



Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Suporte à Cooperação em Computação Móvel e Ubíqua

Fábio Neves (aluno nº 26476)

2º Semestre de 2008/09

29 de Julho de 2009



Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Suporte à Cooperação em Computação Móvel e Ubíqua

Fábio Neves (aluno nº 26476)

Orientador: Prof. Doutor Sérgio Marco Duarte

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2008/09

29 de Julho de 2009

Agradecimentos

Desejo expressar o meu agradecimento a todos os que tornaram possível esta dissertação:

Em primeiro lugar, ao Professor Sérgio Duarte, por me ter proposto para esta dissertação, e pelas recomendações e orientação durante o percurso do trabalho desenvolvido.

De seguida, a todos os amigos e colegas da FCT/UNL, que me apoiaram bastante durante este último ano, em especial ao Ricardo, Pedro, Mário, Rúben, Francisco e Nuno pelos conselhos e presença durante a realização desta dissertação.

Por fim, gostaria de agradecer a toda a minha família, que sem o seu suporte nada deste trabalho poderia ter sido realizado.

Resumo

Com o avanço da tecnologia e crescente miniaturização de dispositivos e componentes electrónicos, existem, hoje em dia, vários dispositivos móveis com grandes recursos computacionais e diversidade de sensores.

Participatory Sensing é uma emergente área da computação móvel que explora a ubiquidade, mobilidade, captura de informação contextual e interacção com os utilizadores destes dispositivos. O seu objectivo é a construção de aplicações de nova utilidade, tirando partido de um espírito colaborativo/cooperativo de uma comunidade.

Esta dissertação foca os problemas do suporte oferecido ao desenvolvimento deste tipo de aplicações e incentivos à partilha e contribuição por parte de uma comunidade. Mais concretamente, neste trabalho propõem-se um conjunto de modelos e descreve-se um protótipo com o intuito de facilitar e fomentar o desenvolvimento de aplicações *Participatory Sensing*. A sua validação ocorre através de simulação e de um caso de estudo.

Palavras-chave: Sensores, Computação Móvel e Ubíqua, Incentivos, Computação Social e Cooperativa, Plataforma de *Middleware*.

Abstract

Steady technological advances in miniaturization of electronic devices and components made possible to have today mobile devices with significant computational resources and equipped with a growing number of sensors. Participatory Sensing is an emergent area of mobile computing, exploring the growing ubiquity, mobility and sensing capabilities of these types of devices. Its broader goal is the development of novel applications, exploring the opportunities offered by the collaborative spirit of a community.

This dissertation focuses on the issues of designing a platform intended to ease the development of Participatory Sensing applications. Specifically, it proposes a set of models to guide the design of these applications and describes the platform prototype implementation. Validation is provided resorting to simulation and an actual case-study application.

Keywords: Sensors, Mobile and Ubiquitous Computing, Incentives, Social and Cooperative Computing, *Middleware* Platform.

Conteúdo

1	Introdução	1
1.1	Motivação	1
1.2	Descrição do Problema	2
1.3	Objectivo	3
1.4	Principais Contribuições desta Dissertação	4
1.5	Estrutura do Documento	5
2	Modelos	7
2.1	Aplicações Modelo (<i>Participatory Sensing</i>)	7
2.1.1	Características fundamentais de uma aplicação <i>participatory sensing</i>	8
2.1.2	Modelo da Aplicação	10
2.1.2.1	Componente Móvel da Aplicação	11
2.1.2.2	Componente Fixa da Aplicação	12
2.2	Modelo de Incentivos	12
2.3	Modelo de Comunicações	14
2.3.1	Qualidade de Serviço	14
2.3.2	Modelo de Interacção	15
2.3.3	Canais	16
2.3.3.1	Escopo Local	16
2.3.3.2	Escopo Ad-Hoc	16
2.3.3.3	Escopo Global	17
2.3.4	Encaminhamento e Filtragem	18
2.3.5	Mobilidade e Desconexão	19
2.3.5.1	Serviço de <i>Binding</i> a um <i>Proxy</i>	19
2.3.5.2	Serviço de Roaming	20
2.4	Modelo da Arquitectura	21
2.4.1	Infra-Estrutura	22
2.4.1.1	Componente Fixa da Aplicação	23
2.4.1.2	<i>Proxy</i>	23
2.4.2	Componente Móvel da Aplicação	23
2.4.3	Componente Aplicação Incentivo	23
2.4.4	Substrato <i>Publish/Subscribe</i>	23
2.5	Modelo de Persistência	24
2.6	Modelo de Programação	25
3	Protótipo	27
3.1	FEEDS	27
3.1.1	Arquitectura	28

3.1.2	Nó	29
3.2	MEEDS	31
3.2.1	Arquitectura	33
3.2.2	Comunicações e Mobilidade	34
3.2.2.1	Tunnel	34
3.2.2.2	Serviços	35
3.2.2.3	Templates	42
3.2.2.4	Transportes e diferentes tecnologias de comunicação	45
3.3	Sensores	46
3.3.1	Arquitectura dos Canais	47
3.3.2	Acesso a Sensores	49
3.3.3	Parametrização de Sensores	50
3.3.4	Sensores Suportados	50
3.3.5	Desenvolvimento de novos Sensores	51
3.3.5.1	Desenvolvimento de <i>Channel Template</i>	51
3.3.5.2	Desenvolvimento de <i>Containers</i> para sensores	52
3.3.5.3	Desenvolvimento de Sensores Virtuais	53
3.4	Validação Experimental	54
3.4.1	Avaliação da perda de pacotes e duplicados	55
3.4.2	Custo médio da comunicação em termos de latência	57
3.4.2.1	Latência entre <i>publisher</i> e <i>subscriber</i>	57
3.4.2.2	Latência entre <i>publisher</i> e infra-estrutura	58
3.4.3	Conclusões	59
4	Caso de estudo	61
4.1	TouristHelper	61
4.1.1	Modo Turista	62
4.1.1.1	Mapa	64
4.1.2	Modo Residente	65
4.2	Implementação	68
4.2.1	Canais, Eventos e Filtros	68
4.2.2	Captura de Informação	73
4.2.3	Componente Fixa da Aplicação e <i>Homepage</i>	73
4.2.4	Submissão de informação associada a um local ou evento	74
4.2.5	Consulta de Informação associada a um local ou evento turístico	74
4.2.5.1	Mecanismo de Persistência	75
4.3	TourismTOP	76
4.3.1	Arquitectura e Implementação da aplicação	76
4.3.1.1	Componente de apresentação	76
4.3.1.2	Componente servidor	77
4.4	Segurança	78

4.5	Validação e Conclusões	78
5	Trabalho Relacionado	81
5.1	Sistemas Sensoriais Desenvolvidos com um Objectivo Específico	81
5.1.1	Sistemas Desenvolvidos Com um Objectivo relacionado com Trânsito	82
5.1.1.1	PotholePatrol	82
5.1.1.2	Nericell	82
5.1.1.3	TrafficView	83
5.1.2	Outros Sistemas Sensoriais	83
5.1.2.1	BikeNet	84
5.1.2.2	MicroBlog	84
5.1.2.3	CenceMe	85
5.1.3	Sumário	85
5.2	Plataformas	86
5.2.1	Plataformas que assentam no modelo P2P	86
5.2.1.1	Mobile Cheddar - A Peer-to-Peer Middleware for Mobile De- vices	87
5.2.1.2	A Design of Service-Based P2P Mobile Sensor Networks e Generic Communication Structure to Integrate Widely Dis- tributed Wireless Sensor Nodes by P2P Technology	87
5.2.1.3	P2PMonitoring	88
5.2.1.4	Global Sensor Network	88
5.2.2	Plataformas que assentam no modelo Cliente-Servidor	89
5.2.2.1	CarTel	89
5.2.2.2	COSMOS	90
5.2.2.3	SenseWeb	91
5.2.3	Plataformas que assentam num modelo Híbrido	92
5.2.3.1	IrisNet	92
5.2.3.2	DEEDS	93
5.2.4	Sumário	96
6	Conclusões	97
6.1	Objectivos Revistos	98
6.2	Trabalho Futuro	99

Lista de Figuras

2.1	Modelo geral de uma aplicação <i>Participatory Sensing</i>	10
2.2	Modelo concreto de uma aplicação <i>Participatory Sensing</i>	11
2.3	Esquema que ilustra o modelo de incentivos.	13
2.4	Fluxo de mensagens possível num canal do escopo ad-hoc	17
2.5	Fluxo de mensagens possível num canal do escopo global	17
2.6	Fluxo de mensagens possível num canal do escopo vizinhança	18
2.7	Serviço de Binding a um Proxy	20
2.8	Serviço de suporte a Desconexão	21
2.9	Modelo da Arquitectura do Sistema	22
2.10	Modelo abstracto de armazenamento de informação produzida pelas aplicações	24
2.11	Modelo abstracto de consulta de informação contextual armazenada	25
3.1	Visão global da comunicação entre entidades no sistema FEEDS	28
3.2	Arquitectura de três camadas de FEEDS, retirado de [5]	28
3.3	Exemplos de possíveis configurações de FEEDS, retirado de [5]	29
3.4	Arquitectura de um nó, imagem adaptada de [5]	30
3.5	Arquitectura segundo uma visão alto nível do componente <i>Processing Pipeline</i> , imagem adaptada de [5]	31
3.6	Estrutura de camadas que ilustra a extensão de FEEDS	32
3.7	Enquadramento das entidades Homebase e Proxy em FEEDS	33
3.8	Funcionamento do tunnel	35
3.9	Processo do Serviço de <i>Homing</i>	37
3.10	Composição de serviços para suporte à mobilidade	38
3.11	Concretização do modelo de persistência, em relação ao armazenamento dos eventos	40
3.12	Concretização do modelo de persistência, em relação à consulta dos dados armazenados	41
3.13	Colocação da camada SEEDS na hierarquia de todas as camadas de suporte.	47
3.14	Esquema da Criação de um Sensor	48
3.15	Arquitectura de classes para o desenvolvimento de Contentores de Sensores	52
3.16	Hierarquia de classes de Sensores Virtuais	53
3.17	Imagem exemplificando uma corrida no simulador	55
3.18	Percentagem de pacotes perdidos ao publicar de um nó móvel para a infraestrutura - Nos instantes iniciais a taxa é elevada devido à não inicialização dos canais nos proxys, reduzindo-se a partir do momento de activação destes nos proxys	56
3.19	Latência Global Média	58
3.20	Latência Média do Tunnel	58

4.1	Mapa da aplicação TourismTop em modo turista	63
4.2	Detalhe de um local turístico	65
4.3	Diferentes níveis de detalhe no mapa, normal, satélite e rua respectivamente	66
4.4	Mapa da aplicação TourismTop em modo residente	67
4.5	Mapa da aplicação TourismTop em modo residente	74
4.6	Mecanismo de Persistência	75
4.7	Arquitetura da aplicação TourismTOP	76
4.8	<i>Screenshot</i> da aplicação TourismTop	77
4.9	Binding da aplicação TOP à aplicação TourismHelper	77
5.1	Esquema que mostra a arquitetura de 3 camadas de DEEDS, retirado de DEEDS [5]	94
5.2	Esquema que ilustra possíveis topologias que DEEDS pode ter, retirado de DEEDS [5]	95

Lista de Tabelas

2.1	API disponibilizada ao programador	26
4.1	Mapeamento entre ícones e keywords	64

1 . Introdução

1.1 Motivação

Com a evolução tecnológica, os telemóveis deixaram de ser apenas utilitários, ganhando poder de computação, recursos sensoriais, comunicação melhorada, tornando-se um alvo muito apetecido para quem pretenda desenvolver um sistema pervasivo e ubíquo. Hoje em dia, esses mesmos telemóveis são cada vez mais abundantes, sendo o seu número, em termos de presença no mercado, superior aos *laptops*. Num futuro próximo, poderá prever-se que estes dispositivos possibilitarão o executar muitas das tarefas que hoje em dia são destinadas a esses mesmos *laptops* [4].

Nos últimos anos, a indústria tem vindo a investir cada vez mais no desenvolvimento de telemóveis [26] com um número significativo de sensores incorporados, o que aliado ao poder de computação e recursos destes dispositivos, abrem possibilidade de desenvolver uma nova panóplia de aplicações.

Deste modo, ao construirmos uma sistema deste tipo, à imagem de outros sistemas como a *wikipedia*, poderá ser interessante utilizar uma filosofia que construa esse sistema à roda de uma comunidade, que permita a sua evolução e o atingir de uma dimensão tal, que a soma de todas as pequenas contribuições de todos os participantes, tornem este sistema bastante rico, visto o total de dados agregado ser enorme.

A um nível mais restrito das aplicações móveis e úbiquas, situa-se o ramo das aplicações *participatory sensing* [17]. Estas são aplicações que através da captura de informação sensorial e contextual, abrem novos horizontes no que a aplicações móveis diz respeito.

Ao combinar e cruzar o grande volume de informação contextual recolhida, conseguem-se inferir abstracções mais elevadas, como por exemplo a detecção de situações de trânsito rodoviário em [23]. Combinando estas situações de elevado grau de abstracção umas com as outras, pode-se chegar ainda mais longe, como foi feito em CenceMe [22], em que é apresentado um sistema que permite obter conclusões acerca de comportamentos sociais, como se uma pessoa é social ou não, etc.

Este é um cenário entusiástico, porém certos autores que já se debruçaram sobre esta área, mostram-nos que existem muitos desafios no desenvolvimento de uma plataforma sobre este paradigma, sendo um indicador que pode explicar a escassez de aplicações com base em informação sensorial e contextual.

Assim, surge a motivação de desenvolver um suporte que fomente o desenvolvimento e construção deste tipo de aplicações, permitindo de uma forma intuitiva o acesso aos recursos necessário a esse desenho, bem como uma forma simples de partilhar informação entre as diversas entidades que compõem a comunidade, sejam ao escopo de simples utilizadores, seja ao nível das diferentes aplicações.

1.2 Descrição do Problema

A ideia de criar uma plataforma que suporte a recolha de informação contextual não é original, no entanto as várias propostas de plataformas não aparentam conseguir satisfazer uma comunidade, dando a impressão que algo falta.

Em [22] e [27], são identificadas bastantes dificuldades no que se refere a problemas encontrados no desenvolvimento de software para os dispositivos móveis.

Ao construirmos um sistema distribuído, somos confrontados com diversos problemas, por exemplo ao nível de escala e comunicações. Neste paradigma, a componente móvel acresce uma quantidade de desafios com que lidar. Neste novo âmbito torna-se necessário suportar problemas como a possível desconexão de alguns componentes, o encaminhamento para esses componentes móveis, recursos energéticos, heterogeneidade de hardware e software e ainda toda uma plataforma de desenvolvimento pouco robusta.

Tendo estes indicadores problemáticos em contexto, fica claro que, para desenvolver uma aplicação sensorial que assente num paradigma destes, será muito trabalhoso e dispendioso a nível de tempo para os programadores. Grande parte do processo de desenvolvimento será passado a cuidar de todas estas possíveis falhas, o que provoca, como é óbvio, entraves ao desenvolvimento de aplicações deste género.

Uma plataforma de trabalho que suportasse e tratasse os problemas anteriores, fornecendo ao programador estas "armas", seria aliciante para o desenvolvimento destas novas aplicações. Com isto em mente, a ideia de produzir uma plataforma *middleware* com estes objectivos, que trate das comunicações, possível desconexão de componentes, recursos energéticos, tolerância a falhas e por fim, que permita aos programadores criarem aplicações num modelo já bastante conhecido como é o Cliente-Servidor ou *publish/subscribe*, seria bastante útil e interessante.

Várias plataformas deste tipo já foram propostas, no entanto estas têm dificuldade em impor-se, em assumir-se junto de uma comunidade.

No geral, estas plataformas, utilizam como base o modelo de arquitectura Cliente-Servidor, centralizado, ou então o formato tradicional de P2P. A convicção de que estes dois modelos individualmente são insuficientes alia-se à constatação de que nenhuma destas plataformas foi aceite por uma grande comunidade.

Os problemas de uma plataforma que se encontre implementada em cima de um modelo Cliente-Servidor, é que esta sofre das limitações desse modelo, isto é, um ponto central de falha, pouca escala no caso de não existirem possibilidades financeiras, não sendo indicado o uso desta arquitectura no arranque de um sistema. Se por acaso a plataforma tiver sido desenvolvida sobre um modelo P2P, sofrerá dos problemas de um sistema P2P, ou seja, perde-se simplicidade e introduzem-se problemas de segurança e de gestão de recursos.

Mas será que são estes os únicos problemas, que levam a que a comunidade não adopte uma plataforma, com o objectivo a que mais e mais aplicações deste cariz venham a surgir?

Claramente que não serão os únicos, possivelmente são uma pequena parte do problema. O outro quinhão tem um cariz social, sendo "*esquecido*" pela comunidade que implementa estas plataformas.

Este factor social pode explicar a escassa adopção dos sistemas já desenhados, estando associado à falta de incentivos para que uma comunidade se estabeleça.

A criação destes incentivos, será possivelmente um dos factores mais importantes no arranque de um sistema deste tipo, e talvez na sobrevivência do mesmo. A existência de uma *killer application* será essencial, no entanto uma aplicação desse tipo terá de ser alimentada por informação sensorial, que no momento de disponibilização do sistema não existe.

Assim, é necessário algo que incentive uma comunidade a formar-se de modo a que estes dados possam ser recolhidos. Na verdade, acreditamos que o modelo de incentivos é um factor determinante à sobrevivência de um qualquer sistema de informação contextual. Tendo por base o público alvo que um sistema destes vai encontrar, é necessário reflectir qual a razão que levará uma pessoa desse grupo a contribuir. A contribuição implicará sempre o consumo de recursos ao utilizador. Se pensarmos num exemplo concreto, um utilizador que use o seu telemóvel para contribuir com informação capturada por este, perde alguma da sua privacidade (expondo dados relativos à sua vida) e existe um consumo acrescido de energia. É forçoso que os *prós* vençam os *contras*, por outras palavras, é necessário estabelecer um modelo de incentivos que permita um aliciante tal, que os utilizadores contribuam aceitando as consequências. Ainda nesta questão dos incentivos, há a dizer que o modelo pensado procura um elevado nível de abstracção, de forma a que se possam formar diversos aliciantes, que amparem e cativem o maior número de utilizadores.

Com este contexto de problemas identificados em mente, é possível identificar um passo para o futuro: a criação de uma plataforma que combine o melhor de dois mundos, o P2P e Cliente-Servidor, e que seja flexível para o programador, com vista a suportar novos sensores e novos protocolos de comunicação; junto com esta plataforma terá que ser necessário ter um modelo de incentivos associado, que case com o sistema de tal forma que se assemelhe a uma simbiose, permitindo a sobrevivência e o formar de uma comunidade em redor deste.

1.3 Objectivo

O objectivo desta dissertação é o desenho de um suporte que fomente o desenvolvimento e subsistência de aplicações móveis e ubíquas, com especial relevo nas aplicações *participatory sensing*. Este suporte, idealmente, deve ser desenvolvido sobre questões de comunicações e mobilidade, captura de informação sensorial e contextual, segurança, e persistência. Como argumentado anteriormente, existe um factor social implícito que terá um papel fundamental no arranque e sobrevivência destas aplicações, devendo estar ainda um modelo de incentivos definido.

Com este objectivo em mente, esta dissertação focar-se-há na definição de um suporte de comunicações e mobilidade, bem como no âmbito de captura de informação sensorial e contextual. Devido ao facto de ser interessante ter um exemplo de persistência nestas aplicações, será apresentado um modelo que permite um suporte rudimentar de armazenamento. Este último tema será apenas ilustrativo, sendo um problema complexo, susceptível a uma dissertação

especializada no assunto. Será ainda feita uma proposta na perspectiva de definir um modelo de incentivos.

Este suporte será desenvolvido em cima de uma plataforma já existente (FEEDS) que, no entanto, não tem suporte a mobilidade nem captura de informação sensorial. Para tal, esta plataforma será estendida, com vista a atingir esse mesmo suporte.

Sumariamente, podemos enunciar o conjunto de objectivos a que nos propomos:

- Através da adaptação da plataforma FEEDS, modelar uma plataforma de suporte a computação móvel e ubíqua, que junte o melhor de dois mundos, P2P e Cliente-Servidor, de forma a ultrapassar certas limitações como por exemplo escala e uma utilização racional dos recursos disponíveis.
- Adaptar a plataforma FEEDS com vista ao suporte de mobilidade e interacção com sensores, através da definição de um conjunto de modelos e concretização desses mesmos na forma de um protótipo.
- Proporcionar o acesso a informação sensorial de uma forma eficiente, simples e uniforme para o programador.
- Criar um modelo de incentivos que permita a subsistência de uma tal plataforma.
- Mostrar, através de um caso de estudo, como se pode usufruir do suporte fornecido bem como instanciar o modelo de incentivos.

1.4 Principais Contribuições desta Dissertação

O trabalho visa desenhar um suporte que fomente a criação de aplicações *participatory sensing*. Este suporte envolve o desenvolvimento de modelos que permitam tanto definir essas aplicações, como corresponder aos requisitos desse tipo de aplicações.

Assim, como principais contribuições deste trabalho podemos enumerar os seguintes pontos:

- Conjunto de modelos que definam o que é uma aplicação *Participatory Sensing* e quais os seus requisitos.
- Através da identificação dos requisitos das aplicações *Participatory Sensing*, desenvolvimento de modelos que suportem os requisitos de mobilidade, comunicações, e captura de informação sensorial.
- Definição de modelos de arquitectura adequados a aplicações *Participatory Sensing*.

1.5 Estrutura do Documento

O restante documento estará dividido em mais cinco capítulos : Modelos (capítulo dois), Protótipo (capítulo três), Caso de Estudo (capítulo 4), Trabalho Relacionado (capítulo 5) e Conclusões (Capítulo 6).

O capítulo dos modelos tem o objectivo de definir quais as aplicações tipo esperadas e como estabelecer um modelos que permitam a sua sobrevivência. Será ainda neste capítulo que serão discutidos os modelos idealizados ao suporte deste tipo de aplicações nomeadamente na definição de uma arquitectura e estrutura de comunicações.

Assim o capítulo foi organizado da seguinte forma:

- Em 2.1 serão definidas as aplicações modelo.
- Em 2.2 será apresentada uma aproximação a um modelo de incentivos que tentará garantir a sobrevivência destas aplicações.
- Em 2.3 serão apresentados os modelos que definem como devem ser executadas as comunicações num suporte a aplicações do género *participatory sensing*.
- Em 2.4 estão definidos os modelos referentes à arquitectura que o suporte a oferecer deve conter.
- Em 2.5 será enunciada uma proposta de um modelo de persistência.
- Em 2.6 será definido o modelo de programação.

O terceiro capítulo pretenderá descrever o protótipo desenvolvido que ilustra a concretização dos modelos definidos no segundo capítulo. Está organizado nas seguintes secções:

- Em 3.1 será apresentada a plataforma base para o desenvolvimento, isto é FEEDS.
- Em 3.2 apresentar-se-há as adaptações executadas a esta plataforma no âmbito do suporte de mobilidade e comunicações.
- Em 3.3 será descrito como foi implementado o suporte à captura de informação sensorial e contextual.
- Em 3.4 descrevemos a validação experimental do protótipo.

O capítulo 4 define o caso de estudo, estando organizado da seguinte forma:

- Em 4.1 descreve-se a aplicação contextual TouristHelper, uma aplicação desenvolvida para telemóveis Android.

- Em 4.2 apresenta-se em mais detalhe a implementação da aplicação contextual e faz-se o mapeamento para os modelos e protótipo.
- Em 4.3 definimos TourismTop, aplicação que exemplifica uma aplicação incentivo, pretendendo ilustrar como se concretiza o modelo de incentivos via o suporte fornecido.
- Em 4.4 alertamos para os problemas ao nível de segurança existentes e não abordados nesta dissertação.
- Em 4.5 indicamos, numa perspectiva qualitativa, as conclusões obtidas através deste caso de estudo.

O capítulo 5 servirá para indicar o trabalho relacionado, estando dividido segundo duas grandes secções: Sistemas sensoriais desenvolvidos com um objectivo específico em 5.1 e Plataformas em 5.2. O último capítulo apresenta as conclusões obtidas do trabalho desenvolvido, e as ideias quanto ao trabalho futuro.

2. Modelos

2.1 Aplicações Modelo (*Participatory Sensing*)

Uma aplicação *Participatory Sensing* pode ser caracterizada por ser uma aplicação móvel e distribuída, tirando assim partido da cobertura espaço-temporal dos seus utilizadores. A aplicação colecciona os dados através dos sensores presentes nos dispositivos móveis que viajam junto dos seus utilizadores, podendo esta captura ser explícita, através da interacção com o utilizador, ou implícita, capturando oportunamente os dados. O processamento destes dados, através do cruzamento de várias capturas feitas por diferentes utilizadores, permite a construção de informação bastante rica. Por exemplo permite a concepção de um sistema de detecção situações anómalas de trânsito, produzindo os resultados através do cruzamento de dados vindos do GPS e acelerómetro[23].

A criação destas aplicações desperta alguma curiosidade, tal a riqueza dos dados obtidos, bem como a variedade de tipos de aplicações se podem desenvolver neste espectro.

O grande factor motivador ao construir uma aplicação deste tipo é o facto de ser possível cobrir, de uma forma mais ou menos simples, uma grande área espacial. Utilizando dispositivos inteligentes com capacidade de recolha de informação contextual, podemos obter dados que, com algum processamento, poderão dar azo a ideias interessantes, de tal forma que se possa produzir aplicações de utilidade tanto para um só utilizador como para uma comunidade.

Além do exemplo anterior, outro poderá ser ilustrado através do exemplo de uma aplicação que dê apoio a surfistas, na qual se pode consultar a situação das ondas nas praias em redor de cada utilizador. Os dados são capturados através de outros utilizadores que decidiram publicar essa informação para todos os interessados. Revisitando o exemplo de uma aplicação que detecta situações anómalas de trânsito, os utilizadores utilizariam o seu dispositivo móvel para trocar dados provenientes dos sensores, como GPS e acelerómetro, informando acerca de situações de congestionamento de trânsito detectadas em determinada localização.

A forma de captura dos dados contextuais não necessita exactamente de ser um sensor presente no hardware. Apesar de em muitos casos isto ser verdade, muita da informação obtida de apenas um sensor será pouco relevante e sem grande significado semântico, se considerada isoladamente. Assim, além do caso em que a informação é capturada por apenas um sensor no hardware, como pode acontecer com o termómetro ou GPS, existirão mais três "tipos de sensores" ditos sensores virtuais.

Estes sensores virtuais podem, por um lado, ser feitos à custa do cruzamento de diferentes sensores *hardware*, como acontece com um sensor que capture os Hotspots na vizinhança através do relacionar da informação obtida pelo GPS com os sinais WiFi capturados pelo dispositivo móvel. Por outro, extrapolando um pouco e levando este pensamento mais além, o próprio utilizador pode ser um sensor, através da submissão de dados capturados por si, como uma fotografia, ou então através da submissão de informação textual produzida por si. Por fim,

o processamento estatístico da informação capturada por qualquer uma das hipóteses enunciadas anteriormente pode ter um valor interessante no desenvolvimento de uma aplicação deste género.

Todo este volume de dados tem um valor não só rico para cada utilizador, mas poderá ter um valor também significativo para uma comunidade. Relembrando o exemplo da aplicação que detecta trânsito, os dados para uma entidade singular significarão apenas que uma situação de trânsito é detectada, fazendo-lhe proveito para se desviar dessa localização onde a detecção foi feita, no entanto, estes dados obtidos podem alimentar uma comunidade que investiga padrões de trânsito, de forma a desenvolver uma aplicação que calcule rotas alternativas retirando partido desses padrões observados.

2.1.1 Características fundamentais de uma aplicação *participatory sensing*

Fazendo uma análise mais cuidada a este tipo de aplicações, podemos identificar um grupo de características essenciais, que levantam alguns problemas e lançam alguns desafios.

Informação sensorial

A informação capturada é um ponto importante de qualquer aplicação *participatory sensing*, visto sem a existência de dados a aplicação não pode existir. Assim faz sentido identificar e diferenciar dois conceitos a nível de informação que podem ocorrer nestas aplicações. A informação capturada em bruto é designada por informação sensorial, isto é, qualquer dado capturado por um sensor presente no *hardware* é considerada informação sensorial.

Informação contextual

Em contraponto com a definição anterior, definimos informação contextual como a informação capturada pelos sensores que possa ter um qualquer significado semântico para uma aplicação *participatory sensing*. A esta informação associamos geralmente o caso de ter sido obtida por um sensor virtual.

Ubiquidade

Uma aplicação deste género geralmente tem como característica estar virtualmente em todo o lado. Isto é, fazendo uso dos dispositivos móveis dos seus utilizadores, a cobertura espacial que estes dispositivos possibilitam atingir é ampla, podendo aceder aos locais mais remotos e capturar dados de todos esses locais.

Agregação e Processamento

A informação capturada pelos sensores pode não ser semanticamente interessante para um determinado tipo de aplicação, sendo necessário que esta seja processada, seja através do cruzamento de informação proveniente de vários sensores, vinda de diferentes locais, seja através de processamento estatístico. Não é indicado que este tipo de operação seja executada num dispositivo móvel, visto a sua computação necessitar de comunicações com entidades remotas ou poder ocupar demasiado tempo e recursos desse mesmo dispositivo.

Comunidade

Nas aplicações do ramo a tratar, a informação em troca é capturada pelos seus utilizadores, sendo que quanto mais informação produzida/capturada existir, melhor e mais ricas serão as aplicações. Assim, associada a uma aplicação *participatory sensing* surge a ideia de uma comunidade, comunidade essa que é composta pelos utilizadores dessa aplicação.

Comunicações

Espera-se que informação produzida pela comunidade que suporta uma aplicação tenha um grande volume. Sendo esta uma aplicação distribuída estes dados terão de ser partilhados e difundidos por todos os utilizadores do grupo envolvente. Neste campo, levanta-se o problema de como são feitas as comunicações entre as diferentes entidades. Existem várias aproximações que se podem ter em conta em relação à forma como atacar o problema, utilizando por exemplo um modelo *publish/subscribe* ou outro modelo do estilo *request/reply*. Os diferentes modelos poderão até coexistir.

Persistência

Certas aplicações precisam de agregar e armazenar a informação capturada, de forma a que no futuro lhe possam aceder e dar algum sentido que no presente possa não existir. Assim, torna-se necessário ter alguma forma de persistência dos dados, de forma a manter esse histórico. O facto desta informação poder vir a ser acedida por bastantes entidades e o volume da informação, transformam o dispositivo móvel pouco adequado à persistência destes dados, pelo que este problema deverá ser resolvido de uma outra forma, nomeadamente em algum tipo de infra-estrutura com grande capacidade de armazenamento. O problema é enunciado aqui mas não será tratado em especial nesta dissertação, destinando-se a trabalho futuro.

Segurança

O facto de a contribuição e captura da informação contextual ser executada pelos utilizadores da aplicação, levanta problemas ao nível de segurança. Este problemas são a nível da privacidade e fiabilidade dos dados. O primeiro caso está relacionado com a questão de ao se capturar dados, se poder expor o utilizador, como acontece no envio de dados relativos à sua posição GPS. O segundo caso existe com o facto de ser possível a um utilizador injectar informação falsa ou imprecisa no sistema. Ambos os casos têm relevância no desenho de uma aplicação deste género visto que a falta de privacidade poderá levar os utilizadores à não contribuição de nova informação e a informação falsa e imprecisa poder ter implicações quanto a satisfação da comunidade de utilizadores que usa a aplicação. Em ambos os casos, a relevância desta questão é significativa visto as consequências subsequentes que podem levar à não sobrevivência de uma aplicação deste género. Contudo, este problema não será tratado nesta dissertação em virtude do tempo disponível, reservando-se para trabalho futuro.

Cooperação

Uma aplicação pode produzir dados que poderão ser significativos para outra. Daqui

surge a ideia de que a informação pode ser importante para não só uma aplicação, mas para um grupo de aplicações que podem ser independentes umas das outras. Assim, a partilha desta informação deve ser possível entre elas. No entanto, a partilha desta informação levanta o problema de como ela é executada, se esta é explicitamente feita através das aplicações, se implícita através de um substrato de comunicação comum a todas. Parece-nos que a forma mais indicada segue o segundo caso, visto no primeiro ser necessário o desenho de protocolos entre as diferentes aplicações de forma a que essa partilha seja possível, tornando o processo de desenvolvimento e partilha bastante dispendioso.

Recursos

Os dispositivos móveis evoluíram ao longo dos anos os seus recursos, essencialmente no campo do armazenamento e capacidade de processamento. No entanto, a sua capacidade energética é ainda bastante reduzida. Atendendo a que o volume de dados produzido será previsível de ser elevado e a aplicação ser distribuída, levará a que seja necessário a existência de algumas comunicações entre as suas diversas componentes. As comunicações são no entanto caras, dando mais significado a este problema energético. Assim, as limitações energéticas terão um papel influente no desenho de uma aplicação deste género, quer no que diz respeito ao desenho da própria aplicação, quer no modelo de comunicações, quer na própria arquitectura subjacente, propagando as limitações para todos os outros recursos.

2.1.2 Modelo da Aplicação

Uma aplicação pode ser vista a dois níveis. A um nível mais abstracto, uma aplicação é um serviço útil oferecido aos seus utilizadores. O serviço pode ser visto como o conjunto de informação contextual contribuída pela comunidade de todos os seus utilizadores, bem como pela partilha de dados provenientes de outras aplicações.

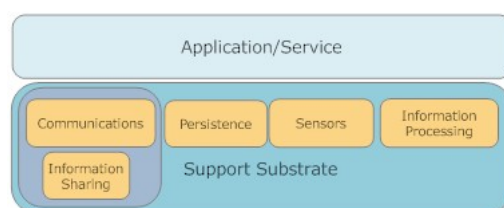


Figura 2.1 Modelo geral de uma aplicação *Participatory Sensing*

Num nível mais concreto, uma aplicação *Participatory Sensing* será uma aplicação distribuída, contendo várias instâncias semelhantes a correr em simultâneo, normalmente uma por utilizador.

As limitações dos dispositivos móveis, em termos energéticos, propagam-se para todos os outros recursos, levando a que se aprecie evitar todo o tipo de computação, comunicação e

armazenamento que possa ocorrer nos dispositivos móveis.

Assim, acreditamos que cada instância de uma aplicação que se enquadre neste paradigma, deva ser composta por duas componentes, uma componente móvel e uma componente fixa.

A componente móvel está focada na capacidade de recolha de informação contextual, junto com o seu alto teor de mobilidade que lhe permite uma grande cobertura espacial, e ainda na interação com o utilizador.

A componente fixa permitirá ultrapassar as limitações da componente móvel. Esta componente considera-se estar bem conectada, com capacidade de armazenamento e processamento suficientes para lidar com o volume de informação capturado.

As aplicações deste género são distribuídas, levando a que seja necessário comunicar entre entidades. Esta partilha de informação leva a crer que deverá existir uma infra-estrutura comum a todas as instâncias, com o intuito de suportar estas comunicações de uma forma organizada e eficiente. Este modelo combina ubiquidade com pervasividade, na medida em que os dispositivos estão virtualmente em todo o lado tal como a computação e processamento de informação.

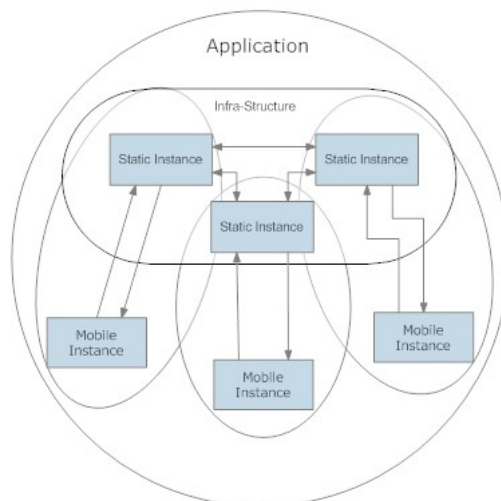


Figura 2.2 Modelo concreto de uma aplicação *Participatory Sensing*

2.1.2.1 Componente Móvel da Aplicação

A componente móvel de uma aplicação será a parte de uma instância com capacidade móvel, capacidade sensorial e interação com o utilizador, com vista a que seja possível a recolha de informação contextual que irá alimentar a aplicação.

O modo de interação com o utilizador, com vista a que este usufrua do serviço oferecido pela aplicação, ocorre também nesta componente, permitindo o acesso ao serviço em qualquer lugar a qualquer momento.

Em suma, a componente móvel serve para capturar e produzir informação contextual, através dos seus sensores e interação com o utilizador explorando a mobilidade deste, e ainda a

função inversa, o consumo do serviço oferecido ao utilizador por parte da aplicação.

2.1.2.2 Componente Fixa da Aplicação

As componentes móveis têm por objectivo a recolha dos dados contextuais. Sendo uma aplicação distribuída, estes dados terão um volume bastante alto, ainda para mais tendo em conta que numa aplicação *Participatory Sensing* estes dados serão obtidos tipicamente por uma comunidade.

As limitações dos dispositivos móveis inviabilizam a estruturação de uma rede para partilha desses mesmos dados apenas com o conjunto de componentes móveis.

A componente fixa tem como objectivo retirar carga e ultrapassar as limitações das componentes móveis de cada instância da aplicação, contribuindo com os recursos em falta nos dispositivos móveis. Assim, para cada componente móvel, existirá uma componente fixa que tratará de aumentar os recursos dessa instância.

A componente fixa deverá estar bem conectada e ter capacidade de processamento e armazenamento de forma a lidar com os dados produzidos pela sua componente móvel, sem estar limitada a nível energético. A partilha destes dados com as outras instâncias será parte da sua responsabilidade.

Propõem-se que as várias componentes fixas se organizem, de forma a que as comunicações sejam eficientes, e que este suporte comum que ambas devem ter, permita a partilha com as outras aplicações *Participatory Sensing*.

Assim, a surge a noção de uma infra-estrutura fixa partilhada e formada pelas várias componentes fixas de cada instância de todas as aplicações, de tal forma que esta ofereça capacidade armazenamento, processamento e difusão dos dados capturados para todos os participantes desta infra-estrutura.

2.2 Modelo de Incentivos

Sendo a aplicação um serviço dado pelo conjunto de instâncias móveis e fixas, através da contribuição de dados oferecidos pelos participantes, existe a preocupação de que todos os utilizadores contribuam. À semelhança do mundo P2P, como no caso do *Bittorrent* ou outra rede do mesmo género, caso os seus participantes não participem activamente na distribuição de novos conteúdos, a aplicação irá perder valor, interesse e possivelmente não sobreviverá.

Neste caso concreto existe o problema acrescido de que a contribuição dos utilizadores é indispensável para que a aplicação funcione. Sendo essa partilha feita através dos dispositivos móveis de cada utilizador, os recursos desses dispositivos serão usados, provocando um consumo superior energético desses mesmos. Assim, levanta-se o problema de como fazer um utilizador contribuir, gastando os seus recursos e diminuindo a autonomia diária do seu dispositivo.

Para tal é definido um modelo de incentivos. Neste modelo, define-se um conjunto de três

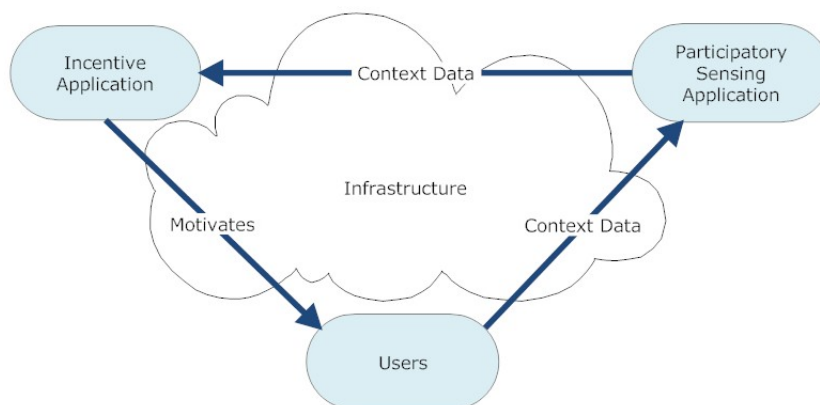


Figura 2.3 Esquema que ilustra o modelo de incentivos.

entidades, que relacionadas entre si pretendem que se resolva o problema da contribuição. São elas: Aplicação *Participatory Sensing*, Aplicação Incentivo (*Incentive Application*), e utilizadores *Users*).

Aplicações Incentivo

As aplicações incentivo, são aplicações que têm uma componente de incentivo que levem os utilizadores a participar com a captura de informação contextual. A aplicação poderá fazer uso da informação contextual capturada pela aplicação *participatory sensing* para enriquecer o serviço que oferece. De notar que esta aplicação poderá ser dos dois tipos, tanto uma aplicação incentivo para outras, como necessitar de incentivos de uma outra aplicação disponibilizada para o efeito.

Aplicações *Participatory Sensing*

As aplicações *Participatory Sensing*, são aplicações que necessitam de informação contextual para poderem funcionar correctamente. Relacionando esta entidade com o modelo das aplicações podemos dizer que são o serviço a oferecer pelo conjunto de todas as instâncias. Algumas destas aplicações não têm um aliciante suficiente para que o seu "*bootstrap*" possa ocorrer, isto é, sem informação contextual inicial a aplicação não poderá arrancar, sendo assim necessário recorrer a uma aplicação incentivo que leve os utilizadores a produzir essa informação desde o primeiro momento.

Utilizadores

Esta entidade tem por objectivo capturar informação contextual. A forma como essa informação contextual é capturada é fazendo uso da componente móvel da aplicação *participatory sensing*.

Processo do modelo

As aplicações *participatory sensing* necessitam de informação contextual com vista a poderem funcionar correctamente e sobreviver, informação essa que é capturada pelos seus utilizadores. Para que exista algo que leve os utilizadores a contribuir com os seus recursos na captura destes dados é necessária uma aplicação incentivo que possa motivar estes utilizadores a contribuir com nova informação, alimentando assim as aplicações *participatory sensing* com esses novos dados, que por sua vez irá fornecê-los a essa aplicação incentivo que fará o melhor uso que puder deles.

Por exemplo, uma aplicação web que mostre um top de contribuidores, poderá estimular suficientemente os utilizadores de uma determinada aplicação *participatory sensing* a contribuir com informação contextual que enriqueça essa mesma aplicação.

2.3 Modelo de Comunicações

As aplicações *participatory sensing* são aplicações distribuídas e por definição necessitam de comunicar. Em concreto oferecem um serviço que faz uso da informação de todas as instâncias da aplicação, levando assim à necessidade de que exista alguma forma de comunicação entre elas, de forma a que esta informação possa ser partilhada.

Essencialmente o problema das comunicações deve ser discutido à volta de dois pontos centrais. O primeiro diz respeito ao modelo de interacção entre as várias componentes. O segundo deve ter a ver com a forma como as mensagens devem circular entre as componentes, isto é, como deve o endereçamento ser realizado.

2.3.1 Qualidade de Serviço

Pretende-se suportar o maior número possível de tipos de aplicações que usem e recolham informação contextual. Sendo que a heterogeneidade em termos de recursos e mensagens que trocam entre componentes é alta, será benéfico oferecer um suporte de diferentes qualidades de serviço.

A qualidade de serviço é afectada pela forma como as mensagens são enviadas, através de que tecnologia são enviadas e qual a sua prioridade nesse mesmo envio.

Por esta razão são mapeadas três qualidades de serviço que julgamos satisfazer a grande maioria dos casos.

Critical Esta qualidade de serviço representa a qualidade de serviço a oferecer a aplicações críticas. As mensagens são enviadas assim que possível. Por exemplo, os serviços internos ao bom funcionamento da plataforma de serviço usam esta qualidade de serviço.

Realtime Nesta qualidade de serviço, pretende-se mapear um serviço que possibilite satisfazer as condições de uma aplicação *realtime*. Assim, estas mensagens têm prioridade no envio sobre todas as outras com a excepção das mensagens críticas.

Deferred Os canais que usam aplicações contextuais que tenham a existência de um critério mais relaxado em termos de tempo de entrega das suas mensagens, devem fazer uso desta qualidade de serviço.

Estes critérios baseiam-se essencialmente em termos de latência. Poderão existir outros atributos associados à qualidade de serviço, como persistência e fiabilidade ou ordenação das mensagens. Acreditamos no entanto que será à roda da latência que a qualidade de serviço das aplicações *participatory sensing* se irá focar.

2.3.2 Modelo de Interação

No modelo definido para as aplicações *participatory sensing*, estas estão divididas em duas componentes, uma componente móvel e uma fixa. A aplicação é o conjunto de todas as instâncias compostas por estas duas componentes necessitando de partilhar a informação contextual produzida por cada instância. Assim, torna-se necessário definir como esta partilha é feita à custa de comunicações entre as várias componentes, isto é, é necessário definir como se processa o diálogo em termos de modelo.

Existem diferentes formas de atacar o problema, podendo ser feito por exemplo através de um tradicional *request/reply* ou então através de um modelo *publish/subscribe*.

Este modelo deve ser definido tendo em conta essencialmente o padrão de cada utilizador em relação a cada aplicação. Assim é possível que um modelo se adequa a um determinado tipo de aplicação, e outro se enquadre mais noutra situação.

Ao imaginarmos uma aplicação que detecte situações de trânsito será espectável que um utilizador arranque a aplicação e fique à espera que avisos de trânsito lhe cheguem, de acordo com os seus interesses. Neste caso o mais adequado será um modelo *publish/subscribe*. Noutro caso, tendo por exemplo uma aplicação que permita a consulta do estado meteorológico de uma determinada zona, o modelo *request/reply* será talvez mais indicado.

A coexistência de ambos os modelos, idealmente, deverá ser possível num suporte a nível de comunicações oferecido a aplicações deste género. Apesar disto ser verdade, acreditamos que a maioria das aplicações deverá cair no modelo *publish/subscribe*. Esta opção tem benefícios que vão mais longe que a singular interação do utilizador com a aplicação. As comunicações serão em menor número, visto o modelo *publish/subscribe* ser um modelo reactivo, não sendo necessário executar múltiplas pesquisas até a informação ficar disponível, o que trará benefícios a nível energético.

Desta forma, defendemos que o modelo prioritário do suporte a oferecer às aplicações deverá ser o modelo *publish/subscribe* e que, idealmente, o modelo *request/reply* seja oferecido. O modelo *publish/subscribe* pode oferecer este serviço à custa da inversão dos papéis de produtor de informação e consumidor, isto é, se por um lado quem faz a pesquisa é um consumidor da informação que existir resultante dessa pesquisa, o próprio pedido de pesquisa poder ser algo que possa ser consumido por uma outra entidade.

2.3.3 Canais

Pensando num modelo *publish/subscribe*, a noção que temos é a de um canal através do qual inserimos mensagens, e através do qual estas chegam aos seus interessados. Este canal oferece a abstracção que leva um fluxo de mensagens semanticamente semelhantes até ao destino, ao qual se dá um nome persistente, fazendo a filtragem baseada nos conteúdos das mensagens pelo meio do processo de transporte, segundo uma determinada qualidade de serviço comum a esse fluxo de dados.

Em termos de qualidade de serviço, como enunciado previamente, existirão várias no suporte que se pretende oferecer. Desta forma, esta qualidade de serviço é modelada nesta abstracção, isto é, para várias qualidades de serviço, existirão diferentes canais que as suportarão. Esta qualidade de serviço é então distinguida nos canais sobre a forma como estes processam a informação e aplicam as suas regras de endereçamento, bem como o tipo de tecnologia de nível de transporte que usam.

Assim, cada aplicação poderá ter a qualidade de serviço indicada usando os canais próprios para o efeito.

Por uma questão de organização e optimização, os canais estão divididos em vários escopos. Essencialmente, esta separação está focada no tipo de comunicação a que se destinam. Esta optimização é orientada tanto em termos de latência como a nível energético, este último caso é detalhado de uma forma mais aprofundada no caso Ad-Hoc.

Assim são definidos três escopos de canais: Local, Ad-Hoc, Globais.

2.3.3.1 Escopo Local

Dentro deste escopo enquadram-se os canais que publicam a informação para ser consumida localmente pelo nó¹. Assim sendo, dentro deste escopo, os canais servirão exclusivamente para acesso a recursos do nó, não sendo injectada qualquer informação na infra-estrutura. Os sensores presentes nos dispositivos móveis são modelados segundo esta abstracção, podendo aceder a um sensor tal como se acede a um qualquer canal de comunicação.

2.3.3.2 Escopo Ad-Hoc

Certos tipos de aplicação podem fazer uso de comunicações ad-hoc com vista a cooperarem entre si de alguma forma ou a aumentarem os seus recursos. Com este intuito definimos um escopo ad-hoc, em que existe comunicação entre dois nós que se situem bastante próximos. Exemplos desta natureza podem ser vistos em aplicações que usem esquemas de *Data Muling* para propagação de informação.

Desta forma, este escopo limita-se a comunicação entre componentes móveis. A comunicação deve então ser feita idealmente com tecnologias que sejam baratas em termos energéticos,

¹No caso corrente um nó representa uma instância da aplicação

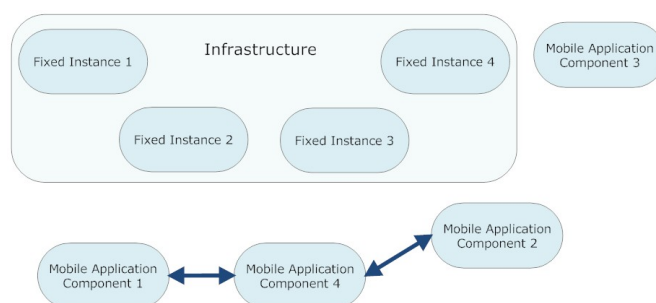


Figura 2.4 Fluxo de mensagens possível num canal do escopo ad-hoc

com vista a que não sejam consumidos muitos recursos do dispositivo móvel. Idealmente, os canais segundo este escopo devem tirar partido da tecnologia Bluetooth, visto ser a comunicação mais barata em termos do par de atributos descoberta/transferência de dados.

2.3.3.3 Escopo Global

Dentro deste escopo caem os canais que possivelmente publicam as mensagens que atingem toda a rede móvel e fixa. Numa forma de âmbito mais geral, poderá dizer-se que potencialmente inundam a rede. Porém, esta comunicação estará sujeita à filtragem inerente ao modelo de encaminhamento com base nos conteúdos das mensagens.

A imagem 2.5 ilustra este fluxo.

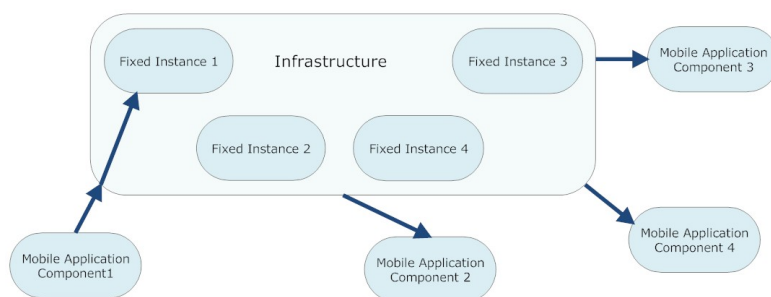


Figura 2.5 Fluxo de mensagens possível num canal do escopo global

Estes fluxos representam o que cada canal neste escopo pode atingir, isto é, um canal pode restringir a comunicação apenas a certas componentes que sejam de interesse, como por exemplo para estabelecer um canal entre componente fixa e componente móvel da aplicação, sendo que esta restrição deve ser feita à custa dos filtros submetidos por cada entidade ao canal.

Dentro deste escopo, por questões de optimização, pode ser definido um sub-escopo, neste caso denominado de Escopo Vizinhança.

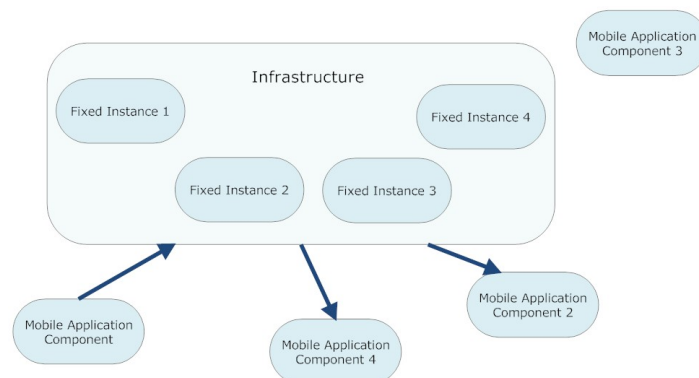


Figura 2.6 Fluxo de mensagens possível num canal do escopo vizinhança

O escopo vizinhança é uma restrição mais predominante aos canais de escopo global. Dentro deste apenas existe troca de informação entre os dispositivos móveis que se encontrem na vizinhança do emissor. A figura a seguinte ilustra o fluxo de mensagens que ocorrem aquando do envio de mensagens segundo um canal deste escopo.

Mais uma vez, apesar de num escopo vizinhança as mensagens poderem fluir até todos os dispositivos móveis que se situem na vizinhança do emissor, não é necessário que assim ocorra. O endereçamento das mensagens poderá ser restringido segundo os filtros submetidos.

2.3.4 Encaminhamento e Filtragem

Admite-se que a informação em fluxo nas aplicações *participatory sensing* tenha uma forte correlação com a zona geográfica em que foi capturada. Esta correlação é explícita no caso de que a informação contextual está sempre associada a uma localização para fazer algum sentido. Atendendo ao padrão de utilização das aplicações *participatory sensing*, a informação contextual será previsivelmente consultada por utilizadores nas suas proximidades. Tomando o exemplo tanto de uma aplicação de apoio a surfistas como de outra que permita a detecção situações anómalas de trânsito, em ambos os casos, as consultas deverão seguir esta ideia, sendo que no primeiro caso os surfistas deverão estar interessados nas praias próximas de onde estão, e no segundo em situações de trânsito nas suas redondezas.

Por outro lado, os utilizadores de uma aplicação deste género não estarão interessados em toda a informação produzida pela aplicação. Estes devem especificar os seus interesses e apenas a informação que se ajustar a esse padrão lhes deverá ser encaminhada.

Assim, o encaminhamento deverá ser efectuado tendo em conta o conteúdo e os interesses do utilizadores, utilizando em simultâneo critérios geográficos de forma a que a sua eficiência seja superior.

2.3.5 Mobilidade e Desconexão

Uma componente móvel permite uma grande cobertura espacial, uma vantagem indubitável para a construção de um sistema deste tipo. No entanto, esta vantagem acarreta um acréscimo de complexidade no que às comunicações diz respeito.

Esta complexidade deve-se exactamente ao facto destas componentes poderem ficar momentaneamente desconectadas do sistema, enquanto estas mudam por exemplo de AP, o que pode levar à perda de dados num intervalo de tempo.

Pretendendo não perder dados devido à desconexão, torna-se necessário a existência de alguma entidade que fique responsável pelos dados em trânsito com destino à componente móvel. Essa entidade poderá ser definida como uma *Homebase*. Esta entidade será bem conhecida pela componente móvel, sendo por isso fácil a comunicação com esta, tendo ainda como requisito capacidade de armazenamento.

Existe ainda o problema de saber quando a componente móvel se desliga do sistema. Este problema dá-se devido ao facto de uma componente móvel poder ficar desconectada largos períodos de tempo, apenas por falta de conexão. No entanto podem acontecer duas situações nesse período: O utilizador desliga a sua componente móvel com a intenção de sair do sistema, ou então o mesmo utilizador espera que essa desconexão seja apenas temporária, continuando com a aplicação a correr e esperando a recepção de eventos.

Como distinguir as duas situações pode ser problemático, no entanto acreditamos que sempre que possível, ao sair do sistema a componente móvel deve informar a sua intenção. Em caso de o utilizador sair do sistema quando a componente móvel está em período de desconexão, o sistema deverá fazer uso de um "*timeout*", suficientemente grande, a partir do qual se considera que a componente móvel se desconectou.

Por outro lado, acreditamos que o acesso à informação contextual será executado maioritariamente por utilizadores que se situem nas proximidades da origem dessa informação. Em virtude da eficiência, fará sentido definir uma outra entidade que seja responsável pela interacção de uma componente móvel com todas as outras do sistema e que esteja ao mesmo tempo situada numa zona próxima de uma componente móvel. Esta componente será denominada de *Proxy*. Esta componente será discutida de uma forma mais aprofundada na secção seguinte, respeitante à arquitectura da solução.

Com estas duas entidades podemos resolver duas questões que dizem respeito às componentes móveis: como é feito o *binding* de uma componente móvel ao sistema (aplicação *Participatory Sensing* distribuída), e como suportar a desconexão destas mesmas entidades tendo a intenção de não perder dados.

2.3.5.1 Serviço de *Binding* a um *Proxy*

Definimos que para uma eficiência das comunicações, uma componente móvel necessita de estar associada ao *proxy* mais próximo. No entanto, por a componente se poder deslocar, o *proxy* mais próximo nem sempre será o mesmo, que faz com que seja necessário ter um serviço que nos permita obter o *proxy* mais próximo.

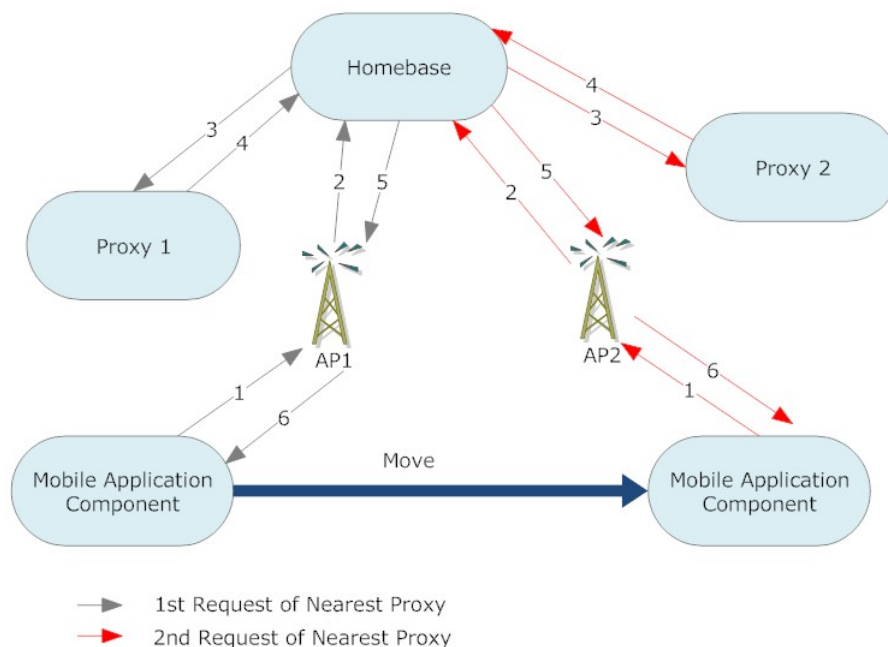


Figura 2.7 Serviço de Binding a um Proxy

A comunicação de entre uma componente móvel e o sistema será feito, normalmente, através de um AP. Os APs têm normalmente um alcance reduzido, pelo que enquanto continuarmos ligados a um não faz sentido verificar se temos um *proxy* mais próximo do que aquele a que estamos ligados actualmente.

Desta forma o pedido para associarmos uma componente móvel ao *proxy* que se situe mais próximo, tem o requisito de acontecer apenas quando existir acesso à infra-estrutura via um AP, podendo ocorrer em duas situações. A primeira acontece quando não existe associação com nenhum *proxy*, isto é, no momento de entrada ou reentrada no sistema. O segundo caso acontece quando existe troca de AP.

2.3.5.2 Serviço de Roaming

A semântica de comunicações pretendida é evitar que dados sejam perdidos por desconexão de uma componente móvel. Será normal que os *proxys* tenham um certo limite de capacidade, limite esse que seja definido à custa quer de um possível raio máximo de abrangência, quer de utilizadores.

Assim, é necessário definir o que acontece quando uma componente móvel não tem conexão com o sistema.

Neste caso é necessário que as mensagens que se destinem à componente móvel sejam enviadas através de GPRS para esta, ou então guardadas até futura conexão da componente móvel. Assim, temos como requisito para entidade que irá substituir a componente móvel a

necessidade de saber como aceder à esta sempre que possível, bem como a capacidade para armazenar esses dados.

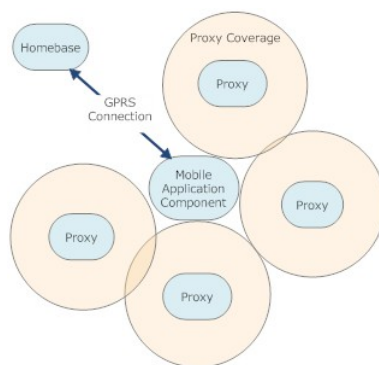


Figura 2.8 Serviço de suporte a Desconexão

Com este intuito, é definida uma *homebase* que tem como requisito saber aceder à instância móvel em qualquer momento. Em caso de possibilidade de conexão GPRS, a *homebase* deverá direccionar toda a informação para a componente móvel servindo como se de um *proxy* se tratasse. Na impossibilidade de contacto com a entidade móvel, armazenará as mensagens até ao próximo pedido de associação a um *proxy* mais próximo, situação na qual entregará todos os dados com destino a essa instância móvel.

Com estes dois serviços, é possível garantir que uma componente móvel apenas estará impossibilitada de receber ou enviar dados para o sistema apenas na situação de não existir uma conexão disponível, como também será possível garantir que todas as mensagens endereçadas a uma componente chegarão a seu destino.

2.4 Modelo da Arquitectura

Um aplicação que se enquadre no âmbito de *Participatory Sensing* é, como já indicado uma aplicação distribuída. Segundo o modelo definido para as aplicações cada instância desta aplicação está definida em duas componentes, móvel e fixa. As componentes fixas são componentes que estão bem conectadas, gozando de poder de armazenamento, processamento, comunicações eficientes e sem limitações de energia. Assim, a organização e estruturação das componentes fixas formará uma infra-estrutura, cujo suporte irá oferecer facilidade de comunicação, capacidade de processamento, armazenamento e colaboração entre as entidades e aplicações *participatory sensing*.

Assim, deixamos de ter a noção de um conjunto de pares "componente móvel - componente fixa" que comunicam entre si de uma forma isolada, para ter uma infra-estrutura na qual as componentes fixas das instâncias da aplicação estão devidamente organizadas, oferecendo os recursos indicados, mas de maneira eficiente e estruturada.

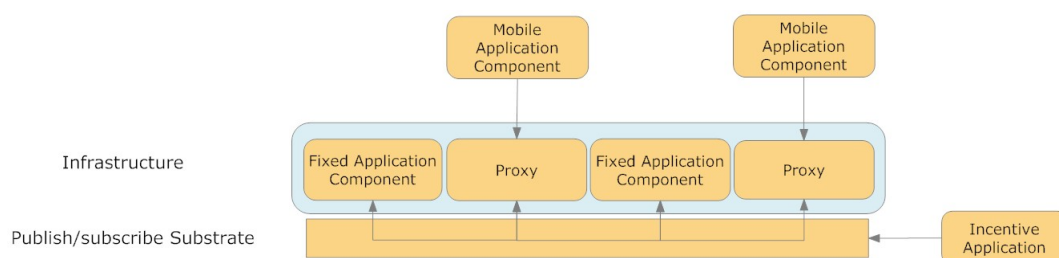


Figura 2.9 Modelo da Arquitectura do Sistema

O modelo geral da arquitectura do sistema será então composto por uma infra-estrutura, que tem como entidades participantes a componente fixa da aplicação e uma outra denominada de *proxy*, pelas componentes móveis de cada instância de uma aplicação e aplicações incentivos.

As componentes móveis têm como função a recolha e produção de informação, bem como a interação com o utilizador. A infra-estrutura, o seu processamento, a sua gestão, armazenamento e endereçamento, tendo como o ponto de ligação com o mundo móvel a entidade *proxy*. Por sua vez, as aplicações incentivo existirão para permitir que o modelo de incentivos se possa processar.

Definimos ainda um substrato *publish/subscribe* como forma de endereçamento, baseado no conteúdo das mensagens, entre as várias entidades. Assim, as comunicações estão sujeitas a filtragem, impondo relevância às mensagens recebidas, existindo um desacoplamento entre produtores e consumidores. Este modelo é ainda assíncrono, sendo útil no uso deste tipo de dispositivos visto, ao contrário de um modelo bloqueante, o utilizador poder interagir sempre com o seu dispositivo.

2.4.1 Infra-Estrutura

A infra-estrutura pretende, através de um suporte comum a todas as componentes, organizar as componentes fixas de cada instância de cada aplicação e os *proxys* numa rede sobreposta, de forma a que os requisitos de cada aplicação sejam satisfeitos de uma forma eficiente.

A infra-estrutura será responsável por processar a informação obtida pelas componentes móveis da aplicação, armazenar esta e ainda endereça-la ao destinatário da forma mais eficiente possível.

A informação contextual de cada aplicação deverá fluir com maior frequência nas imediações de onde foi capturada. Por exemplo numa aplicação que detecte trânsito, os utilizadores desta aplicação deverão estar interessados nas informações de trânsito que se situem perto de onde se encontram, se o caso for o de uma aplicação que de suporte a surfistas também será provável que estes estejam interessados em praias perto da sua localização. A existência de uma entidade *proxy* trará melhorias ao nível da eficiência e distribuição de carga.

Assim, estruturação das entidades na infra-estrutura permite que a carga seja distribuída, dividindo os componentes segundo a sua função específica. Os *proxys* deverão gerir as componentes móveis nas suas imediações, e as componentes fixas devem ocupar-se do processamento

dos dados das suas componentes móveis.

2.4.1.1 Componente Fixa da Aplicação

A componente fixa reside na infra-estrutura de forma a poder processar a informação obtida através da sua componente móvel. Desta forma, esta componente terá recursos ao nível de comunicações e autonomia energética que a componente móvel não possui. Todas as comunicações que não sejam indispensáveis de ser apresentadas ao utilizador, devem ser tratadas por esta componente em vez da sua componente móvel com vista a aumentar a eficiência do sistema e a reduzir o consumo de recursos e comunicações nos dispositivos móveis. Poderemos então indicar que esta componente se destina a preencher a insuficiência de recursos da componente móvel, possibilitando ao mesmo tempo beneficiar da cobertura espacial desta última.

2.4.1.2 *Proxy*

O *proxy* será a sub-componente da plataforma que permite a interacção com o mundo móvel. A sua principal responsabilidade é fazer a ponte entre infra-estrutura e dispositivos móveis.

O *proxy* é o responsável pela gestão dos dispositivos móveis que se situem nas suas redondezas. Esta ideia suscita que uma noção de uma operação *binding*, como referido previamente, entre os dispositivos móveis e a infra-estrutura, bem como uma operação que consiga gerir a desconexão destes, tenha a entidade *proxy* como um dos intervenientes principais.

2.4.2 Componente Móvel da Aplicação

A componente móvel da aplicação será a forma da aplicação interagir com os utilizadores do sistema. Possibilitará a captura e o envio de informação contextual para a infra-estrutura, onde será difundida pelos interessados. Em operação inversa, proporcionará a interacção do utilizador com a aplicação, de forma a que este explore o serviço que lhe é oferecido por esta última.

2.4.3 Componente Aplicação Incentivo

Segundo o modelo de incentivos existirá esta componente denominada de aplicação incentivo. Esta componente tem como missão promover a colaboração entre utilizadores. A sua interligação com as aplicações será feita através da infra-estrutura.

2.4.4 Substrato *Publish/Subscribe*

A necessidade de comunicação e o facto de as comunicações serem semelhantes entre as várias entidades, indicam que um suporte de comunicação comum a todas as entidades é uma ideia com razão de existir. Assim, e segundo o modelo de comunicações previamente descrito, definimos que deve existir um substrato de comunicações *publish/subscribe*, o qual ficará responsável por fornecer as abstracções dos vários canais e diferentes qualidades de serviço.

O substrato deverá permitir a interação entre toda a rede, de uma forma transparente tanto a nível de localização das entidades como de acesso. Assim, o substrato garante um desacoplamento entre as várias componentes, garantindo o endereçamento baseado na filtragem segundo o conteúdo das mensagens, provando-se a forma de interação entre as várias entidades.

2.5 Modelo de Persistência

Numa aplicação *participatory sensing*, fará sentido pensar que os dados adquiridos pelo utilizador e pela comunidade sejam necessários no futuro. Torna-se necessário oferecer algum serviço que possibilite armazenar em memória estável os conteúdos distribuídos por cada aplicação.

Este é um problema complexo, cuja solução específica cai fora do âmbito desta dissertação. No entanto, indicamos uma aproximação bastante abstracta de como oferecer um suporte de persistência a este tipo de aplicações e, de certo modo, ajudar a definir um caderno de encargos a uma futura solução dedicada especificamente ao problema.

Neste problema, existem questões que rondam a forma de como e onde é guardada a informação, qual o tipo de formatos em que a informação é guardada, bem como garantir que exista uma forma de acesso a essa informação e o grau de transparência do processo face ao seu custo.

Para tal será necessário que exista processamento da informação contextual capturada com vista a uma possível normalização que permita a compreensão e armazenamento dessa informação por um outro sistema. Por outra via, será necessário a tradução dos dados guardados no sistema relativo ao armazenamento da informação para dados que sejam legíveis pelas entidades participantes de uma aplicação *participatory sensing*

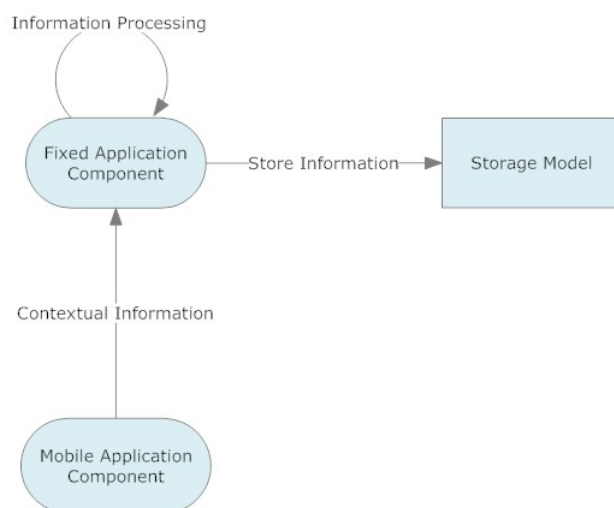


Figura 2.10 Modelo abstracto de armazenamento de informação produzida pelas aplicações

Assim, e fazendo referência às entidades definidas no modelo das aplicações, definimos um

modelo de persistência no qual existe processamento e transformação da informação contextual, com vista a interligar o sistema de persistência com a aplicação. Este processamento deverá ser executado pela componente fixa da aplicação visto esta ter recursos mais abundantes.

Tendo em conta o modelo de comunicações efectuado através de canais, podemos definir um modelo de persistência no qual a informação contextual é encaminhada, via substrato *publish/subscribe*, da componente móvel da aplicação, onde é recolhida, para a componente fixa. Nesta última componente esta informação é processada e normalizada para ser armazenada num local indicado.



Figura 2.11 Modelo abstracto de consulta de informação contextual armazenada

Por sua vez, para consulta desta informação, é "desenhada" uma *query* que é enviada à sua componente fixa, sendo esta *query* então redireccionada para o local onde está armazenada essa mesma informação.

Em sentido inverso vêm as respostas dadas pela entidade que armazena a informação contextual, sendo a normalização dessa informação para um formato legível à componente móvel efectuada pela componente fixa da aplicação.

2.6 Modelo de Programação

O modelo de programação usado é o mesmo modelo existente em FEEDS/DEEDS[5], e orientado ao paradigma *publish/subscribe*, com filtragem baseada no conteúdo das mensagens e cuja API é a indicada na tabela 2.1

O principal conceito deste modelo tem a ver com o canal de eventos activo. Este tem um papel de intermediário lógico entre os produtores e consumidores de informação, desacoplando as suas interações completamente.

Um canal activo representa um fluxo de mensagens agregadas segundo uma determinada lógica, normalmente relacionada com a semântica dos seus conteúdos ou uma determinada qualidade de serviço. Os canais, além da difusão filtrada, e da primitiva *publish*, possibilitam ainda a operação de enviar mensagens no sentido inverso, isto é, dos subscritores para os editores, operação essa denominada de *feedback*.

O encaminhamento baseado no conteúdo é uma das principais características da plataforma FEEDS/DEEDS reflectindo-se na forma de como são especificados os eventos. Um evento é assim composto por duas componentes: uma parte pública, denominada de *envelope*, acessível pelo substrato de encaminhamento; e a parte privada, denominada de *payload*, que diz respeito

ChannelDirectory ← directory ()
Channel ← clone (template, name,...)
Channel ← lookup (name, ...)
Receipt ← publish (envelope, payload)
Receipt ← feedback (receipt, envelope,payload)
void ← reRoute (receipt)
void ← subscribe (criteria, handback, handler)
void ← unsubscribe (handback)
void ← subscribeFeedback (criteria,handback, handler)
void ← unsubscribeFeedback (handback)
→ notify (receipt, envelope, payload)
→ notifyFeedback (receipt, envelope, payload)

Tabela 2.1 API disponibilizada ao programador

apenas à aplicação. A filtragem processa-se apenas sobre o *envelope* da mensagem, na forma de objectos específicos para essa função (*criteria*), decidindo se um dado evento deve ser entregue ou não ao subscritor.

Assim, com as operações disponíveis existem dois tipos habituais de interacção com a plataforma. A primeira aproximação é reactiva e processa-se como um modelo normal de *publish/-subscribe*, isto é, uma aplicação subscreve um canal e espera por eventos que venham a surgir; a segunda segue uma filosofia diferente, tirando partido da primitiva *feedback*, e interagindo de uma forma do estilo *request/reply*, publicando um evento que representa um pedido, obtendo a resposta em *feedback*.

3. Protótipo

Pretendeu-se desenvolver um suporte que facilitasse o desenvolvimento de aplicações *participatory sensing*, que oferecesse como principais requisitos abstrações a nível de comunicações, acesso simplificado a sensores, acesso a uma infraestrutura que permita a realização dos modelos definidos anteriormente, de tal forma que seja possível possibilitar uma qualidade de serviço adequada a uma qualquer aplicação.

Para tal, utiliza-se a plataforma FEEDS, à qual se acrescenta um determinado número de funcionalidades, nomeadamente, o suporte a mobilidade, segundo a concretização dos modelos definidos, e o acesso a sensores. Esta plataforma FEEDS é uma re-implementação de DEEDS[5] focada na infra-estrutura fixa.

Estes sensores são parte integrante de um dispositivo móvel, tendo a plataforma Android sido a escolhida para a concretização do cenário. A escolha da plataforma caiu neste sistema visto o desenvolvimento ser feito na linguagem Java, a mesma na qual FEEDS está implementada, e a API de acesso aos sensores ser bastante mais evoluída e simplificada que nas outras opções disponíveis como J2ME.

3.1 FEEDS

FEEDS é uma plataforma que pretende oferecer um serviço de disseminação de eventos, com filtragem baseada nos conteúdos, contendo propriedades de auto-regeneração e auto-organização.

O modelo de comunicações usado é um mecanismo semelhante ao *publish/subscribe* com a particularidade de que é possível executar uma opção de *feedback* na qual se contacta directamente o nó que produziu o evento recebido. Em ambos os casos é seguida uma filosofia de encaminhamento baseado no conteúdo.

A comunicação nesta plataforma é feita ao nível de uma abstracção canal. Cada canal tem a responsabilidade de enviar um fluxo de mensagens entre as várias entidades, existindo um processo de filtragem durante a fase de transporte das mensagens. Os fluxos de eventos são mensagens que são agrupadas segundo políticas de semântica/lógica da aplicação e/ou questões de QoS.

Estas QoS são definidas na forma de um pequeno módulo denominado de *template*, através da especificação das regras de endereçamento, que levam os eventos a percorrer percursos diferentes atingindo as entidades adequadas e consequentemente diferentes qualidades de serviço.

Cada canal é uma instanciação de um *template*, sendo esse canal sujeito às regras especificadas nesse mesmo *template*, identificando um fluxo de mensagens agregadas segundo algum critério semântico, com um dado nome.

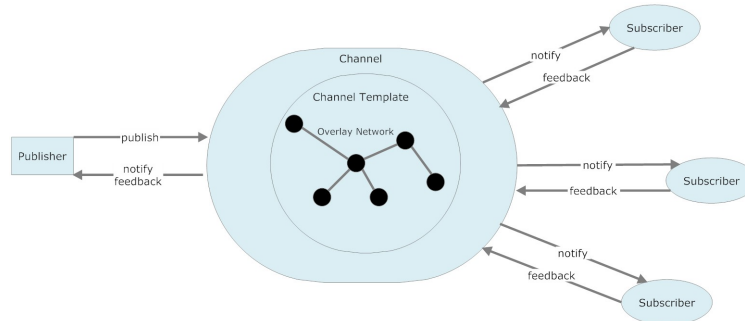


Figura 3.1 Visão global da comunicação entre entidades no sistema FEEDS

3.1.1 Arquitectura

A arquitectura de FEEDS é hierarquizada em três camadas que são populadas por nós de tipos diferentes. Esta hierarquização permite atingir uma escala elevada, bem como uma grande adaptabilidade. Em consequência desta característica, FEEDS pretende possibilitar qualidades de serviços ajustáveis a cada aplicação à custa dos diferentes *templates* que definem cada canal.

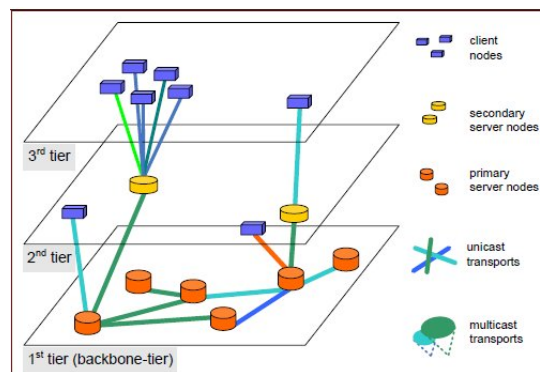


Figura 3.2 Arquitectura de três camadas de FEEDS, retirado de [5]

A primeira camada está definida pelo conjunto dos nós correspondentes aos servidores primários e é conhecida como *backbone tier*. O principal objectivo deste conjunto de nós é a formação e manutenção da rede *overlay*. A operação de ligação ao sistema passará muito por este conjunto de nós, na forma em que estes representam as sementes aos quais os outros nós se devem ligar.

A segunda camada é composta por nós servidores secundários e possivelmente alguns clientes. O segundo caso é mais uma excepção que uma regra, não sendo prevista a existência de muitos nós clientes nesta camada. O seu objectivo passa por libertar os nós primários da gestão directa dos inúmeros clientes.

A terceira e última camada é composta por nós clientes, que correspondem às aplicações. Estes nós normalmente irão ver as camadas superiores, com o intuito de descobrirem e/ou poderem comunicar os restantes nós da terceira camada.

A configuração concreta da rede *overlay* depende do *template* do canal que estiver a ser usado. Podendo existir vários canais instanciados segundo vários e diferentes *templates*, várias são as configurações possíveis de co-existir.

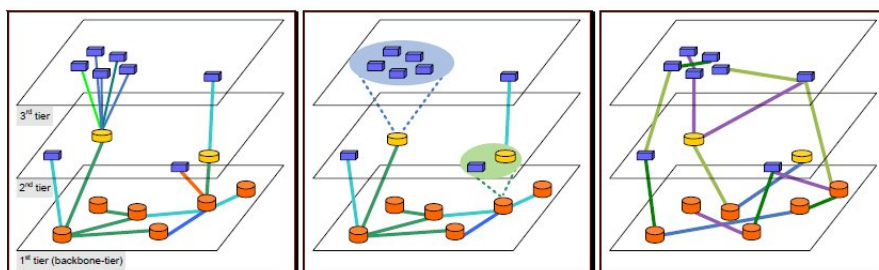


Figura 3.3 Exemplos de possíveis configurações de FEEDS, retirado de [5]

A figura 3.3 representa um exemplo que ilustra várias configurações possíveis de obter em FEEDS. Estas configurações são dependentes do *template*, correspondendo cada uma a um *template* específico. Existe a possibilidade de coexistirem simultaneamente vários canais definidos por diferentes *templates*, o que possibilita a que não exista uma só configuração da rede, mas a possibilidade da coexistência de diferentes configurações em simultâneo.

Por exemplo, será possível especificar através de três diferentes *templates*, uma configuração em árvore de difusão, um modelo completamente P2P ou ainda um modelo cliente-servidor. Estas diferentes configurações co-existirão, partilhando os mesmos nós, se existir a instanciação dos três *templates* em simultâneo.

3.1.2 Nó

Os nós são os componentes que fazem parte da arquitectura enunciada anteriormente. A diferenciação entre eles é feita segundo os componentes que estão activos e o comportamento destes em função do tipo nó em que estão a correr.

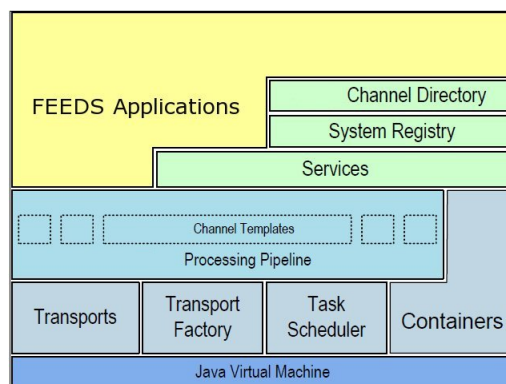


Figura 3.4 Arquitectura de um nó, imagem adaptada de [5]

Os nós são compostos por vários subcomponentes, estando ilustrados na figura 3.4, sendo que cada um tem uma tarefa específica. De todos os componentes destacamos três com relevância para a melhor compreensão do protótipo desenvolvido e apresentado neste capítulo: *Containers*, *Transports*, *Services* e *Event Processing Pipeline*.

Containers

Os *containers* são componentes cuja missão é guardar informação respeitante ao *soft-state* da execução corrente. Mais concretamente, os contentores são uma forma de armazenamento que podem ser monitorizados ou acedidos num padrão *on-demand*, sendo normalmente usados para sincronizar e partilhar dados entre os vários componentes internos do nó.

Transports

Transports são componentes que permitem a interconexão entre dois nós. De uma forma mais ilustrativa, podemos definir que os transportes correspondem às arestas entre os vários nós da figura 3.3.

Estes têm dois tipos de natureza, *incoming* quando estão à escuta de eventos, *outgoing* quando os enviam para um qualquer local. Normalmente são acedidos através de um canal e não directamente por parte de um outro componente. Assim podemos dizer que são a parte de baixo nível dos canais, responsável pelo envio ou recepção explícita dos eventos.

Services

Por vezes existem operações complexas que necessitam da interacção (transparente para o utilizador/programador) entre vários componentes como diferentes canais e *Containers*. Normalmente esta complexidade é depositada num serviço. Desta forma, os serviços englobam as funções dos nós FEEDS/MEEDS que se relacionam com o processamento de eventos trocados por canais do nível de sistema e monitorização de contentores. Nesse sentido, são construídos sob princípios semelhantes aos das aplicações finais. Por exemplo, toda a lógica respeitante ao serviço de descoberta do *proxy* mais próximo está concentrada num serviço que utiliza um canal próprio para esse mesmo efeito.

Processing Pipeline

O *Processing Pipeline* (PP) corresponde ao núcleo de toda a base de comunicações e funcionamento da rede. Todos os eventos produzidos bem como as mensagens de controlo atravessam o PP.

Este PP é composto pelo agregado de um conjunto de componentes simples, em que a diferente interacção entre estas leva à criação de diferentes caminhos tomados pelos eventos. Existe apenas um PP por cada canal instanciado de FEEDS, podendo no entanto existir componentes nos extremos que sejam partilhados por diferentes canais, nomeadamente ao nível dos transportes.

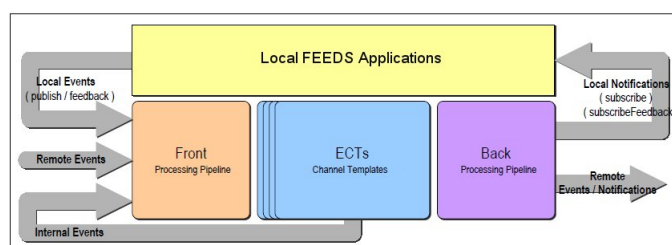


Figura 3.5 Arquitectura segundo uma visão alto nível do componente *Processing Pipeline*, imagem adaptada de [5]

Essencialmente, este componente corresponde ao trajecto interno que os eventos percorrem nos nós. Um evento entra sempre pelo início do *pipeline* (FRONT), através de um transporte de natureza *incoming*, é sujeito a regras de endereçamento e/ou multiplexagem num componente *Channel Template*, e entregue a um transporte que enviará o evento ao destino correspondente (parte final do *pipeline*). Estas regras são essencialmente definidas segundo os métodos *fRoute*, *pRoute* e *cRoute*, presentes nesse objecto, sendo possível definir regras adaptáveis a cada tipo de nó.

Alguns transportes estão definidos e são de uso comum nos *templates*, como LIQ (*local input queue*) para onde todos os eventos são passados antes de serem tratados por um qualquer *Channel Template*, a LOQ (*local output queue*) que direcciona os eventos para serem tratados pelos subscritores, e SOQ (*server output queue*) que corresponde a um transporte que permite o envio dos eventos para uma camada superior da hierarquia de FEEDS.

3.2 MEEDS

O suporte desenvolvido define uma plataforma que pretende fomentar e facilitar o desenvolvimento de aplicações *participatory sensing*. Para tal foi necessário adaptar FEEDS com vista a que seja possível concretizar os modelos relativos a comunicações e suporte de mobilidade descritos no capítulo anterior.

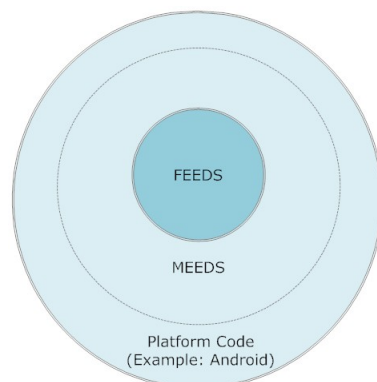


Figura 3.6 Estrutura de camadas que ilustra a extensão de FEEDS

Desta forma, FEEDS possibilita todo o suporte que é necessário à noção de infra-estrutura, isto é, FEEDS possibilita a implementação dos modelos definidos e que se inserem apenas no escopo infra-estrutura definida no modelo aplicacional.

No entanto, para o suporte de mobilidade foi necessário estender esta camada, acrescentando a camada MEEDS. Esta camada diz respeito a todos os serviços e novos componentes que possibilitam o suporte de mobilidade, tendo sido desenhada com vista a que seja possível vir a derivar estes serviços e possibilitar a integração com vários e heterogêneos sistemas (ex: Android, sistema para o qual está produzido o protótipo).

Para toda esta adaptação ser executada de uma forma simbiótica com FEEDS, foi necessário que o desenho novos componentes se enquadrem na arquitectura de FEEDS, estendendo as suas funcionalidades com vista ao suporte da mobilidade. Um exemplo, é a necessidade de novos serviços que possibilitem a descoberta dos *proxys* que aceitem servir um nó móvel (segundo um critério geográfico de proximidade), serviço esse que é construído à volta de novos canais, transportes e contentores, baseados nos homólogos de FEEDS.

Assim, as funcionalidades mais complexas definidas nos modelos, como as funcionalidades de *Homebase* e *Proxying*, são definidas como serviços, e as qualidades de serviço na forma canais e transportes. Estas qualidades não são definidas num ponto de vista quantitativo mas sim qualitativo, isto é, são qualidades específicas a diferentes fluxos de informação, agregados segundo o seu conteúdo, que fluem com regras diferentes definidas nos *templates* dos canais.

Esta camada MEEDS contém então os serviços responsáveis por esse suporte de mobilidade, isto é, o serviço *Homebase*, através do qual é possível obter uma ligação de um dispositivo móvel à infra-estrutura e todo o sistema, o serviço de *Proxying*, onde é descoberto o *proxy* mais próximo ao dispositivo móvel e suportada a comunicação entre o dispositivo móvel e a infra-estrutura, e ainda o serviço de persistência, o qual permite o armazenamento dos eventos produzidos.

Visto existir necessidade de cooperação entre diversos serviços e outros componentes, essa cooperação foi conseguida à custa de contentores.

As próximas secções irão mostrar como se enquadram as duas entidades (*Homebase* e

Proxy) na arquitectura de FEEDS, e ainda os serviços e qualidades de serviços definidos neste suporte. São apresentadas agrupadas segundo os componentes em que estão desenhadas.

3.2.1 Arquitectura

Atendendo à estrutura de três camadas de FEEDS, executámos o mapeamento das entidades do modelo para os nós nas diferentes camadas. As diversas entidades enquadram-se ao nível da *3rd tier* (terceira camada), na qual foram definidos dois tipos especiais de clientes, isto é os *Proxys* e as *Homebases/componentes fixas da aplicação*.

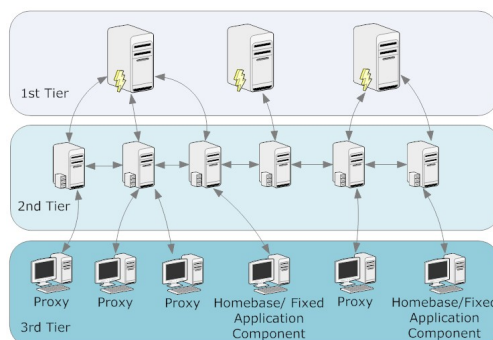


Figura 3.7 Enquadramento das entidades Homebase e Proxy em FEEDS

Estes componentes são mais do que simples clientes FEEDS, visto existir uma interacção com as componentes móveis, tendo por exemplo de integrar serviços como a descoberta do *proxy* mais próximo.

Proxy

O *proxy* está definido como nó cliente FEEDS. No entanto, este nó funciona não só como cliente de FEEDS, mas sim como servidor para as instâncias móveis da aplicação.

A cada *proxy* está associado uma área ao qual está disposto a servir instâncias móveis. Cada instância móvel tenta associar-se a um *proxy* que se situe o mais próximo possível de si, sendo para isso feito pedidos em determinados intervalos de tempo, que levam consigo qual a posição da instância móvel. Esta área deverá ser tipicamente modelada por uma forma geométrica e, no caso do protótipo para tal foi escolhida uma forma circular. No entanto, estando o serviço suportado pelos mecanismos de difusão filtrada, é possível implementar outras políticas, bastando especificar um filtro de outra natureza que aceite ou rejeite um nó móvel com base na sua localização.

Este processo está definido na forma de um serviço, no qual as entidades *proxy* necessitam de activar o serviço *Proxying*.

Homebase/Componente fixa da aplicação

O termo *Homebase*, neste caso, terá um sentido maior que o definido nos modelos. Na realidade esta componente refere-se não só às responsabilidades da entidade *Homebase* definida no capítulo dos modelos, mas também da componente fixa da aplicação, isto é, em cada instância da componente fixa da aplicação estão inicializados os serviços da *Homebase* (*Homing*).

Não é estritamente necessário que a *Homebase* esteja a correr na mesma instância, ou seja, componente fixa da aplicação não tem que estar forçosamente associada à componente responsável pelo *roaming*, no entanto os recursos e requisitos disponíveis e idealizados para ambas as componentes são compatíveis, sendo ainda um forma de uma distribuição mais justa dos recursos.

3.2.2 Comunicações e Mobilidade

Uma das características de uma aplicação *participatory sensing* é a cobertura espacial proporcionada pela mobilidade dos dispositivos móveis.

Esta mobilidade não é suportada em FEEDS, sendo necessário por isso necessário oferecer este serviço de alguma forma, utilizando os componentes já existentes dessa plataforma.

Para o suporte desta mobilidade existem agora canais móveis e canais FEEDS, em que se tira partido ainda das novas entidades (*Proxy* e *Homebase*), através da definição de um conjunto de serviços.

A diferença entre os dois tipos de canais, situa-se na existência de um novo tipo de nó denominado de nó móvel no caso dos canais MEEDS, o que torna a interacção entre eles, sem algum tipo de adaptação, incompatível.

Torna-se então importante suportar a possibilidade de se querer usar, num nó móvel, um *template* exclusivamente FEEDS, no qual não existe a noção de regras para nós móveis, isto é, o *template* foi definido ainda quando não eram contemplados nós móveis. Para que seja possível a um nó móvel utilizar um destes *templates*, de forma transparente para as aplicações, foi criada a noção de *tunnel*.

3.2.2.1 Tunnel

O *tunnel* é uma abstracção que representa um canal de comunicação entre o dispositivo móvel e a infra-estrutura fixa.

Devido à existência da primitiva de *feedback*, existe a necessidade de os nós móveis serem conhecidos na camada FEEDS. Como não é apropriado que estes nós participem directamente nesta camada, torna-se necessário utilizar um intermediário no seu lugar, com identidade conhecida na infra-estrutura fixa.

O túnel é concretizado na forma de um serviço que, quando um nó o tem activo, estabelece o *template* túnel como o *template* a usar por defeito quando não existir uma regra habilitada a tratar o pacote. Mais concretamente, no caso de um nó móvel a usar um *template* FEEDS, este

template não terá definidas as regras de encaminhamento e processamento adequadas, sendo utilizado o *template* por defeito, ou seja, o *tunnel*.

O que este *template* realiza é simplesmente definir que cada mensagem enviada de um nó móvel se dirija ao *proxy* mais próximo, que por omissão será a sua *Homebase*.

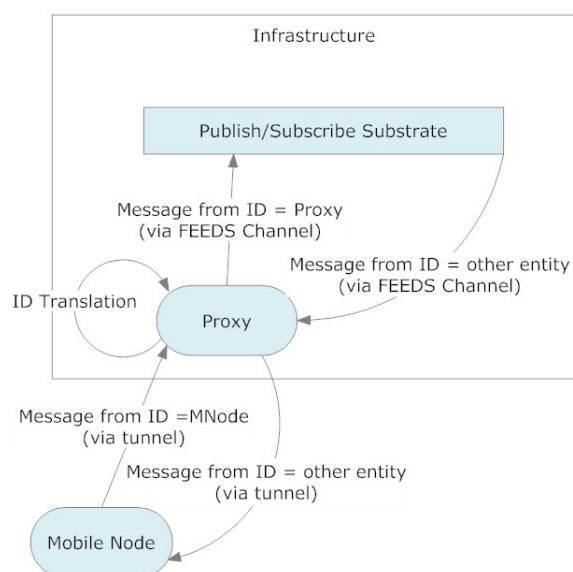


Figura 3.8 Funcionamento do tunnel

Portanto, cada nó que pertencer à camada clientes de FEEDS com o serviço de *tunnel* activo será uma *Homebase* ou um *Proxy* e terá um identificador único. Cada uma destas entidades mantém uma tabela que guarda os nós móveis (um identificador único) e as suas correntes subscrições e/ou canais activos. Estão assim aptas a fazerem o papel de "clientes" desses mesmos canais. Desta forma, sempre que uma mensagem venha de um nó móvel com o intuito de ser difundida na infra-estrutura, o seu envio é feito de novo, só que com o identificador da *Homebase* ou *Proxy*, dependendo do caso. No caso oposto, quando uma mensagem chega a um *Proxy* ou *Homebase*, esta é difundida para cada um dos nós que tenha uma subscrição que aceite essa mesma mensagem.

Em suma, este processo é semelhante ao NAT, funcionando através da tradução entre vários ID de nós móveis, para um ID de uma entidade na infra-estrutura fixa, que serve como intermediária nas comunicações quer num sentido quer noutro.

3.2.2.2 Serviços

Existem operações complexas que, tirando partido da interacção entre diversos componentes como canais e contentores, produzem alterações ao estado do nó. No fundo é nestes serviços que se deve executar o processamento complexo de ao nível de troca de eventos (via canais),

que permita a adição de uma qualquer nova funcionalidade ao sistema. Os serviços são normalmente autónomos (sem necessidade de interacção com o utilizador/programador), tendo as suas operações bem definidas.

Estas mesmas operações serão parte da distinção de cada tipo de nó, mais concretamente os nós que são *Homebase*, *Proxy* e que tenham responsabilidades/capacidade de persistência.

Tal como definido no capítulo dos modelos, o protocolo de mobilidade tem a interacção de três instâncias, o nó móvel, a *Homebase* e a entidade *Proxy*.

Como requisitos, temos a necessidade de a *Homebase* ser bem conhecida do seu nó móvel, bem como que cada *proxy* conhece a sua localização e define um limite máximo de distância que está disposto a servir, na forma de uma raio em redor do seu ponto geográfico. Considerando estes requisitos satisfeitos, a descrição informal do protocolo de mobilidade é a seguinte:

- Periodicamente, um nó móvel envia um pedido à sua *Homebase* requisitando informação sobre o *proxy* mais próximo.
- A *Homebase* ao receber esse pedido, difunde essa informação através do substrato *publish/subscribe*, com intenção de atingir os *proxys*.
- A mensagem ao chegar a um *proxy* que esteja dentro das condições de servir o nó móvel, retorna à *Homebase* a confirmação de que está disposta a servir, adicionando-lhe o seu endereço.
- A *Homebase* colecta as respostas de todos os *proxys*, calcula o mais próximo e, na próxima interacção, devolve o resultado ao nó móvel.
- O nó móvel quando recebe a mensagem, extrai o endereço e envia um pedido de associação ao *proxy*.
- O *proxy* recebe o pedido de associação e devolve uma confirmação de associação estabelecida ao nó móvel.

Para implementação deste protocolo, foram desenvolvidos dois serviços, um que define o comportamento da *Homebase* (denominado de *Homing*), e outro que define comportamento de um *Proxy* (denominado de *Proxying*).

Em acréscimo a estes serviços foi desenvolvido um outro que concretiza o modelo de persistência.

Serviço de Homebase (*Homing*)

O serviço de *Homebase* serve para uma instância móvel estabelecer uma conexão com a sua *Homebase*. Admite-se que a componente móvel tem conhecimento prévio da localização (URL) da sua *Homebase*.

Ambos os nós, tanto os móveis como os que pretendem desempenhar o serviço de *Homebase*, fazem uso de um canal denominado de *HomingChannel*. No caso do serviço que

corre no dispositivo móvel, enquanto o nó móvel não estiver conectado a um *Proxy*, este tenta conectar-se através da publicação de pedidos de associação (*HomingRequest* para o canal).

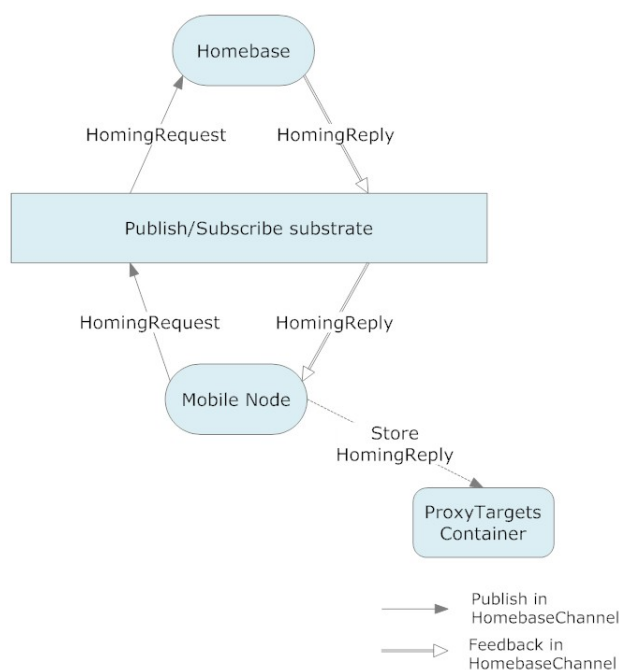


Figura 3.9 Processo do Serviço de *Homing*

O serviço que corre no nó presente na infra-estrutura fixa, e que tem intenção de servir como *Homebase*, inscreverá este canal. Ao receber um pedido, reponderá através de *feedback* ao nó móvel indicando-o qual o nó que está em condições de ser o seu *proxy*. No caso de não existir nenhum *proxy* nas condições de oferecer o serviço ao nó móvel, a *Homebase* desempenhará esse papel.

Ao receber a resposta, o nó móvel guarda a informação respeitante ao *Proxy* num contentor, de forma a que esta mesma informação possa ser acedida por outros componentes. Este contentor está denominado de *ProxyTargets*.

Serviço de *Proxying*

Nos modelos definimos que cada instância móvel de uma aplicação está ou deve estar ligada ao *Proxy* mais próximo. Para tal existem duas fases distintas, uma na qual se descobre qual o *proxy* que se situa mais próximo, e uma segunda onde se tenta conectar com o *proxy* encontrado. Estas duas fases correspondem às duas funcionalidades essenciais de um *proxy*, o de responder a pedidos de descoberta e a pedidos de associação.

Estas duas fases consideram-se distintas, no entanto a segunda está dependente da primeira. Desta forma, foram implementados dois sub-serviços diferentes, o primeiro no qual se executa a descoberta do *proxy* mais próximo e um outro através do qual se conecta a esse mesmo *proxy*.

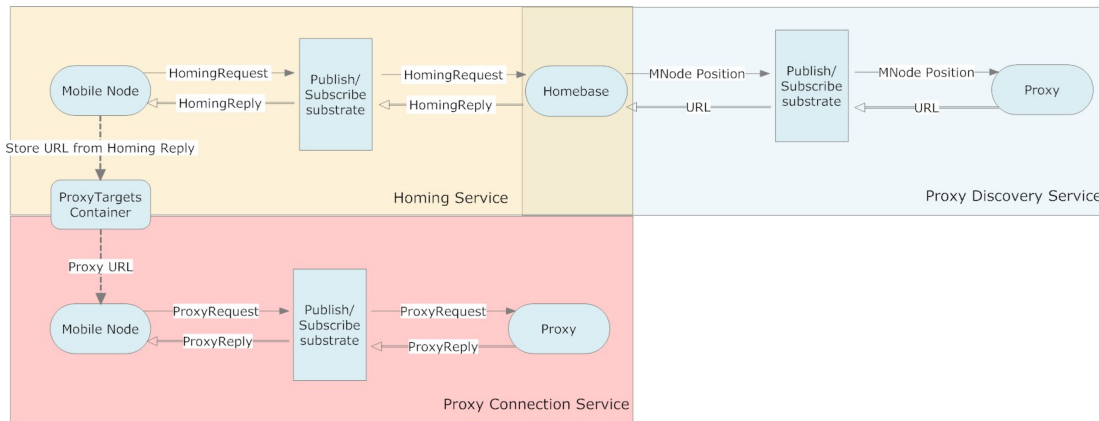


Figura 3.10 Composição de serviços para suporte à mobilidade

Descoberta do *Proxy* mais próximo

Este serviço tem como requisito que o nó que deseje desempenhar o papel de *proxy* conheça a sua localização, definindo um limite máximo (área nas proximidades) a que está disposto a servir. Para tal, foi usado um raio, o qual permite definir um círculo, dentro do qual todos os nós móveis que lá estejam contidos (localização), são servidos, ou pelo contrário rejeitados. Este raio em conjunto com a localização consiste no filtro (Criteria) a usar no canal *ProxyDiscoveryChannel*, que é uma instanciação do *template Catadupa-ProxyChannel*, sendo responsável por implementar a difusão filtrada na infra-estrutura fixa.

Listagem 3.1 Classe representativa do filtro, respeitante à operação de descoberta de um *proxy*

```
class Circle extends Criteria<XY> {
    final XY center;
    final double radiusSq;

    Circle(XY center, double radius) {
        (...)
    }

    public boolean accepts(XY pos) {
        return center.distanceSq(pos) < radiusSq;
    }

    public String toString() {
        (...)
    }

    private static final long serialVersionUID = 1L;
}
```


Este é um *template* FEEDS que passa a ter escopo global de forma transparente, por via do suporte de mobilidade.

Os pedidos deste canal são injectados pelo nó que desempenhar o papel de *Homebase*, que insere neste canal a posição do nó que quer obter o *proxy* mais próximo.

Por *feedback*, o *proxy* devolve uma resposta contendo o seu endereço. De notar que apenas os *proxys* que tiverem cuja área abranja a posição do dispositivo móvel recebem a mensagem, visto esta ser filtrada no processo de difusão de acordo com os filtros submetidos, e que têm um critério baseado em nível geográfico. Este filtro poderá vir a ser melhorado, podendo dar-se o exemplo que o filtro segue processo estocástico, existindo uma probabilidade de aceitar um nó definida com base na distância ao dispositivo móvel.

Conexão ao Proxy

A segunda fase processa-se com a ligação ao *proxy*, sendo para tal utilizado um novo canal denominado de *ProxyChannel*. Este canal terá como objectivo permitir comunicar directamente entre o nó móvel e o *proxy* a que deseja associar.

Estando também implementado na forma de um serviço, o nó móvel, envia pedidos de associação ao *proxy* que esteja disposto a aceitar um nó móvel nas actuais condições. O endereço do *proxy* está guardado num contentor local, sendo o resultado do serviço anterior.

Como subscritor do canal *ProxyChannel*, está o nó da infra-estrutura fixa que tenha intenções de desempenhar o papel de *proxy*. Este ao receber um pedido de associação, responderá por *feedback* uma resposta que terá o intuito de confirmar a associação.

Em virtude da mobilidade existente, os nós móveis irão frequentemente deixar de estar nas condições de serem servidos por um *proxy* para entrarem na área de cobertura de outro. Devido a tal facto, ambos os serviços de descoberta como de conexão são executados periodicamente.

A desconexão do *proxy* "antigo" é executada aquando de uma recepção de uma mensagem vinda deste. O nó móvel ao verificar que a mensagem é vinda de um *proxy* que já não é o "seu" actual, indica a esse *proxy* para deixar de o servir.

Serviço de Persistência

O serviço de persistência pretende ilustrar uma concretização do modelo de persistência. Este suporte é bastante primitivo em virtude da complexidade do tema, e não se engloba no âmbito específico desta dissertação. No entanto, em virtude de não ter sido possível, por questões de tempo, implementar o sistema de *roaming* com fiabilidade, no qual a *Homebase* seria responsável pelos eventos da componente móvel enquanto esta estivesse desligada, este suporte permite a compensação da falta de fiabilidade introduzida por essa ausência.

Assim, este serviço é composto por duas situações, em que a primeira diz respeito ao armazenamento dos eventos e a segunda ao acesso (ou *query*) desses mesmos dados armazenados.

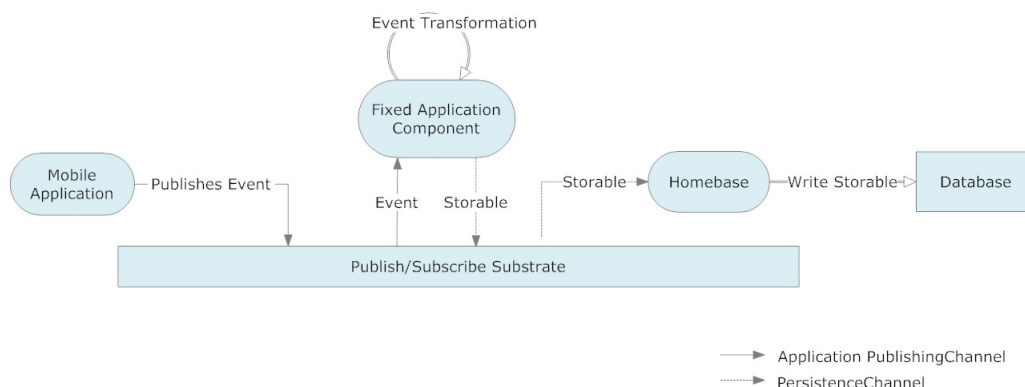


Figura 3.11 Concretização do modelo de persistência, em relação ao armazenamento dos eventos

Para armazenamento da informação está definido um serviço o qual permite definir quais os canais e seus respectivos filtros que irão ser monitorizados com vista ao armazenamento dos eventos. Estes são indicados no momento de activação do serviço, através de um conjunto de pares canal-filtro.

Assim, uma aplicação que deseje ter os seus eventos armazenados deve implementar uma interface *Storable* nesses objectos em que transitam nos canais. Desta forma, estes são objectos que passam a saber como ser guardados na base de dados, isto é, têm definidos dois métodos através do qual é possível segundo *table()* obter a tabela onde devem ser inseridos e um método *values()* que devolve uma *String* no formato SQL dos valores a inserir, do género (*valor1,valor2,"valor3"*).

Listagem 3.2 Interface *Storable*, a qual define um objecto que possa ser guardado

```

public interface Storable{
    public abstract String table();
    public abstract String values();
}
  
```

As *Homebases* que têm o serviço de persistência activo subscrevem um conjunto de canais, que têm de estar definidos segundo um filtro sobre cada canal, que permite a especificação de quais os eventos em que estão interessadas a armazenar.

Desta forma será possível criar uma qualquer distribuição dos eventos pelos nós, sendo essa especificação feita através dos filtros. Um exemplo poderá ser uma distribuição feita com base num critério geográfico, no qual os filtros das *Homebases* aceitam apenas os

eventos que tenham origem na sua zona. Este critério implicará que os eventos venham catalogados com a sua informação geográfica.

Em suma, para um programador que use o suporte fornecido poder usufruir deste serviço, terá apenas de activar o serviço *PersistenceService*, indicando-lhe quais os canais a monitorizar, e os eventos da sua aplicação terão de implementar a interface *Storable*.

Para a operação de consulta dos dados, está definido um canal de escopo global, *PersistenceQuery*. Este canal tem como eventos objectos *Query*.

Listagem 3.3 Código respeitante ao objecto de uma *Query*

```
public class Query implements Serializable{

    private XY center;
    private double radius;
    private String[] keywords;
    private String table;
    private long freshNess;

    public Query(XY center, double radius, String[] keywords, String table, long freshness ) {
        this.center = center;
        this.radius = radius;
        this.keywords = keywords;
        this.table = table;
        this.freshNess = freshness;
    }

    public String table(){
        return table;
    }

    public XY getCenter() {
        return center;
    }

    public double getRadius() {
        return radius;
    }

    public String[] getKeywords() {
        return keywords;
    }

    public long freshness() {
        return this.freshNess;
    }
}
```

Estes objectos permitem definir uma *query* sobre um qualquer objecto de uma base de dados. O canal levará a *query* até às *Homebases* em que a *query* poderá ser satisfeita, isto porque a *Homebase* subscreverá o canal *PersistenceQuery* com um filtro que verificará quais os tipos de eventos que aceitou guardar.

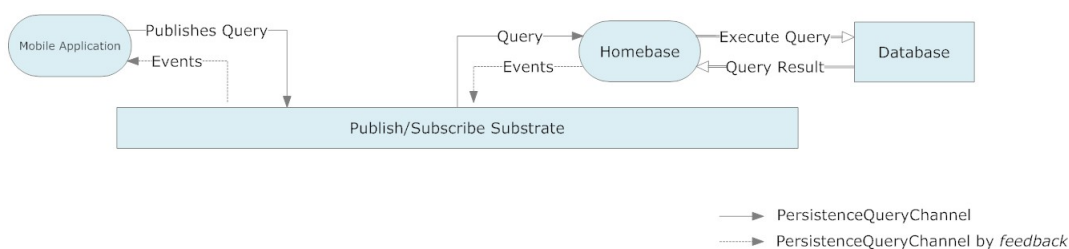


Figura 3.12 Concretização do modelo de persistência, em relação à consulta dos dados armazenados

Após uma *Homepage* aceitar uma *query* esta executará essa mesma numa base de dados, retornando os resultados à *Homepage/componente fixa da aplicação* correspondente à instância da aplicação que iniciou a *query*, através da operação de *feedback* do canal. Por *feedback* são também os eventos enviados à componente móvel da aplicação, após a normalização efectuada pela *Homepage/Componente fixa da aplicação*.

3.2.2.3 Templates

No capítulo dos modelos foram identificados três tipos de escopos que são prováveis de acontecer tendo em conta os outros modelos definidos e a necessidade de comunicações entre as diversas entidades. Estes escopos, são essencialmente definidos à volta do alcance que têm na disseminação das mensagens, estando denominados de *global*, *local* e *ad-hoc*.

Os escopos estão delineados na forma de *templates* de canais, sendo definidas as regras de endereçamento destes *templates* que levam aos diferentes limites de alcance de cada canal instanciado à custa destes mesmos *templates*.

Listagem 3.4 Exemplo de um *template*

```
public class ProxyChannel extends BasicTemplate<Void, ProxyRequest, Void, ProxyReply> {
    public void init() {
        super.init();

        switch (FeedsNode.type()) {

        case mNODE:
            pipeline.setTemplate(new BasicTemplate<Void, ProxyRequest, Void, ProxyReply>(channel()) {

                public void pRoute(pPacket<Void, ProxyRequest> p) throws Exception {
                    (...)
                }

                public void fRoute(fPacket<Void, ProxyReply> p) throws Exception {
                    (...)
                }

            });
            break;

        case cNODE:
            pipeline.setTemplate(new BasicTemplate<Void, ProxyRequest, Void, ProxyReply>(channel()) {

                public void pRoute(pPacket<Void, ProxyRequest> p) throws Exception {
                    (...)
                }

                public void fRoute(fPacket<Void, ProxyReply> p) throws Exception {
                    (...)
                }

            });
            break;

        }
    }
}
```

As regras definem-se essencialmente à custa, dos métodos *pRoute* e *fRoute*. Este métodos tratam os eventos que seguem pelos caminhos de publicação e *feedback* respectivamente. É possível especificar ainda o comportamento de cada método dependendo do nó em que nos encontremos.

Nos modelos descritos anteriormente, defendia-se ainda que o endereçamento deveria ser feito à custa de critérios geográficos. Será no canal de escopo global, que utiliza um algoritmo denominado de *Catadupa* que esse processo é executado, no qual as mensagens apenas necessitam de ser catalogadas com a sua localização e os filtros utilizados especificarem quais as

posições que estão dispostas a aceitar.

Global/Catadupa

São definidos alguns *templates* que permitem a comunicação entre os vários componentes, podendo um evento circular de uma componente móvel até um qualquer outro componente (*Proxy*, *Homebase*, uma aplicação externa) com filtragem baseada no conteúdo da mensagem.

Os *templates* acabam por ser todos uma pequena variação ao *template Catadupa* que representa um algoritmo desenvolvido no âmbito do projecto LiveFeeds ¹[20], sendo Catadupa o nome dado na implementação em FEEDS e onde existe a particularidade de os nós primários tomarem o papel de *Slice Leaders*.

Este algoritmo pretende transpor o problema da difusão de informação com filtros baseados no conteúdo num problema de computação distribuída. O algoritmo dá a garantia de inexistência de falsos positivos e falsos negativos, entregando as mensagens apenas a todos interessados nesse conteúdo.

O algoritmo utiliza uma filosofia baseada *one hop routing*, sacrificando complexidade espacial com vista a melhorar complexidade temporal.

Essencialmente, a sua natureza está dividida em duas partes: algoritmo de filiação e posteriormente difusão das mensagens com filtragem. Ambas as fases fazem uso de árvores de difusão aleatórias, segundo as quais as mensagens são posteriormente distribuídas pelos interessados.

Algoritmo de filiação

Existem duas partes neste algoritmo, a primeira é onde são construídas estas árvores que permitam a difusão eficiente das mensagens pelo sistema, em que se tenta colocar o maior número de nós a tentar conhecer tantos outros quanto possível. Porém, as entradas concorrentes e falhas temporárias de nós levam a que visões inconsistentes destas árvores surjam. Assim, existe uma etapa complementar que, à custa de algoritmos epidémicos, restaura a consistência das árvores.

Cada nó é conhecido através de um identificador numérico aleatório, um filtro e o seu endereço. Os identificadores estão agrupados em pequenos grupos, no qual o identificador mais baixo desse grupo toma o papel de *slice leader*, cujas responsabilidades no algoritmo são diferentes dos outros nós.

Um nó para se juntar a sistema faz uso de um nó conhecido (*semente*) que, consoante o identificador deste novo nó, lhe atribui o *slice leader* correspondente.

¹Projecto PTDC/EIA/76114/2006, Project LiveFeeds - P2P Dissemination of Web Syndication Content, financiado pela FCT/MCTES.

Um *slice leader* tem como função principal agregar pedidos de entrada antes de estes serem difundidos por todos os nós do sistema avisando-os que estes novos nós fazem agora parte do sistema. Cada difusão de informação é acompanhada de uma estampilha temporal (relógio vectorial). A acumulação dura o tempo necessário até a agregação de um número significativo, sendo permitido um período máximo de trinta segundos de acumulação. O evento é então difundido através de uma árvore aleatória de grau n gerada ao longo de todo o processo de difusão. Esta árvore é criada através do *slice leader* que define n grupos com semelhante dimensão, enviando a um nó aleatório de cada grupo a lista de todos os nós que serão da sua responsabilidade de informar. O processo é recursivo até a informação fluir até todos os nós. Um nó sabe que faz parte do sistema quando receber uma mensagem indicando o seu pedido de entrada no sistema.

As falhas de nós e entradas concorrentes geram árvores e visões inconsistentes do sistema. Assim, é executado um algoritmo epidémico no qual periodicamente um nó selecciona outro ao acaso, enviando a estampilha temporal corrente, o que permite identificar e recuperar dessas inconsistências.

Difusão de Conteúdos

Com a visão consistente dos nós do sistema, a difusão de informação é uma pesquisa distribuída dos nós cujos filtros estão de acordo com a mensagem, em que cada nó fica responsável por pesquisar um sub-grupo de todos os nós. Desta forma, se construirmos uma árvore de grau n , dividir-se-há o conjunto de nós em n grupos. Pesquisa-se cada grupo sequencialmente até encontrar um nó que aceite o filtro, indicando-lhe para executar a pesquisa para o resto do grupo em que está inserido. Recursivamente este processo é repetido, construindo-se uma árvore de difusão, em que apenas os interessados são notificados do evento.

Este tipo de *template* (todas as variações do *template Catadupa* permite a instanciação dos canais relativos à persistência, à descoberta dos *proxies*, bem como canais aplicativos que necessitem de um escopo global.

Ad-Hoc

Este *template* define um escopo é do tipo "ad-hoc". Este escopo permite a interconexão entre diferentes dispositivos móveis sem a passagem pela infra-estrutura fixa. Idealmente este *template* deverá fazer uso de transportes que permitam o uso tecnologias de baixo custo, como o *Bluetooth*, ou então combinar o uso de *Bluetooth* com WiFi, com a primeira tecnologia como serviço de descoberta e a segunda para transferência de dados. No entanto, no imediato ainda não existe uma API de *Bluetooth* disponibilizada para o sistema Android. Desta forma, está implementado este canal usando um transporte que simula a proximidade entre dispositivos através de *IP Multicast*.

Em termos de futura implementação, aquando da disponibilidade de uma API para o efeito, bastará alterar os extremos do *pipeline* do canal, isto é, os transportes de entrada e saída utilizados pelo *template*.

Canais de Escopo Local

Os canais de escopo global existem neste protótipo na forma de acesso aos sensores. Existem os canais relativos a sensores de GPS, Tilt, WiFi, Câmara, Acelerómetro e Microfone. A descrição dos sensores, acesso e modo de utilização destes canais está descrita numa secção dedicada (3.3).

3.2.2.4 Transportes e diferentes tecnologias de comunicação

O suporte apresentado por FEEDS em termos de transportes não permite o uso todas as tecnologias disponibilizadas pelos dispositivos móveis. Promovemos então uma discussão que irá mapear a as diversas tecnologias a diferentes transportes.

Bluetooth

A tecnologia *Bluetooth* é uma tecnologia de baixo custo energético. O uso desta tecnologia poderá ser mapeado para um transporte que possibilite a automática conexão entre dois dispositivos que usem esta tecnologia. Estes podem ser definidos através de um transporte que aceite como parâmetros no URL de saída o endereço físico *Bluetooth*. Está implementado um transporte que pretende ilustrar o funcionamento deste tipo de tecnologia essencialmente ao nível da proximidade entre dispositivos. Correntemente, este transporte não usa no entanto a tecnologia *Bluetooth*, em virtude da não existência de uma API disponível para o sistema Android que lide com essa mesma tecnologia. Como tal, para simular o efeito de proximidade da tecnologia Bluetooth, o transporte está implementado sobre IP Multicast.

GPRS e WiFi

Estas duas tecnologias, são dois modos de comunicação sobre IP. Assim sendo, esta tecnologia está dependente da rede à qual o dispositivo móvel está conectado. Um simples transporte é suficiente para ambas as formas de comunicação, no entanto ao desenvolver este transporte deve-se ter em conta que caso seja necessário comunicar via GPRS, o utilizador deverá ser alertado para tal facto.

Decidimos implementar as três qualidades de serviço definidas nos modelos (Critical, Real-time, e Deferred) na forma de transportes. Todos estes transportes fazem uso de um contentor denominado de *SystemStatsContainer* que contém informação sobre a qualidade de sinal de WiFi e Bluetooth, bem como o estado do uso dos três tipos de qualidade de serviço (através da monitorização de três filas de prioridade correspondendo a cada uma qualidade de serviço).

Em qualquer um destes transportes é feita uma agulhagem que permite escolher qual o meio de "sub-transporte" mais adequado para o envio da mensagem, isto é, se por *Bluetooth* ou WiFi. Esta agulhagem baseia-se nos valores de sinal das diferentes tecnologias, obtidos a partir do contentor *SystemStatsContainer*.

Critical

Qualidade de serviço que tem prioridade no envio sobre todas as outras. O transporte tem um pequeno *buffer* que assim que enche despeja todas as mensagens nele presentes para a infraestrutura. Ao mesmo tempo, uma tarefa concorrente despeja essas mesmas mensagens em curtos intervalos de tempo.

Realtime

Qualidade de serviço que representa uma qualidade quase tempo real. Este transporte monitoriza o estado do número de mensagens a enviar pelo transporte *Critical*, sendo as mensagens enviadas quando apenas se verificar um número abaixo de um limite definido.

Deferred

Qualidade de serviço que não dá muita importância a requisitos temporais para o envio das mensagens. Este transporte monitoriza a quantidade de mensagens a enviar por parte dos transportes *Critical* e *Realtime*, sendo que apenas é feito o envio das mensagens quando ambos os valores obtidos acerca das duas filas de transporte estejam abaixo de um certo limite.

Estes transportes implementam um conceito semelhante ao de CafNet do sistema Cartel[14], no qual o mesmo tipo de persistência é conseguido, isto é, através de bufferização, existindo transmissão aquando de condições disponíveis para o efeito.

O uso destas diferentes qualidades de serviço, pode ser feita de diferentes formas. Por um lado a construção de um canal com as qualidades de serviço *hardcoded* é possível, sendo que neste caso, para cada qualidade de serviço seria necessário um *template* específico. No caso de existirem vários fluxos de mensagens semelhantes, com apenas a necessidade de prioridades diferentes, seria necessário reescrever os vários canais substituindo o transporte usado, o que na prática impossibilita a reutilização de código.

Outra forma de usufruir destas qualidades de serviço de uma forma mais reutilizável, seria codificar nos eventos a qualidade de serviço que é desejada por estes. Esta aproximação implica simplesmente a construção de um canal que, consoante a interpretação e análise dos metadados desses eventos, executa a instanciação do transporte mais adequado.

3.3 Sensores

A recolha de informação contextual por parte de uma aplicação *participatory sensing* é executada através da interacção com o utilizador e uso de sensores. Um utilizador, ao desejar incorporar sensores ao nível da recolha de informação contextual, pretende essencialmente capturar informação desses sensores, segundo uma determinada configuração de atributos relativos a esse mesmo sensor. O acesso a estes mesmos sensores poderá envolver diversas APIs, dependendo da sua natureza.

Em acréscimo a esta heterogeneidade de acesso aos sensores, as APIs existentes costumam estar desenvolvidas especificamente para um determinado sistema, dificultando a uniformidade no desenvolvimento de aplicações *participatory sensing* em diferentes sistemas.

Uma forma de acesso comum aos recursos, constituiria um impulso no sentido de uniformizar o desenvolvimento deste tipo de aplicações, diminuindo os conceitos necessários a aprender e a curva de aprendizagem em relação a uma determinada plataforma.

Com este intuito, definimos que o acesso aos sensores será feito com o recurso ao modelo de programação de FEEDS segundo o substrato *publish/subscribe*, mais concretamente através do uso de canais.

Esta aproximação permite uma curva de aprendizagem mais rápida, permitindo ao programador o acesso simplificado aos sensores, facilitando por isso o desenvolvimento de aplicações e acelerando o seu desenvolvimento. Respeitando ainda este modelo, consegue-se manter ainda a modularidade e extensibilidade oferecida por FEEDS, sendo por isso fácil acrescentar novos sensores ou reescrever estes para um outro tipo sistema móvel que não o Android.

Desta forma, definimos esqueletos que pretendem oferecer o suporte ao desenvolvimento de sensores, cuja camada de middleware denominamos de SEEDS.

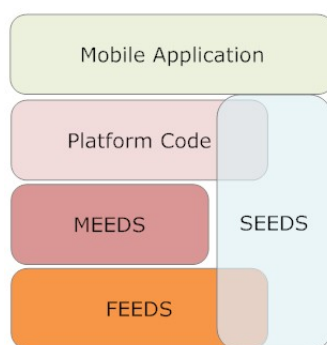


Figura 3.13 Colocação da camada SEEDS na hierarquia de todas as camadas de suporte.

Esta será uma camada que atravessará todas as outras camadas, começando em FEEDS e acabando no que respeita ao código específico da plataforma de desenvolvimento. Assim, é nesta última camada que os verdadeiros sensores estão verdadeiramente implementados, estendendo no entanto os esqueletos existentes.

3.3.1 Arquitectura dos Canais

O uso de canais para sensores situa-se ao nível do escopo local, sendo os seus eventos produzidos e consumidos apenas no que à instância móvel da aplicação diz respeito.

Para construir um canal deste género, é necessário que em algum ponto se faça a ligação com a API de sistema, de forma a que esses mesmos eventos possam ser produzidos de uma

forma transparente ao utilizador e se enquadrem neste modelo de canais. Foram consideradas várias hipóteses tendo em conta a estrutura de componentes de FEEDS.

Sendo que o principal requisito para este tipo de canais é a transparência de acesso ao detalhes da API de sistema, o mais natural é o canal fazer uso de um outro componente para aceder a essas APIs e, de alguma forma, normalizar esse componente para ser fácil a sua extensão para a criação de novos sensores, ou mesmo derivar esses sensores para um sistema específico (Android, Symbian).

Foi considerada a hipótese de fazer a ligação à API de sistema através de *transportes*, no entanto estes não estão preparados para serem reconfigurados e não seria possível a parametrização dinâmica dos sensores. Outra via seria a sua implementação totalmente nos *templates* dos canais, só que este procedimento seria muito específico e pouco modular, sendo complicada a reutilização do código e, consequentemente, perder-se-ia o requisito de extensibilidade para outras plataformas.

A opção tomada foi a de dar a responsabilidade aos contentores (*Containers*) de FEEDS. Esta aproximação permite atingir os requisitos de extensibilidade e reutilização de código idealizados, sendo possível isolar o código relativo a cada sistema de desenvolvimento neste componente.

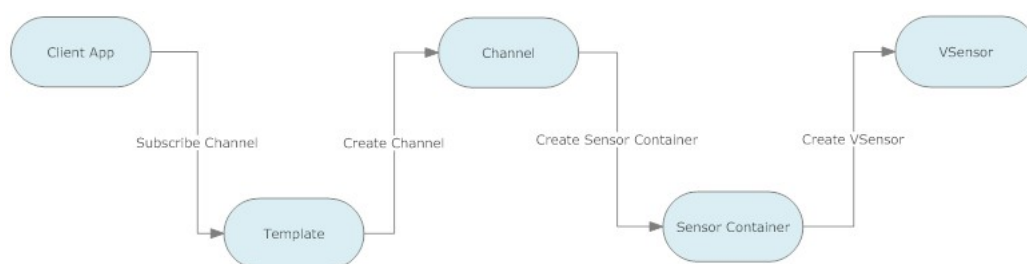


Figura 3.14 Esquema da Criação de um Sensor

A figura 3.14 mostra o processo de interação entre as várias componentes de forma a que se possa criar os sensores desejados. Tal como ilustrado, os sensores são criados aquando do pedido de instanciação do canal. Neste ponto, e devido ao facto de o canal ser local, ao ser criada uma instanciação do canal, é criada também uma instância de um *Container* que por sua vez instancia um objecto *sensor virtual*, que será responsável pela colecta de informação associada à API de sistema existente para uso do sensor do dispositivo móvel.

Listagem 3.5 Exemplo de acesso a um sensor, no caso Acelerómetro

```

final Channel<AccData, Void, ?, ?> channel = Feeds.lookup("/System/Sensors/AccelerometerChannel");
channel.subscribe(new AccCriteria(true,10,10), new Subscriber<AccData, Void>() {
public void notify(Receipt r, AccData e, Payload<Void> p) {
    ....
}
});
  
```

O acesso a todos os sensores faz-se de forma semelhante ao exemplo anterior. A diferença

consiste essencialmente ao nível do nome submetido para *lookup* e ao nível dos tipos de envelope e ainda filtros.

3.3.2 Acesso a Sensores

O acesso aos sensores faz-se através do conceito de canal da plataforma FEEDS. Este canal permitirá que cada actividade detectada por um sensor possa ser chegar a um qualquer interessado (subscritor do canal) na forma de um evento.

O acesso aos sensores segundo o canal pode ser efectuado de duas formas, dependendo do sensor. No primeiro caso, identifica-se os sensores cujo padrão de acesso seja reactivo, isto é, assim que os sensores tenham "novidades" produzem um evento que chegará aos interessados participantes do canal. Neste caso o acesso é feito através da simples subscrição do canal.

Listagem 3.6 Exemplo de acesso a um sensor na forma de uma subscrição

```
final Channel<GpsData, Void, ?, ?> channel = Feeds.lookup("/System/Sensors/GpsChannel");

channel.subscribe(new GpsCriteria(10.0, "gps", 2, 5), new Subscriber<GpsData, Void>() {
    public void notify(Receipt r, GpsData e, Payload<Void> p) {
        (...)
    }
});
```

No exemplo acima, mostra-se a forma como aceder a um sensor na forma de uma subscrição. No dado caso, o acesso é ao sensor GPS, o qual deverá produzir dados em intervalos de tempo (ou após a posição se ter alterado uma determinada distância), bastando tratar o evento no método *notify*.

Outra via de aceder a sensores é através do pedido para uso de um sensor, acontecendo em casos que necessite existir alguma interacção entre utilizador e sensor, como por exemplo no sensor câmara. Esta forma modela-se melhor na forma de um *pedido-resposta*.

Para tal, deve-se publicar para o canal um evento que corresponde a esse pedido, obtendo-se a resposta pelo mecanismo de *feedback* do canal.

Listagem 3.7 Exemplo de acesso a um sensor na forma de pedido-resposta

```
final Channel<CamRequest, ?, CamData, Object> channel = Feeds.lookup("/System/Sensors/Cam");

channel.publish(new CamRequest(PIC_EXTS.JPG, 100, 100), null);

channel.subscribeFeedback(new FeedbackSubscriber<CamData, Object>() {
    public void notifyFeedback(Receipt r, CamData e, Payload<Object> p) {
        (...)
    }
});
```

Neste último exemplo, o acesso é feito ao sensor câmara, sendo que para o seu uso, um pedido é feito ao canal, o que despoletará a abertura da interface relativa à captura de uma foto. O retorno do resultado virá no envelope do evento de *feedback* gerado.

3.3.3 Parametrização de Sensores

Os sensores necessitam de ser parametrizados de acordo com os atributos desejados por quem usa o sensor. Se através de uma API de sistema esta questão é simples e directa de resolver, bastando para isso o acesso a um método disponibilizado por essa mesma API, no caso do canal esta solução não é tão óbvia.

O acesso a estes sensores, como já referido, processa-se segundo um canal e, normalmente através de uma subscrição desse mesmo canal. Assim, a monitorização das subscrições do canal, permite que se possa executar a parametrização desse mesmo canal, utilizando os objectos *Criteria* passados no momento da subscrição. Tendo o conjunto dos atributos passados nesses objectos, podemos computar os atributos mais específicos existentes, sendo a parametrização do canal esses mesmos atributos. No momento de uma nova subscrição, verificamos se os atributos dessa subscrição são mais específicos que os existentes e, em caso afirmativo, parametriza-se de novo o sensor.

Existem alguns sensores cujo acesso segue uma política "*on-demand*", sendo a sua parametrização diferente. A parametrização deste tipo de sensores poderá ser feita através do pedido para uso do sensor, no evento de publicação do canal. Um exemplo deste tipo de sensor é o sensor câmara, o qual so deve ser feito uso quando o utilizador o especificar.

3.3.4 Sensores Suportados

Os sensores são canais de eventos de escopo local, o que se traduz em que os eventos que circulam dentro destes canais são produzidos internamente e consumidos no dispositivo móvel.

GPS

Sensor que acede à API de sensores e retorna a posição GPS. A sua parametrização permite definir os ritmos a que se pretende receber informação vinda de GPS, ou ainda definir se se está interessado em informação GPS quando a posição se altera uma certa distância. O seu acesso é feito de forma normal, isto é, através da subscrição do canal relativo ao GPS.

Acelerómetro

Sensor que acede à API de sensores e obtém os valores do acelerómetro. É possível a parametrização deste sensor definindo a rotação mínima detectada e o ritmo temporal para produção de eventos. O seu acesso é feito através de uma simples subscrição do canal relativo ao acelerómetro.

Tilt

Sensor que informa acerca da orientação do telemóvel, fazendo uso do giroscópio/acelerómetro, com vista a calcular a rotação deste e o *Tilt* mais recente. A sua parametrização possibilita especificar o ritmo temporal a que os eventos devem ser capturados pelo sensor.

WiFi

Sensor que acede ao hardware WiFi para recolher informação sobre os dispositivos adjacentes. A informação capturada traduz-se nas redes WiFi capturadas, a potência de sinal associada e a sua localização GPS. O seu acesso é feito através da subscrição do canal relativo ao sensor WiFi.

Microfone

Sensor que acede ao microfone, capturando excertos de dados áudio. É possível parametrizar o sensor através do ritmo a que se pretenda recolher amostras bem como a duração dessas amostras. O seu acesso é feito através da simples subscrição do canal relativo ao microfone.

Câmara

Sensor que usa a API do sistema de forma a obter imagens através da câmara do dispositivo. O sensor está preparado para ser parametrizado simplesmente sobre o tamanho da imagem a capturar e o formato em que se deve guardar essa mesma imagem. O seu acesso é diferente dos outros sensores, sendo um sensor "*on-demand*" pelo que a sua interação é feita através da publicação de um evento de pedido e os dados são obtidos no evento de *feedback* produzido.

3.3.5 Desenvolvimento de novos Sensores

Para o desenvolvimento de novos sensores são necessárias três etapas: desenvolvimento de um *Channel Template*, desenvolvimento de um *Container* e ainda desenvolvimento de um *VSensor* ou *sensor virtual*.

3.3.5.1 Desenvolvimento de *Channel Template*

Um *Channel Template* tem como objectivo definir como endereçar um fluxo de informação de uma entidade para outra. Para desenvolvimento de um sensor completamente novo será necessário derivar um objecto de *BasicTemplate* que monitorize o contentor respectivo ao sensor em causa, crie os eventos obtidos a partir dos dados obtidos dos sensores virtuais, e os enderece segundo o transporte mais adequado, neste caso, a *local output queue*.

A grande maioria das configurações destes novos *templates* poderá ser feita através da reescrita do método *init()* e/ou dos métodos *pRoute* e/ou *fRoute*.

Para derivar um qualquer sensor dos já suportados para um ambiente específico, simplesmente será necessário implementar dois métodos abstractos dos canais abstractos como está indicado no código em baixo.

Listagem 3.8 Canal de GPS para o sistema Android

```
public class AndroidGpsChannel<E, P, F, Q> extends GpsChannel<E, P, F, Q> {

    protected Container<GpsContainer> getGpsContainer() {
        return Container.byClass(AndroidGpsContainer.class) ;
    }

    protected GpsContainer.Updater getGpsContainerUpdater() {
        return Container.byClass(AndroidGpsContainer.class) ;
    }

}
```

Estes dois métodos têm a função de instanciar o contentor relativo ao sistema para o qual se está a definir o canal.

3.3.5.2 Desenvolvimento de *Containers* para sensores

Um contentor é a componente responsável por instanciar e comunicar directamente com um sensor virtual (VSensor). Para que um VSensor quando tiver informação nova possa informar um contentor, um contentor necessita de implementar a interface *SensorUpdateHandler* que obriga à implementação de um método *update()*. Este método serve para o contentor ser avisado de que pode notificar quem o monitoriza.

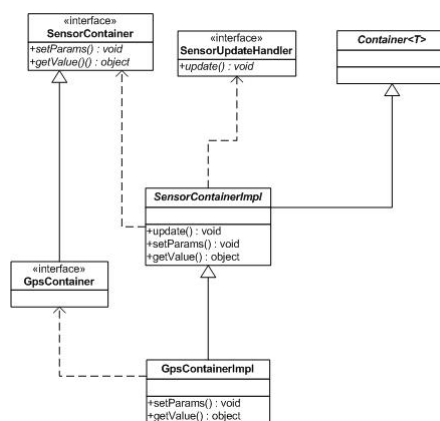


Figura 3.15 Arquitectura de classes para o desenvolvimento de Contentores de Sensores

O modelo apresentado na figura 3.15 ilustra este processo. No caso, para construção de um novo contentor destinado a lidar com um sensor, apenas é necessário declarar uma interface que derive de *SensorContainer* e desenhar um contentor que implemente essa mesma interface e derive de *SensorContainerImpl*, que é o esqueleto para um contentor de sensores.

Estes são os guias, ou caso se prefira esqueleto, a desenvolver para permitir a especificação de um *container* genérico ou *dummy*. Para o desenvolvimento de um contentor para *Android*, por exemplo no caso GPS, o que será necessário será a criação de uma interface que derive de *GpsContainer* e um contentor que derive de *GpsContainerImpl* e implemente a interface respectiva.

Ao desenvolvermos para uma plataforma específica, como Android, precisamos apenas de, no construtor, criarmos o sensor adequado como exemplificado no código em baixo.

Listagem 3.9 Código respeitante à criação de um contentor específico a um sistema móvel

```
public class AndroidGpsContainer_Impl extends GpsContainer_Impl implements GpsContainer {

    public AndroidGpsContainer_Impl() {
        super();
        gps = new AndroidGpsSensor(this);
    }
}
```

3.3.5.3 Desenvolvimento de Sensores Virtuais

Para a criação de novos sensores virtuais, é definido um esquema que pode ser ilustrado pela figura seguinte:

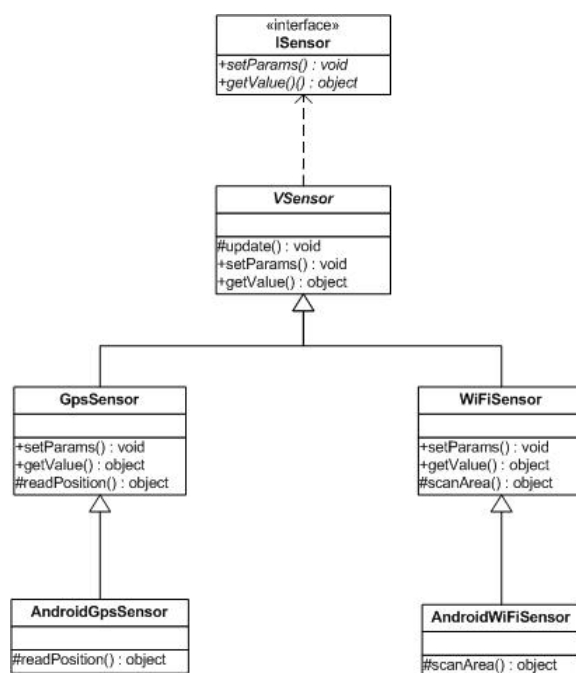


Figura 3.16 Hierarquia de classes de Sensores Virtuais

Este diagrama permite verificar a estrutura de como se pode desenvolver um novo sensor. Assim, é fácil constatar que todos os sensores derivam de uma classe *VSensor*, tendo de implementar os métodos públicos *setParams()* e *getValue()*. Deve ser então desenvolvida uma classe de nível intermédio que permita obter uma estrutura fixa (*template*) da qual se podem derivar então a classe que se pretende desenvolver para um determinado dispositivo. No caso surge o exemplo de dois sensores : *GpsSensor* e *WiFiSensor*.

Este diagrama é respeitante apenas aos sensores que são acedidos de uma forma "normal", isto é, através da subscrição dos canais. Os sensores "*on-demand*" são sensores que geralmente têm interação com o utilizador, como acontece com a câmara fotográfica, o que leva a que o programador ao estender a plataforma desenhe esse seu sensor virtual sendo apenas oferecido o suporte dos *containers*.

3.4 Validação Experimental

A validação experimental faz recurso a um simulador que permite a simulação dos modelos de uma forma semelhante à realidade, podendo correr o mesmo código que está definido para ambiente real.

No ambiente FEEDS, o processamento concorrente é modelado à custa do agendamento de tarefas segundo um escalonador, e a comunicação recorre a transportes. Assim, mudando a implementação destes, consegue-se suportar execução real e simulada em simultâneo, isto é, com o mesmo código dos *templates* dos canais.

Este simulador permite a especificação de várias configurações de rede, sendo possível instanciar o número de nós desejados e simular o seu comportamento, afim de obter métricas de desempenho.

Para este trabalho, a latência da comunicação é uma das questões que foi analisada. Com vista a esse objectivo, o simulador foi configurado para utilizar um modelo de rede em que latência entre pares de nós, posicionados aleatoriamente num plano, é directamente proporcional à sua distância.

O modelo usado é uma aproximação grosseira da realidade e a sua precisão depende muito da escala geográfica considerada. Porém, é de esperar que em muitos cenários possíveis de utilização da plataforma, haverá alguma correlação entre a geografia e a latência. Assim sendo, pretendeu-se avaliar o impacto que o mecanismo de *roaming* tem na latência das comunicações.

Assim, existe uma correlação imperfeita ou limitada entre a geografia e latência. No entanto, caso a correlação se verificar, os resultados podem ser extrapolados para a realidade.

A figura 3.17 mostra uma simulação, na qual está definida uma configuração aleatória. No caso concreto, existem um conjunto de dez nós primários que funcionam como *SliceLeaders* do algoritmo Catadupa, assegurando a componente de auto-organização e auto-regeneração da rede sobreposta. Existe um conjunto de trinta nós secundários que, neste caso têm um papel limitado neste algoritmo. Finalmente existem um conjunto de nós clientes, dez *Homebases* e trinta *Proxys*, e cinco nós móveis que vagueiam pelo cenário.

Para a definição da configuração estão definidas duas sementes, que influenciam a disposição dos nós primários e, consequentemente, de toda a rede. Assim, alterando as sementes, podemos obter configurações diferentes da rede.

A inicialização do simulador é executada através da criação dos nós primários e da sua disposição no plano, posteriormente são desenhados os nós secundários à sua volta. Por fim, são desenhados os nós clientes à volta destes últimos e os nós móveis. Os nós são dispostos de

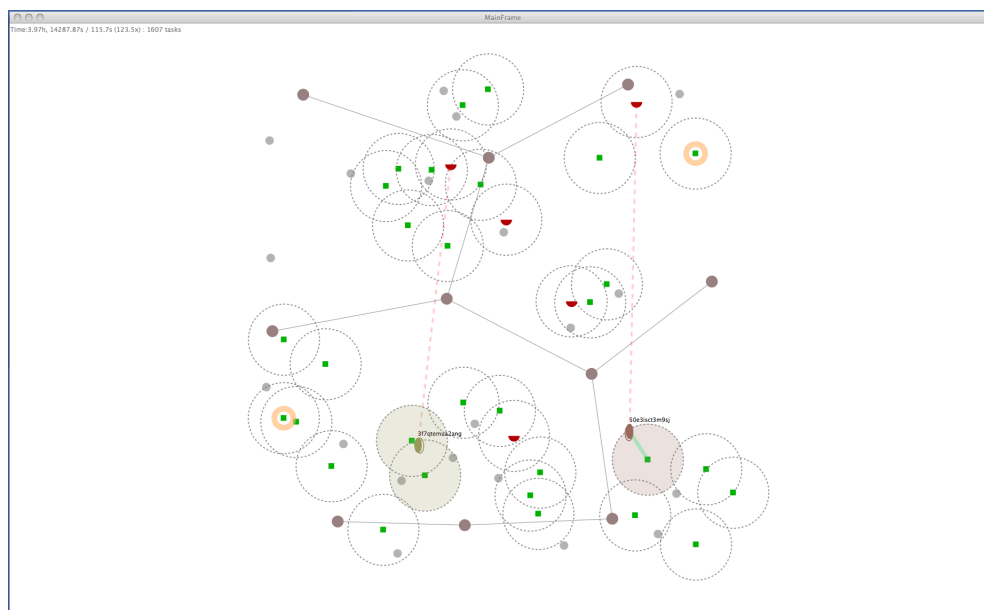


Figura 3.17 Imagem exemplificando uma corrida no simulador

forma a ficarem excessivamente sobrepostos.

O movimento dos nós móveis segue a direcção de um nó escolhido ao acaso (nó atractor) e, ao se encontrar a uma determinada distância (reduzida) desse nó, é escolhido um outro nó ao acaso para tomar esse papel de atractor.

Os valores numéricos acima citados são os mesmos que usados nas várias simulações, tendo sido alteradas as sementes para obtenção das diferentes configurações da rede.

Desta forma, através do simulador descrito, avaliou-se o protótipo desenvolvido, nomeadamente ao nível do algoritmo de *roaming* (suporte de mobilidade).

3.4.1 Avaliação da perda de pacotes e duplicados

Não estando implementado o suporte de fiabilidade definido nos modelos, o qual usa a *Home-base* para guardar os eventos, enquanto um nó móvel executa o processo de troca de um *proxy* por outro, existe a possibilidade que alguns pacotes se possam perder no envio de/para os nós móveis.

Desta forma, duas situações foram testadas: o caso de nós móveis a publicar e nós presentes na infra-estrutura a receber eventos do canal, e o caso inverso.

A experiência consistiu essencialmente na contabilização de quantos pacotes são enviados confrontando-os com o total de pacotes recebidos.

Em respeito ao primeiro caso, no sentido nó móvel - infra-estrutura, não foi detectada duplicação de pacotes mas, em contrapartida, foi detectada perda de pacotes (figura 3.18). Esta perda era esperada, em virtude dos canais activos. Um pacote ao chegar a um *proxy* que ainda

não tenha esse canal activo é perdido, em virtude de o *proxy* ter de executar um *reverse lookup* do canal que, devido ao modelo ser assíncrono, tem a consequência de abrir uma janela na qual ocorre essa perda de pacotes.

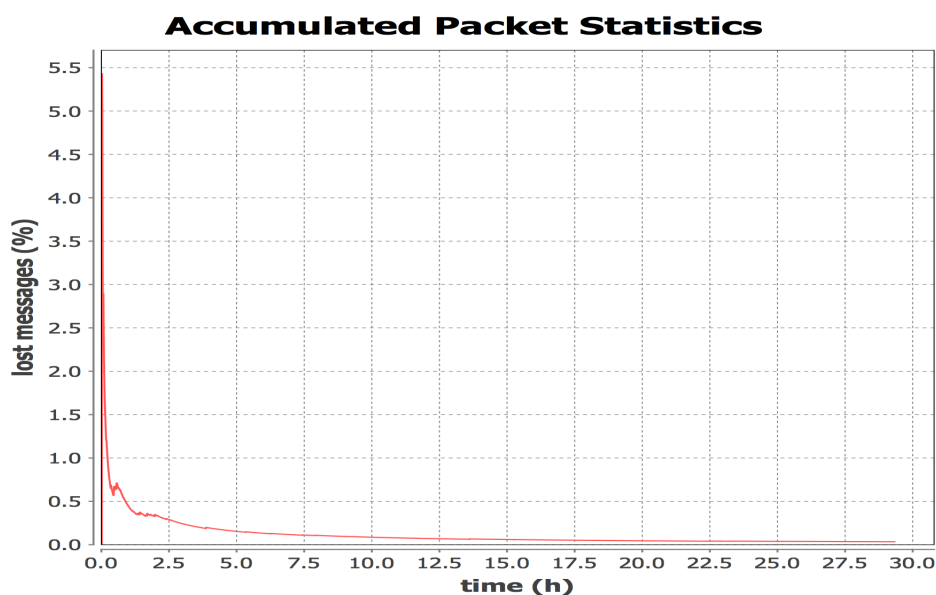


Figura 3.18 Percentagem de pacotes perdidos ao publicar de um nó móvel para a infra-estrutura - Nos instantes iniciais a taxa é elevada devido à não inicialização dos canais nos proxys, reduzindo-se a partir do momento de activação destes nos *proxys*

Este é um problema necessário de resolver e, aparentemente, de solução não muito complexa. Para tal, será necessário implementar um sistema de armazenamento temporário no lado do *proxy*, cuja dimensão não necessita de ocupar muito espaço, em virtude do pequeno intervalo de tempo que demora a activar o canal, guardando os pacotes que sejam recebidos e despejando-os no canal aquando das condições estarem estabelecidas.

O sentido inverso (infra-estrutura - *proxy*) é o caso mais relevante, na medida em que está mais centrado na aplicação que interage com o utilizador. O problema consiste essencialmente no processo de troca de *proxys* por parte de um nó móvel. Relembrando o esquema implementado, um nó móvel tenta estar ligado sempre com o *proxy* mais próximo. O processo de desconexão do *proxy* antigo, dá-se aquando do momento da recepção de um pacote oriundo desse mesmo *proxy* por parte do nó móvel e, verificando-se que não é o actual, se rejeita o pacote e se lhe envia uma mensagem indicando a quebra na associação.

Os testes efectuados indicam-nos que não existiram ocorrências de perda de pacotes nem duplicados neste sentido. No entanto os testes efectuados não foram exaustivos ao nível de explorarem todos os casos possíveis de ocorrer na rede, não podendo garantir por isso a inexistência de perda de pacotes.

Temos a noção que certas condições específicas da rede possam ocorrer situações que levem à perda de pacotes. Um dos casos dá-se quando um pacote já foi enviado há algum tempo mas, por motivos de congestionamento da rede, levou mais tempo a chegar ao *proxy* que, na altura do envio, era o intermediário entre o nó móvel e a rede fixa. Mudando o nó móvel de *proxy* entretanto, esse pacote no momento que fosse entregue seria rejeitado, em virtude da associação com o antigo *proxy* já não ser válida.

Este cenário perfaz um problema a resolver no futuro mais próximo e, acreditamos, a sua resolução não será complexa. Para tal, poderá ser desenhada uma pequena *cache* do lado do nó móvel, que mantém informação sobre os *hashes* dos últimos pacotes recebidos. Esta *cache* declinará pacotes que sejam apenas duplicados, aceitando os pacotes que ainda não tenham sido recebidos. O mecanismo de desconexão ficará no entanto semelhante, indicando as quebras de associação assim que receba um pacote vindo de um *proxy* ao qual não está associado.

Em suma, os indicadores recebidos por esta experiência são animadores, necessitando no entanto de alguns pequenos afinamentos ao nível da implementação, com vista a que o cenário da não existência de duplicados e perda de pacotes se possa confirmar.

3.4.2 Custo médio da comunicação em termos de latência

As experiências realizadas no âmbito da latência envolveram dois casos específicos, isto é, a latência entre um editor e um subscritor, e a latência entre infra-estrutura e nó móvel.

A experiência consistiu em montar um cenário tendo 10 configurações de redes distintas, executando para cada configuração 6 testes cuja diferença é o aumento do raio de alcance dos *proxys* (0, 25, 50, 75, 100, 125), tendo a duração de 20000 segundos para cada simulação. Estas configurações consistem essencialmente em alterar a disposição geográfica dos nós na rede.

No final, para cada teste, realizou-se a média entre as dez configurações de rede, ficando assim com seis séries por cada medição, uma por raio de alcance do *proxy*.

Com este cenário, pretendemos observar qual o impacto do uso de um *proxy* na arquitectura da plataforma ao nível da latência, sendo que para o caso de o raio ser zero, cada nó móvel usa a sua *Homepage* como o *Proxy* mais próximo. A tese defendida é de que quanto maior o raio, menor será a latência.

3.4.2.1 Latência entre *publisher* e *subscriber*

Esta experiência consistiu em colocar uma estampilha temporal nos pacotes que eram publicados por um grupo de nós móveis, calculando posteriormente a diferença com o tempo actual nos nós subscritores. Visto ser executado em ambiente de simulação, o relógio é o mesmo em ambos os casos, não existindo necessidade de sincronização de relógios, ou de um procedimento que meça o RTT dos pacotes.

Os resultados obtidos (figura 3.19), mostram que numa arquitectura sem *proxy* (caso do raio igual a zero), a latência é superior a uma arquitectura com *proxy*. Podemos ainda observar que quanto maior for o raio de cobertura de um *proxy*, mais rápido desce a latência para os valores

mínimos. Por fim, em virtude da latência ser essencialmente derivada de factores geográficos, podemos extrapolar que quanto maior for a distância entre dois nós, maior será o impacto (positivo) que a arquitectura com *proxy* terá.

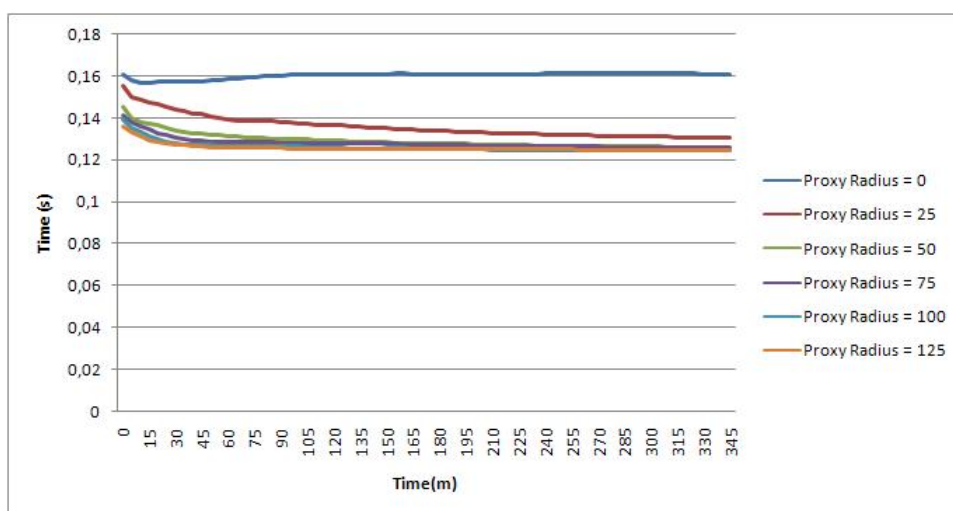


Figura 3.19 Latência Global Média

3.4.2.2 Latência entre *publisher* e infra-estrutura

Para cálculo da latência, colocou-se uma estampilha temporal nos pacotes de controlo do *Tunnel* e, assim que chegados a um nó cliente (infra-estrutura), calcula-se a diferença com o tempo actual.

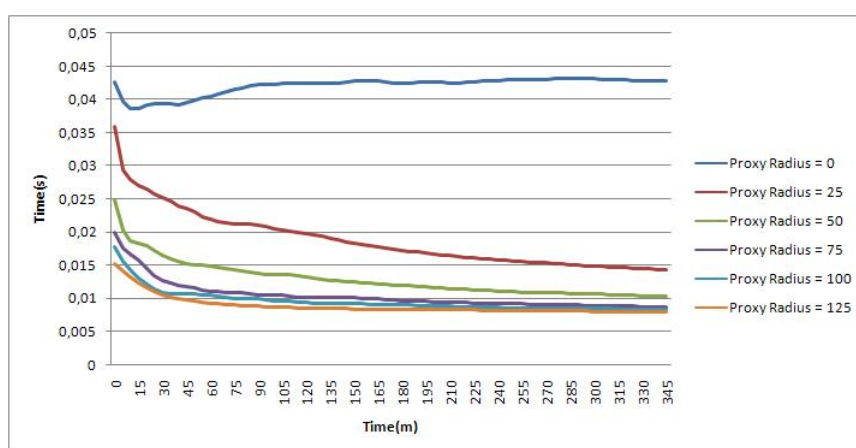


Figura 3.20 Latência Média do Tunnel

Os resultados em relação ao *Tunnel* mostram mais uma vez, que quanto maior for o raio,

menor será a latência. Este era o resultado esperado, visto aquando do raio ter o valor zero, a *Homebase* fará de *Proxy* aos nós móveis, o que não terá a noção de proximidade oferecida pelo *proxy*, levando a que a latência seja superior.

3.4.3 Conclusões

A validação experimental centrou-se à volta da análise do algoritmo de *roaming* e do impacto da arquitectura *Homebase/Proxy* + Tunnel na latência.

Foram conduzidas experiências com vista à detecção da possível perda de pacotes, no que diz respeito ao primeiro caso, e ao nível da latência média entre produtor e consumidor, entre nó móvel e infra-estrutura, no que diz respeito ao segundo caso.

Em relação à experiência de detecção da perda de pacotes e contabilização de duplicados, está detectada uma situação de perda de pacotes no que diz respeito à publicação de pacotes por um nó móvel. Esta situação, é provocada pelo modelo de canais activos, passando a sua solução pela colocação de um sistema de armazenamento temporário no *proxy*. Ao nível da duplicação de pacotes, não foi encontrada qualquer ocorrência.

Quanto à situação de subscrição de um canal por parte de um nó móvel, não foram detectados pacotes perdidos nem duplicados. Este facto é apenas um indicador animador para o futuro, em virtude da não ter sido possível especificar todas as condições da rede nos testes efectuados. Como tal, é possível que possam ocorrer situações de perda de pacotes no processo de transição de *proxy*. No entanto, esta perda não será significativa em virtude da reduzida janela de tempo na qual um nó móvel está sujeito a este tipo de omissões. Em relação aos duplicados o mesmo se aplica, a possibilidade de ocorrência é mínima, mas existe.

Em ambos os casos anteriores, a solução aparenta não ser complexa de implementar, passando pelo desenho de uma *cache* dos últimos pacotes recebidos do lado do nó móvel, devendo ser implementada num futuro próximo.

Ao nível da latência, os resultados obtidos levam-nos a acreditar que a hipótese de arquitectura avançada, beneficia, em termos de latência média, da entidade *proxy*. Apesar de simulada, a latência apresenta correlações com a distância geográfica entre nós e, caso exista reciprocidade entre a realidade e os parâmetros de simulação, os resultados podem ser extrapolados ao caso real.

Em suma, os resultados obtidos não permitem garantir a não duplicação e perda de pacotes, no entanto este é um problema passível de resolução num futuro próximo, através da implementação dos mecanismos enunciados. Quanto à latência os resultados foram positivos e, embora obtidos através de simulação, acreditamos que se possam aplicar à realidade.

4. Caso de estudo

O caso de estudo pretende ilustrar um cenário de uma aplicação *Participatory Sensing* desenvolvida com ajuda do suporte oferecido, sendo composto por duas aplicações.

A aplicação principal pretende, em termos gerais, ilustrar o suporte à computação móvel e ubíqua desenvolvido no contexto desta dissertação, mais especificamente na parte relacionada com a captura e manipulação da informação contextual e sensorial.

O serviço oferecido por esta aplicação, denominada de TouristHelper, tem como objectivo principal informar um turista de possíveis locais de interesse que se situem nas redondezas deste. Os dados obtidos poderão ser tanto de restaurantes, como eventos musicais ou um outro qualquer acontecimento que seja interessante em determinada localização. Neste contexto podemos afirmar que é uma aplicação com um conceito semelhante à aplicação MicroBlog [9].

A segunda aplicação é um ensaio que pretende explorar o modelo de incentivos definido. A sua natureza é um pouco exploratória, e pretende dar uma ideia de como se pode desenvolver uma aplicação que, apesar de externa a todo um sistema de recolha de informação contextual, possa interagir com este, com vista a promover uma qualquer aplicação que esteja nesse sistema.

Essa interacção com outra aplicação é feita ao nível dos dados, mostrando que com o suporte oferecido é possível partilhar esses mesmos dados entre aplicações que usem o suporte de uma forma fácil, podendo esses dados serem em bruto ou já alvo de algum processamento.

A um nível mais concreto, esta segunda aplicação desenvolvida é um *Top* de todas as contribuições executadas por parte dos utilizadores da aplicação TouristHelper. A sua apresentação é feita na forma de uma página Web, no entanto existe toda uma componente desenvolvida por trás, que faz a ligação com a aplicação TouristHelper.

Com estas duas aplicações torna-se possível ter uma visão mais clara de como desenvolver todo um sistema que se baseie em informação contextual, que tenha capacidade de cooperação entre aplicações, podendo ainda estabelecer um caso prático do modelo de incentivos definido.

4.1 TouristHelper

A aplicação TouristHelper tem como objectivo principal mostrar o desenvolvimento de uma aplicação que manipule informação contextual e sensorial, usando o suporte oferecido.

A sua principal funcionalidade é permitir a um turista consultar locais ou acontecimentos que possam ser relevantes para si, denominados de *actividades turísticas*. Assim, ao turista é possível declarar em que categorias de informação está interessado e ainda visualizar essa informação, preferencialmente, na forma de marcas num mapa navegável. Por exemplo, é possível visualizar a indicação de existência de um restaurante em determinada localização, ou então a presença de um concerto, através de um ícone disposto num mapa.

Em termos de implementação a aplicação deposita grande parte da sua complexidade neste

modo de visualização. É possível ver o mapa em diferentes níveis de detalhe, bem como interagir com as diferentes marcas, acedendo a informação mais detalhada sobre a *actividade turística*.

O uso do suporte desenvolvido ofereceu uma camada de abstracção significativa, o que permitiu investir um pouco de tempo no campo da interacção entre o utilizador e a aplicação.

A aplicação não é uma só instância que corre num singular telemóvel, mas sim o conjunto de todos esses dispositivos, bem como todos as *actividades turísticas* capturadas por estes. Sendo uma aplicação distribuída existe a necessidade da concretização de algum tipo de colaboração de forma a que esses mesmos dados capturados possam chegar a todos os interessados. Extraímos daqui duas ideias importantes: a primeira é a de que a aplicação necessita de comunicar entre as várias instâncias, a segunda de que é neste tipo de instâncias, presentes nos dispositivos móveis, que existirá a capacidade de interacção com o meio e captura de informação contextual.

Desta forma, são definidos dois modos de interacção com TouristHelper, um modo *residente* no qual é possível produzir informação e um outro, modo *turista* no qual é possível consumir essa mesma informação.

Fazendo o paralelismo com o modelo das aplicações, esta aplicação que corre no dispositivo móvel corresponde à componente móvel, onde a interacção com o utilizador está presente em ambos os modos de funcionamento. A captura de informação contextual neste caso é feita ao nível do modo residente, em que essa informação contextual é composta por dados provenientes de sensores *hardware*, nomeadamente a posição GPS, e sensores virtuais, nos termos da inserção textual e captura de uma foto por parte do utilizador.

Existe ainda uma componente fixa desta aplicação. Esta componente tem a função de transformação, processamento e persistência de eventos, bem como servir de *Homebase* e todas as suas responsabilidades associadas.

4.1.1 Modo Turista

A aplicação TouristHelper destina-se primariamente a ajudar um turista com as sugestões de locais ou acontecimentos atractivos que ocorram nas proximidades desse turista.

A aplicação, neste modo, é essencialmente apresentada na forma de um mapa, no qual um conjunto de marcas são apresentadas. Essas marcas correspondem a diferentes *actividades turísticas* e têm um ícone representativo da actividade associada. Durante o tempo em que a aplicação está ligada, esta irá apresentando novos eventos que caiam dentro dos interesses de cada utilizador e que sejam provenientes de outras instâncias da aplicação.

A especificação dos interesses de um utilizador são feitos sob a forma de um conjunto de *keywords* que representam os temas de interesse e ainda um campo relativo ao raio de um círculo, correspondente à área geográfica originária dos eventos. O centro dessa circunferência será a posição corrente do utilizador, e aquando da deslocação deste essa mesma área deslocar-se-à também.

Os últimos parágrafos levantam a ideia de distribuição da aplicação. As comunicações são então necessárias ao correcto funcionamento da aplicação e, segundo o pretendido pela

aplicação, na qual as marcas vão surgindo à medida que são produzidas por alguém, existe a sugestão de um modelo de comunicações reactivo, o qual deverá suportar filtragem de conteúdos segundo os critérios especificados pelo utilizador.

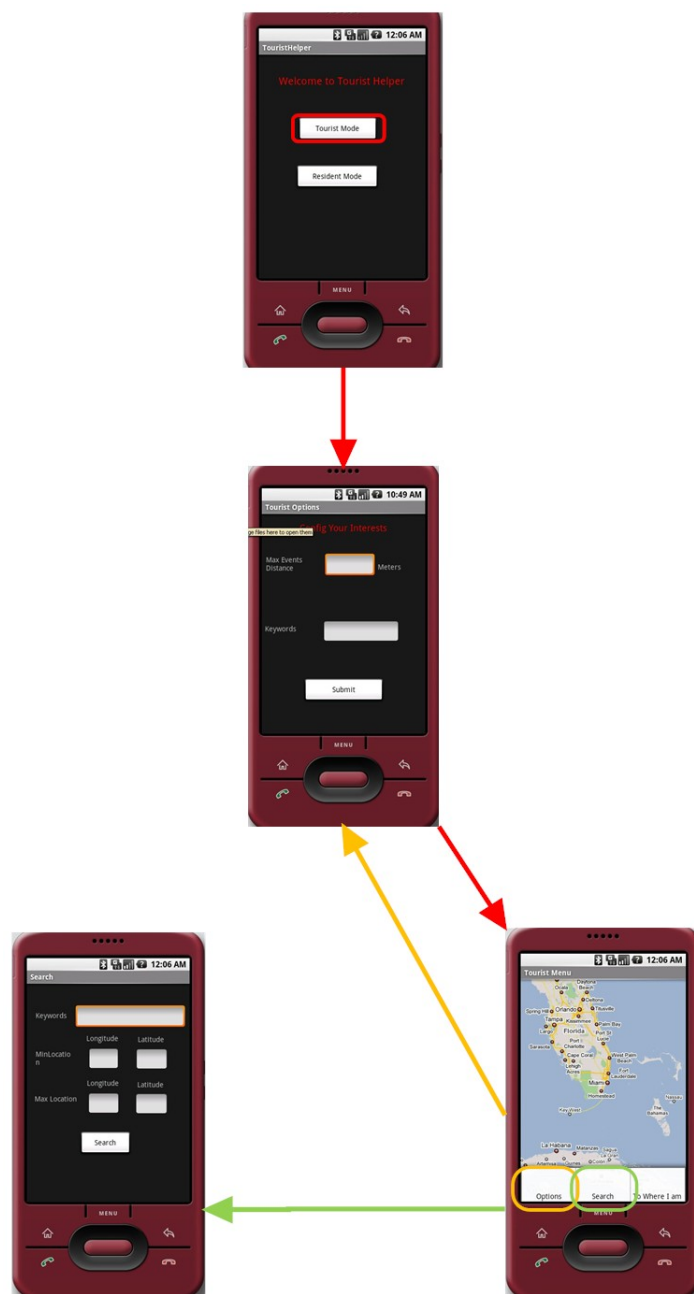


Figura 4.1 Mapa da aplicação TourismTop em modo turista







Icon	Keywords
	restaurant, food
	transports, public transports
	airport
	music, festival
	museum, monument
	valor <i>default</i>

Tabela 4.1 Mapeamento entre ícones e keywords

4.1.1.1 Mapa

Grande parte da aplicação centra-se no modo *turista*, essencialmente ao nível do mapa. É possível "viajar" por todo o globo utilizando os mapas da aplicação, no caso implementadas à custa das APIs do serviço GoogleMaps para Android[11].

No entanto, à data este suporte ainda não era semelhante ao disponibilizado por exemplo para PC, tendo sido necessário desenvolver uma pequena biblioteca, com recurso a algum código de baixo nível, que permite desenhar no mapa um conjunto de ícones personalizados e com capacidade de interacção.

Desta forma, em cada local correspondente a um evento turístico que caia dentro dos parâmetros definidos pelo utilizador, é apresentado um ícone representativo desse evento. Existem alguns ícones pré-definidos, que são colocados no mapa dependendo das *keywords* que caracterizam o evento. Esse mapeamento poderá ser encontrado na tabela 4.1.

O utilizador ao seleccionar um ícone disposto no mapa poderá aceder a alguns detalhes, como o nome e uma figura descritiva do evento (figura 4.2).

Caso o utilizador deseje saber ainda mais sobre um evento, ao clicar sobre um balão, é redireccionado para uma página correspondente a um blogue que descreve esse evento turístico.

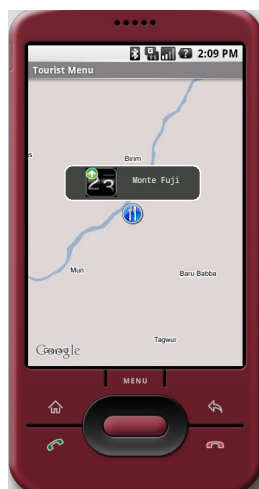


Figura 4.2 Detalhe de um local turistico

Uma aplicação deste género deve privilegiar a interacção com o utilizador, de forma a que um utilizador consiga obter o máximo de informação possível de uma forma intuitiva. Neste contexto, foram definidos três tipos de detalhe possíveis de visualizar no mapa:

Normal É apresentada uma vista na qual é possível visualizar as ruas de diferentes locais, na forma de um mapa, onde será possível observar ícones respeitantes a locais de interesse, fazer *zoom* e movimentar-se pelo mapa.

Satélite Semelhante ao anterior, mas a vista é feita através de imagens de satélite. É possível aceder a este modo clicando duas vezes em cima do mapa quando este se encontra com o nível de detalhe normal.

Rua Neste nível é possível aceder a um nível de detalhe na rua onde se pretender, podendo obter uma imagem do local a 3D, e navegar pelo local como se lá estivesse presente. Assim, e fazendo uso de alguns sensores como bússola ou acelerómetro, é fácil navegar por uma rua "virtual" disponibilizada no ecrã, interagindo manualmente neste último ou simplesmente rodando o telemóvel (provoca a rotação no espaço, como se uma pessoa estivesse parada e tentasse obter uma visão panorâmica do que se situa à sua volta). É possível aceder a este nível de detalhe aquando da presença do nível de detalhe Satélite, clicando duas vezes em cima do local onde se pretenda entrar neste nível de visualização.

Estas funcionalidades avançadas foram executadas tirando partido das APIs disponibilizadas pela Google, mais concretamente, das relativas ao GoogleMaps e GoogleStreetView.

4.1.2 Modo Residente

A aplicação ao correr em modo *residente* permite a um utilizador promover um determinado local. A informação é capturada através da aplicação fazendo recurso aos sensores necessários

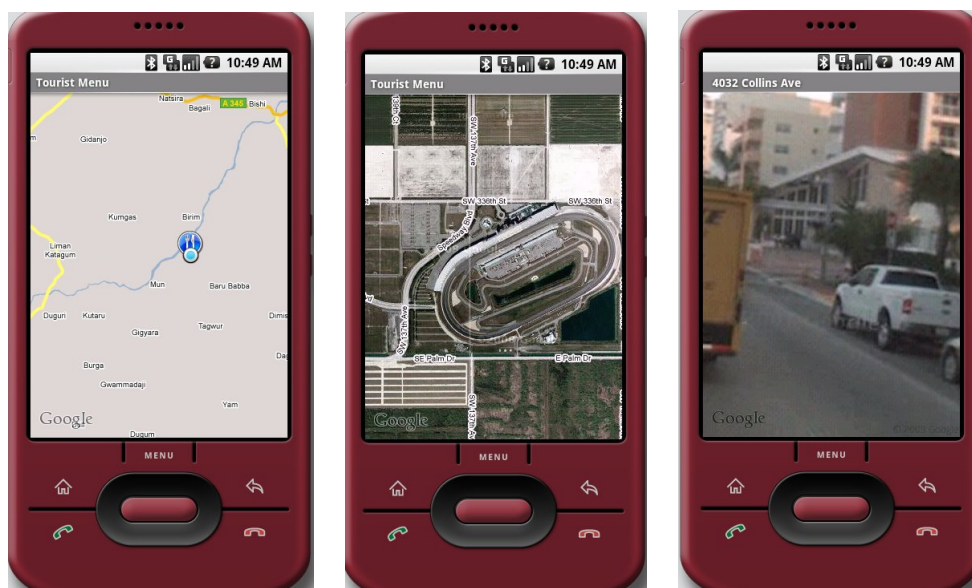


Figura 4.3 Diferentes níveis de detalhe no mapa, normal, satélite e rua respectivamente

e à interacção com o utilizador. Desta interacção é possível extrair texto e dados multimédia.

Na aplicação implementada, presentemente é possível a partilha de eventos turísticos que fazem uso do sensor físico GPS e de um conjunto de dados de texto inseridos manualmente pelo utilizador. É ainda possível documentar este acontecimento turístico com uma foto capturada no momento através da câmara do telemóvel.

Esta partilha poderá ser despoletada tanto por interesse espontâneo de um utilizador, como em resposta a um pedido por parte de um turista. Os pedidos dos turistas são disponibilizados numa lista visível ao utilizador. Cada pedido que está na lista tem as *keywords* que o turista terá submetido, bem como a área geográfica em que este último está interessado. Em termos de configuração, é possível ainda indicar uma opção na qual um "residente" declara que está disponível para que assim que chegue um pedido seja avisado de tal ocorrência.

Os parágrafos anteriores descrevem duas funcionalidades que permitem a ilustração da capacidade de recolha de informação contextual na componente móvel, bem como a colaboração intra-aplicação. Neste último caso, a colaboração segue um padrão *request/reply* conseguido à custa de dois canais, o *canal colaboração* e o *canal de partilha de eventos turísticos*. Ambos se situam no âmbito do escopo global definido anteriormente.

Apesar de neste caso a colaboração ser apenas entre diferentes instâncias da mesma aplicação, o processo para colaboração processa-se de forma semelhante.

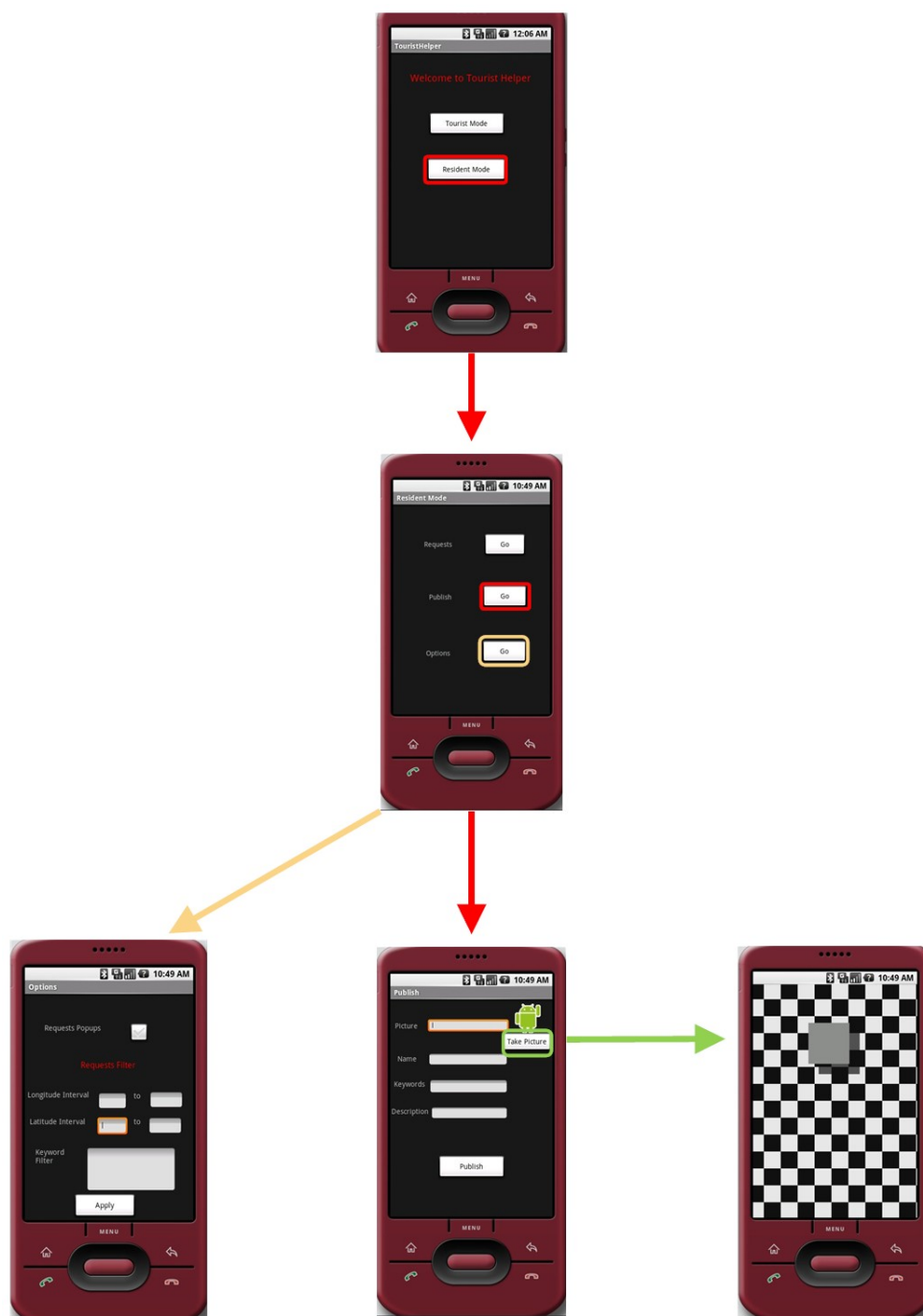


Figura 4.4 Mapa da aplicação TourismTop em modo residente

4.2 Implementação

A necessidade de comunicação por parte da aplicação é satisfeita através do suporte desenvolvido e descrito no capítulo referente ao protótipo. Este suporte permite a comunicação através de um modelo *publish/subscribe*, no qual é possível abstraírm-nos da complexidade dessas mesmas comunicações através do conceito de canal.

Cada canal permite o envio de fluxos de mensagens, agregados sobre o mesmo tema semântico, de tal forma que cada canal tem a sua qualidade de serviço, sendo responsáveis pela entrega dessas mensagens ao destino. Como destino entende-se qualquer interessado no conteúdo de uma mensagem.

A especificação de interesses é feita segundo um filtro, na forma de um objecto *criteria*.

Cada mensagem que flui num canal está dividida em duas partes: *envelope* e *payload*. Na primeira seguem dados relativos às *actividades turísticas*, públicos e que podem ser usados pelo canal para o endereçamento. Na segunda parte do evento poderá seguir qualquer tipo de dados que sejam privados a um evento, e apenas os interessados terão acesso a esses mesmos dados.

Assim a comunicação da aplicação usando o suporte fornecido, simplesmente necessita da subscrição ou então através da publicação de um evento num canal.

Listagem 4.1 Exemplo de uma subscrição de um canal, no caso GPS

```
final Channel<GpsData, Void, ?, ?> channel = Feeds.lookup("/System/Sensors/GpsChannel");
channel.subscribe(new GpsCriteria(10.0, "gps", 2, 5), new Subscriber<GpsData, Void>() {
    public void notify(Receipt r, GpsData e, Payload<Void> p) {
        ...tratar evento...
    }
});
```

No caso ilustrado em cima, a subscrição refere-se a um canal que recolhe informação de GPS, ao qual é passado um objecto *Criteria* que servirá para definir quais os eventos em que estamos interessados, isto é um filtro. É possível ainda não definir nenhum objecto *Criteria*, pelo que se aceitará todo e qualquer um evento produzido pelo canal.

Assim, é possível evitar o uso de APIs nativas e genéricas que, apesar de bastante flexíveis, introduzem uma complexidade demasiado alta e um dispêndio de tempo muito elevado, variando ainda de plataforma para plataforma, impedindo o focar do trabalho nas funcionalidades da aplicação. Com a noção de canal, esta heterogeneidade desaparece, facilitando o desenvolvimento de aplicações em qualquer tipo de sistema, visto o acesso aos recursos, seja esse recurso um sensor ou comunicação, ser efectuado sempre da mesma maneira.

O mapeamento entre os canais existentes e o tipo de mensagens que fluem nesses mesmos canais, e os filtros associados são apresentados de seguida.

4.2.1 Canais, Eventos e Filtros

A aplicação sendo distribuída necessita de comunicar entre as várias instâncias que a compõem. Para tal, e usando o suporte desenvolvido, estas comunicações fazem-se à custa de canais. Os

vários canais são instâncias de *templates* FEEDS/MEEDS que identificam os três escopos definidos: local, global e ad-hoc.

Em cada canal seguem ainda fluxos de mensagens agrupados pela semântica do seu conteúdo, na forma de eventos (*envelope* e *payload*), sendo que é executada a filtragem das mensagens em cada canal segundo os conteúdos e os filtros especificados pelos utilizadores.

De seguida são apresentados os canais, tipos de eventos e filtros utilizados pela aplicação.

Canal GPS O canal GPS é um canal de escopo local, permitindo a recolha de informação associada ao sensor GPS. Na aplicação corrente, este canal é usado para poder catalogar os eventos turísticos que cada utilizador deseje produzir com a sua localização. A produção destes eventos é interna e o seu consumo pela aplicação.

Evento GPS Os eventos GPS ilustram uma posição GPS capturada pelo sensor GPS, circulando no canal GPS. São unicamente compostos por metadados, que contêm informação acerca da posição (latitude e longitude) corrente e prévia, bem como o tempo (data) a que estas amostras foram tiradas.

Listagem 4.2 Constituição de um Evento GPS

```
<GpsEvent>
  <longitude>12.0</longitude>
  <latitude>12.0</latitude>
  <currentTime>123767868.0</currentTime>
  <previousLongitude>11.0</previousLongitude>
  <previousLatitude>18.0</previousLatitude>
  <previousTime>123237912.0</previousTime>
</GpsEvent>
```

Filtro GPS O filtro GPS permite a especificação do ritmo (temporal) a que se pretende receber notificações da posição GPS, bem como ainda a especificação de uma noção espacial, isto é, só existirem notificações quando a posição variar uma certa distância.

Canal Multimédia O canal multimédia pertence ao escopo global, tendo a intenção de endereçar eventos com dados de cariz multimédia até um local de persistência, presente na infra-estrutura. A produção destes eventos ocorre via a componente fixa da aplicação, em que esses eventos seguem caminho até serem guardados em memória persistente.

Evento Multimédia Um evento multimédia é composto por metadados e dados em binário (*envelope* e *payload*). Nos metadados irá o nome, extensão e tamanho dos dados binários.

Listagem 4.3 Constituição de um Evento Multimédia

```
<MultimediaEvent>
  <fileName>filename</fileName>
  <ext>jpg</ext>
  <size>123767</size>
</MultimediaEvent>
```

Filtro Multimédia O canal é acedido apenas na forma de publicação. Os eventos que o percorrem pretendem ilustrar as imagens submetidas pelos utilizadores, sendo apenas enviadas para um canal que os levará a serem armazenados.

Canal de Colaboração Uma das funcionalidades da aplicação a funcionar em modo *turista* é a de permitir a um turista requisitar informação com uma certa categoria para um determinado local. Esta funcionalidade faz uso deste canal que usa um *template* de um escopo global. A produção destes eventos é feita pelos utilizadores em modo turista, e destinam-se aos utilizadores em modo residente.

Evento Pedido de Colaboração Estes eventos têm a intenção de representar um pedido por parte de um utilizador em modo *turista* acerca de uma determinada área geográfica e com um conjunto de conjunto de características em que está interessado (representadas na forma de um conjunto de *keywords*). Estes dados referem-se apenas aos metadados que compõem o *envelope* não existindo qualquer *payload*.

Listagem 4.4 Constituição de um Evento Pedido de Colaboração

```
<RequestEvent>
  <keywords>
    <string>keyword1</string>
    <string>keyword2</string>
  </keywords>
  <min_latitude>10.0</min_latitude>
  <min_longitude>10.0</min_longitude>
  <max_latitude>20.0</max_latitude>
  <max_longitude>20.0</max_longitude>
</RequestEvent>
```

Filtro Colaboração O canal de colaboração é subscrito pelos turistas em modo residente. A sua filtragem é feita de acordo com o especificado por estes nas suas opções, nomeadamente através da especificação de um conjunto de *keywords* em que estão interessados em colaborar bem como a definição da sua área de contribuição (limite máximo mínimo de valores de latitude e longitude).

Canal de Partilha de Eventos Turísticos Este é o canal que permite a obtenção de eventos turísticos a um turista. A subscrição é feita através de um objecto *criteria* que define o filtro de interesses de um turista. Os eventos que forem inseridos neste canal chegarão aos interessados e serão dispostos no mapa com o ícone correspondente. Numa implementação mais ajustada ao mundo real este canal deveria permitir obter simultaneamente os eventos já ocorridos no passado e os do presente, estes últimos em tempo real. No entanto, actualmente apenas é permitido o segundo caso. A noção de passado implica a resolução do problema persistência, que poderá ser feita ao nível do canal e seu *template*, no entanto a persistência por si só é um problema complexo e não abordado explicitamente no âmbito desta dissertação. A persistência está implementada na forma de um serviço que monitoriza este canal e a possibilidade de aceder aos eventos armazenados terá de ser feita explicitamente através de um canal para o efeito (*PersistenceQuery*). Estes eventos, após a transformação de um *evento de publicação turística* pela componente fixa da aplicação, são injectados por esta no canal, sendo enviados a todos os utilizadores em modo *turista* que tenham os seus interesses de acordo com o evento.

Evento Turístico Os eventos turísticos circulam dentro do canal de partilha de eventos turísticos. A sua informação contém dados suficientes para a apresentação de uma marca

num mapa, correspondente a uma *actividade turística*. São compostos unicamente por metadados.

Listagem 4.5 Constituição de um Evento Turístico

```
<TourismEvent>
  <id>nick</id>
  <uri>image description uri</uri>
  <keywords>
    <string>keyword1</string>
    <string>keyword2</string>
  </keywords>
  <description>description</description>
  <eventName>name</eventName>
  <latitude>10.0</latitude>
  <longitude>10.0</longitude>
</TourismEvent>
```

A cada evento está associado um nome, uma localização (latitude e longitude) capturada pelo sensor GPS, bem como a uma lista de *keywords* que permitem a sua catalogação e posterior filtragem no suporte de comunicação. Ainda na composição do evento existe a noção de uma descrição desse mesmo evento através de texto, um URI da imagem que o descreve e o *nick* de quem publicou o evento.

Filtro de Turista O canal ao ser subscrito por turistas em modo *turista* deve poder especificar quais os interesses desses mesmos turistas. Assim, esta especificação faz-se através de um conjunto de *keywords* e através do raio em que o turista está interessado em receber informação.

Canal de Publicação de Eventos A publicação de um evento produzido pela componente móvel não é feita directamente no canal de partilha de eventos. É necessário que exista algum processamento do evento produzido por esta componente através da componente fixa, com o objectivo de adicionar novos campos aos metadados do evento produzido e ainda dividir o fluxo de informação associado aos dados multimédia. Este canal tem por objectivo permitir a publicação de eventos associados a uma *publicação turística*, associando-se a um escopo global, sendo a comunicação restrita à componente móvel e componente fixa da aplicação, através do filtro correspondente.

Evento Publicação Turística Os eventos deste tipo são os eventos produzidos por um utilizador em modo *residente*. A sua existência prolonga-se atingirem a componente fixa da aplicação, onde são transformados noutro tipo de evento. O evento é composto por metadados e dados binários (*Envelope* e *Payload*). Os campos existentes nos metadados, são os preenchidos pelo utilizador, nome de evento, *keywords* de caracterização do evento e descrição, bem como a posição capturada por GPS. Em termos de dados binários segue a imagem descritiva do evento.

Listagem 4.6 Constituição de um Evento de Publicação Turística

```
<PublishEvent>
  <id>nick</id>
  <keywords>
    <string>keyword1</string>
    <string>keyword2</string>
```

```

    <string>keyword3</string>
  </keywords>
  <description>description</description>
  <eventName>name</eventName>
  <latitude>10.0</latitude>
  <longitude>10.0</longitude>
</PublishEvent>

```

Filtro Publicação de Eventos O canal em questão permite levar a publicação de um evento de um dispositivo móvel até à sua componente fixa, para ser sujeito a processamento e transformação e posteriormente injectado no *canal de eventos turísticos*. Assim, o filtro permite a identificação do sujeito que executou uma publicação. Essa identificação tem em conta o *nick* de quem produziu o evento.

Canal Descoberta de Utilizadores Uma outra funcionalidade da aplicação em modo turista apresenta no mapa os utilizadores da aplicação que se situem à sua volta. Este canal permite que essa funcionalidade seja conseguida, em que os eventos recebidos por este canal se referem às posições e identificadores desses mesmos utilizadores. Este canal centra-se no escopo ad-hoc.

Evento Utilizador Os eventos utilizador circulam dentro do canal de descoberta de utilizadores, sendo compostos exclusivamente por metadados que contêm apenas a posição GPS e o identificador desse mesmo utilizador.

Listagem 4.7 Constituição de um Evento Utilizador

```

<UserEvent>
  <id>nick</id>
  <longitude>10.0</longitude>
  <latitude>20.0</latitude>
</UserEvent>

```

Filtro de utilizadores] Não existe qualquer filtro para este canal. A subscrição aceitará todos os eventos que chegam por este canal.

Canal PersistênciaQuery O canal persistência é de escopo global e tem a intenção de permitir a recolha de eventos que estejam armazenados de forma não volátil. Este canal tem nos "extremos" a entidade *utilizador em modo turista*, em termos de injeção de eventos *pesquisa persistência* e obtenção das respostas via *feedback*, e ao nível da subscrição do canal todas as componentes fixas existentes.

Evento de Pesquisa Persistência Um evento desta natureza tem como objectivo permitir a pesquisa de eventos turísticos presentes em memória estável. Circulam no canal persistência, sendo constituídos pelos atributos que compõem o filtro especificado pelo utilizador em modo turista. Apesar de poderem ser usadas *keywords* na *Query*, tal como definido no capítulo anterior, esta *query* apenas está a fazer uso da definição de uma área de pesquisa, e em termos de frescura, isto é, limitar a pesquisa apenas a critérios geográficos e temporais.

Listagem 4.8 Constituição de um Evento Pesquisa Persistência

```
<PersistenceQueryEvent>
  <radius>200.0</radius>
  <latitude>10.0</latitude>
  <longitude>10.0</longitude>
  <fresherThan>1232381239</fresherThan>
</PersistenceQueryEvent>
```

Filtro de Pesquisa Persistência A interação neste canal é feita através da publicação de uma *query* que corresponde a evento. Todas as componentes que subscrevam este canal, *Homebases* com o serviço de persistência activo, especificam os atributos geográficos em que estão segundo os quais estavam dispostos a armazenar os eventos.

4.2.2 Captura de Informação

A captura de informação processa-se no modo residente. Esta captura, como indicado anteriormente pode ser explícita, pela via da vontade espontânea de um utilizador, ou implícita, através da resposta a um pedido de colaboração de um turista.

Em ambos os casos faz-se uso de sensores. Em termos de implementação são usados 3 tipos de sensor: 2 sensores *hardware* e um *sensor virtual*.

Em respeito aos dois primeiros, estes são o sensor GPS e o sensor câmara. Nestes é possível obter a localização com se poderá catalogar um evento turístico ou então capturar uma imagem que ilustre melhor um mesmo evento.

O terceiro sensor diz respeito a um sensor que se engloba na categoria definida por nós de *sensor virtual*. Este sensor refere-se ao próprio utilizador, o qual insere informação textual, que fará parte da informação contextual apresentada a um turista.

Neste caso, está mapeada a capacidade de recolha de informação contextual pela componente móvel da aplicação.

4.2.3 Componente Fixa da Aplicação e *Homebase*

No modelo das aplicações é definida uma entidade Componente Fixa da Aplicação. Em termos de implementação esta entidade toma dois papeis.

O primeiro é servir de *Homebase* à componente móvel. Tal como definido anteriormente, neste papel, a componente fixa tem de suportar os eventos que tenham como destino a instância móvel enquanto esta esteja sem conexão ao sistema.

Desta forma, esta entidade está desenvolvida como um pequeno cliente de Feeds, em que subscreve qualquer canal que a sua instância móvel tenha subscrito, no entanto esta subscrição é feita sem filtros. Como consequência todos os eventos que tenham como destino a componente móvel serão também aqui recebidos, estando esses especificados dentro do filtro da componente móvel ou não. Este ponto é importante para o suporte primitivo de persistência que se pretende oferecer.

A segunda missão desta componente será a de processar os eventos produzidos pela sua componente móvel. Este processamento tratará de converter as publicações de eventos turísticos

produzidos pela componente móvel em eventos que sejam legíveis por outras componentes móveis. O processo indicado na figura 4.5 ocorre nesta componente, dividindo-se o evento produzido em dois eventos distintos.

4.2.4 Submissão de informação associada a um local ou evento

A submissão de informação de um evento é feita num formulário presente na aplicação, na qual são requeridos o conjunto de dados que compõem o *evento de uma publicação turística*.

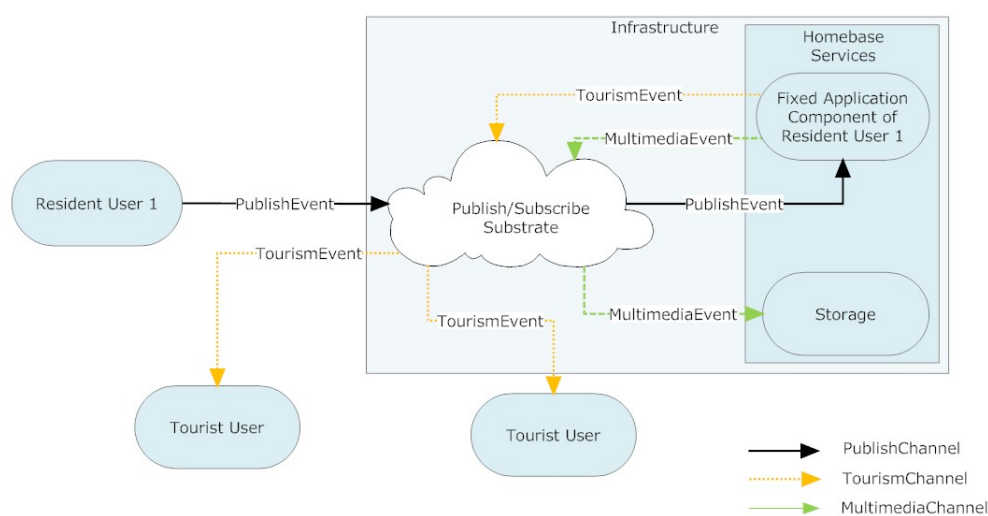


Figura 4.5 Mapa da aplicação TourismTop em modo residente

O destino desta informação será a componente fixa da aplicação. Nesta última entidade enunciada, a informação é partida em dois eventos, um de exclusivamente de metadados e outro com os dados multimédia. Estes dois tipos de informação seguem então destino de duas formas, isto é, através de dois fluxos. A imagem segue por um canal respeitante a dados multimédia com vista a serem guardados em algum servidor. Os restantes dados seguem para a infra-estrutura fixa na forma de um envelope (meta-dados) do evento, segundo o canal de eventos turísticos, com a intenção de serem consumidos pelos interessados.

4.2.5 Consulta de Informação associada a um local ou evento turístico

As consultas de informação num sistema *participatory sensing* devem ter em conta não só os interesses dos utilizadores, mas essencialmente conceitos de mobilidade associados a estes.

Assim, as consultas devem ter em algum aspecto algo que as relacione com a área à volta do utilizador da instância da aplicação.

Desta forma, as consultas de informação turística presentes na aplicação, permitem a definição dos interesses dos utilizadores através da submissão de um grupo de *keywords* respeitantes

aos interesses destes, e um raio à volta do qual estará interessado em eventos turísticos.

Este conjunto de atributos está denominado de filtro, sendo possível de modificar ao longo do tempo, bastando para isso aceder de novo às opções do modo turista.

A *Homepage*, de acordo com o defendido nos modelos, deve guardar os eventos que se dirijam para a sua componente móvel enquanto esta esteja sem conexão do sistema, sendo estes enviados assim que possível.

Por motivos de escassez de tempo este serviço de fiabilidade não é implementado desta forma, sendo compensado por um mecanismo primitivo de persistência.

Este mecanismo de persistência permite mascarar algumas omissões na comunicação através do pedido explícito de re-emissão.

4.2.5.1 Mecanismo de Persistência

A questão da persistência é pertinente no que diz respeito a este género de aplicações. Porém esta é bastante complexa, merecendo um trabalho dedicado explicitamente ao tema.

Apesar deste facto desenvolvemos um mecanismo de persistência que, além do seu próprio valor para aplicação e segundo o qual é possível consultar informação sobre o histórico dos eventos já produzidos, permite compensar a não implementação do modelo de *roaming* com fiabilidade.

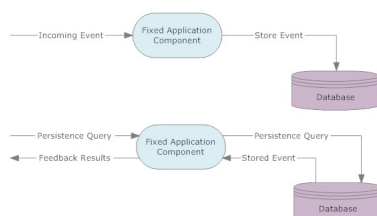


Figura 4.6 Mecanismo de Persistência

Este serviço é desempenhado pela *Componente fixa da aplicação/Homebase*. Assim, na inicialização do serviço de persistência nesta componente, faz-se através da indicação para monitorização do canal de partilha de eventos turísticos, e ainda da especificação de um objecto *Criteria* focado em atributos geográficos, o qual toma a forma de uma figura geométrica, mais concretamente um círculo.

A consulta destes dados é feita através do canal *PersistenceQuery*, através de um evento respeitante a uma *Query*, em que a resposta é devolvida através do mecanismo de *feedback* da plataforma de suporte, na forma de um *DBRecord*, sendo transformado em evento turístico antes de enviado à instância móvel.

4.3 TourismTOP

Esta aplicação ilustra como desenvolver uma aplicação externa que tenha um objectivo de fomentar a partilha de recursos por parte da comunidade desenvolvida à volta de uma aplicação contextual.

No caso concreto, a aplicação TourismTop indica na forma de um Top os maiores contribuidores da comunidade estabelecida à volta da aplicação TourismHelper.

Esta aplicação está dividida em duas componentes. A primeira tem a ver essencialmente com uma camada de apresentação ao utilizador, desenvolvida sobre a forma de uma página Web.

A segunda terá uma missão diferente, na medida em que é esta componente que faz o ponto de ligação com a aplicação contextual.

A existência desta segunda aplicação permite exemplificar como estabelecer as três entidades definidas no modelo de incentivos, utilizando para isso o suporte à computação móvel e ubíqua desenvolvido no âmbito desta dissertação.

4.3.1 Arquitectura e Implementação da aplicação

A aplicação está dividida em vários componentes, tendo um componente de apresentação ao utilizador na forma de uma página Web AJAX, uma componente presente no servidor que fará a interacção com a base de dados e um componente que fará a ligação com a aplicação contextual.

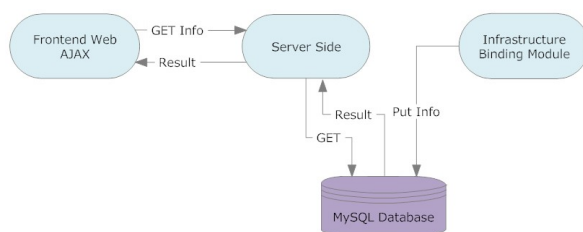
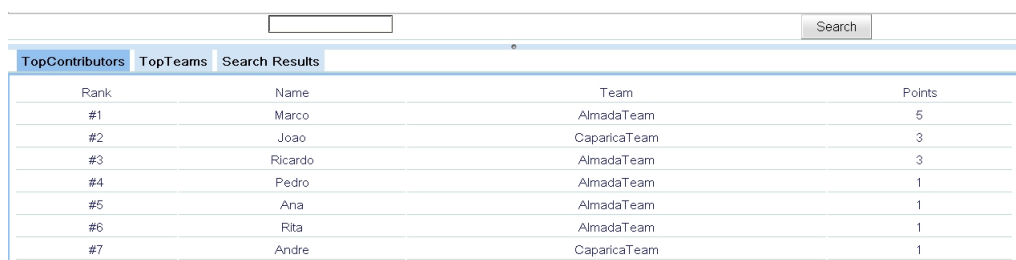


Figura 4.7 Arquitectura da aplicação TourismTOP

4.3.1.1 Componente de apresentação

Nesta componente é apresentada a informação sobre os maiores contribuidores da comunidade de TourismHelper ao utilizador. Esta informação pode ser consultada de uma forma geral, onde é apresentada uma lista de utilizadores individuais, juntamente com o nível de créditos obtidos. Cada crédito será visto como cada contribuição única por parte de um utilizador.



The screenshot shows a web application interface with a search bar at the top right. Below it, there are three tabs: 'TopContributors', 'TopTeams', and 'Search Results'. The 'TopContributors' tab is active, displaying a table with the following data:

Rank	Name	Team	Points
#1	Marco	AlmadaTeam	5
#2	Joao	CaparicaTeam	3
#3	Ricardo	AlmadaTeam	3
#4	Pedro	AlmadaTeam	1
#5	Ana	AlmadaTeam	1
#6	Rita	AlmadaTeam	1
#7	Andre	CaparicaTeam	1

Figura 4.8 Screenshot da aplicação TourismTop

Por outro lado é possível consultar o TOP segundo equipas. Esta ideia segue o a lógica de que uma pessoa se pode associar a uma equipa para promover uma zona, tendo cada utilizador interesse em que esse grupo seja reconhecido.

Por fim será ainda possível pesquisar pelo nome de um utilizador de forma a saber em que posição está no TOP geral de contribuidores, sendo este apresentado exclusivamente.

A informação disponibilizada é refrescada em intervalos de tempo pré-definidos.

Esta componente foi desenvolvida através da ferramenta GWT (Google Web Toolkit)[12], a qual permite tanto a especificação do *frontend* como da aplicação, como a parte que corre em servidor e que será descrita de seguida.

4.3.1.2 Componente servidor

A componente servidor tem como objectivo recolher a informação acerca dos contribuintes a partir da base de dados. Esta responde a pedidos executados pelo *frontend* acerca dos contribuintes.

Componente de ligação à aplicação contextual

Para que seja possível apresentar alguma informação acerca dos contribuintes é necessário que exista algum tipo de conexão com o sistema de recolha de informação contextual.

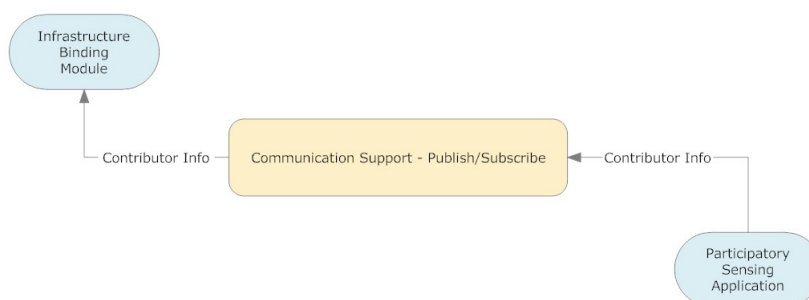


Figura 4.9 Binding da aplicação TOP à aplicação TourismHelper

Assim, esta componente interliga-se com a aplicação TouristHelper através do substrato *publish/subscribe* presente no suporte oferecido às aplicações contextuais. Mais concretamente,

esta funciona como se de um cliente se tratasse e, através da subscrição do *canal de eventos turísticos*, no qual circulam eventos com informação sobre os utilizadores que produziram a informação, captura os dados com que depois preencherá a base de dados. A cada recepção de um evento novo, é contabilizado um crédito ao utilizador (identificado pelo seu *nick*) através da inserção de um tuplo na base de dados.

Esta última ideia mostra que para uma aplicação poder partilhar dados com outra, ou colaborar com outra, apenas necessita de subscrever ou publicar para um canal comum a ambas.

Com esta componente podemos mapear o modelo de incentivos definido. A aplicação TourismTop, em especial esta componente de ligação à aplicação contextual, associa-se à entidade de aplicação incentivo. A entidade aplicação *Participatory Sensing* diz respeito à aplicação descrita anteriormente TouristHelper e os utilizadores aos utilizadores em *desta modo residente*. Os dados em trânsito correspondem aos eventos publicados por estes utilizadores, já transformados, e que circulam no *canal de eventos turísticos*. O modelo fica assim totalmente mapeado, ficando ilustrado tanto como é possível interligar duas aplicações distintas bem como efectuar a mesma partilha e colaboração entre estas duas aplicações.

Seria fácil concretizar um outro tipo de TOP, mais atractivo, que se baseasse por exemplo em apresentar as fotos descritivas dos locais ou acontecimentos turísticos. Para tal bastaria a esse TOP subscrever o canal das publicações turísticas que contém as fotos e mostrá-las segundo a uma interface indicada para o efeito.

4.4 Segurança

Num aplicação deste género a privacidade, bem como a fidelidade e fiabilidade dos dados são conceitos importantes. Nas aplicações implementadas nenhum conceito acima está implementado, isto é, não estão desenvolvidos mecanismos que escondam qual o utilizador que produziu algum evento, nem em nenhum caso está previsto a inserção de eventos errados.

Estas são situações importantes, que não são no entanto, abordadas no âmbito desta dissertação, que se centra mais no suporte de comunicação, colaboração e partilha de informação contextual inter e intra-aplicações.

Assim, estes conceitos de segurança ficarão para trabalho futuro, ou mesmo relegados para um trabalho que aborde esta problemática de uma forma sistemática.

4.5 Validação e Conclusões

O caso de estudo tem por objectivo verificar se os modelos definidos são exequíveis bem como adequados. Desta forma, as duas aplicações construídas mostram o uso desses modelos no desenvolvimento de uma aplicação *participatory sensing* e de uma aplicação incentivo.

Na execução do caso de estudo os modelos provaram-se adequados na medida em que estes foram implementados de forma natural, sem qualquer adaptação.

A implementação da aplicação TouristHelper permitiu a concretização dos modelos definidos para uma aplicação *participatory sensing*. Esta aplicação tem uma componente móvel, a qual corresponde à instância que corre no telemóvel que, tal como definido nos modelos, processa a interação com o utilizador e captura a informação contextual. Tem também uma componente fixa, a qual desempenha as responsabilidades identificadas no modelo da aplicação. A primeira componente beneficia desta, na medida que é nela que é suportado o conceito de *Homepage*, sendo também importante no processo de transformação de eventos e persistência. Assim, concretiza-se a ideia de que a existência de uma componente fixa pode e deve aumentar os recursos não existentes na instância móvel.

Nesta aplicação foi ainda possível verificar que com o modelo *publish/subscribe* se torna bastante mais intuitivo o desenho de uma aplicação deste género. Este modelo permite uma desacoplação implícita das componentes do sistema, o que permitiu o desenvolvimento de cada uma de forma independente. Este modelo permite ainda a transparência ao nível de programação no que diz respeito às comunicações, visto apenas ser necessário a especificação do canal para o qual deseje enviar ou monitorizar informação.

Estes canais permitem ainda agregar informação do mesmo género, o que permite a especificação de diferentes QoS dependendo do fluxo semântico de cada canal. Isto foi conseguido facilmente através da instanciação de diferentes canais, tendo nos canais persistência e de eventos turísticos um exemplo de diferentes QoS e fluxos semânticos.

O modelo de arquitetura *Homepage/Proxy + Tunneling* mostra-se uma forma plausível de implementar um suporte de mobilidade transparente para a aplicação, de tal forma que o programador não necessita de se preocupar com esse problema, podendo concentrar-se nas funcionalidades da aplicação.

A captura de informação sensorial foi facilitada devido ao seu acesso ser feito segundo canais. Esta aproximação tornou a programação da aplicação bastante estruturada, bastando especificar o canal segundo o qual se ao recurso desejado.

A aplicação TourismTop ilustra como concretizar o modelo de incentivos, isto é como criar uma aplicação que dê suporte a outra a nível de incentivos. No entanto não foi possível avaliar o verdadeiro valor deste modelo de incentivos. Esta mesma aplicação permitiu ainda a exemplificação de como partilhar informação entre diferentes aplicações. Para isso, retira-se mais uma vez proveito do modelo *publish/subscribe*, na medida em que um canal se torna partilhado por todas as aplicações que o subscrevam.

5. Trabalho Relacionado

Este capítulo pretende explorar melhor o contexto no qual esta dissertação se irá realizar e indicar qual o espectro das soluções existentes hoje em dia. Tendo em conta que a área em se insere, área de computação móvel e ubíqua, é quase imperativo citar uma famosa frase de Mark Weiser [28]:

- *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"*

Traduzindo, fica algo com um sentido parecido com: as tecnologias mais importantes são as que não se vêem, que se fundem com o dia a dia de cada um de tal forma que o seu uso seja inconsciente.

Desde esta afirmação, os dispositivos com que se pretende tornar realidade esta filosofia, sensores, e hoje em dia *"smartphones"*, sofreram um avanço significativo, tendo actualmente grande poder de computação e recursos disponíveis. Esta última afirmação é verdadeira para a maioria dos recursos com a excepção da capacidade energética disponível [27].

Assim, com vista a atingir os objectivos propostos por Mark Weiser, vários sistemas foram entretanto desenvolvidos. Focando-nos mais no que à recolha de informação sensorial diz respeito, podemos observar dois grandes grupos de aproximações tomadas:

- Software aplicacional com recolha de informação sensorial, desenvolvido para um objectivo específico.
- Plataformas de Middleware que possibilitem uma reutilização de código com vista a facilitar o desenvolvimento de aplicações com base neste paradigma.

5.1 Sistemas Sensoriais Desenvolvidos com um Objectivo Específico

No campo dos sistemas desenvolvidos para um objectivo concreto, podem ser enunciados como objecto de interesse: PotholePatrol [7], BikeNet [6], CenceMe [22], Nericell[23] ou TrafficView [25].

Uma observação mais cuidada acerca destes sistemas permite constatar uma correlação entre os demais citados, isto é, existe uma clara tendência para explorar aplicações com relação com tráfego viário, grupo no qual se incluem PotholePatrol, Nericell, TrafficView.

Uma razão para que isto aconteça, é o facto de se poderem automatizar serviços como a detecção de engarrafamentos, ou acidentes, através de dados fidedignos obtidos por sensores, dispositivos esses que gozam de uma grande cobertura espacial devido ao facto de estarem instalados em veículos.

Com isto em mente, os vários sistemas são descritos nas sub-secções seguintes, e podem dar-nos uma ideia de que tipo de aplicações se podem executar utilizando informação sensorial.

5.1.1 Sistemas Desenvolvidos Com um Objectivo relacionado com Trânsito

5.1.1.1 PotholePatrol

Pothole Patrol é um sistema computacional baseado numa rede de sensores com o objectivo analisar o estado de conservação de uma estrada, sendo que para o efeito, o sistema faz uso de um acelerómetro de 3 eixos (presente num telemóvel) e de GPS, ambos instalados em veículos (neste caso taxis).

O sistema consiste num conjunto de veículos, equipados com sensores, e um servidor central para o qual é enviada toda a informação. O sistema tira partido dos veículos com vista a obter uma grande cobertura espacial, alcançando assim um enorme número de quilómetros de estrada. A detecção de situações anómalas é executada inicialmente por uma máquina de aprendizagem automática que aplica filtros sobre as amostras, e posteriormente através de agregação de forma a descartar falsos positivos. A comunicação é feita via GSM ou WiFi quando possível.

Este sistema mostra-nos um exemplo de uma aplicação interessante e útil, que poderia ser suportada através de MEEDS. São ainda tidas como contribuições o trabalho realizado ao nível do GPS e acelerómetro e, neste último caso, o alerta para que os dados recolhidos dependem da sua localização no veículo.

5.1.1.2 Nericell

O sistema Nericell foca-se na monitorização do tráfego e condições da estrada, utilizando para o efeito, um acelerómetro de 3 eixos, um sistema de localização (por meio de GSM ou GPS) e um microfone, todos eles presentes nos telemóveis inteligentes usados, sendo a comunicação executada através das redes GSM, GPRS ou UMTS. A detecção do trânsito é efectuada tanto pela detecção dos padrões obtidos pelo acelerómetro como pelo microfone que fica à escuta de buzinas.

A sua arquitectura é centralizada, tendo como principais responsabilidades a agregação e processamento para posterior visualização, da informação capturada pelos dispositivos móveis. O sistema foca-se no entanto mais na captura de informação sensorial/contextual, deixando para trabalho futuro a questão da implementação da agregação.

O sistema trata em especial duas situações a que damos bastante valor. A primeira é uma re-orientação dos eixos do acelerómetro a partir de um referencial virtual, através de um conjunto de cálculos segundo uma metodologia conhecida como *ângulos de Euler*. A re-orientação é importante na medida em que ajuda a distinguir as travagem de outros tipos de vibração como por exemplo lombas. A outra área na qual foi feito um trabalho importante é o estudo de outros tipos de localização. Para tal são descritos testes de localização sobre GSM, segundo o algoritmo SS (*signal strength*) que, no entanto, apresentam taxas de desvio algo elevadas.

O trabalho apresenta ideias interessantes, nomeadamente ao nível do trabalho feito em termos de re-orientação do acelerómetro e ao nível da noção de procura de novos tipos de localização. Por outro lado os testes realizados em termos do padrão obtido pelos acelerómetros

indicam que é relativamente simples distinguir situações de, por exemplo, um utilizador pedestre e outro que se situe num carro. Mais uma vez, este é sistema apresenta um tipo de aplicação que pode ser desenvolvido no suporte oferecido.

5.1.1.3 TrafficView

TrafficView é uma plataforma que tem como objectivo disseminar e recolher informação de veículos que se desloquem na estrada. O sistema debruça-se essencialmente, em discutir como captar e agregar a informação, bem como no processo de disseminar esta informação em movimento. A questão da disseminação da informação em movimento é um desafio acrescido, visto em comunicações de curta distância existirem alguns problemas, nomeadamente ao nível do Bluetooth[24]. As mensagens em difundidas têm a função de informar os condutores da ocorrência de acidentes e/ou situações anómalas na estrada, através da adição de informação complementar no GPS tradicional.

Para tal, o sistema utiliza a nível de arquitectura, simplesmente um computador Compaq iPAQ com um receptor de GPS e uma placa de Wireless por veículo, apresentando um modelo distribuído.

As principais contribuições do sistema TrafficView são os estudos realizados em termos de difusão de conteúdos, bem como a ideia reforçada que é preciso ter em conta os limites das tecnologias usadas para comunicação no sistema, levando à necessidade de agregação dos dados a um nível semântico. Estes são problemas relevantes para o trabalho desenvolvido nesta dissertação.

5.1.2 Outros Sistemas Sensoriais

No entanto, e tal como previsto por Mark Weiser, outras áreas de interesse podem beneficiar dos sistemas com base em informação sensorial. BikeNet mostra-nos um sistema que mapeia actividade de ciclistas através de uma rede móvel de sensores, CenceMe mostra-nos como relacionar informação captada através de sensores de telemóveis e como a combinar com redes sociais como MySpace ou Facebook.

Estes sistemas mostram que, com criatividade, podemos criar um qualquer tipo de aplicação com base nestes dados sensoriais, que poderão almejar tanto a utilidade funcional como mesmo o lazer.

Ambos os sistemas, BikeNet e CenceME, e um outro de características semelhantes, Micro-Blog, são descritos nas páginas seguintes, procedendo-se depois à análise do resto do espectro existente da área: as plataformas de código reutilizável.

5.1.2.1 BikeNet

BikeNet[6] é um sistema que visa mapear a actividade de ciclistas, através de uma rede de sensores móvel. BikeNet faz uso de um Moteiv Tmote Invent ¹, personalizado e de um Nokia N80 para as suas capacidades sensoriais. Como funcionalidades suportadas destacam-se o suporte de persistência, do mapeamento da actividade do ciclista e ambiente em redor deste, análise dos dados totais recolhidos, personalização dos dados a recolher pelos sensores, pesquisa remotas despoletadas através de um portal Web, onde a informação (ex: fotos) é apresentada ao utilizador.

A arquitectura é composta por três camadas, uma com um conjunto de servidores com grande capacidade de processamento e armazenamento, uma outra camada denominada de SAP, a qual faz a ponte com os dispositivos móveis, como que uns *gateways* que permitem a interacção mais próxima com os dispositivos móveis. A terceira camada é a responsável pela captura de informação sensorial/contextual, através de dispositivos móveis. Esta última camada tem ainda o objectivo de permitir algum processamento da informação capturada antes do envio para a infra-estrutura.

Em termos de comunicação, este sistema faz uso ainda de técnicas de *DataMuling*, sendo que no caso normal dos dados a comunicação é executada entre camada SAP e dispositivos móveis segundo *Bluetooth* e *WiFi*.

O trabalho realizado ilustra mais uma aplicação que pode ser suportada pelo *middleware* desenvolvido. Esta aplicação é de um âmbito mais social, alargando o escopo de aplicações que caem neste paradigma. Por outro lado, podemos retirar semelhanças ao nível da arquitectura, na qual a camada SAP efectua uma operação semelhante ao *Proxy*, entidade presente no modelo da arquitectura.

5.1.2.2 MicroBlog

Os telemóveis são dispositivos com capacidade de recolha de informação contextual através dos sensores que possuem. Devido a terem capacidade de inteligência é possível computar e agregar informação nestes, e assim construir aplicações centradas na pessoa. Com estas capacidades, e inúmeros telemóveis espalhados pelo mundo, é possível criar uma visão do mundo em geral, através da criação de múltiplos blogues que descrevem possíveis locais interessantes. MicroBlog [9] é um sistema cujo objectivo é que perfazer esta missão.

A informação é transportada dos dispositivos móveis até uma base de dados, acessível pela Internet, por uma qualquer estrutura de rede sem fios. Os blogues são então colocados numa plataforma geo-espacial como o GoogleMaps. Os utilizadores podem ainda fazer zoom sobre locais e ver informação mais detalhada, ou usar WebServices para aceder a toda esta informação. É possível executar um pedido sobre uma localização não catalogada, pedido esse que será apresentado aos utilizadores que estejam em condições de responder. Por outro lado, existem ainda *microblogs* que ao passarmos no local onde estes se situam, estes aparecem no dispositivo

¹Um dispositivo sensorial estático

do utilizador que por lá passe, de forma a obter a sua atenção.

MicroBlog utiliza uma arquitectura tradicional de cliente-servidor, em que os dispositivos de captura de informação contextual (telemóveis) enviam os dados catalogados com a posição geo-espacial e temporal directamente para esse mesmo servidor. Os telemóveis estarão sempre ligados ao sistema e, periodicamente, informam o servidor da sua posição. Aquando de uma *query*, o servidor irá verificar se existem alguns resultados na base de dados de *microblogs*, caso isso não aconteça, irá averiguar quais dos telemóveis presentes na região especificada estão aptos a responder a *query*, lançando-a de seguida para esses dispositivos alvo.

Este é um sistema que é bastante semelhante ao caso de estudo efectuado. Mostra-nos uma aplicação que tenta tirar partido de informação com relevo ao nível social, de forma a criar simultaneamente uma ferramenta com uma utilidade interessante, por exemplo, ao nível do turismo. Podemos ainda retirar deste trabalho a ideia de que a informação é muito mais simples de aceder se estiver disposta numa ferramenta geo-espacial como o GoogleMaps.

5.1.2.3 CenceMe

O sistema CenceMe [22] tem por objectivo combinar informação recolhida pelos sensores disponíveis nos telemóveis de última geração, com aplicações de âmbito social como MySpace ou Facebook.

O sistema tem uma arquitectura de duas camadas, com agentes sensores (telemóveis) e servidores. A partir dos dados recolhidos pelo acelerómetro, microfone, câmara, GPS, Bluetooth, recursos estes presentes nos telemóveis (no caso em questão Nokia N95), são feitas classificações quanto à actividade actual da pessoa. Estes classificadores, podem trabalhar directamente sobre as amostras recolhidas ou então, através de troca de informação com os servidores (executada através de XML-RPC), cooperando com estes de forma a estabelecer alguma relação lógica.

Através do processamento e de cruzamento de informação, os servidores obtêm relações de um nível de abstracção mais elevado, podendo-se inferir por exemplo através do cruzamento de informação acerca da posição GPS com informação acerca de ocorrência de um evento social, que uma determinada pessoa está nesse evento.

Podemos retirar deste trabalho, a importante ideia de que é possível construir abstracções de nível mais elevado através do cruzamento de informação contextual de várias fontes, mostrando que a criatividade é o limite para a construção de novas abstracções. Este é uma ideia chave que nos levou a idealizar que deve ser possível permitir partilhar informação entre as diversas aplicações que usem o suporte concebido nesta dissertação.

5.1.3 Sumário

Os sistemas apresentados na secção anterior têm como objectivo dar uma noção de que tipo de aplicações podem ser construídas com base em informação contextual. O estudo destas aplicações mostra também o que a plataforma a desenvolver, idealmente, deve suportar, isto

é, definir uma estrutura de recolha de informação sensorial, que permita a recolha de vários e heterogéneos sensores, apresentando simultaneamente uma grande flexibilidade, com vista a adequar a qualidade de serviço a um qualquer tipo de aplicação. Desta forma, dos sistemas enunciados, podemos retirar de PotholePatrol a noção de como se lida com acelerómetros e GPS, bem como problemas em relação à mobilidade e suporte de intermitência de conectividade de rede; Nericell apresenta uma solução muito interessante em relação à re-orientação dos eixos de um acelerómetro num telemóvel; TrafficView mostra os problemas de comunicação com dispositivos em movimento, indicando a agregação semântica como factor de redução de custos de comunicação; BikeNet mostra uma solução criativa, que liga uma actividade social (andar de bicicleta) com um sistema deste tipo, permitindo a partilha de experiências entre utilizadores; CenceMe apresenta uma ideia do que se pode inferir ao cruzar informação sensorial, podendo dar-se como exemplo a funcionalidade de permitir verificar se uma pessoa é sociável ou não.

Estas aplicações mostram a utilidade que sistemas que têm por base informação sensorial e contextual podem oferecer, bem como dar uma perspectiva sobre que aplicações uma plataforma deverá conseguir suportar. Assim, investigámos que género de suporte à construção deste tipo de aplicações existe actualmente e, que diferentes tipos de soluções são propostas.

A próxima secção fará esse levantamento, distinguindo e agrupando as plataformas segundo o modelo em que estão implementadas, ou seja, P2P, Cliente-Servidor e soluções híbridas.

5.2 Plataformas

Tal como referido anteriormente, as plataformas de código reutilizável, são outra área de grande actividade de investigação. No geral, o que se constata é que existem dois grandes grupos nos quais uma plataforma se pode acoplar: plataformas que assentam num modelo P2P e, em contraponto, plataformas que se enquadram num modelo cliente-servidor tradicional.

Serão descritos sistemas, tendo em atenção a estruturação da arquitectura da plataforma, serviços oferecidos e ainda soluções de interesse a problemas que tenham sido detectados.

Desta forma, a estruturação para a exposição destes sistemas, que decorre nas páginas seguintes, prossegue segundo a ordem lógica identificada no primeiro parágrafo, acrescentando ainda uma secção para plataformas que possuem um modelo híbrido.

5.2.1 Plataformas que assentam no modelo P2P

As plataformas que se irão explicar nas próximas sub-secções, contêm muitos pontos em comum, pelo que se irão descrever alguns sistemas em conjunto, quando tal semelhança o justificar. Em termos de afinidades, surge um componente que surge em maior destaque, o *proxy* ou *gateway* (como muitas vezes é chamado). Esta componente tem o papel de permitir a interconexão com os dispositivos móveis, que nem todos suportam comunicação por IP. Em adição a este componente, também se pode referir o uso à plataforma JXTA [21] para camada P2P, usada em várias das plataformas analisadas. Dito isto, irá passar-se à descrição de tais sistemas,

procedendo à identificação de similitudes e diferenças com o trabalho a realizar.

5.2.1.1 Mobile Cheddar - A Peer-to-Peer Middleware for Mobile Devices

Mobile Cheddar [16] centra-se na descrição de um *middleware* para uma rede P2P móvel. O sistema apresentado não é feito de raiz, sendo uma extensão à rede Cheddar P2P [3], acrescentando o suporte a mobilidade. O seu objectivo é providenciar uma API que permita o desenvolvimento rápido de aplicações no modelo P2P, agora com suporte também a mobilidade. Mais concretamente as duas funcionalidades a que o sistema aponta com um maior foco são registo dos recursos do dispositivo móvel na rede e descoberta de recursos noutros dispositivos móveis.

A arquitectura do sistema é composta por nós especiais (*gateways*) e nós móveis, estando os primeiros ligados numa rede sobreposta sobre a Internet, e os segundos ligados aos *gateways* através de Bluetooth. Existe a possibilidade de que um nó móvel desempenhe o papel de *gateway*, desde que para tal tenha capacidade de conexão de longo alcance.

A plataforma suporta um modelo *publish/subscribe*, em que um peer que subscreve um recurso, fica imediatamente apto a publicar esse recurso. actualmente apenas texto e imagens são suportados.

O sistema tem semelhanças com o desenvolvido ao nível do modelo de interacção que segue uma filosofia *publish/subscribe*, apresentando no entanto uma estrutura pouco hierarquizada e por conseguinte pouco escalável. Existe ainda a possibilidade de encaminhar mensagens entre duas redes móveis, que é um dos objectivos desta dissertação (suporte de mobilidade), apesar de no entanto existirem algumas diferenças na tecnologia usada.

5.2.1.2 A Design of Service-Based P2P Mobile Sensor Networks e Generic Communication Structure to Integrate Widely Distributed Wireless Sensor Nodes by P2P Technology

Em [8] e [18], são propostas duas aproximações bastantes semelhantes.

Ambos os sistemas propõem uma *framework* para interligação de redes de sensores móveis, "*service-based*", utilizando o modelo P2P, fazendo por isso recurso ao *middleware* JXTA.

O sistemas, estão organizados de forma a que cada rede de sensores sem fios esteja conectada a um *proxy* (em [18] denominado de P2PBridge), facto que permite maior facilidade no suporte à heterogeneidade dessas redes. Este *proxy*, será um nó pertencente a uma rede P2P conectada através da Internet. Nesta rede P2P, estão ainda presentes nós com papel de clientes do sistema, que podem ser PCs ou telemóveis, sendo que estes últimos ligam-se a um *proxy* que servirá de intermediário entre o dispositivo móvel e a rede P2P. O *proxy*, tem ainda como função, retirar carga do telemóvel, por forma a poupar bateria neste.

Para o funcionamento do sistema, foi desenvolvida uma interface que permite a execução de pesquisas, modificar e adicionar serviços, num estilo SQL. Os comandos são inseridos em SQL, e depois convertidos para uma mensagem XML.

Em [8], adicionam à plataforma JXTA, um modelo de comunicações activo, isto é, um componente que permite identificar, segundo um conjunto de regras definidas pelo utilizador (ABLE Rule Language) e que operam tanto sobre o estado da rede como do sensor (energia),

qual o melhor meio de comunicação disponível, ou o serviço mais indicado ao requerido pelos clientes.

O acesso aos dados sensoriais é feito segundo um modelo de *queries*. Em [8] é feito no estilo SQL, sendo a informação processada pelos *proxys* e convertida num ficheiro XML, sendo difundida pela rede P2P. No caso de [18] o serviço de descoberta dos sensores faz uso do serviço de descoberta da camada P2P (por exemplo uma DHT), podendo aceder aos nós descobertos directamente, inquirindo-os sobre os recursos que deseje. Ao contrário do outro sistema, que obtém a informação "*on-demand*", neste é possível subscrever uma série de atributos em que se está interessado, sendo a respectiva informação enviada até aos nós subscretores.

Ambos artigos são interessantes no ponto de vista do suporte de heterogeneidade de sensores, mas também ainda se apresentam-se ainda algo embrionários. É possível retirar ideias interessantes como a especificação de regras para obtenção de dados sensoriais, relacionando-se ainda com o trabalho desenvolvido nesta dissertação em termos do modelo *publish/subscribe* aplicado à colecta de difusão e informação sensorial.

5.2.1.3 P2PMonitoring

Em [2], é apresentada a arquitectura de um sistema P2P, que tem como objectivo a monitorização de múltiplas redes de sensores sem fios, respeitante ao sistema ShareSense. ShareSense ambiciona ser uma plataforma que permite a interconexão de heterogéneas redes de sensores, com grande capacidade de escala, que permita a sua reutilização para a construção de aplicações com base em informação sensorial.

Em termos de arquitectura está definido por cada rede de sensores um nó especial denominado de *gateway*, que participa numa rede P2P através da internet sobre a plataforma JXTA. Nesta rede P2P participam ainda os *Hosts* de cada utilizador. Para monitorização das redes de sensores é usada a plataforma jWebDust [15]. Assim, as aplicações baseadas neste sistema têm a possibilidade de executar *queries* às redes de sensores.

Um ponto a ressaltar nestas *queries* é o trabalho realizado no processamento destas. Cada uma é analisada e dividida, de forma a que cada *subquery* corra no gateway mais indicado para tal. Estas são *queries* executadas com base em atributos geográficos, sendo os resultados apresentados na plataforma GoogleEarth.

A plataforma desenvolvida é, no fundo, mais uma que apresenta uma arquitectura habitual nos sistemas estudados, um *gateway* por rede de sensores que faz parte da rede P2P, viabilizando a interacção entre clientes e sensores. O trabalho dá ainda informação em relação a uma plataforma que lida com sensores, ressaltando-se a ideia das *queries* elaboradas segundo critérios geográficos como interessante. Contudo, o trabalho reflecte-se apenas sobre sensores estáticos.

5.2.1.4 Global Sensor Network

GSN [1] é uma plataforma que pretende oferecer um *middleware* que suporte a gestão e integração dinâmica de redes de sensores, dando a possibilidade de executar *queries* às redes de

sensores, móveis e fixas. A mobilidade é suportada segundo o IEE1451 standard, mais conhecido por *Transducer Electronic Data Sheet* (TEDS).

A arquitectura da plataforma está dividida em várias entidades, consistindo essencialmente em redes de sensores que comunicam com a plataforma através de nós especiais (nós *sink*), que por sua vez está ligado a um PC, com maiores recursos computacionais, que faz parte de uma rede sobreposta definida sobre a Internet. É ainda disponibilizada uma camada que permite o acesso à GSN através da WEB, com controlo de acessos para prevenção de acesso indevido.

De forma a que exista maior simplicidade, no que às redes de sensores diz respeito, é criada a noção de sensor virtual. Um sensor virtual pode ser um qualquer dispositivo que produza dados, ou uma combinação de sensores virtuais, sendo este descrito na forma de metadados num formato XML.

As *queries* estão divididas da especificação temporal em que se está interessado, com o intuito de proporcionar uma especificação mais detalhada dos requisitos temporais, por conseguinte, em GSN, está implementada a noção de histórico, permitindo a execução de *queries* que comparem atributos de diferentes intervalos temporais.

O trabalho mais valioso do sistema, assenta no suporte dinâmico de sensores e na abstracção de sensores virtuais, bem como na ideia de executar *queries* sobre intervalos temporais. Em termos de similitudes com o nosso trabalho, a plataforma tem o mesmo objectivo da recolha de informação sensorial e, tentar através da disponibilização de uma API, fomentar e facilitar a construção de aplicações com base nesses dados auferidos, diferindo no entanto nos modelos defendidos.

5.2.2 Plataformas que assentam no modelo Cliente-Servidor

As plataformas não foram desenhadas todas segundo o mesmo modelo. Com o objectivo de explorar um modelo mais estável, bem estudado e fácil de trabalhar, é normal que tenham aparecido as plataformas segundo a arquitectura Cliente-Servidor. Na realidade, estas plataformas apresentam todas uma estrutura bastante semelhante, com pequenas variações entre elas. Se é verdade que as plataformas ficam mais robustas, pela simplicidade e conveniência do modelo, no que concerne à recolha de informação contextual, a sua escalabilidade, em contraposição com o modelo P2P, sofre mais com o crescimento utilizadores de dispositivos móveis. Algumas soluções interessantes foram desenvolvidas e, de seguida, passam a explicar-se os sistemas que cobrem o âmbito dessas soluções.

5.2.2.1 CarTel

CarTel [14], é uma plataforma desenhada para lidar com mobilidade de sensores, disponibilizando ainda uma interface de forma a permitir a visualização dessa informação. Os sensores são instalados em "*objectos*" com mobilidade, tendo por exemplo de maior referência os auto-móveis.

A arquitectura de CarTel foi desenhada de forma a suportar a operação de desconexão dos

componentes móveis, bem como permitir o acomodamento de um número elevado de redes sensoriais e ainda fácil integração de novos tipos que venham a surgir mais tarde.

A arquitectura apresentada utiliza um conceito semelhante ao *proxy* das plataformas P2P, neste caso denominado de *Adapter*, os dispositivos sensoriais equipados em unidades móveis e um servidor central denominado de Portal. A responsabilidade de um *Adapter* é fazer a ponte entre os dispositivos móveis e o servidor central (desacoplando as duas componentes), normalizando ainda os dados obtidos pelos dispositivos de captura sensorial. O portal por sua vez pretende armazenar e processar a informação capturada, e permitir o acesso a esta na forma de uma GUI disponibilizada ao utilizador, para efectuar as suas consultas apresentando os resultados na plataforma GoogleMaps.

A comunicação ocorre tirando partido de tecnologias como WiFi ou Bluetooth. De forma a que a informação seja transmitida de/para os dispositivos sensoriais é possível fazer *buffering* (sub-sistema denominado de *CafNet*) dos dados, existindo também suporte para a técnica de *Data Muling*, em que a informação é disseminada para outros dispositivos que, possivelmente, a farão chegar ao destino pretendido.

Um ponto a referir em relação ao bom trabalho desenvolvido, é o conceito de "*adapter*" (semelhante ao *Proxy*), que não só permite mais facilmente as operações de desconexão, mas a fácil integração e grande heterogeneidade de sensores, podendo ser configurados em *runtime*, tal como um *device driver*. Por outro lado, o sistema armazena os dados na forma de uma base de dados distribuída, facto relevante para uma abordagem a um modelo de persistência.

5.2.2.2 COSMOS

Cosmos [19] é uma plataforma desenhada de forma a permitir a recolha de dados de várias e heterogéneas redes de sensores. A recolha é baseada numa interface chamada *sensor network common interface*. Esta interface estabelece um protocolo normalizado para a comunicação entre as redes de sensores e a plataforma Cosmos.

A plataforma suporta várias funções comuns a aplicações. Estas funções são dependentes da aplicação, nas quais se incluem o acesso e monitorização de sensores, registo e submissão de metadados e ainda suporte para *queries* de dois tipos: "*on-demand*" ou numa espécie de modelo *publish-subscribe*.

A sua arquitectura é composta por um servidor central e por um conjunto de nós que implementam a *sensor network common interface*, ficando responsáveis pela interacção com as várias redes de sensores. A nível de segurança, estão implementadas funcionalidades, tais como o controlo de acessos à plataforma COSMOS por parte de redes de sensores. Existe ainda um monitor de rede que possibilita a monitorização das redes de sensores em tempo real.

Para uso da plataforma é disponibilizada uma API para o efeito, denominada de *Open API*.

É necessário salientar o interesse no motor de regras, em que uma aplicação indica, através de XML, qual a informação contextual em que está interessado capturar, ideia com estreita afinidade com a filtragem, conceito abordado na plataforma DEEDS e que a dissertação irá fazer uso.

COSMOS aparenta ser um *middleware* que pode facilitar a vida no que respeita ao desenvolvimento de aplicações com base em informação sensorial. No entanto não está previsto o suporte de componentes móveis.

5.2.2.3 SenseWeb

SenseWeb [13], é uma arquitectura desenvolvida em cima do princípio enunciado no início deste documento, que defende que um sistema construído em cima de uma comunidade será bastante mais rico em termos de agregado de dados, e que visa possibilitar que várias aplicações concorrentes a partilhem recursos sensoriais entre si. Deste modo, podem obter-se dados a partir de estruturas sensoriais já definidas, melhorar o aproveitamento destas com vista a criar aplicações que possam usufruir destes dados. A grande vantagem desta arquitectura é a que permite o armazenamento dos dados para que outras aplicações possam usufruir desses mesmos.

Quanto à arquitectura, o sistema é composto por várias entidades: Coordenador, Sensores e *Gateways* e *Mobile Proxys*. O coordenador corresponde no fundo a um servidor central. É composto por dois módulos, Tarefas e SenseDB, em que o primeiro tem a missão de processar os pedidos de informação sensorial por parte de um cliente, e o segundo processar, agregando ou sub-dividindo essas *queries*. Os sensores são os responsáveis pela recolha de dados, podendo ser estáticos ou móveis. Os *Gateways* têm a função de interagir com os diferentes e heterogêneos de sensores, escondendo essa complexidade do resto do sistema, sendo o acesso a estes executado através de WebServices. Cada contribuidor de dados de sensores pode manter o seu próprio *gateway* definindo as suas políticas de privacidade e outros tipos de gestão que pense apropriados para os seus dados. *Mobile Proxys* é uma entidade cuja missão é semelhante à dos *gateways*, no entanto é específica para os sensores móveis, tendo como objectivo específico recolher e catalogar os dados destes sensores com a sua localização.

Em Senseweb, certas metas são propostas, como o suporte à heterogeneidade de sensores, escalabilidade, uma política de incentivos e ainda privacidade. A resolução destes problemas não é trivial, e algumas destas metas representam um paradoxo em relação a outras. O problema referente à heterogeneidade a nível de conexão, é resolvido através do modelo de WebServices. Quanto à heterogeneidade de sensores, o problema é resolvido aquando da conexão de um destes a um *gateway*, em que existe a construção de metadados, que descrevem o sensor que se ligou. Sofrendo de alguns problemas de escalabilidade, o sistema efectua funções de agregação de dados antes de os transferir, bem como políticas de gestão dos dados a recolher nos *gateways*, para minimizar esse problema. Um grande problema destes sistemas são os incentivos, ou seja, a questão pela qual alguém tem interesse em contribuir para o sistema. É discutida a necessidade de existirem benefícios para quem participar no sistema, benefícios esses que não são necessariamente monetários. Esta descrição peca por ser demasiado abstracta, num dos termos em o trabalho a realizar nesta dissertação se insere e dá importância.

Por fim, um outro tipo de problema ronda a área da segurança, mais propriamente a privacidade. Devido à existência de dados que o utilizador possa considerar privados, é dada a opção de os partilhar apenas a um grupo restrito ou então enviar apenas sumários desses, de forma

a que tal informação não permita a identificação do emissor. Restam ainda os problemas que podem ser introduzidos pela produção de dados corruptos, em que as soluções previstas são o *rating* por parte da comunidade dos produtores de dados.

Esta plataforma mostra várias semelhanças com a desenvolvida, tanto a nível de arquitectura como em termos dos objectivos a que ambas se propõem alcançar. A diferença em relação à arquitectura, reside num maior esforço em distribuir o armazenamento da informação sensorial segundo um cariz geográfico, procurando situá-los próximo dos locais onde foram produzidos. Outra diferença centra-se no que respeita ao modelo de comunicações: em SenseWeb as comunicações executam-se através de WebServices, ao contrário do nosso sistema.

5.2.3 Plataformas que assentam num modelo Híbrido

As plataformas referenciadas até agora, apresentam ou um modelo P2P puro, ou um modelo Cliente-Servidor. Se ambos os modelos têm problemas, que são colmatados por um e outro, é natural que exista alguma plataforma que recolha o melhor dos dois mundos. As plataformas IrisNet e DEEDS fazem uso desta ideia, tentando resolver os problemas de um e outro modelo, através de uma arquitectura mista, um híbrido entre P2P e Cliente-Servidor.

5.2.3.1 IrisNet

IrisNet [10] apresenta uma arquitectura de duas camadas, a camada de *Sensing Agents* (SAs) e a camada de *Organizing Agents* (OAs), implementando um modelo híbrido entre o modelo P2P e Cliente-Servidor. A plataforma foi alvo de vários casos de estudo, podendo enunciar-se um localizador de espaços de locais de estacionamento e serviço de imagem costeira (no qual é feito vigilância ao oceano nas zonas costeiras). Estes protótipos só puderam ser desenhados pois a plataforma tem um largo número de objectivos a que se propôs alcançar:

- Pretende-se recolher dados de sensores a uma escala elevada e com intenção de guardar histórico, notando-se uma necessidade evidente que os dados fiquem perto das suas fontes.
- Deve ser possível aplicar filtros que actuem sobre um ou mais sensores, alterando ou indicando os dados a recolher.
- Permitir a consulta dos dados sensoriais como se de uma consulta Google se tratasse.
- Tolerância a falhas como *hosts* inatingíveis ou indisponibilidade de recursos previamente disponíveis.
- Proporcionar uma API com uma abstracção elevada que possibilite transparência das operações de recolha de dados dos sensores.

Para uma melhor compreensão do significado de *Sensing Agent* e de *Organizing Agent* a sua definição é apresentada de seguida:

Sensing Agent Um SA, tem como função servir de interface de ligação entre um (ou vários) dispositivo sensorial e a camada de OAs. Numa forma mais abstracta, são como um sensor virtual. Um SA pode ser programável, através de submissões de filtros a serem usados por estes sobre os dados em bruto. Os dados recolhidos são encaminhados para o OA que estiver mais próximo do SA, sendo depois enviados ao OA que seja dono desse mesmo SA.

Organizing Agent Os OAs, são nós nos quais a informação sensorial é guardada depois de recolhida, formando uma base de dados distribuída.

As aplicações construídas em cima de IrisNet são designadas por Serviços. A construção de um serviço é no fundo, a criação de uma série de OAs, ficando cada OA responsável por apenas participar no serviço ao qual está destinado. Note-se que uma única máquina pode correr ao mesmo tempo vários OAs. Cada OA é responsável pela forma como os dados obtidos dos SAs são guardados, bem como no processamento das *queries* do serviço a que se destinam.

Os dados resultantes dos sensores são descritos num ficheiro XML, catalogando os valores com informação geo-espacial e outros atributos de interesse à aplicação. Estes dados são guardados numa base de dados particionada pelos vários OAs, segundo os algoritmos definidos por IrisNet. As *queries* são executadas através de XQuery.

Visto a escala a que o sistema se propõe ser enorme, cada OA faz cache das *queries* que executa, para melhorar a performance, e replicação para tolerância de falhas. Desta forma, certas *queries* podem não reflectir o estado actual da base de dados, no entanto, o sistema permite indicar numa *query* qual o nível de consistência exigido aos resultados.

O sistema apresenta uma grande ambição: a criação de uma rede mundial de sensores. Para essa ambição, muita da ênfase do problema é colocada na sua arquitectura, com a separação entre SA e OA a permitir diferentes topologias P2P a diferentes níveis (Sensores e *Servidores Base de Dados*). Este é de facto um dos pontos em que a semelhança com o trabalho a realizado é maior, no entanto a topologia do trabalho desenvolvido será dinâmica, podendo mesmo, coexistirem várias a serem usadas simultaneamente por uma aplicação. No que ainda respeita a IrisNet, não é explicado como é executada a configuração de topologias em qualquer uma das camadas, bem como o particionamento da base de dados, isto é, se é feito a nível geográfico ou segundo outro critério. De referir ainda que é utilizado um modelo de *queries* original, no que respeita às outras plataformas estudadas: XQuery.

5.2.3.2 DEEDS

DEEDS [5] é uma plataforma que implementa um serviço de disseminação de eventos, extensível e distribuído, que permite o desenho dos modelos de disseminação e qualidade de serviço personalizáveis, isto é, a plataforma permite o seu ajustamento ao modelo que o programador desejar e que mais se adequa aos objectivos deste. O modelo de comunicação utilizado na plataforma é uma variação do modelo *publish/subscribe*.

A plataforma tem como objectivo adoptar um modelo de disseminação de eventos versátil, com vista a garantir que DEEDS seja adaptável a qualquer tipo de aplicação. Desta forma, o sistema utiliza o conceito de canais de eventos activos, em que a disseminação de eventos assenta num esforço cooperativo e cuja arquitectura é baseada numa rede lógica activa, a qual é extensível e personalizável em termos de comunicação, estruturação e qualidade de serviço.

O seu desenho foi efectuado com vista a que DEEDS seja bastante extensível, suportando a adição de novas funcionalidades ao longo do tempo, sob a forma de módulos.

Estão modelados conceitos de segurança, transparência (o uso da plataforma é completamente transparente para as aplicações, não sendo expostas quaisquer comunicações) e heterogeneidade. DEEDS tem ainda uma característica muito importante que é o ser uma plataforma autónoma, isto é, a rede contém propriedade de auto-organização e auto-regeneração, poupando ao programador o problema de manutenção da rede sobreposta.

Modelo

O modelo da plataforma assenta numa rede lógica de três camadas, em que a primeira, conhecida como *backbone tier*, é composta por "nós servidores primários", a segunda por "nós servidores secundários" e alguns nós clientes e, por fim, a terceira será composta exclusivamente por *nós clientes*. A figura seguinte ilustra esta estrutura:

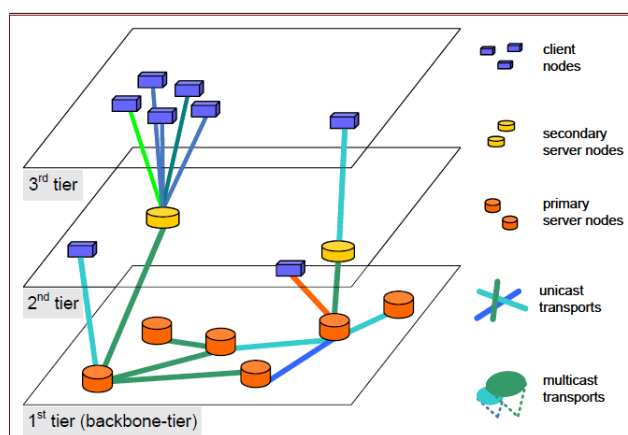


Figura 5.1 Esquema que mostra a arquitectura de 3 camadas de DEEDS, retirado de DEEDS [5]

As várias camadas são uma forma de lidar com os problemas de escala, mantendo ao mesmo tempo uma hierarquização e estruturação eficiente. Esta opção de distribuir carga pelas várias camadas torna-se importante no que concerne à segunda camada, de forma a que a primeira camada, de servidores principais não tenha de lidar com tantos clientes.

Primeira Camada ou *Backbone Tier* Esta camada é formada por um pequeno conjunto de servidores, que estão dispostos numa rede "*tightly-coupled*", cuja função é de

gestão e manutenção de toda a rede sobreposta. É assumido que um nó para pertencer a esta camada terá de possuir altos recursos tanto a nível computacional como a nível de largura de banda, e dar garantias de ser estável, isto é, não ficar *offline* muitas vezes. Com esta descrição, a sua função na rede é quase exclusivamente servir de âncoras ou "*beacons*" para os nós da segunda camada, sendo ainda o repositório de segurança da plataforma.

Segunda Camada Nesta camada, os nós diferem dos primeiros em muitos casos. Por defeito, não são conhecedores dos outros nós da mesma camada, sendo responsáveis por fornecer conectividade aos nós clientes presentes na 3ª camada. Idealmente, estes servidores devem dar conectividade a nós clientes na sua área geográfica, ou um domínio aplicacional específico.

Terceira Camada Esta camada é composta exclusivamente por clientes, que representam as aplicações. Por defeito, e à semelhança da camada anterior, estes nós não conhecem nenhum outro nó da rede, com a excepção que do nó no qual estão pendurados. No entanto este facto não impede a comunicação num estilo P2P entre nós. Caso exista esse desejo, um nó pode descobrir um outro nó através de pedidos aos nós da camada acima, à qual está ligado e, após a descoberta, comunicar directamente com o nó destino.

A descrição acima pode induzir o erro de se pensar que a estrutura da rede será sempre em árvore. Isto em nada corresponde à verdade, pois a plataforma possibilita a reconfiguração dos estratos como se bem entender, através da criação de um objecto do tipo *Event Channel Template*, que permite a definição de como os nós se organizam.

A figura seguinte mostra várias configurações possíveis de obter, não exclusivas, e que podem coexistir na plataforma.

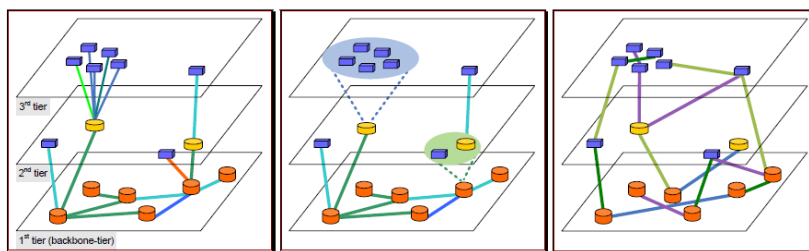


Figura 5.2 Esquema que ilustra possíveis topologias que DEEDS pode ter, retirado de DEEDS [5]

De relevante há a referir o suporte à heterogeneidade de DEEDS. Essa heterogeneidade é suportada na forma de protocolos de comunicação entre dois nós. Não é definido apenas um tipo de comunicação, mas dada a hipótese de construir o *link* e protocolo de comunicação entre dois nós segundo a vontade do programador. Estes protocolos e tipo

de comunicação podem ser definidos à custa da implementação de um objecto do tipo *Transport*.

Particularidades do modelo de comunicação *publish/subscribe/feedback*

Tal como enunciado anteriormente o modelo para comunicações utilizado nesta plataforma não é um tradicional *publish/subscribe*, mas sim uma pequena variação. O modelo implementado adiciona ao tradicional e bem conhecido *publish/subscribe* uma operação *feedback*, e ainda uma outra operação *reroute*.

Esta operação *feedback* permite que os produtores de eventos e consumidores possam iniciar uma comunicação um com o outro em paralelo com o modelo *publish/subscribe*. Por seu turno, *reroute* permite a republicação de um evento já ocorrido.

Esta é uma plataforma que apresenta um modelo reactivo de interacção, tendo como características o facto de ser uma rede *overlay* hierarquizada, com capacidade de auto-organização e auto-regeneração. Não apresenta no entanto suporte para mobilidade nem de interacção com sensores, o que poderá ser reformulado no futuro graças à sua extensibilidade e modularidade.

5.2.4 Sumário

Nesta secção, foram apresentadas várias soluções referentes a *middleware* de suporte a aplicações construídas em cima de informação sensorial. Após a análise deste conjunto de soluções, é notório a dificuldade de conciliar todos os requisitos na mesma arquitectura. Apesar de tudo, são enunciados exemplos de soluções interessantes, em que estes problemas são ultrapassados de uma forma engenhosa. IrisNet e DEEDS são os exemplos estudados que seguem uma filosofia híbrida, oferecendo soluções para grande parte destes problemas relacionados com a arquitectura.

O que se pode constatar é que apenas uma plataforma faz referência a um modelo de incentivos, SenseWeb, não propondo no entanto, nenhuma solução em concreto. Este ponto, em adição à constatação que nenhuma destas plataformas foi adoptada por uma grande comunidade, bem como à escassez de aplicações desenvolvidas neste paradigma, mostra que este factor social é actualmente bastante menosprezado, dando força à nossa tese de que um modelo incentivos associado a uma plataforma flexível e ajustável à aplicação poderá ser um passo em direcção ao futuro.

6. Conclusões

O trabalho realizado nesta dissertação relata o objectivo de modelar e desenhar um suporte que permitisse a facilitar e fomentar a construção de aplicações *participatory sensing*.

A partir da análise de um conjunto de trabalhos na área de *participatory sensing*, nomeadamente ao nível do estudo de aplicações e das plataformas existentes hoje em dia, surgiu a motivação de que era possível fazer algo que pudesse oferecer novas soluções. Por um lado, as aplicações estudadas revelaram-se bastante interessantes, abordando um novo ponto de vista ao nível das aplicações distribuídas, nomeadamente a possibilidade de chegar a todo lado, obter e cruzar informação contextual, de forma a produzir aplicações que sejam de uma utilidade ou de tema recreativo renovado. Por outro, as plataformas de hoje dia aparentam ser insuficientes no objectivo de facilitar e fomentar o desenvolvimento deste tipo de aplicações. Algumas destas plataformas apresentam soluções interessantes e engenhosas, no entanto todas elas apresentam o que nosso entender são lacunas no suporte ao desenvolvimento de aplicações *participatory sensing*. Mais concretamente, as arquitecturas das plataformas são estáticas, sofrendo dos problemas habituais de arquitecturas P2P e cliente-servidor, não oferecendo também QoS ajustáveis a cada aplicação. Notámos também que as soluções ao nível de mobilidade e gestão de recursos apresentam lacunas e, como tal, pensámos oferecer o nosso contributo nesta área. Identificámos ainda a falta de uma componente social nas plataformas desenvolvidas. Esta componente está direccionada com os incentivos necessários a sobrevivência de uma aplicação, isto é, que leve os utilizadores a contribuir com a informação que fluirá pelo sistema e tenha o condão de fazer sobreviver a aplicação.

Em FEEDS vislumbrámos uma plataforma que oferece alguns dos requisitos identificados anteriormente. Mais concretamente, propriedades de auto-organização, auto-regeneração, QoS ajustáveis às aplicações por intermédio do seu funcionamento via *templates* e ainda um modelo de programação reactivo, com um baixo conjunto de primitivas, facilitando a curva de aprendizagem no que diz respeito ao uso do suporte. No entanto esta plataforma não oferecia qualquer serviço ao suporte da mobilidade e interacção com sensores.

Assim, partimos para o desenvolvimento de suporte a aplicações de *participatory sensing*, essencialmente através do desenho de um suporte a mobilidade, comunicações e interacção com sensores. Foi feita ainda uma proposta no âmbito de um modelo de incentivos que levem os utilizadores a contribuir com nova informação e garanta a sobrevivência das aplicações, bem como um modelo de persistência que dá uma noção a futuros trabalhos sobre o caderno de encargos a desenvolver.

De seguida, apresentamos os objectivos enunciados no capítulo da introdução, confrontando-os com o que foi atingido no trabalho efectuado nesta dissertação.

6.1 Objectivos Revistos

Acreditamos que os objectivos a que nos propusemos foram atingidos. O suporte desenvolvido ajuda e facilita a construção de aplicações *partipatory sensing*, facto sustentado através do caso de estudo efectuado, o qual nos ofereceu indicações a possibilidade de implementar uma aplicação deste género à custa do protótipo desenvolvido.

Assim, tendo em conta os objectivos a que nos propusemos relativamente ao suporte a desenvolver, discutimos de seguida os que objectivamente foram alcançados.

Arquitectura A arquitectura definida à custa de *proxys* e *homebases*, permite uma distribuição da carga, sendo ainda um modo de redução da latência ao nível do encaminhamento.

Mobilidade Foram desenvolvidos um conjunto de modelos com vista ao suporte de mobilidade. Este foram concretizados na forma da adição de uma camada à plataforma alvo FEEDS. Através do protótipo e do caso de estudo mostrámos que são uma forma exequível para o suporte de mobilidade.

Fiabilidade Defendemos ao nível dos modelos não perder qualquer informação aquando do *roaming* dos dispositivos móveis. Este é um problema complexo em termos de implementação e não existiu tempo para o seu desenvolvimento ao nível do protótipo. No entanto foi desenvolvido um suporte de persistência embrionário que permite armazenar os dados e, consequentemente, compensar essa falta de fiabilidade ao nível do *roaming*, mascarando ainda os períodos de desconexão. Porém, como consequência, perde-se alguma transparência para as aplicações.

QoS FEEDS é uma plataforma que permite uma qualidade de serviço (qualitativa) ajustável a cada aplicação. Os modelos desenvolvidos tiveram em conta os modelos da plataforma original, estendendo-os mas ao mesmo tempo preservando os requisitos já oferecidos. A arquitectura *Homebase/Proxy + Tunnelling* mostraram-se uma forma possível de fazer a ponte entre o mundo FEEDS e MEEDS, garantindo a interacção entre os dois tipos de *templates* e consequentemente o prolongamento das qualidades de serviço ajustáveis a cada aplicação.

Interacção com sensores Foi desenvolvido de raiz um modelo de interacção com sensores. Este tem mais uma vez em conta os modelos de FEEDS, sendo o acesso a estes sensores efectuado sob a forma de canais, simplificando o acesso a estes diferentes recursos. O seu desenho contemplou a possibilidade da adição no futuro de novos sensores e extensibilidade a outras plataformas, estando toda uma arquitectura de esqueletos desenvolvida com esse intuito.

Modelo de incentivos Foi feita ainda uma incursão no campo dos incentivos. O modelo de incentivos proposto tem o intuito de dar um passo no sentido de colmatar essa falha identificada em todos os outros sistemas estudados. O modelo é ilustrado no caso de estudo,

através da criação de uma aplicação incentivo (TourismTop) para a aplicação contextual (TourismHelper). Apesar de tudo, não foi possível testar a eficácia deste modelo.

6.2 Trabalho Futuro

Nesta secção irão ser enumeradas um conjunto de funcionalidades e actividades identificadas como importantes e prioritárias num futuro próximo. Estas centram-se essencialmente em questões do suporte de funcionalidades de âmbito mais específico, como a segurança e persistência.

Assim podemos enumerar um conjunto de futuras adições a este trabalho:

- Avaliação e refinamento do modelo de incentivos.
- Refinamento do modelo de suporte à persistência, procurando que esta possa eventualmente ser modelada como um atributo de qualidade de serviço dos canais de comunicação, visando maior transparência para as aplicações.
- Adição de uma solução ao nível de comunicações que permita a escolha eficiente e inteligente das diferentes tecnologias que estão ao dispor.
- Modelação e implementação de um suporte de segurança ao nível do suporte oferecido, nomeadamente focando as questões da privacidade dos utilizadores e na fidelidade dos dados contextuais.

Bibliografia

- [1] M. S. A. Aberer, K. Hauswirth. Infrastructure for data processing in large-scale interconnected sensor networks. In *2007 International Conference on Mobile Data Management*, pages 198–205, Distrib. Inf. Syst. Lab., Ecole Polytech. Fed. de Lausanne, Lausanne, 2007.
- [2] A. Antoniou, I. Chatzigiannakis, A. Kinalis, G. Mylonas, S. Nikolettseas, and A. Papageorgiou. A peer-to-peer environment for monitoring multiple wireless sensor networks. In *PM2HW2N '07: Proceedings of the 2nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pages 132–135, New York, NY, USA, 2007. ACM.
- [3] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, and J. Vuori. Chedar: peer-to-peer middleware. *Parallel and Distributed Processing Symposium, International*, 0:252, 2006.
- [4] J. J. Barton, S. Zhai, and S. B. Cousins. Mobile phones will become the primary personal computing devices. *Mobile Computing Systems and Applications, IEEE Workshop on*, 0:3–9, 2006.
- [5] S. Duarte. *DEEDS - A Distributed and Extensible Event Dissemination Service*. PhD thesis, Departamento de Informática FCT/UNL, 09 2005.
- [6] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The bikenet mobile sensing system for cyclist experience mapping. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 87–101, New York, NY, USA, 2007. ACM.
- [7] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *The Sixth Annual International conference on Mobile Systems, Applications and Services (MobiSys 2008)*, Breckenridge, U.S.A., June 2008.
- [8] H.-W. Fang, L.-C. Ko, and H.-Y. Fang. A design of service-based p2p mobile sensor networks. *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, 2:152–157, June 2006.
- [9] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: sharing and querying content through mobile phones and social participation. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 174–186, New York, NY, USA, 2008. ACM.
- [10] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, 2003.

- [11] Google. Android. <http://developer.android.com/index.html>.
- [12] Google. Google web toolkit. <http://code.google.com/intl/pt-PT/webtoolkit/>.
- [13] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8–13, Oct.-Dec. 2007.
- [14] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. K. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006.
- [15] S. N. I. Chatzigiannakis, G. Mylonas. jwebdust : A java-based generic application environment for wireless sensor networks. In *Proceedings of the 1st International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 376–386, 2005.
- [16] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar "a peer-to-peer middleware for mobile devices. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 86–90, Washington, DC, USA, 2005. IEEE Computer Society.
- [17] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, and A. T. Campbell. Urban sensing systems: opportunistic or participatory? In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 11–16, New York, NY, USA, 2008. ACM.
- [18] M. B. M. Isomura, C. Decker. Generic communication structure to integrate widely distributed wireless sensor nodes by p2p technology. In *Proc. of the seventh International Conference on Ubiquitous Computing (Ubicomp 2005)*, September 2005.
- [19] Y. J. L. Marie Kim, Jun Wook Lee and J.-C. Ryou. Cosmos: A middleware for integrated data processing over heterogeneous sensor networks. *ETRI Journal*, 30(5), October 2008.
- [20] S. Mata, M. José, S. Duarte, and M. Mamede. Análise do custo e da viabilidade de um sistema P2P com visibilidade completa. In *INFORUM 2009 - Simpósio de Informática*, Setembro 2009.
- [21] S. Microsystems. Project jxta. <http://www.jxta.org>.
- [22] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, S. Eisenman, H. Lu, M. Musolesi, X. Zheng, and A. Campbell. Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application. In *Proceedings of ACM SenSys 2008*, November 2008.

- [23] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of ACM SenSys 2008*, November 2008.
- [24] P. Murphy, E. Welsh, and P. Frantz. Using bluetooth for short-term ad-hoc connections between moving vehicles: A feasibility study. In *IEEE Vehicular Technology Conference (VTC)*, pages 414–418, 2002.
- [25] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode. Trafficview: traffic data dissemination using car-to-car communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3):6–19, 2004.
- [26] O. Riva and C. di Flora. Contory: A smart phone middleware supporting multiple context provisioning strategies. *Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on*, pages 68–68, July 2006.
- [27] O. Riva and J. Kangasharju. Challenges and lessons in developing middleware on smart phones. *Computer*, 41(10):23–31, 2008.
- [28] M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.