde o código de cada aplicação *Android* é executado.

A máquina virtual *Dalvik* [4] executa *bytecodes Dalvik* (armazenado em ficheiros com formato *.dex*) que são o resultado da compilação das classes de uma aplicação *Android*. Ao contrário da máquina virtual do *java*, esta tem uma arquitetura baseada em registos em vez de pilha e está otimizada para dispositivos com baixa memória, ocupando menos espaço. É ainda possível existirem múltiplas instâncias da máquina virtual em simultâneo, proporcionando segurança e isolamento.

As camadas de mais alto nível são a camada *Application Framework* [5] e a camada de nível aplicacional (*Applications*). A camada *Application Framework* é fornecida aos programadores de forma a possibilitar a criação de aplicações para a plataforma. Esta camada é bastante rica e permite aos programadores tirarem partido do *hardware*, do sistema de localização, alarmes, notificações, entre outras.

Por fim, existe a camada de nível aplicacional que disponibiliza as aplicações que existem no dispositivo. Entre elas destacam-se o gestor de contactos, gestor de mensagens e marcador de chamadas.

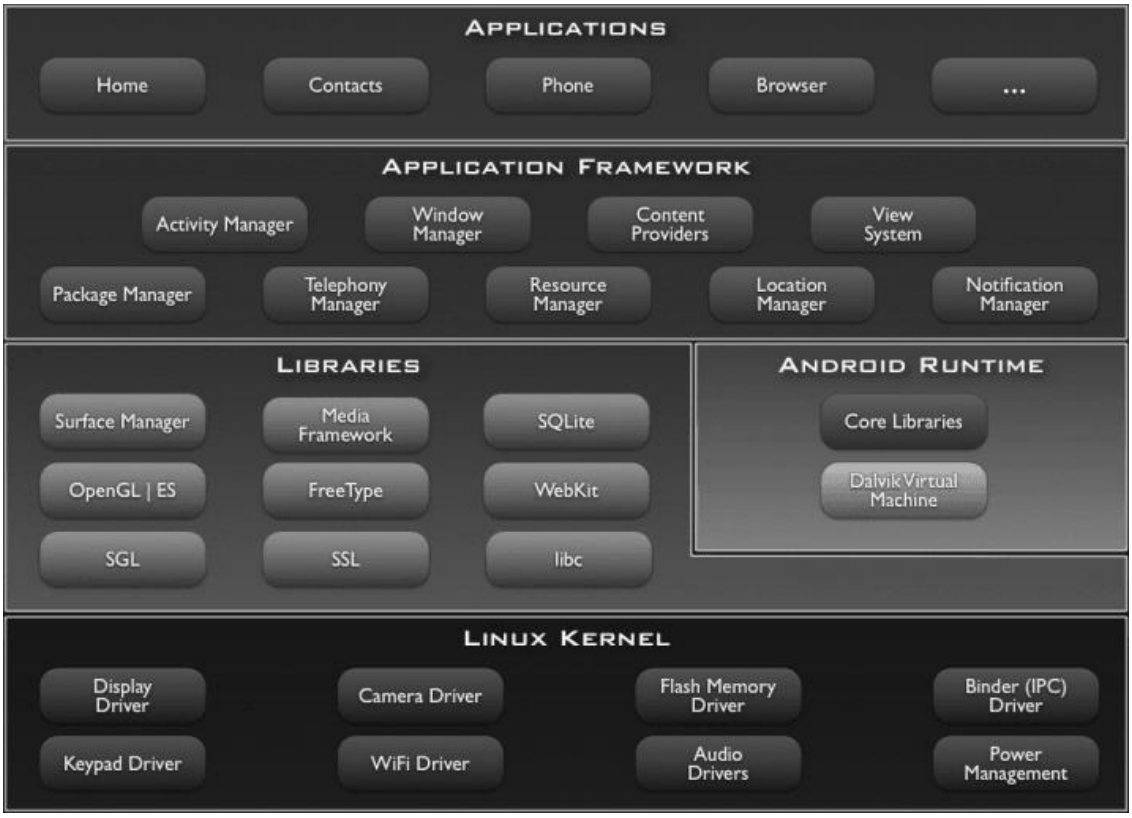

Figura 2.2: Arquitetura da plataforma *Android* [6]

O *Android* é uma plataforma cujo sistema operativo é caracterizado por executar cada aplicação num processo distinto da outra. Grande parte da segurança é garantida pelo sistema operativo *Linux* ao nível do processo. Existe também um mecanismo de segurança de mais alto nível (nível de aplicação) que obriga o programador a especificar em *Extensible Markup Language* (XML*)* num ficheiro de manifesto (*Manifest*) quais as permissões que pretende requerer na sua aplicação. Este tipo de permissões diz respeito, por exemplo, ao acesso à rede sem fios ou ao acesso à localização. No momento da instalação de uma aplicação *Android* todas as permissões são exibidas ao utilizador para que o mesmo possa aceitar ou rejeitar as mesmas.

No que diz respeito ao desenvolvimento, a *Google* fornece um conjunto de ferramentas designadas por *Java Development Tools* (*JDK*). Estas ferramentas são constituídas pelo código fonte da *Application Framework Android*, um *Integrated Development Environment* (IDE) de nome *Eclipse* para desenvolvimento, e um

depurador de erros (*debugger*) integrado ao simulador e ao *Eclipse* que permite executar e testar a plataforma *Android* e as respetivas aplicações.

2.2 Organização da estrutura do código fonte da plataforma *Android*

No decorrer do estudo da plataforma *Android*, houve a necessidade de consultar algum do código fonte disponível. Mais precisamente, código relacionado com a biblioteca de sensores e código relacionado com ferramentas e com o simulador. O esforço subjacente à compreensão da estrutura do código fonte foi elevado dado que a sua dimensão é elevada. A título de referência, é de seguida resumida a estrutura de pastas de primeiro nível mais relevante do código fonte:

- *Bionic* - Contém a distribuição em C da *Berkeley Software Distribution* (BSD). Esta distribuição contém componentes para o *kernel* do *Linux* como o *scheduler*, gestor de memória, controladores de dispositivos, etc. A este nível existe também a biblioteca da linguagem C;

- *Bootable* - Contém exemplos de código de iniciação (*bootloaders*) para alguns processadores como o *armv6*. Este diretório inclui um conjunto de ferramentas para manipulação do sector de arranque e imagens de recuperação;

- *Build* - Contém todos os ficheiros necessários para dar início ao processo de compilação da plataforma *Android*. É possível fazer a compilação individual de qualquer módulo, por exemplo do simulador, do código de modo de utilizador, do código de virtualização para *VirtualBox*. A este nível encontram-se também as ferramentas necessárias para possibilitar o processo de compilação;

- *Cts* - Contém testes de compatibilidade. Encontra-se dividida em pastas conforme as diferentes partes da plataforma *Android* a serem testadas. Aqui são testados componentes como: aceleração gráfica, segurança de uma aplicação, áudio, vídeo. Neste diretório é possível utilizar um conjunto de ferramentas que não fazem parte do processo de compilação. A sua utilidade está relacionada com funções como a análise de ficheiros *Dalvik*;

- *Dalvik* - Contém todo o código relacionado com a máquina virtual *Dalvik*. Entre as várias funcionalidades destacam-se o gerador código *Dalvik*, a geração de métodos presentes num ficheiro *Dalvik*.

- *Development* - Contém um conjunto de aplicações que não são incluídas no sistema operativo mas que são úteis para o desenvolvimento. É disponibilizada a interface da *Native Development Kit* (NDK), que disponibiliza a possibilidade de utilizar numa aplicação *Android* código nativo. São disponibilizadas várias aplicações de exemplo. Por fim, existe também um conjunto de recursos com variadas funções desde depuração de erros, como testes à ligação *Transmission Control Protocol* (TCP) e configurações para os ambientes de desenvolvimento *Eclipse* e *IntelliJ*;

- *Device* - Contém código de configuração específico para dispositivos *Samsung*, *Motorola* e *HTC*. Contém informação relativa à criação de bibliotecas partilhadas sem haver a necessidade de alteração da plataforma;

- *Docs* - Contém tutoriais e referências relacionadas com o *Android Open Source Project* (AOSP);

- *External* - Contém todas as bibliotecas externas utilizadas na compilação. Por exemplo: *Apache-http*, *Chromium*, *Guava*, *SQLite*, *Skia*;

- *Frameworks* - Contém o código da camada *Application Framework* da arquitetura da plataforma *Android*;

- *Hardware* - Inclui o código que faz a abstração do *hardware*, por exemplo: GPS, áudio, câmara, sensores;

- *Libcore* - Contém bibliotecas para realização de testes unitários e utilização de XML. Inclui a implementação da *JavaScript Object Notation* (JSON);

- *Ndk* - Contém a implementação e documentação da biblioteca NDK que permite usar as linguagens C e C++ em aplicações *Android*;

- *Packages* - Contém as aplicações que fazem parte da plataforma *Android*, por exemplo: aplicação para uso de *Bluetooth*, motor de pesquisa, calendário.

Contém ainda aplicações experimentais que não fazem parte da plataforma e um conjunto de *Content Providers* (explicado mais à frente) para os diferentes tipos de dados disponíveis na plataforma;

- *Sdk* - Contém aplicações que não fazem parte da plataforma mas são úteis para o programador;

- *System* - Contém o código fonte do *Bluetooth*, ferramentas para a manutenção da plataforma *Android* (*adb*, *fastboot*) e aplicações para teste de várias funcionalidades.

2.3 Biblioteca de sensores da plataforma *Android*

Grande parte dos dispositivos que executam a plataforma *Android* possuem sensores que permitem medir a orientação, deslocação e várias condições ambientais. Estes sensores têm uma precisão e exatidão elevada, permitindo ao programador tomar decisões em tempo real, por exemplo, detetando se o dispositivo está a ser abanado ou invertido.

2.3.1 Tipos de sensores

A plataforma *Android* categoriza os sensores em três tipos diferentes:

i) Os sensores de movimento, que medem forças de aceleração e rotação em três eixos (x, y, z);

ii) Os sensores de ambiente, que medem parâmetros como a temperatura, pressão do ar, iluminação e humidade;

iii) Os sensores de posição, que medem a posição física do dispositivo.

Os sensores podem ser constituídos por *hardware* ou por *software*. Os sensores constituídos por *hardware* resultam de componentes que vêm previamente instalados no dispositivo móvel e reportam dados medidos diretamente do ambiente. Os de *software*, também conhecidos como sensores virtuais, são constituídos por um ou mais sensores

de *hardware*. Tipicamente, os sensores de aceleração linear e de gravidade são sensores concebidos por *software.*

Grande parte dos dispositivos comercializados tem apenas alguns sensores instalados, nomeadamente, o acelerómetro, o magnetómetro e sensor de proximidade.

De seguida é feita a descrição dos sensores existentes, separados pelo seu tipo. A sua descrição é constituída pelo identificador do sensor e uma breve explicação sobre o mesmo.

Sensores do tipo *hardware*:

i) *TYPE_ACCELEROMETER* - Acelerómetro, mede a força da aceleração aplicada nos três eixos do dispositivo, incluindo a força da gravidade em metros por segundo quadrado (m/s^2);

ii) *TYPE_AMBIENT_TEMPERATURE* - Termómetro, mede a temperatura do ambiente em graus *Celsius* (°C);

iii) *TYPE_GYROSCOPE* - Giroscópio, mede a taxa de rotação do dispositivo em radianos por segundo para cada um dos três eixos;

iv) *TYPE_LIGHT* - Sensor de iluminação, mede o nível da luz ambiente em *Lux* (lx);

v) *TYPE_MAGNETIC_FIELD* - Magnetómetro, mede o campo magnético do ambiente para cada um dos três eixos em micro *Tesla* (μT);

vi) *TYPE_PRESSURE* - Sensor de pressão, mede a pressão do ar ambiente em micro *Bar* (mbar);

vii) *TYPE_PROXIMITY* - Sensor de aproximação, mede a proximidade de um objeto em centímetros relativamente ao ecrã do dispositivo;

viii) *TYPE_RELATIVE_HUMIDITY* - Sensor de humidade, mede a humidade relativa do ambiente em percentagem;

ix) *TYPE_TEMPERATURE* - Termómetro, mede a temperatura do dispositivo em graus *Celsius*. Este sensor foi substituído pelo sensor

TYPE_AMBIENT_TEMPERATURE na versão 4.0 da plataforma *Android* dado que existiam diferentes implementações em diferentes dispositivos.

Sensores do tipo *software*:

i) *TYPE_ORIENTATION* - Sensor de orientação, mede os graus de rotação que o dispositivo faz em torno dos três eixos.

Sensores dos dois tipos:

i) *TYPE_ROTATION_VECTOR* - Sensor de rotação, mede a orientação do dispositivo fornecendo os três eixos num vetor de rotação;

ii) *TYPE_LINEAR_ACCELERATION* - Sensor de aceleração linear, mede a força de aceleração aplicada ao dispositivo nos três eixos excluindo a força da gravidade em m/s^2;

iii) *TYPE_GRAVITY* - Sensor de gravidade, mede a força da gravidade que é aplicada no dispositivo nos três eixos em m/s^2.

2.3.1.1 Disponibilidade dos sensores

A disponibilidade de sensores varia de dispositivo para dispositivo. Existem dispositivos que possuem um número elevado de sensores e outros com número reduzido. Esta variação está normalmente associada ao preço do dispositivo, normalmente, os dispositivos mais caros têm um número maior de sensores. No entanto, existe também variação de versão para versão da plataforma *Android*. Isto acontece porque ao longo da evolução da plataforma *Android* foram feitas alterações à biblioteca de sensores. Por exemplo, alguns sensores foram introduzidos na versão 1.5 da plataforma *Android* mas foram implementados apenas na versão 2.3. Na versão 1.6 da plataforma *Android* foram também introduzidos e implementados novos sensores. Ao longo do tempo foi desencorajado o uso de dois sensores, nomeadamente o *TYPE_ORIENTATION* e o *TYPE_TEMPERATURE*, dado que existiam inconsistências na sua implementação em diferentes sistemas operativos de diferentes fabricantes.

2.3.1.2 Sistema de coordenadas

A biblioteca de sensores usa um sistema de coordenadas baseado em três eixos. Para a maior parte dos sensores o sistema de coordenadas usado é relativo ao ecrã do dispositivo quando este se encontra na sua posição por omissão, como pode ser observado na Figura 2.3.

Quando o dispositivo se encontra na sua posição por omissão o eixo da coordenada x está na horizontal e aponta para a direita, o eixo da coordenada y está na vertical e aponta para cima e o eixo da coordenada z aponta para a frente na direção de fora do ecrã, ou seja, na parte de trás do ecrã o eixo da coordenada z tem valores negativos.

Este sistema é utilizado nos sensores *TYPE_ACCELEROMETER*, *TYPE_GRAVITY*, *TYPE_GYROSCOPE*, *TYPE_LINEAR_ACCELERATION* e *TYPE_MAGNETIC_FIELD*.

Quando o dispositivo é rodado, o eixo permanece fixo. Em casos onde seja necessário usar o dispositivo de lado é necessário fazer o remapeamento das coordenadas.

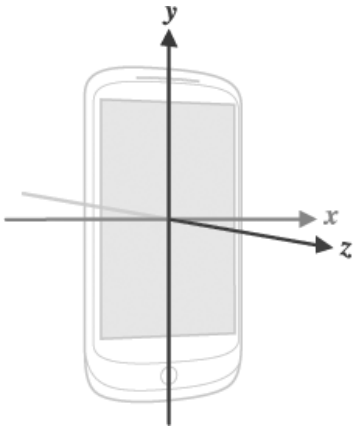

Figura 2.3: Representação dos eixos em relação ao dispositivo

2.3.2 Operações associadas aos sensores

Numa utilização típica da biblioteca de sensores são necessárias duas operações básicas: enumerar os sensores disponíveis e monitorar os eventos dos mesmos. A enumeração dos sensores é feita em tempo de execução, permitindo saber quais os sensores disponíveis e com essa informação tomar decisões do ponto de vista aplicacional. A monotorização dos eventos dos sensores representa a forma como é

possível adquirir dados dos mesmos. Após ter sido requerida esta informação, ocorre um evento, sempre que um sensor tiver novos dados prontos para serem consumidos.

A biblioteca de sensores é constituída por três classes e uma interface que possibilitam a implementação de operações associadas aos sensores. De seguida será feito um resumo das responsabilidades de cada uma.

i) `SensorManager` - Esta classe é o ponto de partida para a utilização das funcionalidades da biblioteca de sensores. Ela fornece métodos que permitem aceder à lista de sensores bem como requerer os eventos de um dado sensor assincronamente;

ii) `Sensor` - Esta classe representa um sensor e permite saber detalhes sobre o mesmo. Estes detalhes incluem o seu nome, potência, fabricante, etc;

iii) `SensorEventListener` - Interface que corresponde ao *listener* de eventos dos sensores. Esta permite definir dois *callbacks:* um onde são capturadas as alterações de precisão do sensor e outro que é disparado sempre que novos eventos são desencadeados. Os *callbacks* implementados são invocados pela plataforma *Android*;

iv) `SensorEvent` - Classe que representa um evento. A biblioteca faz a gestão destes eventos fazendo com que estes sejam enviados assincronamente após feito o registo do evento. Um evento é constituído pela seguinte informação: o sensor que originou o evento, uma marca temporal correspondente ao evento, a precisão do evento e os dados do sensor na forma de um vetor.

A título de exemplo é descrito de seguida o conjunto de passos que permite fazer a utilização dos elementos referidos anteriormente:

1) Adquire-se uma instância de `SensorManager` e através desta é pedido, por exemplo, o sensor de iluminação, assumindo que este está disponível;

2) Definem-se os *callbacks* que implementam a interface `SensorEventListener`;

3) Faz-se o registo dos eventos para o sensor de iluminação utilizando os *callbacks* definidos anteriormente.

2.4 Visão global das soluções existentes

Foi feito um estudo sobre quais as soluções que são utilizadas no mercado e que permitem usar sensores em ambiente de simulação. Das soluções existentes foram selecionadas três. A primeira é fornecida no pacote de desenvolvimento da plataforma *Android*, a segunda é disponibilizada pela *OpenIntents* [7] e por fim, a terceira, que é desenvolvida pela *Samsung* [8].

2.4.1 *Android Tools - Hardware Emulation*

O simulador que é disponibilizado com a distribuição da plataforma *Android* tem suporte experimental para sensores. O código do simulador foi alterado para que este possa estabelecer uma ligação a um dispositivo real que tenha instalado a aplicação *SdkControllerSensor* fornecida nas ferramentas da distribuição. Esta aplicação permite que o utilizador escolha quais os sensores que quer disponibilizar. A comunicação é realizada através de uma ligação *Universal Serial Bus* (USB) via interface de *socket*.

Relativamente aos sensores disponibilizados pela aplicação, existem limitações. A sua disponibilidade encontra-se definida no código. Esta decisão determina à partida o número máximo de sensores que podem ser suportados, tornado a solução pobre no que diz respeito à versatilidade.

O simulador injeta os dados recebidos dos sensores no sistema operativo para que estes estejam disponíveis através da biblioteca de sensores da plataforma *Android*. Esta funcionalidade apesar de ser experimental, apenas funciona em dispositivos móveis reais cujas versões da plataforma *Android* são superiores à 4.0.

2.4.2 *OpenIntents - Sensor Simulator*

Esta solução integra um conjunto de componentes produzidas pela *OpenIntents* que têm como objetivo simular dados de sensores e transmiti-los para o simulador. A solução é constituída pelas seguintes aplicações:

i) Aplicação para o computador (*desktop*);

ii) Aplicação para o simulador;

iii) Aplicação para um dispositivo móvel real.

A interação pode ser feita entre as três aplicações ou apenas entre a aplicação para o computador e a aplicação para o simulador. Por conseguinte, é possível fazer a simulação dos dados dos sensores de dois modos. Um primeiro modo, através da aplicação instalada no computador e outro através da aplicação para o dispositivo real. Na figura que se segue encontra-se a interface de simulação da aplicação instalada no computador.

Figura 2.4: Aplicação gráfica de simulação da solução *OpenIntents*

A aplicação instalada no computador é responsável por fazer a ligação ao simulador. A partir dela é possível selecionar quais os sensores que se pretende utilizar e para os mesmos pode ser feita a simulação através da utilização do rato. A simulação pode também ser feita através do dispositivo real caso este se ligue à aplicação instalada no computador através da rede sem fios. Os dados dos sensores transferidos podem ser gravados editados e reproduzidos mais tarde.

Esta solução suporta um número vasto mas ainda assim incompleto de sensores. A sua implementação é pouco rica, permitindo apenas saber a informação dos valores dos sensores, ou seja, a restante informação associada aos eventos não está disponível. A

implementação das restantes funcionalidades disponíveis na biblioteca da plataforma *Android* para além de receção dos valores dos sensores não está presente.

2.4.3 *Samsung Sensor Simulator*

A *Samsung* disponibiliza uma versão modificada do simulador da plataforma *Android* e oferece uma extensão (*plugin*) para o *Eclipse*. O simulador permite que os dados dos sensores possam ser alterados através da extensão, oferecendo assim um modelo visual de um dispositivo com os seus vários sensores onde é possível fazer a manipulação do dispositivo virtual ou das forças que agem sobre o mesmo, como pode ser observado na figura que se segue.

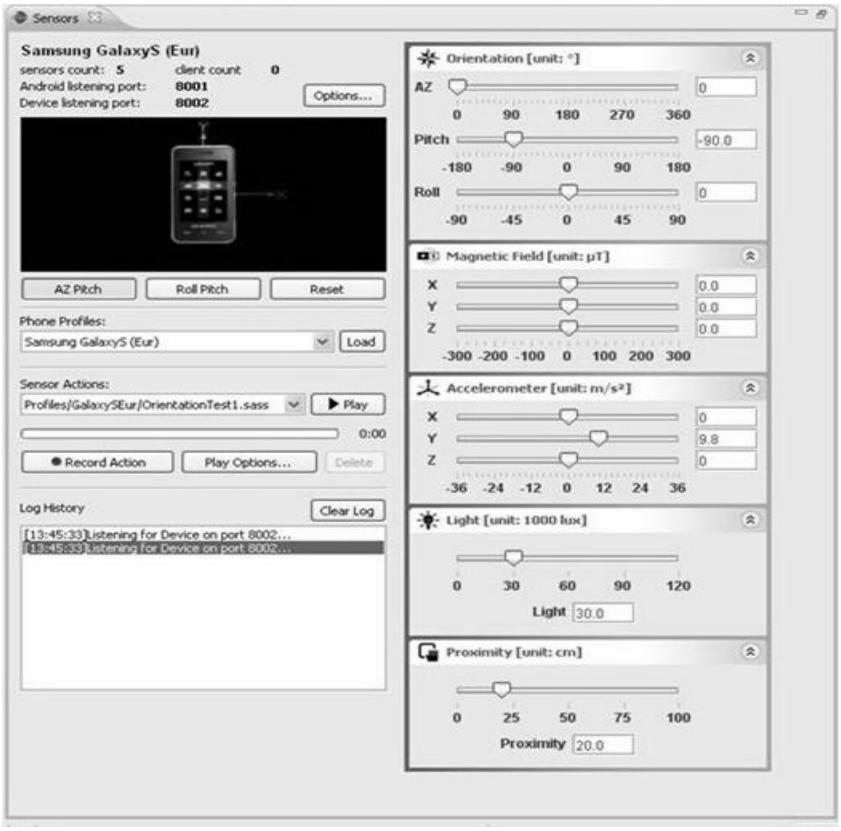

Figura 2.5: Extensão para o *Eclipse* concebida pela *Samsung* [8]

Esta extensão permite ao utilizador concretizar as seguintes tarefas:

i) Executar *scripts* que contêm alterações físicas ao dispositivo;

ii) Ligar um dispositivo real para gravar os dados dos sensores para um ficheiro;

iii) Reproduzir dados gravados através dispositivo real e executar *scripts* escritos pelo utilizador.

São suportados cinco sensores cujos valores podem ser gerados através da extensão para fazer a manipulação do dispositivo.

A comunicação entre o dispositivo real e o simulador é feita através da rede sem fios e cada um dos participantes tem de ter previamente instalada uma aplicação que permita a comunicação.

Esta solução caracteriza-se por ser incompleta porque apenas disponibiliza cinco sensores e não suporta simulação em tempo real.

2.5 Sumário

Este capítulo começou por apresentar a plataforma *Android*, salientando as suas características principais. De seguida foi apresentada a biblioteca de sensores da plataforma *Android*, onde foi dado ênfase quanto ao número de sensores disponíveis e às operações associadas aos mesmos. Foi introduzido o sistema de coordenadas que está associado ao dispositivo móvel e aos sensores, tendo sido explicada qual a disponibilidade existente relativamente aos sensores de um dispositivo.

Foram apresentadas as soluções estudadas, tendo sido destacadas as soluções disponibilizadas pela plataforma *Android* e pela *OpenIntents*. Ambas as soluções solucionam o problema da utilização de sensores no simulador, no entanto apresentam falhas no que diz respeito à generalização da solução. A solução da *Samsung* é uma solução integrada mas não permite que se possa fazer a simulação sensorial em tempo real.

Capítulo 3

Arquitetura da solução

Este capítulo descreve a arquitetura da solução proposta para a infraestrutura de simulação sensorial para a plataforma *Android*. O capítulo inicia-se com a apresentação da diretriz que influenciou o seu projeto, de seguida é descrita a solução, referenciando as propriedades da mesma. O capítulo termina com a decomposição funcional da biblioteca de simulação sensorial.

3.1 Diretriz de projeto

A solução da infraestrutura de simulação sensorial para a plataforma *Android* tem o objetivo de disponibilizar em ambiente de simulação a biblioteca de simulação sensorial usando os dados de sensores de um dispositivo real. A solução desenvolvida designa-se *Simple Sensor Simulator* (SSS) e deve ser passível de usar em qualquer simulador que execute a plataforma *Android*, estando concebida para suportar atualizações da interface de programação da plataforma *Android* e diferentes dispositivos reais.

A solução foi desenhada em prol das propriedades apresentadas na secção 1.1 Objetivos e contribuição.

Recordando:

i) Portabilidade - O código realizado usando a infraestrutura de simulação sensorial deve poder ser executado diretamente na plataforma *Android* sem ter de sofrer alterações significativas de semântica;

ii) Atualização - A infraestrutura deve suportar futuras atualizações da API de sensores da plataforma *Android*;

iii) Independência - A infraestrutura deve poder ser executada em qualquer versão da plataforma *Android*, desde que esta seja superior à versão 2.3 (*Gingerbread*);

iv) Completude - Todas as funções essenciais e sensores devem poder ser utilizados, desde que estejam disponíveis no dispositivo real;

v) Tolerância a falhas – Em caso de falta de rede a solução deve ser capaz de transmitir essa informação ao programador;

vi) Desempenho - Deve ser possível fazer o uso da infraestrutura em tempo real sem que seja observado impacto visual significativo.

Foi tida em conta outra característica na arquitetura do projeto que consiste em utilizar a comunicação através da rede sem fios como sendo um recurso crítico. O uso excessivo do mesmo leva a que o desempenho da infraestrutura seja prejudicado significativamente, como pode ser constado na seguinte alusão [9]:

"An interesting observation about network-based applications is that the best application performance is obtained by not using the network."

Esta afirmação, que aparenta ser contraditória, enfatiza que as aplicações devem fazer um uso controlado da rede, minimizando o número de interações ou o volume de dados a ser transferido.

Na conceção da infraestrutura este princípio foi tido em conta e a rede é apenas utilizada para realizar operações exclusivamente dedicadas ao uso da biblioteca de sensores. No decorrer deste capítulo este tema será desenvolvido adequadamente.

3.2 Arquitetura da infraestrutura

O SSS é uma infraestrutura composta por dois componentes de *software*: uma aplicação móvel de captura e uma biblioteca de simulação sensorial. A interação entre

ambas é feita através de mensagens trocadas através de uma rede sem fios utilizando o padrão cliente-servidor.

A aplicação móvel de captura executa num dispositivo real e é responsável por detetar quais os sensores disponíveis no dispositivo móvel. Os sensores detetados são disponibilizados de forma a poderem ser acedidos através da biblioteca de simulação sensorial.

A biblioteca de simulação sensorial é executada num simulador da plataforma *Android* e substitui a biblioteca nativa da plataforma *Android*, fornecendo todas as características e funcionalidades disponíveis na mesma, ou seja, esta biblioteca é uma cópia da biblioteca de sensores da plataforma *Android*, contudo, as suas chamadas são encapsuladas em chamadas invocadas remotamente na aplicação móvel de captura. A Figura 3.1 mostra a arquitetura lógica da solução, onde é possível observar os intervenientes referidos anteriormente e as mensagens trocadas entre os mesmos.

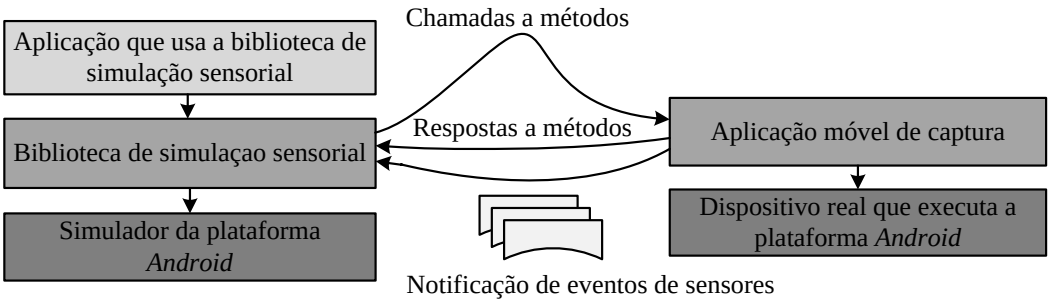

Figura 3.1: Arquitetura lógica da infraestrutura

3.2.1 Aplicação móvel de captura

A arquitetura da aplicação móvel de captura é baseada numa separação clara entre a interface com o utilizador, o serviço de captura e a comunicação e manipulação de tipos.

Esta aplicação é constituída por três camadas, estando uma delas dividida em duas partes, como se pode observar na Figura 3.2. As três camadas encontram-se assentes na camada Plataforma *Android*, que é meramente ilustrativa, uma vez que a mesma corresponde ao sistema operativo presente no dispositivo.

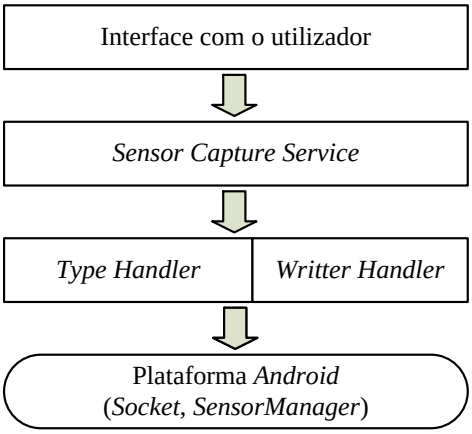

Figura 3.2: Arquitetura da aplicação móvel de captura

A organização em camadas foi adotada de forma a simplificar o desenvolvimento e a alteração de funcionalidades em caso de necessidade futura. A camada que eventualmente pode sofrer alterações caso a biblioteca de sensores seja ampliada é a camada de manipulação de tipos (*Type Handler*), caso seja inserido um novo tipo.

A camada de interface com o utilizador representa os detalhes ligados com interação com o utilizador do dispositivo. A sua função consiste em dar ao utilizador a informação relativa aos sensores presentes no seu dispositivo e permite disponibilizar o serviço de captura.

A função da camada de captura (*Sensor Capture Service*) consiste em receber e processar pedidos com operações oriundas da biblioteca de simulação sensorial e responder com o respetivo resultado.

A terceira camada é composta por duas partes distintas, a parte de manipulação de tipos e a parte de gestão de escritas (*Writter Handler*), que têm como propósito fazer adaptações de dados e retornar respostas, respetivamente.

Antes de descrever quais as técnicas utilizadas na implementação de cada camada, que consta no Capítulo 4, será feita uma descrição funcional desta aplicação realçando as seguintes funcionalidades prestadas:

i) *Endpoint* para troca de mensagens através de um protocolo bem definido;

ii) Exibição dos sensores disponíveis e controlo sobre o início e fim da captura;

iii) Invocação de métodos de origem remota, provenientes da biblioteca de simulação sensorial;

iv) Envio de respostas de forma sincronizada;

v) Conversão de tipos e criação instâncias.

3.2.1.1 Protocolo de mensagens

O protocolo de troca de mensagens entre a aplicação móvel de captura e a biblioteca de simulação sensorial funciona de duas formas:

i) Pedido-resposta: é feito um pedido e é aguardada a reposta;

ii) Sentido único: é feito um único envio sem haver resposta.

A estrutura das mensagens é a mesma independentemente da forma que seja feita a comunicação. A mensagem é composta por um identificador da operação a desencadear e pelos dados a operar. Desta forma é possível coexistirem diferentes operações, dado que cada uma pode ter associados diferentes tipos de dados de operação.

3.2.2 Biblioteca de simulação sensorial

A arquitetura da biblioteca de simulação sensorial é também baseada numa separação por camadas, onde é feita a separação entre a comunicação com a aplicação móvel de captura e a implementação da biblioteca de simulação sensorial. Esta biblioteca é constituída por duas camadas, como retrata a Figura 3.3.

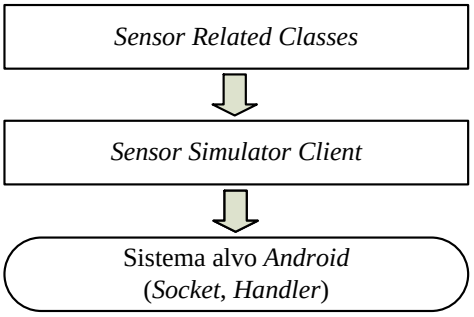

Figura 3.3: Arquitetura da biblioteca de simulação sensorial

A organização em camadas foi adotada de forma a isolar dependências e para permitir que as alterações possam ser feitas isoladamente. A camada sujeita a modificações é a *Sensor Related Classes*, aquando do crescimento da biblioteca de simulação sensorial.

A biblioteca é responsável por comunicar com o dispositivo real e traduz as chamadas à biblioteca, em pedidos a serem executados remotamente no dispositivo real.

A camada *Sensor Related Classes* inclui as seguintes funções:

i) Fornecer uma interface de programação de sensores com a mesma semântica que a da plataforma *Android*;

ii) Gerir sensores e respetivos *listeners* de eventos;

iii) Gerir falhas de rede.

A camada *Sensor Simulator Client* representa o cliente da aplicação móvel e presta as seguintes funcionalidades:

i) Invocação de métodos remotamente;

ii) Receção de eventos dos sensores;

iii) *Endpoint* para troca de mensagens através de um protocolo bem definido.

3.3 Sumário

Neste capítulo foram apresentados os objetivos da solução proposta e foi descrita de forma global a interação entre os constituintes da solução.

Resumindo, a solução é constituída pela aplicação móvel de captura, onde são recebidos os pedidos provenientes da biblioteca de simulação sensorial que executa no simulador. Estes pedidos são processados no dispositivo real e o seu resultado é enviado para a biblioteca de simulação sensorial.

A questão de desempenho associada aos recursos de rede é tratada usando um protocolo de comunicação constituído por mensagens que são utilizadas única e exclusivamente em operações relacionadas com a biblioteca de simulação sensorial.

O capítulo termina com a descrição da arquitetura da biblioteca de simulação sensorial, onde se destacam as características da mesma.

Capítulo 4

Aspetos de implementação

Este capítulo descreve os aspetos mais relevantes da implementação. Começa-se por fazer uma breve análise sobre os componentes existentes na plataforma *Android*, de seguida é detalhada a implementação da aplicação móvel de captura, destacando os aspetos de comunicação, gestão de tipos, sincronização de escritas e notificação de eventos. Por último, são apresentados os detalhes da biblioteca de simulação sensorial salientando o atendimento de chamadas remotas, os mecanismos de sincronização e a gestão de eventos.

4.1 Componentes da plataforma *Android*

A plataforma *Android* disponibiliza um conjunto de componentes para a construção de aplicações. Estes componentes são de elevada importância tendo em conta que cada um deles desempenha um papel específico e têm um ciclo de vida próprio, assim como uma interação distinta. Alguns componentes são pontos de partida que podem ser iniciados pelo utilizador, outros, só podem ser desencadeados através de outros componentes. Existem quatro componentes diferentes, cada um com características e responsabilidades distintas. De seguida é feito um resumo dos mesmos:

i) `Activity` (atividade) - Uma atividade representa um ecrã com uma interface de utilização. Uma aplicação pode ser constituída por uma ou mais atividades. Uma atividade pode ser iniciada e utilizada por outra aplicação;

ii) `Service` (serviço) - É um componente que não tem interface de utilizador e que executa em segundo plano (*background*) com o objetivo de realizar operações demoradas. Os serviços podem ser de dois tipos: o *Started*, que após ser iniciado permanece a executar indefinidamente, e o *Bounded*, que após ser iniciado disponibiliza uma interação cliente-servidor;

iii) `Content Provider` - Este componente faz a gestão de um conjunto de dados. Os dados podem ser colocados no sistema de ficheiros, numa base de dados *SQLite* ou na *World Wide Web* (WEB). Um `Content Provider` pode ser partilhado entre aplicações, para que estas possam interrogar e modificar dados;

iv) `Broadcast Receiver` - É um componente que responde a um *broadcast* enviado pelo sistema ou pelo utilizador. Este componente não tem interface de utilizador. Ele pode ser definido, por exemplo, para ser executado quando o nível de bateria está próximo do fim, ou quando a ligação de rede sem fios foi perdida.

4.1.1 Detalhes da plataforma que influenciaram a implementação

A plataforma *Android* possibilita que um componente de uma aplicação seja iniciado a partir de outra aplicação. Esta característica possibilita a reutilização de comportamento, e foi considerada como aspeto fundamental no decorrer da implementação.

Quando um componente é iniciado, este é iniciado num novo processo, caso o processo associado à aplicação em causa não se encontre em memória. Como cada aplicação é executada num processo distinto, cujas permissões restringem o acesso a outras aplicações, a ativação de outro componente tem de ser feita com um pedido que expresse essa necessidade, através de uma mensagem que manifeste essa ação. A este tipo de mensagem dá-se o nome de `Intent`.

A classe `Intent` representa mensagens assíncronas que permitem solicitar funcionalidades de outras componentes da plataforma *Android*, entre atividades, ou entre uma atividade e um serviço. Pode ser usado para lançar uma nova atividade a partir da atividade atual, ou associar-se a um serviço a correr em segundo plano.

Quando uma aplicação é iniciada, da plataforma *Android* cria um processo *Linux* para essa aplicação com apenas um fio de execução. Por omissão, todos os componentes dessa aplicação são executados no fio de execução desse processo. Este fio de execução é denominado por fio de execução principal (*main thread*). A plataforma *Android* oferece mecanismos para que seja possível fazer execução de código noutros fios de execução, que não o fio de execução principal.

Relativamente ao ciclo de vida de um processo pode dizer-se que o sistema tenta manter esse processo em memória o máximo de tempo possível, mas eventualmente pode haver a necessidade de remover processos antigos, de forma a obter memória para um processo novo ou para um processo mais prioritário. De forma a determinar quais os processos que permanecem no sistema existe uma hierarquia baseada no estado dos componentes que estão a ser executados e que fazem parte do processo. Os processos com menor importância são os primeiros a serem eliminados. Os níveis de hierarquia e a sua importância podem ser aprofundados na documentação [10]. Os componentes referidos na secção 4.1 Componentes da plataforma *Android*, têm também um ciclo de vida próprio e distinto.

4.2 Aplicação móvel de captura

Esta aplicação, como já foi referido anteriormente, tem como função disponibilizar os sensores e as suas operações à biblioteca de simulação sensorial. Para tal, a base da implementação está centrada em torno do mecanismo de receção de invocações de métodos de origem remota, na gestão do envio das respostas dos mesmos e no envio de dados dos sensores.

Esta aplicação é constituída por dois componentes da plataforma *Android*: um serviço denominado `SensorCaptureService` e uma atividade denominada `SensorCaptureActivity`.

A atividade é responsável pela interface com o utilizador. Esta interface não é de especial interesse em termos estéticos, por isso o seu aspeto é simples e prático.

Na Figura 4.1 encontra-se ilustrada a interface da aplicação móvel de captura, onde é possível observar quais os sensores disponíveis no dispositivo, bem como iniciar e parar o serviço de captura através do botão circular.

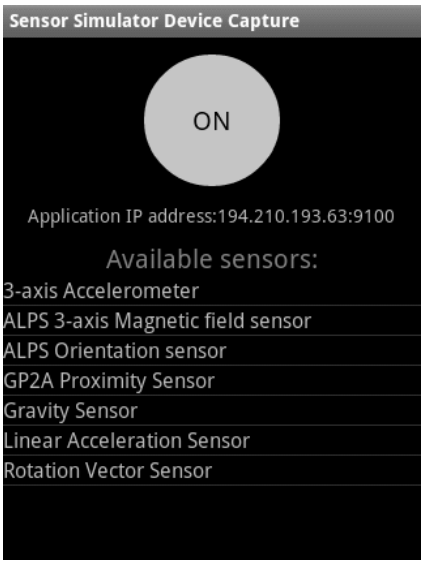

Figura 4.1: Interface de utilização da aplicação móvel de captura

A atividade contém informação relativa ao endereço de rede do dispositivo (explicado mais à frente).

O serviço `SensorCaptureService` é o componente central desta aplicação. A partir dele são desencadeados outros dois componentes que fazem a gestão da comunicação. O serviço é lançado quando é iniciado o processo de captura de sensores. Este serviço tem três responsabilidades fundamentais:

i) Aguardar pedidos (`SocketReader`);

ii) Retornar respostas (`SocketWritterHandler`);

iii) Reportar eventos dos sensores (`SensorCaptureService`).

Na Figura 4.2 encontra-se o diagrama *Unified Modeling Language* (UML) com as classes responsáveis pelas três funções indicadas anteriormente.

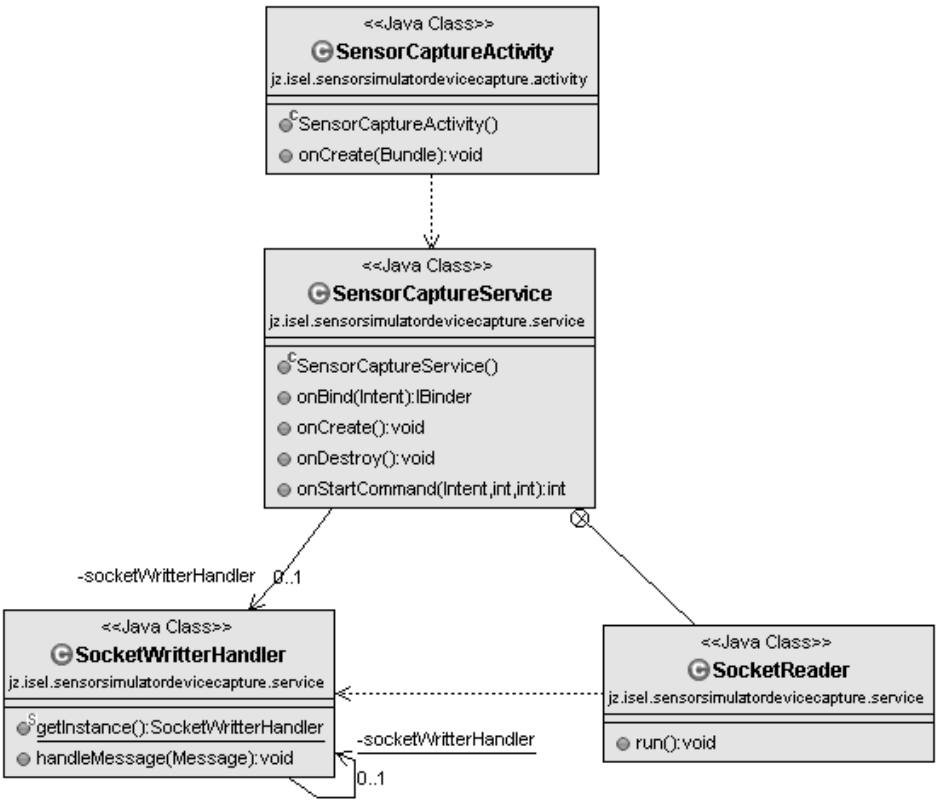

Figura 4.2: Diagrama UML da estrutura base da aplicação móvel

A classe `SocketReader` é responsável por aguardar pedidos. Esta classe resulta na implementação da interface `Runnable`. As classes que implementam esta interface são tipicamente utilizadas com a finalidade de serem executadas no contexto de um fio de execução. Esta classe é executada em fio de execução alternativo logo que o serviço é iniciado, decorrendo por tempo indefinido. A sua função está relacionada com o atendimento de chamadas de métodos remotos. Os detalhes desta funcionalidade encontram-se descritos em pormenor na secção 4.2.2 Tratamento de chamadas remotas. O resultado de uma chamada remota é retornado à biblioteca de simulação sensorial através da classe `SocketWritterHandler`.

A classe `SocketWritterHandler` é iniciada pela classe `SocketReader`. Esta classe é de igual modo executada em fio de execução alternativo e a sua função consiste em aguardar por dados para serem enviados. Os detalhes deste componente encontram-se explicados na secção 4.2.1.1 Sincronização de escritas.

Por fim, é feito o reporte de eventos dos sensores (no contexto do `SensorCaptureService`), que se encontra aprofundado na secção 4.2.4 Transmissão de eventos de sensores.

4.2.1 Mecanismo de comunicação

Como referido na secção 3.2.1.1 Protocolo de mensagens, os dados a transferir são caracterizados por um identificador da operação e pelos dados dessa operação. Desta forma, optou-se por armazenar esta informação numa classe parametrizável com o tipo dos dados a transferir. Na Figura 4.3 é possível observar o diagrama UML da classe `OperationData`.

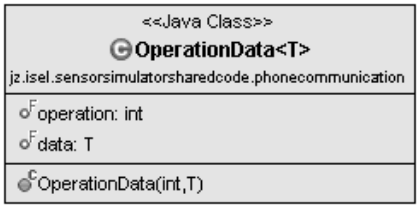

Figura 4.3: Diagrama UML da classe que representa uma operação

Para satisfazer as necessidades de comunicação da infraestrutura foi utilizado como mecanismo de comunicação, o mecanismo de *sockets* do *java*. Esta decisão foi tomada dado que existe facilidade de se poder ler e escrever objetos através de um `OutputStream`. Desta forma utilizaram-se as classes `ObjectOutputStream` e `ObjectInputStream`, que encapsulam um `OutputStream`, para fazer a escrita e leitura de objetos, respetivamente. Toda a comunicação da infraestrutura é suportada através de um único *socket*, por questões de desempenho.

4.2.1.1 Sincronização de escritas

Na solução existem duas fontes de produção de dados, mas poderia haver mais. Desta forma existe a possibilidade das escritas concorrerem entre si, visto que apenas existe um canal para escrita. Para isso, foi desenhada uma solução que suporta várias fontes de produção de dados tirando partido de um elemento da plataforma *Android*, denominado `Handler`. O `Handler` [11] permite que lhe sejam enviadas mensagens que são processadas mais tarde pelo fio de execução que lhe está associado. As mensagens entregues são colocadas na fila de mensagens associadas ao fio de execução. O `Handler` implementado (`SocketWritterHandler`) tem associado um fio de execução dedicado para o envio de dados. Desta forma garante-se que as escritas são sempre feitas única e exclusivamente por um fio de execução, mantendo a ordem de chegada à fila.

Este componente é partilhado por um conjunto distinto de componentes. Esta partilha está associada à natureza deste componente, ou seja, o envio de informação. O componente foi implementado através da utilização do padrão de desenho *singleton* [12]. Neste caso em particular, foi discutida a hipótese de se usar um serviço em vez do padrão *singleton*. No entanto, dado que o `Handler` depende do `ObjectOutputStream` no momento da sua instanciação, optou-se por usar o padrão *singleton* pois só este permite satisfazer esta dependência no momento da sua instanciação.

4.2.2 Tratamento de chamadas remotas

A classe `SocketReader` aguarda por pedidos provenientes da biblioteca de simulação sensorial. Estes pedidos são sempre caracterizados por serem invocações a métodos da biblioteca de sensores da plataforma *Android*. O pedido contém a informação do método, denominada `MethodCallData`, armazenada no campo `data` do objeto `OperationData`. Esta estrutura, ilustrada na Figura 4.4 contém os argumentos, os tipos dos argumentos, o tipo de retorno, o tipo de retorno esperado, a classe associada ao método e o nome do método.

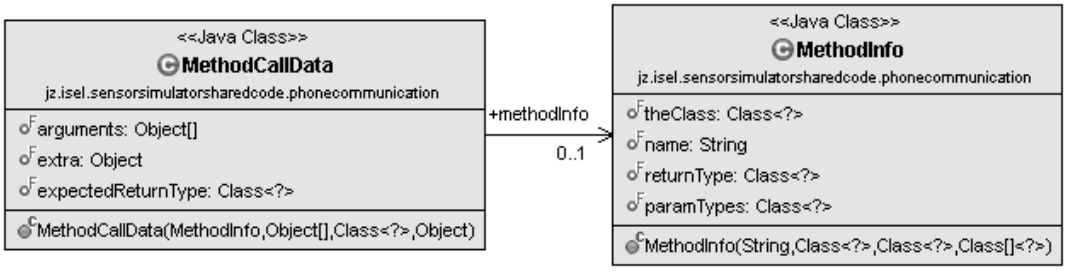

Figura 4.4: Diagrama UML das classes associadas a invocações remotas

Com esta informação é possível, através de reflexão [13], adquirir o método que se pretende invocar. Após invocado o método, o seu retorno é enviado para a biblioteca de simulação sensorial. No caso da ocorrência de exceção, a mesma é capturada e encapsulada numa exceção específica da aplicação relatando a causa da mesma. A exceção é enviada para a biblioteca de simulação sensorial.

4.2.3 Conversão de tipos e criação de instâncias

Os métodos invocados através de reflexão pertencem à biblioteca de sensores da plataforma *Android*. Para fazer a invocação através de reflexão é necessário providenciar os parâmetros relativos à chamada bem como a instância respetiva. Os parâmetros são fornecidos pela entidade que faz a invocação remota. Os parâmetros que não são seriáveis (classe `Sensor`, por exemplo), não podem ser transferidos. Desta forma, estes parâmetros têm de ser copiados e encapsulados numa estrutura que possa ser seriada. Este facto leva a que exista uma diferença entre os parâmetros recebidos e os parâmetros necessários para fazer a invocação. Posto isto, é necessário que haja uma conversão dos parâmetros em causa e que seja fornecida a instância que contém o método a ser invocado. Este problema acontece de igual modo com o tipo de retorno.

A solução pensada para resolver este problema opta por utilizar um mecanismo de conversão que em tempo de execução converte o tipo recebido no tipo adequado. A solução implica a existência de um conjunto de conversores onde cada conversor, individualmente, sabe converter um único tipo. Este paradigma resulta da aplicação do padrão *Composite* [14] tal como pode ser observado na estrutura de classes presente na Figura 4.5.

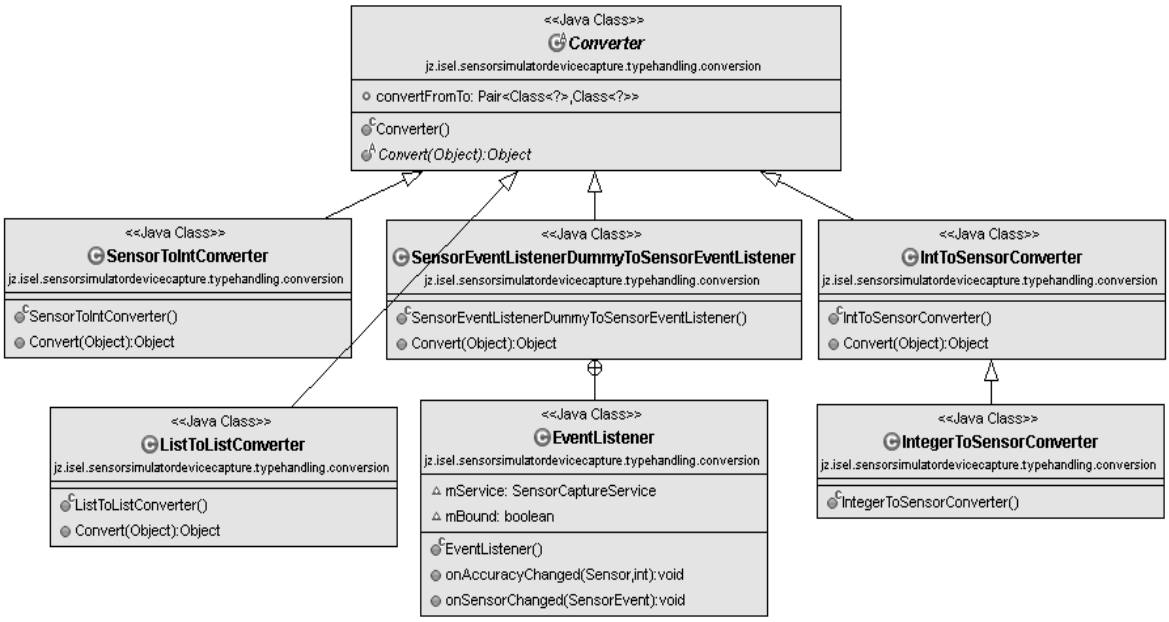

Figura 4.5: Diagrama UML das classes conversoras de tipos

Cada conversor tem de derivar a classe `Converter`, o que faz com que cada conversor seja autossuficiente no que diz respeito à conversão. Esta classe contém

informação relativa à origem e destino do tipo a ser convertido. Na implementação do método `Convert` deve ser feita a conversão para o tipo em causa. Desta forma podem existir tantos conversores quantos forem necessários.

No que diz respeito à criação de instâncias, foi desenvolvida uma solução semelhante à dos conversores. Existe a classe base `GetInstance` de onde as classes que pretendem disponibilizar instâncias devem derivar. Cada classe contém um campo com a classe que é capaz de criar. No contexto deste projeto apenas existem duas classes que sabem fazer a criação de instâncias, uma vez que são as únicas classes da biblioteca de sensores que disponibilizam métodos que podem ser invocados pelo utilizador. Estas classes são a `SensorManager` e a `Sensor` como pode ser observado na Figura 4.6.

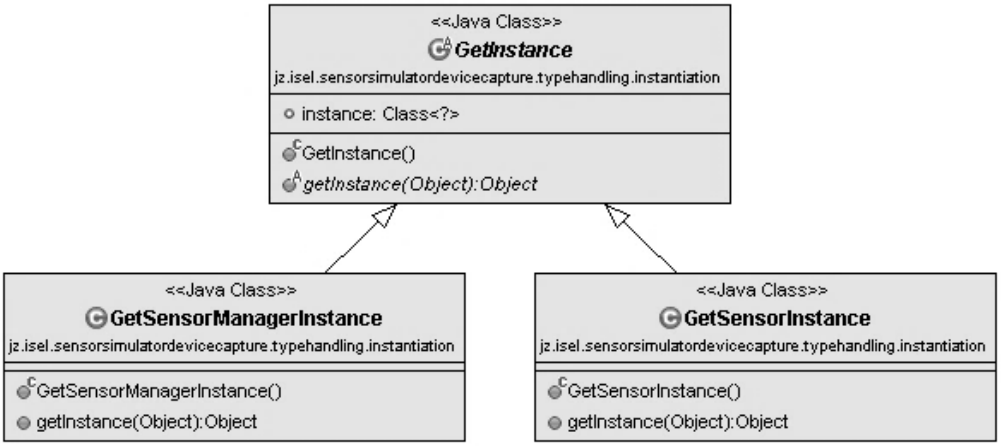

Figura 4.6: Diagrama UML das classes capazes de criar instâncias

Por si só, os conversores e criadores de instância não têm qualquer utilidade no contexto do problema referido. Surge a necessidade de existir uma entidade que faça o seu carregamento, diga-se dinâmico, e a partir dela são disponibilizados métodos para fazer a conversão e criação de instâncias em tempo de execução. Esta entidade tem o nome de `TypeHandler` e faz a gestão e o carregamento dinâmico de todos os conversores e criadores de instância existentes.

4.2.3.1 Carregamento dinâmico de conversores e criadores de instâncias

Uma aplicação *Android* depois de sofrer o processo de compilação, ver Figura 4.7, é armazenada na forma de um *Android Application Package* (APK). Este ficheiro contém a informação necessária ao processo de instalação no dispositivo. Esta

informação pode ser caracterizada, em traços largos por ser constituída pelo executável *Dalvik* (formato .dex), recursos pré-compilados (binário de um XML por exemplo), recursos não compilados e o ficheiro de manifesto.

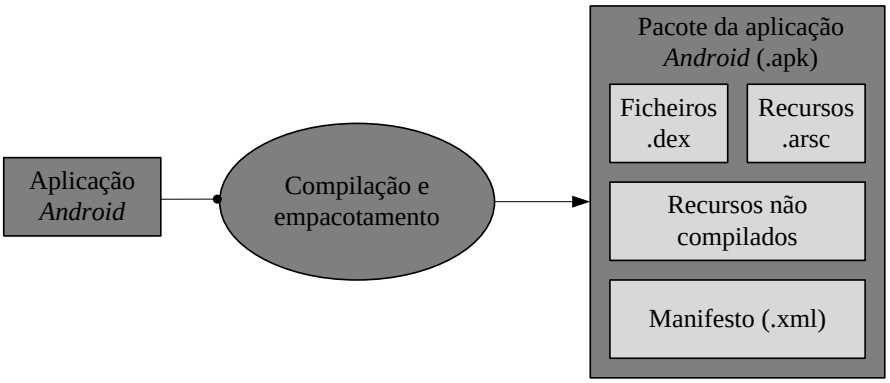

Figura 4.7: Compilação e empacotamento de uma aplicação *Android* [15]

Quando um APK é instalado [16] [15] [17], a única forma de aceder às classes que o constituem é através da leitura do ficheiro .dex.

Para fazer o carregamento dinâmico de todos os conversores e criadores de instância é, em primeiro lugar, necessário saber quais as classes que derivam da classe `Converter` e `GetInstance`. A plataforma *Android* fornece a classe `DexFile` [18] que tem como função manipular ficheiros .dex, bastando para isso localizar o mesmo. Através desta classe é possível saber todas as classes que constituem um ficheiro *Dalvik* e com essa informação é possível, através de reflexão, determinar quais as classes que derivam das classes `Converter` e `GetInstance`. Depois de encontradas as classes em causa, estas são instanciadas e armazenadas num contentor associativo distinto. No caso dos conversores a chave do contentor é um par de classes contendo o tipo que se quer converter e o tipo para qual vai ser feita a conversão. No caso dos criadores de instância a chave do contentor é o tipo da instância. Com estas duas estruturas de dados é possível fornecer métodos de conversão e criação de instâncias que sabem qual o conversor e criador adequado em tempo de execução. A entidade responsável pelo carregamento dinâmico dos conversores e criadores de instância chama-se `TypeHandler` e pode ser observada na Figura 4.8.

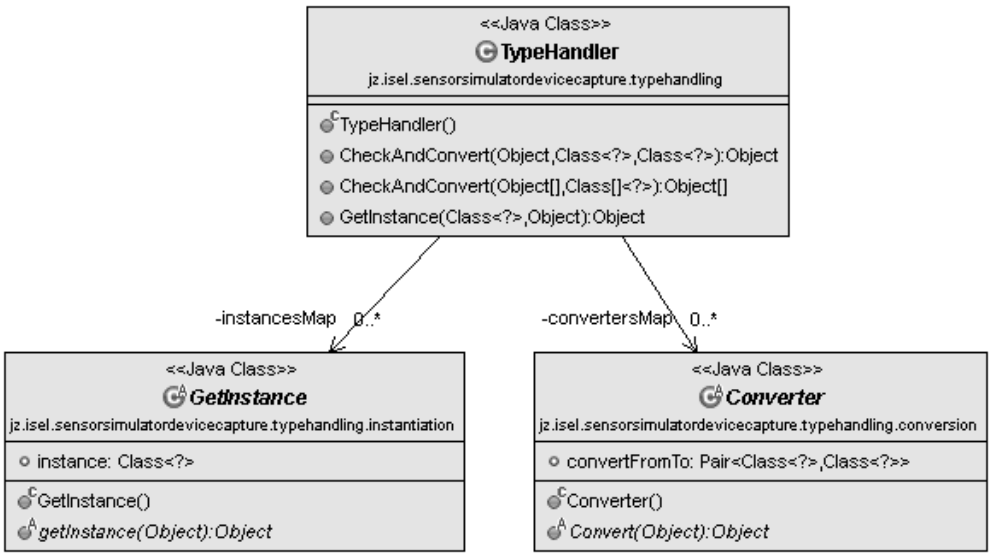

Figura 4.8: Diagrama UML das classes que suportam o carregamento dinâmico

4.2.4 Transmissão de eventos de sensores

Quando é invocado o método `registerListener`, da classe `SensorManager`, que tem como função fazer o registo de eventos para um dado sensor, é necessário passar como parâmetro um *listener* que é invocado sempre que os dados do sensor são atualizados. Tirou-se partido da solução (de conversão) referida anteriormente na secção 4.2.3 Conversão de tipos e criação de instâncias, para que o resultado da conversão de um *listener* seja sempre o mesmo *listener*. Desta forma sempre que algum evento de qualquer sensor registado é gerado, é sempre invocado o código do mesmo *listener*. O *listener* implementado faz o envio do evento em causa através da classe `SocketWritterHandler`.

4.3 Biblioteca de simulação sensorial

Como já foi dito anteriormente, a biblioteca de simulação sensorial substitui a biblioteca de sensores da plataforma *Android* disponibilizando uma interface de programação idêntica. A biblioteca de simulação possibilita que a utilização de sensores e suas operações associadas sejam feitas através da utilização de sensores de um dispositivo real.

A utilização da biblioteca de simulação resume-me essencialmente às operações que estão presentes na classe `SensorManager` e na classe `Sensor`. Ambas as classes têm a necessidade de fazer invocações remotas à aplicação móvel.

O código destas classes é semelhante ao código da biblioteca de sensores da plataforma *Android* de forma a possibilitar uma utilização idêntica.

4.3.1 Invocação de chamadas remotas

As invocações remotas são feitas através de um componente (`SensorSimulatorDeviceCaptureClient`) que comunica com a aplicação móvel de captura, permitindo que ambos os extremos troquem mensagens. Esta funcionalidade é partilhada pelas classes `SensorManager` e `Sensor`.

Dado que a biblioteca de simulação é uma biblioteca (*Library*), não é possível ter acesso a determinados componentes de uma aplicação *Android*, nomeadamente ao contexto (`Context`). O contexto é fundamental para fazer o vínculo a um serviço do tipo *Bound*, que neste caso seria a componente indicada a implementar esta funcionalidade. Posto isto, optou-se por implementar a classe `SensorSimulatorDeviceCaptureClient`, que representa um cliente da aplicação móvel, que é capaz de desencadear chamadas remotas. Esta classe é caracterizada por ser uma instância *singleton* porque tem de ser partilhada pela classe `SensorManager` e a classe `Sensor`. Esta prática é bastante comum no desenvolvimento de componentes da plataforma *Android* no que diz respeito à partilha de componentes que não podem ser partilhados por via de um serviço, por exemplo.

Na figura que se segue pode ser observada a estrutura de classes usada para responder a esta necessidade.

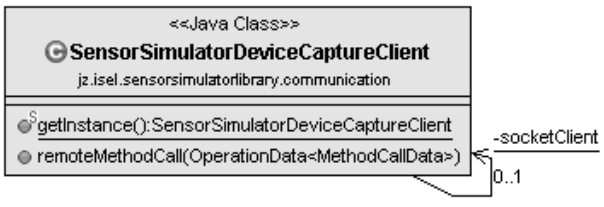

Figura 4.9: Diagrama UML da classe cliente da aplicação móvel

O componente, cliente da aplicação móvel, encontra-se dividido em duas partes:

i) Invocação remota de métodos (`SensorSimulatorDeviceCaptureClient`);

ii) Leitura de dados (`SocketReaderThread`).

A invocação remota de métodos tem de ser executada no fio de execução principal de forma a respeitar o comportamento da biblioteca de sensores da plataforma *Android*. Esta invocação é caracterizada pelo envio do objeto `OperationData` tipificado com os dados relativos à chamada remota, `MethodCallData`, para a aplicação móvel. Após feito o envio dos dados relativos à chamada, o fio de execução bloqueia-se até que que lhe seja entregue a reposta respetiva.

A leitura dos dados é feita em fio de execução alternativo. Quando existem novos dados recebidos, estes são analisados de forma a determinar qual o seu tipo. Caso sejam a reposta a uma invocação remota os dados são entregues ao fio de execução bloqueado de forma a serem consumidos. Caso sejam eventos, os mesmos são processados e encaminhados de forma a serem executados pelo *listener* respetivo. Os detalhes da gestão de eventos estão explicados na secção 4.3.2 Gestão e encaminhamento de eventos.

Na classe `SensorManager` existe um conjunto de métodos estáticos que são responsáveis por fazer cálculos auxiliares relacionados com a orientação, inclinação, altitude, etc. Estes métodos são fornecidos no código da classe `SensorManager` da plataforma *Android* sendo possível desta forma a sua execução diretamente no simulador, mas por razões de coerência são invocados também remotamente.

4.3.2 Gestão e encaminhamento de eventos

Quando é requerido o registo de eventos, a partir do método `registerListener`, para um determinado sensor, através do uso da biblioteca de simulação, via classe `SensorManager`, é necessário executar as seguintes ações (no contexto da implementação do método):

i) Guardar o *listener* associado ao registo do evento;

ii) Guardar o `Looper` associado à operação de registo;

iii) Guardar o sensor associando-o ao seu *listener*.

O `Looper` é uma classe que processa infinitamente uma fila de mensagens. Está normalmente associado a um fio de execução. O fio de execução principal tem um `Looper` associado por omissão. Todos os restantes fios de execução não estão associados a um `Looper` mas é possível fazer essa associação. A fila de mensagens processa mensagens (`Message`) que são enviadas através de um `Handler`. É relevante fazer a salvaguarda do `Looper` associado ao fio de execução principal de forma a poder usá-lo para invocar código.

A estrutura de dados que armazena toda a informação necessária para fazer esta gestão resulta de um contentor associativo cuja chave é o *listener* de evento e o valor é o tipo `SensorEventQueue` que pode ser observado na Figura 4.10. Este tipo guarda para cada par, *listener*/`Looper` todos os sensores que lhe estão associados.

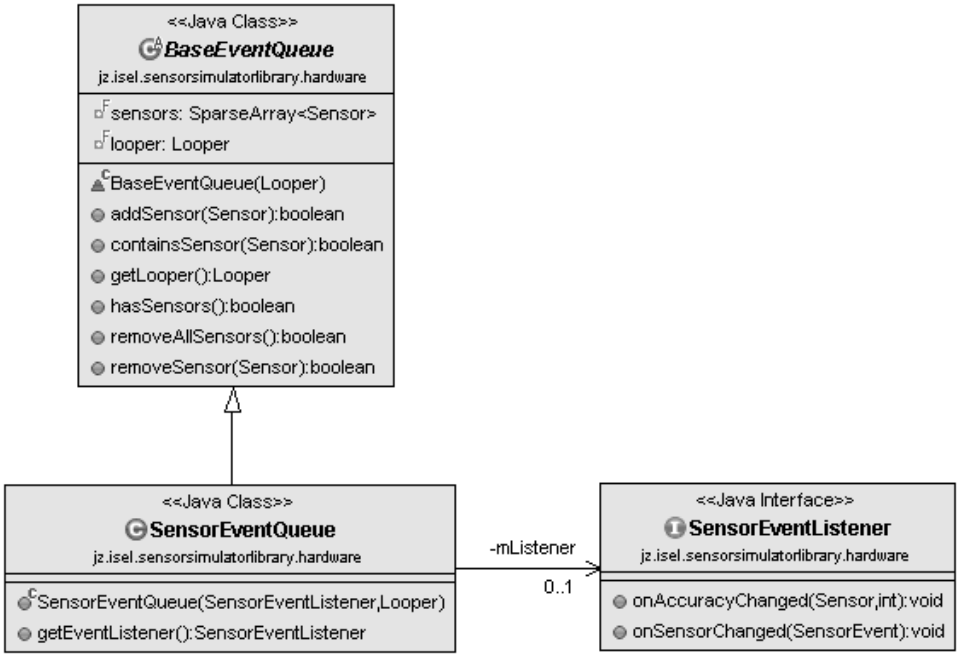

Figura 4.10: Diagrama UML das classes que estruturam os eventos

O encaminhamento dos eventos é feito no contexto da componente cliente da aplicação móvel. Quando este faz a leitura de dados que dizem respeito a eventos é desencadeado o processo de encaminhamento. Este processo inicia-se recolhendo do conteúdo do evento qual o sensor que o desencadeou, de seguida é procurado no contentor quais os `SensorEventQueue` que têm associado o sensor em causa. Através da utilização de um `Handler`, cujo `Looper` foi retirado do `SensorEventQueue` adequado, é injetada uma mensagem fazendo a invocação do método de sinalização (`onSensorChanged` ou `onAccuracyChanged`). A

informação do `Looper` é crucial porque é a única forma de garantir que a invocação acontece no fio de execução à qual o *listener* está associado.

No caso onde o método `registerListener` é invocado com um `Handler` como parâmetro, o `Looper` a ser usado é retirado do `Handler`, garantindo assim que a execução é feita no fio de execução associado ao `Handler`.

De forma a tornar a compreensão desta interação mais clara, na Figura 4.11, é ilustrado o diagrama de sequência de todas as atividades que constituem o processo de gestão e encaminhamento de eventos. É de notar que esta interação é feita entre a biblioteca de simulação sensorial e a aplicação móvel de captura.

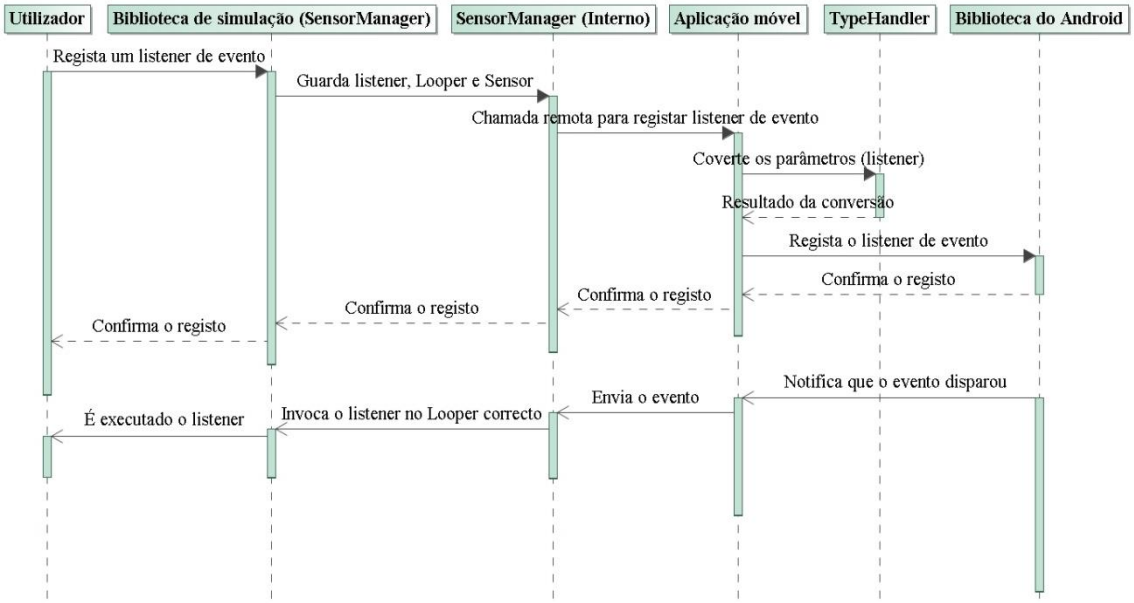

Figura 4.11: Diagrama de sequência da gestão e encaminhamento de eventos

4.4 Considerações sobre a implementação

Um dos objetivos deste projeto caracteriza-se por realizar um protótipo que traga funcionalidade e eficiência ao programador enriquecendo a produção de aplicações. No entanto, existem lacunas no que diz respeito à sua implementação como um todo, pois o objetivo centrou-se em resolver os problemas essenciais não tendo sido feita uma implementação completa de toda a biblioteca de sensores. As funcionalidades que não foram implementadas apenas dizem respeito à classe `SensorManager`. Desta forma enunciam-se as funcionalidades deixadas por implementar e as respetivas razões.

i) Todos os métodos obsoletos não foram implementados - No contexto do projeto realizado não faz sentido implementar estes métodos, pois o seu modo de utilização e implementação é pouco eficiente e o seu uso é desencorajado pela documentação;

ii) Qualquer funcionalidade relacionada com `Triggers` não foi implementada - O `TriggerEventListener` é um *listener* semelhante ao `SensorEventListener` com uma única diferença, é apenas disparado uma única vez. Isto é útil no caso do programador apenas estar interessado numa única notificação. No entanto, existe a possibilidade de voltar a registar esse evento. Em termos de funcionalidade, é possível atingir as mesmas características usando o `SensorEventListener`;

iii) O método *flush* não foi implementado - Este método tem uma utilidade pouco interessante no desenvolvimento de aplicações relacionadas com sensores.

4.4.1 O problema cliente-servidor

Como já foi referido anteriormente, a comunicação entre as duas componentes da infraestrutura, é feita através do mecanismo de *sockets*, respeitando a arquitetura cliente-servidor. O cliente para fazer a ligação necessita de saber qual o endereço de rede e o porto do servidor. O mais adequado, no contexto deste projeto, consistiria em implementar o cliente na aplicação móvel de captura e antes de ser iniciado o processo de captura, deveria ser inserido o endereço de rede e porto da biblioteca de simulação sensorial.

O simulador é executado na máquina hospedeira como uma aplicação comum. A comunicação entre estes é feita através do mecanismo de *sockets* por via da interface de *loopback* (endereço de rede 127.0.0.1) num porto definido para o efeito, normalmente a partir de 5554.

Assumindo que o servidor está implementado na biblioteca de simulação, é preciso que haja reencaminhamento de portos para a comunicação ser feita desde o dispositivo móvel até ao simulador. Para tal devem ser feitos dois reencaminhamentos: um que reencaminhe o tráfego do porto do servidor do simulador para um porto da máquina

hospedeira; e outro que reencaminhe o tráfego do porto que dá acesso ao exterior na máquina hospedeira para o endereço e porto da interface de *loopback*. A título de exemplo a Figura 4.12 ilustra o processo referido anteriormente.

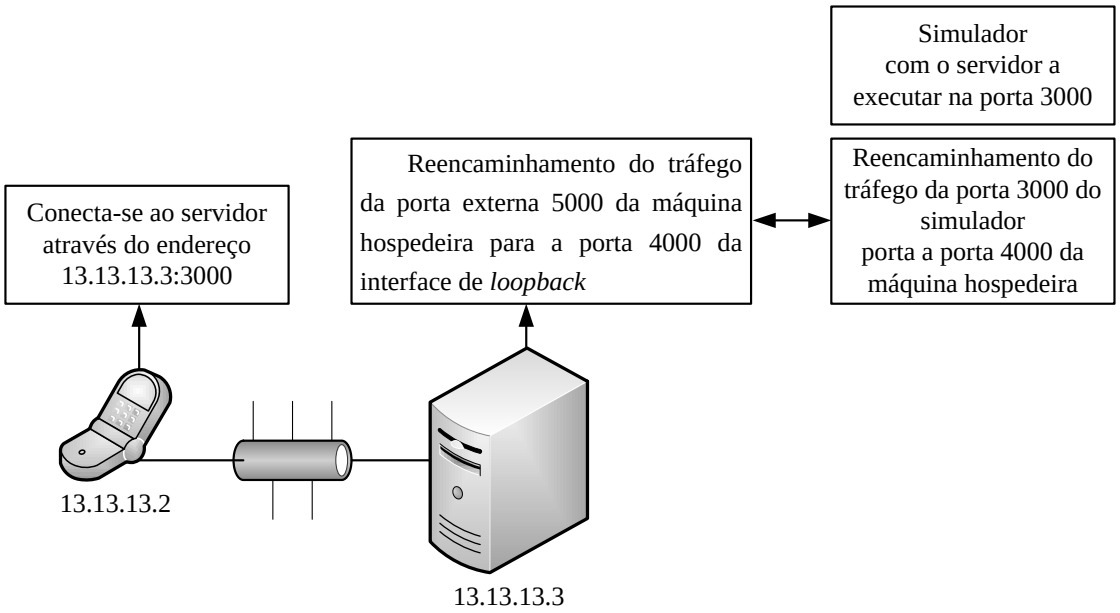

Figura 4.12: Reencaminhamento de portos subjacentes ao cliente-servidor

Dado que esta solução, apesar de ser a melhor, requer manipulação da rede, em dois componentes diferentes (simulador e máquina hospedeira), decidiu-se optar por uma solução mais prática, mas menos eficaz e menos realista. A solução adotada passa por inverter os papéis e colocar a aplicação móvel de captura como servidor e a biblioteca como cliente. Esta solução tem a desvantagem de que sempre que o endereço de rede da aplicação móvel sofre alteração, a biblioteca de simulação tem de ser recompilada com o novo endereço de rede. Isto acontece porque no desenvolvimento de bibliotecas na plataforma *Android* não existe forma de ter acesso a um ficheiro de configuração para, neste caso, facilitar a alteração do endereço de rede. Do ponto de vista de utilização em ambiente académico, esta inversão de papéis não é significativa.

4.5 Suporte de alterações

Com a velocidade exponencial com que o *software* sofre alterações é necessário ter em conta que esta infraestrutura irá sofrer alterações. As alterações, à partida, estão relacionadas com a biblioteca de sensores. Considerando que as alterações podem ser de três tipos, constata-se as seguintes alterações à biblioteca de simulação sensorial:

i) Adição de métodos;

ii) Adição de novos tipos de retorno/parâmetro;

iii) Adição de novos sensores.

A adição de métodos, no caso em que não acrescenta novos tipos, resume-se a alterar a classe subjacente, que neste caso pode ser a classe `Sensor` ou a classe `SensorManager`.

No caso de serem adicionados novos tipos de retorno/parâmetro, devem ser feitas alterações nas classes `Sensor` ou `SensorManager` e devem ser adicionados conversores que suportem os novos tipos.

A adição de novos sensores influencia as classes `SensorManager` e `Sensor`, dado que estes têm de ser acrescentados.

4.6 Organização da implementação

A solução encontra-se materializada na forma de um projeto *Eclipse*. O projeto encontra-se dividido em quatro componentes, como pode ser observado na Figura 4.13.

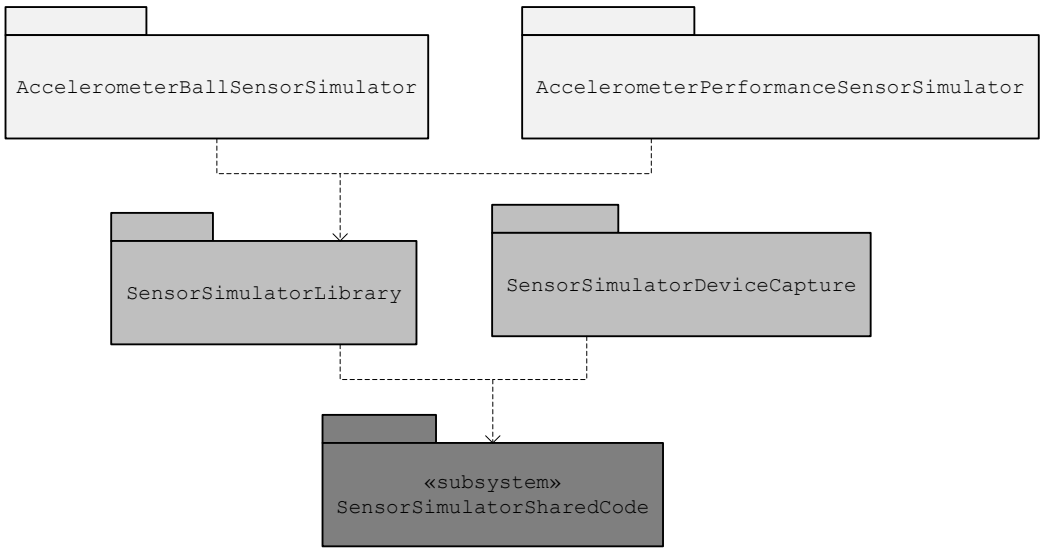

Figura 4.13: Organização da estrutura da implementação

A aplicação de teste da infraestrutura encontra-se no pacote `AccelerometerBallSensorSimulator`. Esta aplicação depende da biblioteca de simulação sensorial. A biblioteca encontra-se definida no pacote

`SensorSimulatorLibrary`. A aplicação móvel de captura está definida no pacote `SensorSimulatorDeviceCapture`. Tanto o pacote `SensorSimulatorLibrary` como o `SensorSimulatorDeviceCapture` dependem do subsistema `SensorSimulatorSharedCode`. Este subsistema contém código que é comum a ambos.

A aplicação `AccelerometerPerformanceSensorSimulator` contém um teste que será explicado mais à frente. Ambas as aplicações de teste da infraestrutura de simulação sensorial estão também disponíveis para dispositivos reais.

4.7 Sumário

Neste capítulo foram descritos todos os aspetos mais relevantes no desenvolvimento da solução. O capítulo começou por fazer a descrição dos aspetos gerais da plataforma *Android* que influenciaram na escolha de componentes e mecanismos para a solução. De seguida foram descritos os detalhes mais relevantes da aplicação móvel de captura e da biblioteca de simulação sensorial. O capítulo termina com a descrição das funcionalidades não implementadas e com a informação relativa ao suporte de alterações.

Capítulo 5

Avaliação da solução desenvolvida

Este capítulo começa por caracterizar o ambiente de desenvolvimento e teste utilizado durante o desenvolvimento da plataforma. É de seguida descrita a forma de utilização da infraestrutura, salientando as diferenças práticas entre a biblioteca de simulação sensorial e a biblioteca de sensores da plataforma *Android*. Por fim, é feita uma avaliação relativa ao desempenho da plataforma.

5.1 Ambiente de desenvolvimento e teste

Durante o desenvolvimento dos protótipos e de testes, foi usado o ambiente de desenvolvimento *Eclipse* com a versão 4.3.1, hospedado no sistema operativo *Windows 7* com arquitetura de 64 *bits* da *Microsoft*. As principais características da máquina hospedeira são:

i) Processador - *Intel(R) Core(TM)2 Duo CPU P8600* com 2.40GHz de velocidade de processamento;

ii) Memória – 2 x 2 *Gigabyte DDR2 SDRAM* com 800MHz de velocidade de funcionamento;

iii) Placa gráfica - *ATI Mobility Radeon HD* 3650 com 512 *Megabyte* de memória dedicada e com 700MHz de velocidade de processamento.

A versão do *Eclipse* utilizada integra a extensão *Android* SDK incluindo o simulador para testes. A versão da plataforma *Android* usada no simulador corresponde

à 4.4.2 (*KitKat*, API nível 19). Para avaliação dos vários protótipos foram usados dois *smartphones*:

i) *Samsung Galaxy Mini S5570* com processador de 600MHz *armv6*, com 384 *Megabyte* de memória e com a versão da plataforma *Android* 4.0.4 (*Ice Cream Sandwich*, API nível 15);

ii) *ZTE V875* com processador de 800MHz, com 420 *Megabyte* de memória e com a versão da plataforma *Android* 2.3.5 (*Gingerbread*, API nível 10).

Por motivos de eficiência, na atualização do comportamento gráfico, optou-se por utilizar outro simulador, o *Genymotion* [19]. No entanto, inicialmente o desenvolvimento foi feito através do simulador da plataforma *Android*. O *Genymotion* é um simulador que pode ser obtido gratuitamente ou mediante pagamento. A versão gratuita tem suporte de um número reduzido de funcionalidades. As funcionalidades (pagas e gratuitas) mais relevantes deste simulador podem ser descritas da seguinte forma:

i) Suporte de GPS (Gratuita);

ii) Suporte de acelerómetro (Paga);

iii) Suporte de multitoque (Paga).

Segundo alguns testes que foram executados, este simulador tem um desempenho superior ao fornecido pela plataforma *Android* porque usa originalmente uma arquitetura de virtualização x86, embora seja possível instalar no simulador da plataforma *Android* um componente que faça esta virtualização [20]. O simulador *Genymotion* utiliza a aceleração do *hardware* na implementação do *OpenGL*, obtendo também um desempenho superior em relação à implementação do *OpenGL* feita pelo simulador da plataforma *Android*. Ambos os simuladores integram com o ambiente de desenvolvimento *Eclipse*.

De forma a aumentar a diversidade dos testes de comunicação foram usadas duas infraestruturas distintas de rede sem fios. Uma das infraestruturas de rede pertence ao Instituto Superior de Engenharia de Lisboa (ISEL), a outra consiste numa rede doméstica. Os testes executados foram feitos com ambos os dispositivos (dispositivo móvel e computador) na mesma rede.

5.2 Experiência de utilização da infraestrutura

Para poder ser feita a utilização da infraestrutura devem ser assegurados os seguintes requisitos:

i) Existência de uma rede sem fios;

ii) A aplicação móvel de captura deve estar instalada, iniciada e com acesso à rede sem fios no dispositivo real;

iii) A biblioteca de simulação sensorial deve fazer parte das dependências da aplicação a conceber (assumindo que o endereço de rede do dispositivo real já foi inserido e a biblioteca compilada com o mesmo).

De forma a ilustrar a utilização da biblioteca de simulação, foi concebida uma aplicação cuja funcionalidade é pouco interessante mas contém os aspetos essenciais da utilização. A aplicação tem como finalidade saber qual o fornecedor do sensor de luminosidade. Para isto deve ser feito a importação das classes `Sensor` e `SensorManager`. Adicionalmente é também necessário fazer a importação da classe responsável pelas exceções que podem ocorrer, `SensorSimulatorException`.

De seguida é implementada a atividade, e no código da sua criação é pedida uma instância do `SensorManager` para através desta adquirir o sensor de luminosidade. Através da instância do sensor de luminosidade invoca-se o método `getVendor` e o seu conteúdo é guardado numa variável. Como os métodos das classes `SensorManager` e `Sensor` podem estar sujeitos a lançar exceção em caso de falha de rede, envolve-se os mesmos numa cláusula para capturar a exceção. Não é feito qualquer processamento com a exceção por motivos de simplificação. Na Figura 5.1 é possível observar o troço de código descrito anteriormente.

```
import jz.isel.sensorsimulatorlibrary.hardware.Sensor;
import jz.isel.sensorsimulatorlibrary.hardware.SensorManager;
import jz.isel.sensorsimulatorsharedcode.exceptions.SensorSimulatorException;
// ... Outros importes relativos ao Android.
public class SimpleDemoSS extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // ... Código de inicialização da Activity.
        try {
            SensorManager sm = SensorManager.getSystemService(Context.SENSOR_SERVICE);
            Sensor sensor = sm.getDefaultSensor(Sensor.TYPE_LIGHT);
            String vendor = sensor.getVendor();
        } catch (SensorSimulatorException e) {
            // Captura e manipulação da exceção.
        }
    }
}
```

Figura 5.1: Exemplo do uso da biblioteca de simulação sensorial

Um dos objetivos deste projeto visa uma experiência de utilização o mais semelhante possível à biblioteca de sensores da plataforma *Android*. Com vista a deixar este objetivo completo, é ilustrada na Figura 5.2 a implementação da aplicação referida anteriormente usando a biblioteca de sensores da plataforma *Android*. É de notar que todas as diferenças de utilização existentes estão incluídas no seguinte exemplo.

```
import android.hardware.Sensor;
import android.hardware.SensorManager;
// ... Outros importes relativos ao Android.
public class SimpleDemoAnroid extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // ... Código de inicialização da Activity.
        SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        Sensor sensor = sm.getDefaultSensor(Sensor.TYPE_LIGHT);
        String vendor = sensor.getVendor();
    }
}
```

Figura 5.2: Exemplo do uso da biblioteca de sensores da plataforma *Android*

As diferenças que se podem encontrar entre a utilização da biblioteca de simulação sensorial e a biblioteca de sensores da plataforma *Android* são: o nome dos módulos importados, a sintaxe no momento de adquirir uma instância de `SensorManager` e a adição do mecanismo de captura de exceções.

5.3 Desempenho em tempo real da infraestrutura

O desempenho em tempo real da infraestrutura é um dos objetivos principais deste projeto de mestrado, pois a biblioteca de sensores é normalmente utilizada em aplicações que requerem informação em tempo real.

De forma a medir o tempo de resposta da biblioteca foi criada uma aplicação de teste que calcula o número de eventos disparados por segundo, em particular para o acelerómetro. Esta aplicação possui uma interface gráfica, que pode ser observada na Figura 5.3.

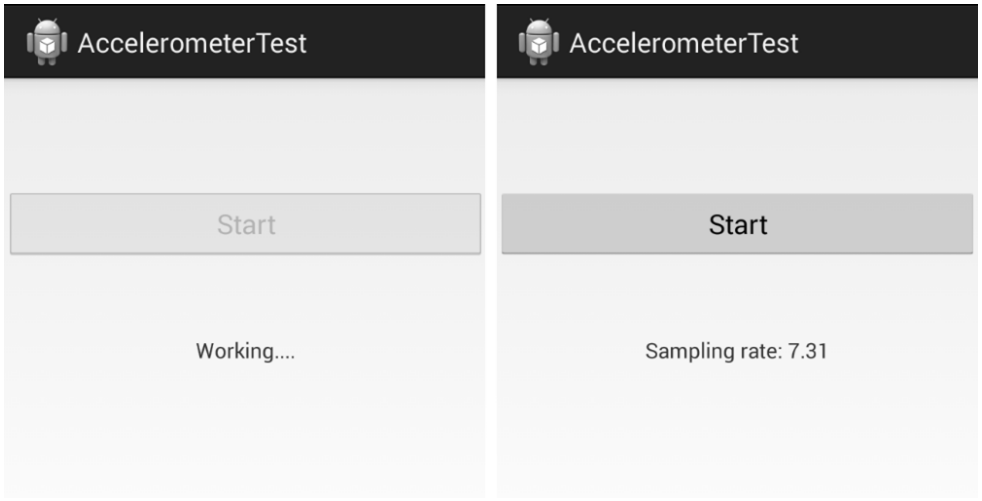

Figura 5.3: Interface gráfica da aplicação de testes de tempo de resposta

A interface tem um botão para dar início ao teste e uma caixa de texto que serve para apresentar o resultado do teste. Quando o botão é clicado, é registado um *listener* para o acelerómetro e durante um período definido em segundos *a priori*, são contadas quantas vezes é invocado o *listener*. Após terminado o tempo é calculado o número de invocações feitas num segundo. Através desta aplicação pode fazer-se uma comparação entre os resultados obtidos utilizando o dispositivo real e a biblioteca de simulação sensorial.

A plataforma *Android* permite que se indique um atraso pré-definido entre cada a receção de cada evento no momento em que se faz o registo de um *listener*. Este atraso é categorizado por ordem decrescente, em termos de atraso, da seguinte forma:

i) *SENSOR_DELAY_NORMAL*;

ii) *SENSOR_DELAY_UI*;

iii) *SENSOR_DELAY_GAME*;

iv) *SENSOR_DELAY_FASTEST*.

O atraso é apenas uma referência para a plataforma, quer isto dizer que os eventos podem ser recebidos mais rapidamente ou mais lentamente. Posto isto, foi utilizado o ritmo mais elevado (*SENSOR_DELAY_FASTEST*) nos testes realizados.

5.3.1 Resultados obtidos

A escolha da duração e da dimensão dos testes realizados para a infraestrutura baseou-se nos dados observados durante testes feitos no desenvolvimento.

Para cada teste foram feitas cinco medições. Cada medição tem a duração de trinta segundos. Foram feitos testes com durações superiores a trinta segundos e observou-se resultados muito semelhantes. Trinta segundos não é um valor muito elevado, tornando o processo de testes mais rápido.

Em todos os testes foi utilizado o dispositivo real o *ZTE V875*. A biblioteca de simulação sensorial foi testada no simulador da plataforma *Android* e no simulador da *Genymotion*.

A Tabela 1 apresenta nas colunas os três testes realizados. Nas linhas estão as cinco medições que foram feitas. A última linha contém a média das medições de cada teste. As unidades de cada medição são dadas em eventos por segundo.

O primeiro teste foi realizado recorrendo ao dispositivo real a executar a biblioteca de sensores da plataforma *Android.* O segundo teste foi realizado no simulador da plataforma *Android* utilizando a infraestrutura de simulação e o terceiro teste foi realizado no simulador da *Genymotion* utilizando a infraestrutura de simulação. Os resultados obtidos podem ser observados na Tabela 1.

Medições (dadas em eventos por segundo)	Biblioteca de sensores da plataforma *Android* a executar no *ZTE V875*	SSS a executar no simulador da plataforma *Android* com o dispositivo real *ZTE V875*	SSS a executar no simulador *Genymotion* com o dispositivo real *ZTE V875*
Medição 1	7.51	7.25	7.53
Medição 2	7.69	7.36	7.30
Medição 3	7.56	7.33	7.74
Medição 4	7.79	7.03	7.60
Medição 5	7.75	7.42	7.37
Média	7.66	7.28	7.51

Tabela 1: Comparação do resultado dos tempos de resposta

A diferença entre a média de cada teste é pouco significativa. Mas é possível observar que o desempenho no simulador da *Genymotion* está muito próximo do teste realizado com a biblioteca de sensores da plataforma *Android* a executar diretamente no dispositivo real.

5.3.1.1 Desempenho da biblioteca em ambiente 2D

Depois do objetivo relacionado com o desempenho em termos do tempo de resposta da biblioteca estar realizado. Decidiu-se testar a utilização da biblioteca mais perto de um ambiente real relacionado com o ambiente gráfico. Tendo em conta que o objetivo principal deste projeto em termos de desempenho já foi validada, este teste é de importância secundária. Desta forma, a medição do desempenho não foi feita com máxima precisão. Ou seja, os resultados apresentados sobre este teste em particular são o reflexo da observação pessoal.

Foi criada uma aplicação minimalista de teste que tira partido do acelerómetro para fazer o controlo de uma bola em duas dimensões. Esta aplicação é distribuída com a solução para permitir testar se a infraestrutura está corretamente operacional.

Nos testes realizados foi apenas comparado o tempo de resposta observado no dispositivo real *ZTE V875* e nos simuladores da plataforma *Android* e da *Genymotion*.

Claramente o desempenho do tempo de resposta gráfico é superior usando apenas o dispositivo real. Já na utilização dos simuladores da plataforma *Android* e da

Genymotion a executar a biblioteca de simulação sensorial o desempenho é visivelmente superior no simulador da *Genymotion.* Comparando o tempo de resposta da biblioteca de simulação observado no simulador da *Genymotion* e na utilização da biblioteca de sensores da plataforma *Android* a executar no dispositivo real, pode afirmar-se que o tempo de resposta observado no simulador da *Genymotion* é relativamente inferior. A diferença entre o desempenho atingido entre o dispositivo real e o simulador da *Genymotion* pode variar entre cerca de um e dois segundos.

A figura que se segue mostra a aplicação em execução. Esta aplicação funciona em modo de ecrã inteiro e os limites do ecrã estão sujeitos à colisão da bola.

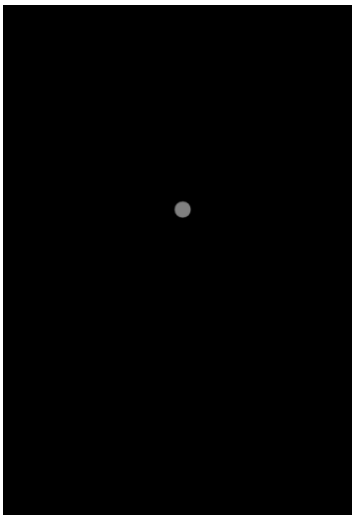

Figura 5.4: Exemplo da execução da aplicação de teste em duas dimensões

5.4 Sumário

O capítulo inicia-se com a descrição do ambiente de desenvolvimento utilizado para fazer o desenvolvimento e testes. Foram apresentados os dispositivos móveis e simuladores utilizados. De seguida foi explicada a experiência de utilização e feita a comparação entre o uso da biblioteca de sensores da plataforma *Android* e o uso da biblioteca de simulação sensorial.

Foi apresentado o desempenho da infraestrutura face à utilização da biblioteca de sensores da plataforma *Android*.

Por fim, foi apresentado o desempenho da biblioteca numa aplicação minimalista que utiliza uma bola em duas dimensões que é controlada pelo acelerómetro.

Capítulo 6

Conclusão

Os *smartphones* desempenham um papel fulcral no acesso e processamento de informação. Estes dispositivos possuem um conjunto de sensores integrados que os torna capazes de gerar informação de elevada precisão. O desenvolvimento de aplicações móveis, quando executado em plataformas hospedeiras através de simuladores, é incompleto pois o suporte de sensores não é genérico nem completo.

Neste contexto, a origem da investigação da plataforma *Android* foi, em grande medida, inspirada pela tentativa de compreender o seu funcionamento a um nível mais aprofundado de forma a colmatar o problema descrito anteriormente, sugerindo uma arquitetura e implementação o mais genérica e completa possível. Progressivamente foram estudadas as soluções que emergiram na tentativa de solucionar este dilema, de forma a compreender quais as suas abordagens e eventuais lacunas. Com base na informação adquirida, traçaram-se um conjunto de objetivos e características que a solução tentou perfilhar.

De forma a suportar a implementação de uma infraestrutura com diferentes tipos e níveis de complexidade, foi definida uma arquitetura que tornasse a implementação o mais genérica possível. Essa arquitetura é caracterizada por um conjunto de mecanismos base que concretizam e tiram partido da integração entre uma biblioteca e um dispositivo móvel real que alcançam a simulação sensorial.

A implementação resulta num conjunto de componentes que, por meio de comunicação sem fios, interagem de forma a fazer a simulação da biblioteca de sensores disponibilizada pela plataforma *Android*.

Os casos experimentais apresentados mostram a viabilidade da abordagem proposta, validando a solução apresentada neste documento. Constatou-se que a simulação, tendo como percursor um dispositivo móvel real é possível, permitindo assim a utilização de sensores em tempo real, facilitando o desenvolvimento de aplicações móveis.

6.1 Principais dificuldades

No decorrer deste projeto foram sentidas algumas dificuldades, nomeadamente pelo facto do tema abordado não ser de divulgação elevada, tornando a pesquisa uma tarefa difícil.

De forma a poder ser compreendida a implementação dos sensores, numa fase inicial, onde ainda não tinha sido decidida qual a abordagem a emergir, houve alguma dificuldade em aceder ao código fonte da plataforma *Android*, dado que a sua dimensão é bastante elevada (a versão 2.3.3 tem cerca de nove *Gigabytes*) e os métodos de extração descritos na página da internet da plataforma encontram-se desatualizados.

A divergência existente na implementação da biblioteca de sensores da plataforma *Android* ao longo do tempo fez com que fosse perdido algum tempo, que acabou por ser desnecessário no âmbito deste projeto. A atual implementação da biblioteca de sensores da plataforma *Android* surgiu de uma implementação mais rudimentar e mais complexa.

Durante o estudo das soluções já existentes foram sentidas algumas dificuldades no sentido em que existe falta de informação relativa à sua instalação e utilização, nomeadamente na solução fornecida pela *Samsung* e pela plataforma *Android*. No que diz respeito ao código das soluções não foi possível aceder ao código da solução da *Samsung* dificultando assim a sua compreensão.

O período de implementação foi caracterizado por ser o mais extenso, havendo um conjunto de fatores que contribuíram para o mesmo. O principal fator deveu-se à utilização do simulador da plataforma *Android.* Este simulador apresenta, no ambiente utilizado, um desempenho bastante deplorável desde o tempo de arranque, como o tempo de carregamento de aplicações para depuramento. A descoberta do simulador *Genymotion* sucedeu sensivelmente a um terço do fim da implementação.

A natureza da solução está inerente a uma maior dificuldade de depuramento dado que a comunicação é feita através da rede sem fios. A origem distribuída da infraestrutura aumentou a taxa de erros e dificultou a correção dos mesmos. Tal como

referido na secção 4.4.1 O problema cliente-servidor, este problema potenciou algum atraso, dado que a informação associada à origem do problema não constava na documentação.

6.2 Contribuição

A solução descrita neste relatório de projeto de mestrado traz uma série de vantagens face às soluções que existem atualmente. Estas vantagens estão essencialmente relacionadas com completude, portabilidade, e generalização.

A implementação base é completa em termos de funcionalidades subjacentes à utilização padrão de sensores. Todos os sensores estão disponíveis, não havendo limitações relacionadas com as operações disponibilizadas.

A solução caracteriza-se por ser portável, ou seja, tem a capacidade de ser executada em qualquer versão da plataforma *Android* desde que seja superior à 2.3. Independentemente da versão que o dispositivo real execute, só podem ser executadas as funcionalidades que dizem respeito a essa versão. A infraestrutura pode ser executada em qualquer simulador que suporte a plataforma *Android*.

A generalização é uma característica importante tendo em conta que permite que a biblioteca possa sofrer alterações futuras sem comprometer a estrutura implementada. Esta vantagem permite que possam ser feitas melhorias, ou acrescentadas novas funcionalidades.

É de destacar a semelhança no acesso à biblioteca de simulação sensorial em relação à biblioteca da plataforma *Android*, tornando possível a transposição de uma aplicação de ambiente de teste, numa aplicação real, procedendo a alterações mínimas.

Por fim, é acrescentado valor a esta solução pelo facto do seu desempenho ter-se revelado elevado nos testes realizados, o que faz desta solução uma solução eficaz.

6.3 Trabalho futuro

A implementação corrente do *Simple Sensor Simulator* contém toda a parte central de funcionamento, nomeadamente a definição do sistema de comunicação e as definições das operações essenciais. O sistema de conversão prevê a extensibilidade da infraestrutura, relativamente à adição de novos tipos.

A implementação pode ser melhorada em diferentes aspetos, de forma a possibilitar uma melhor experiência de utilização.

A aplicação móvel de captura deveria disponibilizar na interface de utilização a função de seleção restringida de sensores. Há a necessidade de monitorar que sensores estão a ser reportados e eventualmente guardar um registo com as operações requeridas.

A biblioteca de simulação sensorial, pelos motivos referidos na secção 4.3 Biblioteca de simulação sensorial, deve sofrer algumas melhorias. A conversão dos tipos de retorno não primitivos na classe `SensorManager` foram feitos manualmente de forma a acelerar o processo de implementação. No entanto a solução que foi utilizada na aplicação móvel de captura para fazer a conversão de tipos seria a escolha certa a implementar neste caso. Seria útil criar um mecanismo de geração das classes `SensorManager` e `Sensor` a partir das fontes da plataforma *Android*, visto que grande parte do código é repetitivo ou padronizável.

Com o objetivo de amadurecer esta infraestrutura, seria interessante adicionar a simulação de novos periféricos. Um dos periféricos com maior interesse é o GPS. Este 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. O suporte de multitoque seria outra funcionalidade com elevado interesse. Esta funcionalidade é coberta pelas ferramentas da plataforma *Android* mas sofre dos mesmos problemas referidos na solução dos sensores na secção 2.4.1 *Android Tools - Hardware Emulation*. Por fim, a mais ambiciosa adição, seria integrar as simulações referidas anteriormente no sistema de localização fornecido na plataforma *Android*. Este sistema de localização utiliza diversas técnicas para calcular a localização. O GPS é usado, mas também existem outras opções que fazem uso de sensores de movimento.

Referências bibliográficas

1. **Ken Schwaber, Mike Beedle.** *Agile Software Development with Scrum.* s.l. : Prentice Hall, Edition 1, 2001.

2. **Google.** Android Developers. *Android Developers.* [Online] Abril de 2014. https://developer.android.com.

3. —. Dashboards. *Android Developers.* [Online] Setembro de 2014. https://developer.android.com/about/dashboards/index.html.

4. **Ehringer, David.** The Dalvik Virtual Machine. [Online] 2010. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf.

5. **elinux.** Android Architecture. *elinux.* [Online] Janeiro de 2014. http://elinux.org/Android_Architecture.

6. **Brahler, Stefan.** Analysis of the Android. [Online] Junho de 2010. http://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf.

7. **Openintents.** Sensor Simulator for simulating sensor data in real time. [Online] Junho de 2014. https://code.google.com/p/openintents/wiki/SensorSimulator.

8. **Samsung.** *Samsung Sensor Simulator*. [Online] Junho de 2014. http://developer.samsung.com/android/tools-sdks/Samsung-Sensor-Simulator.

9. **Fielding, Roy Thomas.** Architectural Styles and the Design of Network-based Software Architectures. [Online] Maio de 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

10. **Google.** Processes and Threads. *Android Developers.* [Online] Julho de 2014. http://developer.android.com/guide/components/processes-and-threads.html.

11. —. Handler. *Android Developers.* [Online] Agosto de 2014. http://developer.android.com/reference/android/os/Handler.html.

12. **Goswami, Mainak.** *Singleton Design Pattern – An introspection and best practices.* [Online] Julho de 2014. http://www.javacodegeeks.com/2013/02/singleton-design-pattern-an-introspection-and-best-practices.html.

13. **Oracle.** *Java SE Documentation.* [Online] Junho de 2014. http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Method.html.

14. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley Professional, Edition 1, 1994.

15. **Google.** Building and Running. *Android Developers.* [Online] Junho de 2014. http://developer.android.com/tools/building/index.html.

16. **Ehringer, David.** *David Ehringer Blog.* [Online] Março de 2010. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf.

17. **Parmar, Ketan.** *In Depth : Android Package Manager and Package Installer.* [Online] Outubro de 2012. http://www.kpbird.com/2012/10/in-depth-android-package-manager-and.html.

18. **Google.** DexFile. *Android Developers.* [Online] Junho de 2014. http://developer.android.com/reference/dalvik/system/DexFile.html.

19. **Genymobile.** *Genymotion.* [Online] Maio de 2014. http://www.genymotion.com/.

20. **Google.** *Configuring Virtual Machine Acceleration.* [Online] Agosto de 2014. http://developer.android.com/tools/devices/emulator.html.

21. **Yi Zhi-An, Mu Chun-Miao.** The Development and Application of Sensor Based. *IEEE* . [Online] Junho de 2012. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6269263.

22. **Won-Jae Yi, Weidi Jia, Jafar Saniie.** Mobile sensor data collector using Android smartphone. *2012.* [Online] Agosto de 2012. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6292180.

23. **Greg Milette, Adam Stroud.** *Professional Android Sensor Programming.* s.l. : Wrox, Edition 1, 2012.

24. **Frank Ableson, Robi Sen, Chris King, C. Enrique Ortiz.** *Android in Action.* s.l. : Manning Publications, Edition 3, 2011.

25. **Iggy Krajci, Darren Cummings.** *Android on x86: An Introduction to Optimizing for Intel® Architecture.* s.l. : Apress, Edition 1, 2013.

26. **J. Paek, J. Kim and R. Govindan.** *Energy-efficient rate-adaptive GPS-based positioning for smartphones.* 2010.

27. **Google.** Android Platform Frameworks Base. *GitHub.* [Online] Junho de 2014. https://github.com/android/platform_frameworks_base/tree/master/core/java/android/hardware.

28. —. Application Fundamentals. *Android Developers.* [Online] Março de 2014. http://developer.android.com/guide/components/fundamentals.html.

29. —. Hardware Emulation. *Android Developers.* [Online] Junho de 2014. http://tools.android.com/tips/hardware-emulation.

30. —. Message Class. *Anroid Developers.* [Online] Agosto de 2014. http://developer.android.com/reference/android/os/Message.html.

31. **International Data Corporation.** *Smartphone OS Market Share, Q2 2014.* [Online] Setembro de 2014. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

32. **Huang, Chih-Wei.** Android-x86. *Porting Android to x86.* [Online] Abril de 2014. http://www.android-x86.org/getsourcecode.

33. **Samsung.** *Samsung Sensor Simulator*. [Online] Junho de 2014. http://developer.samsung.com/android/tools-sdks/Samsung-Sensor-Simulator.