Copyright

DESENVOLVIMENTO DE UMA INFRA-ESTRUTURA DE SOFTWARE PARA SUPORTAR CONCEITOS DE EFICIÊNCIA ENERGÉTICA USANDO SISTEMAS MÓVEIS PRÓ-ACTIVOS

Agradecimentos

Ao Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa pelo apoio financeiro e institucional e pela cedência das instalações onde efectuei grande parte do desenvolvimento deste projecto.

Ao UNINOVA, Instituto de Desenvolvimento de Novas Tecnologias pelo apoio financeiro e institucional na aquisição de hardware.

Ao Tiago pelo apoio no desenvolvimento e troca de ideias sempre úteis para o bom sucesso deste projecto. À sua boa disposição e alegria capazes de fazer superar todos os momentos de frustração.

Ao Ricardo pela troca de ideias que me ajudaram a resolver alguns problemas de implementação e pelo companheirismo ao longo de todo este percurso académico.

A todos os colegas da sala 1.4 do Ed. X da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa por me aturarem ao longo de todos estes meses.

Á minha família pelo seu apoio incansável ao longo de toda a minha vida.

Finalmente, ao Prof. Doutor Celson Lima, orientador da dissertação, por me guiar ao longo de todo o processo de construção desta dissertação, pelas palavras de apoio que foram realmente motivadoras e contribuíram para que nunca desistisse. Pelo espírito de trabalho e de responsabilidade que me incutiu. Pela amizade.

«O valor das coisas não está no tempo que elas duram,
mas na intensidade com que acontecem.
Por isso existem momentos inesquecíveis,
coisas inexplicáveis e pessoas incomparáveis.»
Fernando Pessoa

Resumo

Desde sempre que o homem, como ser pensante e racional, utiliza diversas formas de energia para a sua sobrevivência e bem-estar. Energia, nas suas mais diversas formas, faz mover o mundo e avançá-lo tecnologicamente pelo que o seu consumo é cada vez mais intenso e presente em todo o lado, nas nossas casas, nos nossos automóveis, aviões e navios, sendo essencial para o seu funcionamento. Não se pode, contudo, dissociar o consumo descontrolado que muitas vezes leva a desperdícios energéticos graves.

Os combustíveis fósseis são um recurso energético não renovável e à falta de uma alternativa limpa e capaz de responder às necessidades do mundo, o seu desperdício poderá causar graves problemas em termos de escassez energética no futuro, bem como o grande impacto ambiental que já se faz sentir no nosso planeta.

O propósito deste trabalho é o de desenvolver uma infra-estrutura de software baseada em dispositivos móveis (i.e. telemóveis) que contribua para uma melhor gestão dos recursos energéticos de um edifício, promovendo a eficiência energética, a redução das emissões de gases poluentes e consequentemente a poupança dos recursos energéticos, através da monitorização do fluxo de pessoas pelos diferentes espaços de um edifício, das condições dos próprios espaços e da previsão do fluxo de pessoas nesses espaços numa determinada altura do dia, mês ou estação do ano.

O sistema desenvolvido basea-se no standard IEEE 802.11 (ou Wi-Fi) que especifica um conjunto de normas para as comunicações sem fios em redes locais, com especial ênfase nas redes ad hoc, que por terem uma natureza pró-activa podem ser criadas em qualquer lugar e em qualquer altura. Esta característica das redes ad hoc é especialmente útil para a monitorização continua de diferentes comportamentos sociais que podem levar a desperdícios de energia. Ao se fazer este tipo de monitorização pode-se perceber onde há desperdícios energéticos, quer seja por mau comportamento das pessoas, quer seja por equipamentos mal regulados ou avariados e actuar de uma forma instantânea nesses mesmos sistemas.

Palavras-Chave: Energia, Eficiência Energética, Redes sem fios, IEEE 802.11, Wi-Fi

Abstract

Ever since man as a rational thinking being, use various forms of energy for their survival and well-being. Energy, in its various forms, is moving the world and technologically advancing it, and so, energy consumption is becoming more intense and present everywhere: in our homes, our cars, airplanes and ships, being essential for their operation. We can't, however, dissociate the uncontrolled consumption of energy that leads to severe energetic wastes.

Fossil fuels are a nonrenewable energy resource, and the reluctance of men in using clean and renewable energy sources, can bring severe problems in the future in terms of energetic shortages, not to mention the high environmental impact that we are already experiencing in our planet.

The purpose of this work is to develop a software infrastructure based on mobile devices (i.e. mobile phones) to contribute to a better managment of the energetic resources of a building, promoting energy efficiency, reduction of greenhouse gases emission and therefore energy resource savings. This is achieved by monitoring the flow of people across different areas of a bulding, the conditions of the spaces themselfs and by predicting the flow of people in those spaces at any given time of the day, month or season.

The developed system builds on the IEEE 802.11 standard (or Wi-Fi), that specifies a set of standards for wireless communications in local areas, with emphasis on ad hoc networks. Because of their proactive nature ad hoc networks can be created anywhere and can be used to continually monitor different social behaviours that can lead to energy wastes. By doing this type of monitorization we can understand where energetic wastes are occuring, either by bad behaviour of men, whether by malfunctioning equipments and act instantly on those systems.

Keywords: Energy, Energy Efficiency, Wireless networks, IEEE 802.11, Wi-Fi

Conteúdo

1 Introdução **1**

1.1 Motivação 2

1.2 Objectivos 3

1.3 Contexto de Desenvolvimento 3

1.4 Estrutura 4

2 Redes Ad-Hoc Sem Fios: Conceitos e Aplicações **7**

2.1 IEEE 802.11: O *standard* do WiFi 7

2.1.1 Divisão do espectro electromagnético 8

2.1.2 Versões do IEEE 802.11 9

2.1.3 Segurança 11

2.2 Redes Ad-Hoc: História e características 11

2.3 Dispositivos Móveis 12

2.4 Mobile Ad hoc Network: Redes ad hoc móveis 13

2.5 WSN: Redes de sensores sem fios 14

3 Sistemas de Gestão de Edifícios **17**

3.1 O que é um Sistema de Gestão de Energia? 17

3.1.1 Principais Funções do SGE 18

3.1.2 Características de Hardware 18

3.1.3 Características de Software 19

3.1.4 Capacidades básicas de um SGE 20

3.1.5 Vantagens do SGE 21

3.2 O ANEE como um Sistema de Gestão de Edifício 22

3.2.1 Visão Global do ANEE 22

3.2.2 Pré Requisitos Operacionais 22

4 ANEE: Modelo Conceptual **25**

4.1 Visão Funcional 27

4.2 Diagrama Arquitectural 30

4.2.1 Nível *Core* 31

4.2.2 Nível *Application* 34

4.2.3 Application Programing Interface (API) 36

4.2.4 Universal Tag Deliver - UTD 36

4.3 Diagrama de Entidades e Relacionamentos 36

5 ANEE: Implementação **41**

5.1 Tecnologias 41

5.2 Diagramas de Classes 43

5.2.1 Mobile Device Application - MDA 43

5.2.2 Person Density Controller - PDC 48

5.2.3 Hotspot Application - HA 50

5.3 Diagramas de Sequência 51

5.4 Contabilização do número de pessoas em cada divisão 54

5.5 Diagrama de Entidades e Relações 54

5.6 Map Designer: Criação de mapas de edifícios 56

6 ANEE: Exemplo Ilustrativo de Aplicação **59**

7 Conclusões e Perspectivas Futuras **65**

7.1 Síntese Geral 65

7.2 Contribuição da Investigação 66

7.3 Trabalhos Futuros 66

Bibliografia **67**

Apêndices

A Classe *Sender* **71**

B Classe *Receiver* **77**

C Classe *Serializer* **83**

D Classe *Broadcast* **85**

E Classe *AHPAM* **89**

F Classe *BroadcastArduino* **95**

G Classe *UserProfile* **97**

H Esquema XML da classe *Map* **99**

I Ficheiro XML da classe *Map* **101**

Lista de Figuras

2.1 (a) Rede sem fios com ponto de acesso. (b) Rede Ad-hoc. [Tanenbaum and Wetherall, 2010] 7

2.2 Divisão do espectro electromagnético em canais. [Gauthier, 2009] 8

3.1 Visão Global do ANEE. 22

4.1 Contabilização do número de pessoas por parte do PDC. 26

4.2 Sistema de distribuição de grupos. 26

4.3 Diagrama de casos de uso do ANEE. 28

4.4 Arquitectura do ANEE. 31

4.5 Detalhe da arquitectura do ANEE. 32

4.6 Processo de conversão de (a) objecto em XML e de (b) XML em objecto. 33

4.7 Estrutura da classe sender. 33

4.8 Estrutura da classe receiver. 33

4.9 Diagrama de entidades e relacionamentos do MDA. 37

4.10 Diagrama de entidades e relacionamentos do PDC. 38

5.1 Arduino UNO com WiShield. 42

5.2 Diagrama de classes do MDA. 44

5.3 Diagrama de classes do PDC. 49

5.4 Diagrama de Sequência - Troca de mensagens entre o Marco e a Rita. 52

5.5 Diagrama de Sequência - Visualização dos grupos do utilizador. 52

5.6 Diagrama de Sequência - Envio de convites para um grupo. 53

5.7 Diagrama de Sequência - Recepção de convites de um grupo. 53

5.8 Detalhes de implementação do DER do MDA. 55

5.9 Detalhes de implementação do DER do PDC. 55

5.10 Exemplo de um mapa gerado pelo *Map Designer*. 56

6.1 Listagem de grupos de um utilizador no MDA. 59

6.2 Janela de chat do MDA. 60

6.3 Posição do utilizador mostrada no mapa no MDA. 60

6.4 *Map Designer* para a criação de mapas de um edifício. 61

6.5 Janela principal do PDC sem grupos criados. 61

6.6 Janela de criação de grupos. 62

6.7 Janela principal do PDC com um grupo criado. 62

6.8 Contabilização do número de pessoas por divisão. 63

Lista de Tabelas

4.1 Exemplo de uma tabela de IPs e portos, mantida por cada elemento da rede ad-hoc. 32

5.1 Detalhe da entidade *UserProfile*. 43

5.2 Detalhe da entidade *Group*. 45

5.3 Detalhe da entidade *Map*. 45

5.4 Detalhe da entidade *Level*. 45

5.5 Detalhe da entidade *Space*. 45

5.6 Detalhe da entidade *Space*. 46

5.7 Detalhe da entidade *Broadcast*. 46

5.8 Detalhe da entidade *Sender*. 46

5.9 Detalhe da entidade *Receiver*. 47

5.10 Detalhe da entidade *Serializer*. 47

5.11 Detalhe da entidade *Chat*. 47

5.12 Detalhe da entidade *Groups*. 48

5.13 Detalhe da entidade *Maps*. 48

5.14 Detalhe da entidade *Groups* do PDC. 50

5.15 Detalhe da entidade *Maps* do PDC. 50

5.16 Organização da memória EEPROM do Hotspot 51

5.17 Exemplo da organização da memória EEPROM do Hotspot 51

5.18 Esquema matricial em que se baseia a construção dos mapas do *Map Designer*. . . . 56

Acrónimos

.NET CF .NET Compact Framework. 41

AES Advanced Encryption Standard. 11

AMS Ad-hoc Managment System. 30, 31

ANEE Ad-hoc Networks for Energy Efficieny. 3–5, 17, 22, 25, 27, 30, 31, 34, 36, 41, 43, 51, 54, 59

AP Acess Point. 8

API Application Programming Interface. 3, 30, 31, 36

ASIC Application Specific Integrated Circuit. 15

AVAC Aquecimento, Ventilação e Ar Condicionado. 18

BACnet Building Automation and Control NETworks. 19

BAS Building Automation Systems. 17

BMS Building Management System. 4, 17–22, 25, 29, 35, 62

CCK Complementary Code Keying. 9

CPU Central Processing Unit. 14

DER Diagrama de Entidades e Relações. 36, 54

DM Dispositivo Móvel. 12, 30–35, 41, 57

DSSS Direct Sequence Spread Spectrum. 9, 10

E/S Entrada / Saída. 42

EEPROM Electrically-Erasable Programmable Read-Only Memory. 42, 50, 51

EMCS Energy Management Control Systems. 17

EMS Energy Management Systems. 17

FCC Federal Communications Comission. 9

FHSS Frequency Hopping Spread Spectrum. 9, 10

GloMo Global Mobile Information Systems. 12

GPS Global Positioning System. 65

GUID Globally Unique Identifier. 36, 38, 45, 62

HA Hotspot Application. 27, 29, 30, 35, 41, 42

IEEE Institute of Electrical and Electronic Engineers. 7, 9

IIS Internet Information Services. 42

ISM Industrial, Scientific and Medical. 8, 10

LAN Local Area Network. 7

MAC Media Access Control. 11

MANET Mobile Ad hoc NETwork. 12–15

MDA Mobile Device Application. 25, 27, 29, 30, 34–36, 39, 41, 48, 51, 52, 54, 57, 59, 60

MIMO Multiple Input, Multiple Output. 10

MVC Model-View-Controller. 43

OFDM Orthogonal Frequency Division Multiplexing. 10

PDC People Density Controller. 25–27, 29–31, 35, 36, 38, 39, 41, 48, 50, 54, 56, 59–62, 66

PNG Portable Network Graphics. 56

PRNET Packet Radio Network. 12

QoS Quality of Service. 10

SGBD Sistema de Gestão de Base de Dados. 42

SGE Sistema de Gestão de Edifícios. 17, 18

SRAM Static Random Access Memory. 42

SURAN Survivable Adaptive Network. 12

TCP Transmition Control Protocol. 32, 33, 46, 51, 52

TKIP Temporal Key Integrity Protocol. 11

UDP User Datagram Protocol. 52

UML Unified Modeling Language. 25

US DARPA United States Defense Advanced Research Project Agency. 12

USB Universal Serial Bus. 15

UTD Universal Tag Distributer. 26, 27, 30, 36, 41, 42, 62

VoIP Voice Over IP. 10

WEP Wired Equivalent Privacy. 11

Wi-Fi Wireless Fidelity. 7, 41, 42, 66

WPA Wired Protected Acess. 11

WPA-2 Wired Protected Acess 2. 11

WSN Wireless Sensor Network. 14, 15

XML eXtended Markup Language. 2, 3, 32, 33, 38, 43, 47, 56, 57

Capítulo 1

Introdução

As tecnologias de redes sem fios têm vindo a revolucionar o mundo no que toca a troca de informação entre indivíduos e organizações. A forma rápida e cada vez mais exigente com que os consumidores nos dias de hoje procuram estas formas de comunicação (com crescimentos de 600% em 2009 [Daly, 2010]) exigem que se desenvolvam sistemas capazes de corresponder às expectativas do mercado contribuindo para uma melhor qualidade de vida a nível de segurança, conforto, entretenimento e produtividade.

O rápido crescimento que se tem verificado nos equipamentos móveis, nomeadamente nos smartphones, em termos de processamento, capacidade de armazenamento e capacidades multimédia permite potenciar as tecnologias de redes sem fios para um panorama onde a troca de informação pode ocorrer em qualquer lugar, a qualquer momento [Kunz, 2006].

As redes sem fios ad-hoc não são uma tecnologia nova, existem desde a década de 70 e, com o aparecimento de novas normas, as comunicações sem fios são feitas com cada vez mais segurança. É um tipo de rede onde não existe um ponto central, ou um ponto de acesso, para onde todas as ligações convergem e depois são encaminhadas para o destino. O próprio nome ad-hoc, do latim "para este fim" ou "com este objectivo" [Moraes et al., 2011], diz isto mesmo: uma solução destinada a resolver uma necessidade específica e imediata.

Como se consegue criar uma rede de utilizadores *on-the-fly* é possível trocar informação relevante a diversos fenómenos que ocorram num determinado instante e local, de uma forma quase imediata. De um ponto de vista de alertas podemos avisar todos os utilizadores de uma auto-estrada de um acidente acabado de acontecer ou alertar os vizinhos de possíveis fugas de gás num prédio. Do ponto de vista cooperativo é possível criar grupos de pessoas que discutem sobre os mais variados temas e mostrar a sua localização geográfica quando próximos uns dos outros. Pode-se também verificar algumas vantagens ao nível da eficiência energética pois torna-se possível verificar a densidade de pessoas num determinado espaço, ajustando automaticamente os sistemas de climatização e iluminação. De um ponto de vista comercial (e.g. lojas, cinemas, teatros e restaurantes) é possível divulgar pela rede promoções e catálogos, permitindo um contacto mais próximo com os clientes.

1.1 Motivação

Para que seja possível colocar as redes adhoc num nível em que a sua utilização seja fácil e imediata é necessário criar uma infra-estrutura de software que mantenha a rede adhoc capaz de constantemente trocar informação entre os diversos elementos da rede, sejam eles quem sejam, sensores, telemóveis, computadores portáteis, etc.

Os protocolos de encaminhamento existentes fazem com que a informação transmitida seja encaminhada pelos vários elementos da rede adhoc desde o remetente até ao destinatário com sucesso, mas apenas tratam essa informação de um ponto de vista de muito baixo nível, no sentido em que olham para essa informação como um conjunto de bits e bytes.

É interessante se abstrair deste baixo nível e explorar novas formas de conseguir comunicar entre dispositivos com mensagens que estes dispositivos conheçam, como linguagens universais como é o caso do eXtended Markup Language (XML). Quando as aplicações que têm como base a rede adhoc conseguirem comunicar de uma forma mais simples, então é possível construir-se aplicações com propósitos mais alargados. A preocupação deixa de ser a comunição dentro da rede e passa a ser as funcionalidades da aplicação, aquilo que se oferece ao utilizador final.

No contexto desta dissertação a funcionalidade principal é contribuir para uma gestão mais rigorosa dos recursos energéticos dentro de um edifício, contudo poderiamos ter outras funcionalidade completamente distintas que tivessem como base a rede adhoc, como é o caso do exemplo já dado do sistema de alertas de acidentes nas auto estradas.

Segundo a Associação Portuguesa de Energias Renováveis 66% da energia gasta em edifícios é desperdício [Vilarigues, 2007] e muitas vezes as pessoas não se apercebem deste desperdício e os impactos que têm nos seus orçamentos. A maior fonte de consumo de energia são os sistemas de iluminação com 17% do consumo energético, seguidos dos sistemas de climatização com 15% e de seguida os sistemas de electrónica de consumo com uma fatia de 11% [Roth and McKenney, 2007].

Tendo em conta estes dados é legítimo pensar que é possível automatizar a regulação destes sistemas de modo a reduzir o consumo energético. Pode-se adequar automaticamente as condições físicas de um espaço de acordo com a quantidade de pessoas que lá se encontram. Determinar a densidade de pessoas nas diversas divisões de um edifício, através da rede adhoc formada por essas pessoas, é onde esta dissertação se enquadra.

Se por um lado se percebe em que sistemas se deve actuar, é necessário saber como se deve actuar e em que informação é que se deve basear de modo a tornar estes sistemas o mais eficiente possível. Se é tendo em vista o conforto e bem-estar das pessoas dentro dos edifícios, sejam estes públicos ou privados, então é as pessoas que devem ser a fonte de dados, fornecendo a informação necessária de modo a actuar eficazmente em termos de eficiência energética.

1.2 Objectivos

No sentido de se criar uma rede adhoc cuja forma de comunicação seja o mais transparente possível do ponto de vista das aplicações que usam a rede como base em relação aos protocolos de encaminhamento, o primeiro objectivo é o de criar pacotes de dados baseados em linguagens universais como é o caso do XML.

O segundo objectivo é tornar transparente à aplicação a forma como esses pacotes de dados são enviados através da rede. A aplicação não deve saber como e por que elementos da rede é que a informação passa até chegar ao destino, apenas deve saber se essa informação chegou ou não.

De modo a se poder interligar as pessoas entre si e com os próprios edifícios é necessário utilizar algo que os ligue e que os mantenha conectados. A resposta está no uso de redes ad-hoc sem fios através dos telemóveis das pessoas. Ao se utilizar este tipo de tecnologia é possível trocar informação em qualquer lugar e a qualquer momento. O terceiro objectivo é **permitir terceiros a utilizarem os recursos da rede**, pelo que é necessário desenvolver uma Application Programming Interface (API) para que a ligação à rede ad-hoc sem fios e a troca de mensagens entre os elementos da rede seja o mais transparente possível.

Por último, é necessário **identificar a localização das pessoas nas diversas divisões dos edifícios** pelo que o quarto objectivo é criar um mecanismo que identifique em que divisões é que as pessoas se encontram, para que depois seja possível fazer uma contabilização do número de pessoas em cada divisão do edifício.

Para cumprir todos estes objectivos, foi desenvolvida uma infraestrutura de *software* que se chama *Ad-hoc Networks for Energy Efficieny (ANEE)* que será apresentada ao longo deste documento.

1.3 Contexto de Desenvolvimento

O ANEE permite fazer monitorização dos comportamentos dos diferentes elementos que integram um edifício, sejam estes humanos ou sensoriais. Com base no tratamento dos dados gerados pela monitorização pode-se actuar automaticamente nos sistemas responsáveis pela gestão energética de um edifício, de modo a conseguir atingir reduções nos níveis de energia consumida, necessária para alimentar estes sistemas, bem como adequar as condições do espaço de acordo com o bem estar das pessoas.

O trabalho apresentado nesta dissertação faz parte de um conjunto de três trabalhos. Para além de criar toda a infra-estrutura de software que controla uma rede adhoc que permite uma componente de monitorização dos elementos humanos, nomeadamente a sua distribuição dentro do edifício. Os outros dois trabalhos focam-se, o primeiro na monitorização ao nível das características físicas dos espaços e o segundo na actuação nos equipamentos do edifício de acordo com a

informação recebida pelas duas componentes de monitorização.

O uso das tecnologias sem fios está em crescendo hoje em dia tendo inúmeras aplicações, desde as comunicações de rádio e televisão, as redes sem fios para aceder à Internet em casa, nos escritórios, nos aeroportos, bares ou parques, até à rede celular em que os telemóveis funcionam. Em 2007, um terço dos utilizadores da Internet acederam à Internet utilizando ligações sem fios, quer seja em casa, no trabalho ou noutro sítio [Lewis, 2007]. O grande desafio é, portanto desenvolver aplicações que façam uso do melhor que as redes sem fios têm para oferecer.

A rede sem fios será usada com recurso ao uso de telemóveis, uma vez que são dispositivos que permitem atingir um grande número de pessoas, num curto espaço de tempo, capitalizando assim na grande base de utilizadores que os telemóveis têm hoje em dia.

1.4 Estrutura

Por forma a se conseguir uma melhor organização da dissertação apresentada neste documento criou-se uma estrutura para o mesmo, apresentada de seguida.

Capítulo 2

O capítulo 2 apresenta de uma forma sumária os conceitos de uma rede ad-hoc, um pouco da sua história e os princípios básicos de funcionamento. Apresenta também o papel dos telemóveis como dispositivos da rede.

Capítulo 3

O capítulo 3 apresenta a forma como o ANEE se integra com o Building Management System (BMS), sistema que controla automaticamente um edifício através de informação relativa ao mesmo. Apresenta-se a visão geral e os requisitos do sistema desenvolvido como suporte ao BMS.

Capítulo 4

O capítulo 4 mostra o modelo conceptual desenvolvido na fase inicial do projecto, fase de estudo e de discussão do problema em questão. Apresenta-se um diagrama funcional, um diagrama arquitectural, um diagrama de classes e um modelo de dados.

Capítulo 5

O capítulo 5 discute como foi efectuada a implementação e as tecnologias usadas. Inclui alguns diagramas de sequência de modo a mostrar os diferentes passos de implementação, desde que o utilizador interage com o sistema até que lhe é apresentado os resultados dessa interacção.

Capítulo 6

O capítulo 6 mostra um exemplo de utilização em que o utilizador se liga à rede ad-hoc dentro de um edifício vendo a sua localização no mapa do mesmo.

Capítulo 7

O capítulo 7 apresenta as conclusões desta dissertação, apresentando ainda algumas linhas de futuros trabalhos que podem ser desenvolvidos, como continuação deste.

Anexos

Os anexos apresentam a implementação das classes mais importantes do ANEE.

Capítulo 2

Redes Ad-Hoc Sem Fios: Conceitos e Aplicações

2.1 IEEE 802.11: O *standard* do WiFi

Desde que os computadores portáteis apareceram que é desejável poder entrar em casa ou no escritório e ficar automaticamente ligado à Internet, sem a necessidade de fios. Tendo em conta esta necessidade diversos grupos de investigação começaram a pensar em diversas formas de atingir este objectivo. A forma mais simples de o fazer era equipar todos os computadores portáteis com transmissores e receptores rádio por forma a comunicarem entre si. Isto levou ao aparecimento de diversos sistemas Local Area Network (LAN) sem fios, mas com um grave problema, não eram compatíveis entre si pelo que a indústria decidiu então que um standard de LAN sem fios era necessário e a responsabilidade recaiu sobre o comité Institute of Electrical and Electronic Engineers (IEEE) que já tinha standarizado as LAN com fios [Tanenbaum and Wetherall, 2010]. O standard criado foi o IEEE 802.11, ou mais conhecido por Wireless Fidelity (Wi-Fi).

Figura 2.1: (a) Rede sem fios com ponto de acesso. (b) Rede Ad-hoc. [Tanenbaum and Wetherall, 2010]

O standard IEEE 802.11 teria de funcionar em dois modos, nomeadamente na presença de uma estação base e sem a presença de uma estação base (figura 2.1). No primeiro caso todas as comunicações teriam de passar por uma estação base, ou um Acess Point (AP) na terminologia IEEE 802.11. No segundo caso todos os computadores enviavam um para o outro directamente, sendo este último caso também designado rede ad-hoc.

O IEEE 802.11 não é apenas um *standard*, mas sim um conjunto de normas que utilizam um conjunto de modulações de sinal rádio através do mesmo protocolo básico. Os mais populares são o 802.11b e 802.11g que saíram como correcções ao 802.11 original, sendo estes dois os primeiros realmente aceites pela comunidade internacional. A norma 802.11i veio corrigir alguns problemas de segurança que existiam inicialmente propositadamente para cumprir as exigências de alguns governos, após algumas alterações legislativas. O 802.11n é uma norma que apresenta uma modulação *multi-streaming*, enquanto que o resto das normas (c-f, h, j) são apenas correcções a normas anteriores.

2.1.1 Divisão do espectro electromagnético

O segmento de rádio frequência utilizado pelo 802.11 varia de país para país. De um modo geral o 802.11 utiliza a banda Industrial, Scientific and Medical (ISM) dividindo-a em sub-bandas, chamadas canais. Por exemplo, a banda dos 2.4 GHz até aos 2.4835 GHz está dividida em treze canais de 22 Mhz cada e separados de 5 Mhz, onde o canal 1 esta centrado nos 2.412 Ghz e o canal 13 nos 2.472 Ghz. O Japão adiciona um 14º canal 12 Mhz acima do canal 13, como mostrado na figura 2.2.

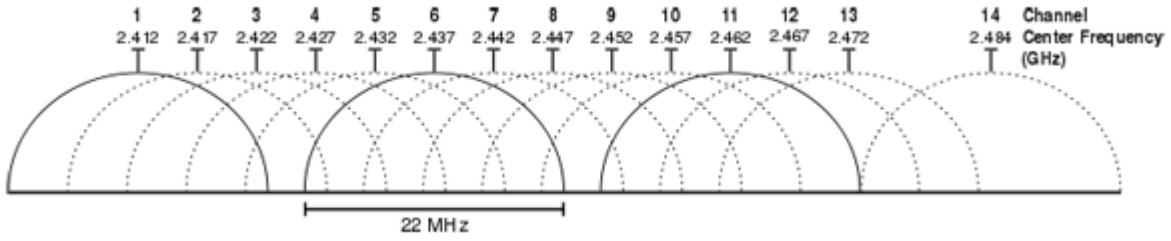


Figura 2.2: Divisão do espectro electromagnético em canais. [Gauthier, 2009]

A norma não especifica quantos canais podem ser usados, o número de canais possíveis de utilizar é definido pelas entidades competentes de cada país. Se por um lado o Japão permite o uso de 14 canais, os países da América do norte, centro e sul apenas permitem o uso do espectro até ao canal 12 e 13. A grande maioria dos países europeus permite o uso do espectro até ao canal 13.

Apesar de não definir o número de canais que devem ser usados, a norma especifica quais as frequências centrais de cada canal e uma máscara espectral que define a distribuição de potência permitida em cada canal. A máscara determina que tem de haver uma atenuação de pelo menos 30 dB desde o pico do sinal a $\pm$11 Mhz da frequência central do canal. Uma das consequências é que para não haver sobreposição dos canais, cada dispositivo pode apenas utilizar um em cada 4 canais. Por exemplo, no caso europeu um dispositivo que transmita no canal 1, só pode utilizar o

canal 6 e o canal 11. A razão pela qual se utiliza um canal a cada quatro, é por se assumir que a cada 4 canais a potência do sinal foi atenuada o suficiente para não interferir com o canal seguinte.

2.1.2 Versões do IEEE 802.11

802.11

A primeira versão do IEEE 802.11 foi apresentada em 1997 e com o aparecimento de novas normas é agora conhecida como IEEE 802.11-1997 ou IEEE 802.11 *Legacy*. O comité IEEE determinou que deveria ser utilizada a banda ISM do espectro electromagnético uma vez que a tecnologia era baseada em ondas rádio, mais especificamente na banda dos 2.4 Ghz até aos 2.4835 Ghz. A taxa de transmissão de dados era de 1 a 2 Mbps onde era possível utilizar duas técnicas de modulação, a Direct Sequence Spread Spectrum (DSSS) e Frequency Hopping Spread Spectrum (FHSS). Qualquer umas das técnicas permite a divisão do espectro em canais, contudo o DSSS constrói segmentos da informação a transmitir e envia-as nos diferentes canais, enquanto que o FHSS utiliza um esquema de salto na frequência, utilizando um determinado canal durante um breve período de tempo saltando depois para outro canal e assim sucessivamente. Por esta razão o FHSS é ligeiramente mais lento do que o DSSS, mas consegue ser mais imune a interferências [InfoWester, 2011].

802.11b

Dois anos depois do lançamento do IEEE 802.11 *Legacy*, em 1999, surgiu uma nova versão da norma, o IEEE 802.11b, que introduzia um aumento significativo na velocidade de transmissão de dados, com taxas de transmissão de 1 Mbps, 2 Mbps, 5.5 Mbps e 11 Mbps. Uma vez que o FHSS não cumpria com as normas da Federal Communications Comission (FCC) para velocidades superiores a 2 Mbps, apenas o DSSS seria utilizado com o auxílio de uma técnica chamada Complementary Code Keying (CCK).

Em condições normais o IEEE 802.11b garantia alcances de cerca de 400m em ambientes abertos, ou até cerca de 50m em ambientes fechados, apesar destes alcances poderem ser altamente limitados por interferências de certos objectos que impedem a propagação do sinal. Esta versão da norma, contudo, estava preparada para ir diminuindo a velocidade de transmissão de dados sucessivamente até à sua velocidade mínima de 1 Mbps conforme um dispositivo fica mais longe da sua estação base.

802.11a

Quase ao mesmo tempo da versão 802.11b, no final de 1999, surgiu a versão 802.11a, que permitia grandes velocidades, desde os 6 Mbps, 9 Mbps, 12 Mbps, 18 Mbps, 24 Mbps, 36 Mbps,

48 Mbps e 54 Mbps. Apesar do aumento significativo na velocidade esta versão da norma não teve tanto sucesso como a 802.11b, uma vez que utilizava a banda ISM nos 5 Ghz, banda essa onde muitos países não tinham legislação e que provocava incompatibilidades com as versões anteriores da norma.

Em vez do DSSS ou do FHSS, a versão 802.11a utilizava uma técnica de modulação chamada Orthogonal Frequency Division Multiplexing (OFDM). Neste tipo de modulação, a informação é dividida em pequenas partes que são enviadas simultaneamente em diversas frequências, separadas de forma a que não interfiram uma na outra, o que torna esta técnica altamente satisfatória.

802.11g

Se o 802.11a falhou em relação ao 802.11b, o mesmo não aconteceu com a versão 802.11g, que por ser completamente compatível com a versão 802.11b é vista como o seu sucessor. Surgiu em 2003 e permitia velocidades até 54 Mbps ao mesmo tempo que operava na banda ISM aos 2.4 Ghz. Apesar de a técnica de modulação utilizada ter passado a ser a OFDM, sempre que um dispositivo com a norma 802.11g comunica com um dispositivo com a norma 802.11b, a técnica utilizada é a DSSS às velocidades limitadas da versão 802.11b.

802.11n

A versão 802.11n surgiu nos finais de 2009 e tal como todas as suas antecessoras trouxe um aumento significativo na velocidade de transmissão de dados, aumentado a velocidade dos 54 Mbps para os 600 Mbps. Apesar de continuar a funcionar na banda ISM aos 2.4 Ghz, os canais podem ter larguras de banda de 40 Mhz em adição aos tradicionais 20 Mhz.

A principal inovação do 802.11n é, contudo, a implementação de um esquema Multiple Input, Multiple Output (MIMO), que aumenta exponencialmente a velocidade de transmissão de dados, tornando possível utilizar múltiplas antenas quer no emissor quer no receptor para a transmissão de dados. O esquema de modulação utilizado é o OFDM, com algumas adaptações para funcionar com o esquema MIMO, podendo alcançar distâncias superiores a 400m.

Outras versões

O 802.11 tem e terá certamente muitas mais versões que não foram aqui mencionadas. O 802.11d é utilizado em alguns países quando por alguma razão não é possível utilizar as versões mencionadas anteriormente. Para comunicações onde se deve ter uma grande robustez face a interferências como é o caso do Voice Over IP (VoIP), foi criada a versão 802.11e que tinha como principal foco a qualidade de serviço (Quality of Service (QoS)).

2.1.3 Segurança

Um dos problemas que as redes sem fios se deparam é o acesso de pessoas não autorizadas à rede. Sem nenhum mecanismo de segurança qualquer dispositivo equipado com *hardware* compatível com a norma IEEE 802.11 pode aceder a uma rede e aproveitar todos os seus recursos, como por exemplo o acesso à Internet.

Um dos mecanismos de segurança é o Wired Equivalent Privacy (WEP) e existe desde o 802.11 *Legacy* e pode funcionar de forma aberta ou limitada por chaves. No primeiro caso qualquer dispositivo se pode ligar à rede, sendo apenas feita uma autorização por parte da estação base. No segundo caso é utilizada uma chave pré-definida (de 64 bits ou 128 bits), que para além de dar acesso à rede a um dispositivo também é utilizada para encriptar todos os dados transmitidos na rede.

O WEP tem graves falhas de segurança, podendo as chaves ser facilmente descobertas mesmo quando é uma chave de 128 bits. Todas as chaves WEP apresentam 24 bits de inicialização que tornam os restantes 40 bits (no caso de uma chave de 64 bits) facilmente descobertos.

Por forma a resolver estas falhas, foi criado em 2003 o mecanismo Wired Protected Acess (WPA) que tal como o WEP permite a autenticação de um dispositivo perante a rede, bem como a encriptação dos dados transmitidos pela rede. A base deste mecanismo de segurança é o protocolo Temporal Key Integrity Protocol (TKIP), que utiliza uma chave de 128 bits combinada com o endereço Media Access Control (MAC) de cada dispositivo. Como cada endereço MAC é único para cada dispositivo consegue-se ter uma chave única para cada dispositivo. A chave é trocada periodicamente entre o dispositivo e a estação base e a palavra passe definida aquando da configuração da estação base é utilizada para estabelecer a ligação entre um dispositivo e a estação base.

Para conseguir um nível de segurança superior foi criado o Wired Protected Acess 2 (WPA-2), que não é mais do que a norma 802.11i. Esta segunda versão do WPA utiliza o protocolo Advanced Encryption Standard (AES), que apesar de ser seguro e eficiente também necessita de grande capacidade de processamento, podendo trazer alguns problemas com dispositivos com fraca capacidade computacional.

2.2 Redes Ad-Hoc: História e características

O aparecimento dos computadores portáteis permitiu uma comodidade extra aos seus utilizadores por estarem sempre disponíveis em qualquer lugar, mas também exigiu que a troca de informação pudesse ser feita da mesma forma. A necessidade de haver um administrador que fosse capaz de interligar todas as máquinas não era nem cómoda nem rápida, exactamente o oposto daquilo que se pretende com "portabilidade".

O aparecimento das tecnologias sem fios permitiu o acesso remoto de um dipositivo a outro dipositivo, sendo a sua utilização claramente vantajosa no caso dos dispositivos móveis. Com esta facilidade de acesso começou-se a pensar em formas de partilhar a informação entre os dispositivos sem que fosse necessário recorrer a um administrador sempre que essa partilha fosse necessária e consequentemente conseguir-se efectuar essa partilha em qualquer lugar. É neste contexto que as redes ad-hoc aparecem.

O conceito de rede ad-hoc começou a ser pensado no início da década de 1970 para fins militares com o projecto Packet Radio Network (PRNET) desenvolvido pela United States Defense Advanced Research Project Agency (US DARPA) com vista a explorar o uso de redes na comunicação de pacotes via rádio [Jubin and Tornow, 1987]. Mais tarde e já nos anos 80, a US DARPA iniciou o projecto Survivable Adaptive Network (SURAN) de modo a expandir o projecto inicial PRNET para que este suportasse grandes redes e se adaptasse às rápidas mudanças de um ambiente militar. Em 1994 a US DARPA desenvolveu o último dos projectos chamado Global Mobile Information Systems (GloMo) que apesar de ter ainda grande foco nas aplicações militares, começou também a ter outras finalidades como a busca e salvamento.

As redes ad-hoc são um tipo de rede que não possuem um ponto central (ou ponto de acesso) para onde todas as ligações convergem sendo depois reencaminhadas para o seu destino, como é o caso muito comum dos *routers*, nem nenhuma topologia pré definida. Ao invés, nas redes ad-hoc cada elemento da rede é por si só um encaminhador, passando a informação que lhe chega aos seus dispositivos vizinhos seguindo um algoritmo de encaminhamento bem definido.

Cada elemento da rede, ou dispositivo, é definido como um nó. Cada nó presente na rede comunica directamente com outros nós (por fios ou sem fios) e só pertencem à rede quando se encontram na proximidade da mesma. Só mais recentemente é que este tipo de rede começou a promover mecanismos mais robustos de segurança pela utilização de standards mais modernos (IEEE 802.11).

Numa rede ad-hoc móvel ou Mobile Ad hoc NETwork (MANET) um conjunto de nós móveis (MNs) formam redes autónomas, dinâmicas e independentes de qualquer infra-estrutura. Como os nós são móveis a topologia da rede está em constante mudança e pelo limitado alcance que estes têm em termos de rádio, é necessário que a informação vá passando de nó em nó até chegar ao destino final. Esta necessidade exige que cada nó seja ao mesmo tempo um router e um host, sendo esta também a razão por que se chama a rede de *multi-hop* (múltiplos saltos).

2.3 Dispositivos Móveis

O Dispositivo Móvel (DM) é parte integrante das redes ad-hoc sem fios pois é nele que se centra toda a acção. Cada dispositivo móvel representa um nó da rede ad-hoc e é neles que todos os bytes trocados na rede são traduzidos em informação útil para o utilizador. Entende-se por dispositivo móvel, um equipamento electrónico com capacidades de *input* e de *output*, e de se manter em constante contacto com outros dispositivos mesmo quando se move geograficamente. São exemplos os

telemóveis, computadores portáteis, PDAs, rádios, smartphones e televisões portáteis.

No contexto desta dissertação, os elementos relevantes são os smartphones por serem dispositivos que seguem as normas de comunicação sem fios IEEE 802.11 e por serem dispositivos que permitem o desenvolvimento de software que faça uso das potencialidades das redes sem fios. Por serem telemóveis com grandes capacidades de processamento e de armazenamento de dados são o exemplo perfeito para o que se pretende mostrar com esta dissertação, o uso de uma rede ad-hoc que forneça informação que contribua para a eficiência energética.

2.4 Mobile Ad hoc Network: Redes ad hoc móveis

A qualidade das ligações entre nós numa rede ad-hoc pode se alterar rapidamente num curto espaço de tempo, especialmente quando a rede é constituída por nós que estão em constante movimento. Podem aparecer e desaparecer e re-aparecer com o passar do tempo e a rede tem de ser capaz de manter a ligação entre os nós da rede que não desaparecem. Neste caso, com conectividade sem fios, a robustez da ligação é um problema muito mais complicado de lidar do que uma ligação com fios.

Como as redes ad-hoc não são constituídas por nenhuma estação base, são vistas como pequenas LANs, especialmente quando a ligação é feita sem fios. Em alguns casos os dispositivos móveis fazem parte da rede por pequenos períodos de tempo, apenas quando estão no alcance do resto da rede.

Características

Numa MANET os nós da rede são dispositivos móveis que comunicam entre si através de ondas rádio. Todos os nós que estiverem ao alcance uns dos outros podem comunicar directamente entre si, enquanto que os nós que não estejam no alcance têm de usar nós intermédios para comunicar com o nó de destino.

Como os dispositivos, ou nós, são móveis, uma MANET tem uma topologia dinâmica, ou seja, que varia ao longo do tempo, o que dificulta ainda mais a construção da rede ad-hoc. Para além disso, os dispositivos dependem de baterias para operar pelo que o consumo de energia tem de ser racionado. Tipicamente o consumo energético é dividido em três partes, uma para processamento de dados, outra para transmitir informação para o destino e, por fim uma parte para quando o dispositivo é usado como um encaminhador, i.e., quando precisa de reencaminhar pacotes de um dispositivo para outro [Saching, 2011].

Os dispositivos móveis têm recursos limitados pelo que dependem fortemente de terceiros para processamento e armazenamento de dados. Uma topologia de rede fiável tem de ser conseguida através de protocolos robustos especificamente desenvolvidos a pensar nas redes ad-hoc.

As MANETs apresentam as seguintes características principais:

1. Baseadas em comunicações sem fios.
2. Os nós da rede funcionam como encaminhadores e como *hosts*.
3. Não existe uma estação base.
4. Têm uma topologia de rede dinâmica.
5. São autónomas.
6. Podem ser criadas em qualquer sítio.
7. Apresentam limitações em termos de energia.
8. São pouco seguras.

2.5 WSN: Redes de sensores sem fios

Inicialmente desenvolvidas para aplicações militares como vigilância em campo de batalha, as redes de sensores sem fios (Wireless Sensor Network (WSN)) são hoje em dia usadas de forma alargada em diversas aplicações industriais e civis como monitorização e controlo de processos industrias, monitorização do ambiente e habitat, aplicações médicas, automação do lar e controlo de tráfego.

Uma WSN consiste numa rede de sensores autónomos espacialmente distribuídos de forma a medir diversos aspectos como o som, temperatura, pressão, movimento e poluição. A rede é constituida por um conjunto de nós que, para além do sensor que mede as diversas condições, contêm também um transceptor rádio (ou outro tipo de tecnologia de comunicação sem fios), um microcontrolador e uma fonte de energia, geralmente uma bateria.

Um nó numa WSN pode variar em tamanho desde um grão de areia até uma caixa de sapatos, sendo que os custos de produção de cada sensor nestes nós pode variar também desde centenas de euros até alguns cêntimos. É de notar que quanto maior a complexidade de um sensor mais caro este é pelo que limitações em termos de tamanho e custo provocam limitações em termos energia, memória, velocidade de processamento e largura de banda.

Uma WSN é constituida por uma rede sem fios ad-hoc (MANET) que corre um algoritmo de encaminhamento multi-hop, onde diversos nós encaminham pacotes de dados para a estação base.

Características

Os nós numa rede WSN podem ser vistos como pequenos computadores, muito limitados em termos de componentes e de interface, consistindo num Central Processing Unit (CPU) com proces-

samento e memória muito limitados, sensores, um dispositivo de comunicação sem fios como um transceptor rádio e uma fonte de energia (bateria). Existe a possibilidade de adicionar diversos módulos aos nós, tais como Application Specific Integrated Circuit (ASIC), dispositivos de comunicação extra (Universal Serial Bus (USB) e RS-232), e dispositivos de obtenção de energia como células fotovoltáicas.

As estações base agem como *gateway* entre os sensores e o utilizador final e possuem muito mais capacidade de processamento, energia e comunicação. É aqui por isso que se processa todos os dados recolhidos pelos sensores ao longo da WSN de forma a apresentar ao utilizador os dados completamente tratados e prontos a utilizar.

As WSN, por não serem mais do que uma MANET onde os dispositivos são sensores, têm características muito semelhantes às de uma MANET, contudo apresentam alguns traços únicos, especialmente por usarem uma estação base que recolhe toda a informação adquirida pelos nós da rede, nomeadamente:

1. Os nós têm pouca capacidade de guardar ou recolher energia.
2. Conseguem aguentar condições meteorológicas muito adversas.
3. Têm capacidade para lidar com falhas nos nós.
4. São robustas a falhas de comunicação.
5. A capacidade de nós é escalável, apenas limitada pela largura de banda do *gateway*.

Capítulo 3

Sistemas de Gestão de Edifícios

O ANEE é uma infraestrutura de software que acenta fortemente nos conceitos de um Sistema de Gestão de Edifícios (SGE), do inglês Building Management System (BMS) como caso de aplicação. Este capítulo apresenta as noções básicas de um sistema de gestão de energia de um edifício e como o ANEE se insere dentro deste contexto.

3.1 O que é um Sistema de Gestão de Energia?

Um SGE é um sistema controlado por computador que monitoriza e controla equipamentos de um edifício que consomem energia, nomeadamente: ventilação, iluminação, sistemas de controlo de incêndios e sistemas de segurança [URVIL, 2011]. Permite automatizar o funcionamento de todos estes sistemas e adequar o seu funcionamento de acordo com determinados aspectos físicos relevantes, contribuíndo para a poupança de energia, com poupanças anuais de cerca de 30%, ao mesmo tempo que mantém um ambiente confortável nos diferentes espaços dos edifícios [Kamm, 2007].

O uso da palavra SGE começou a ser adoptado no início da década dos anos 70, quando começaram a aparecer os primeiros sistemas electrónicos capazes de armazenar dados por forma a actuar em sistemas de energia, iluminação, aquecimento, etc.. Antes dos SGE controlados por computadores, eram utilizados sistemas electromecânicos por forma a controlar um edifício, com painéis luminosos que indicavam o estado dos diferentes elementos do edifício para que os responsáveis pelo mesmo pudessem detectar alguma situação anómala. Para além destes painéis eram também usados alarmes sonoros que alertavam alguma avaria no sistema.

Hoje a utilização de sistemas de controlo de energia são uma prática comum, principalmente em edifícios não residenciais [Stum et al., 1997]. Estes sistemas são também conhecidos por Energy Management Systems (EMS), Energy Management Control Systems (EMCS) e Building Automation Systems (BAS).

3.1.1 Principais Funções do SGE

As principais funções do SGE são monitorizar, controlar e optimizar os equipamentos que consomem energia dos edifícios. É normalmente composto por sistemas de Aquecimento, Ventilação e Ar Condicionado (AVAC), sistemas eléctricos, sistemas, de segurança, de alarmes de incêndio entre outros.

A forma mais simples de controlo de consumo de energia em sistemas AVAC é a utilização de termostatos que reduzem o consumo energético numa relação tempo - temperatura. Outro método de redução do consumo de energia é a utilização de controladores que fazem uso de técnicas *on - off* para diminuir o consumo. Outros tipos de controlo, mais complexos, fazem uso de esquemas de previsão através da quantidade de energia consumida num determinado momento.

Do ponto de vista dos sistemas de iluminação, o consumo de energia depende muito do tipo de equipamento que está instalado nos edifícios. A utilização de lâmpadas fluorescentes são mais eficientes em termos energéticos do que as lâmpadas incadescentes, apresentando poupanças de energia na ordem dos 60% [Binod, 2011]. A disposição dos pontos de luz nos diferentes espaços de um edifício também contribuem para uma maior redução do consumo de energia. Pontos de luz mal distribuídos não oferecem iluminação suficiente levando a adição de pontos de luz adicionais.

Os sistemas eléctricos se forem mal desenhados, i.e. com cabalagem mal distribuída na altura da projecção do edifício, ou cablagem de má qualidade, são muitas vezes factores que contribuem para o desperdício de energia. Os SGE têm portanto de actuar directamente nos paineis eléctricos de um edificío por modo a poder controlar todo o sistema eléctrico do edifício.

3.1.2 Características de Hardware

A rápida evolução da tecnologia faz com que mesmo os SGE mais recentes necessitem de ter capacidade de actualização uma vez que surjem computadores com maior capacidade de processamento, sensores mais precisos, programas de controlo mais complexos, etc.. Por esta razão o *hardware* de um SGE tem de ter algumas características especiais.

Expansiblidade

Um SGE tem de ter capacidade de se expandir. Má escolha de *hardware* pode levar a que um SGE não consiga se actualizar para novas versões de programas, para um novo modelo sem ter de alterar toda a estrutura do sistema e que não consiga cobrir as novas necessidades resultantes de renovações.

Interfaces com outros equipamentos e controlos

Um problema comum dos SGE é o facto de muitos dos equipamentos terem controlos autónomos, pelo que é necessário garantir que todos os equipamentos que compõem um SGE conseguem comunicar entre si.

Interoperabilidade

Por forma a facilitar a interoperabilidade entre os diferentes equipamentos do SGE existem alguns protocolos que os fabricantes utilizam, nomeadamente o Building Automation and Control NETworks (BACnet) e o LonWorks.

CPU e Interface Gráfica

Para sistemas com gráficos a cores e interfaces mais complexas é necessário que as características de *hardware* de um terminal de monitorização de um SGE sejam muito semelhantes àquelas de um computador pessoal de alta gama. Se as reconfigurações de sistema necessitam que os terminais estejam desligados, então é comum utilizar-se mais do que um terminal.

Painéis Distribuídos

Por forma a facilitar a actualização futura do SGE os painéis devem ter entradas e saídas universais. Isto faz com que cada entrada, ou cada saída, consigam ter associados a nível de software como digitais ou analógicas. Outra alternativa é o painél utilizado ser modular, podendo se adicionar módulos digitais ou analógicos ao painel. É desejável que estes painéis consigam funcionar de forma autónoma, no caso das comunicações falharem com o terminal central. Dessa forma torna-se possível conectar directamente um portátil, ou um equipamento equivalente, ao painel.

Rede

A forma como a rede é estruturada afecta a forma e a velocidade com que a informação chega aos terminais de controlo. A taxa em que os painéis trocam informação afectam essa velocidade, bem como o tempo necessário para recuperar de uma falha de sistema.

Impressoras e Portáteis

Se os terminais suportarem a inclusão de impressoras ou portáteis então é mais fácil trocar a informação que o SGE gera, seja para relatórios, diagnósticos ou configuração do sistema.

Sensores

A qualidade e a precisão dos sensores que compõem um SGE são necessários para garantir um controlo eficiente. Por exemplo, se um sensor estiver a ser utilizado para controlar a temperatura de uma divisão pode haver algum erro associado, contudo se o objectivo é controlar os custos que um equipamento tem em termos financeiros, então é necessário garantir um erro menor.

Válvulas e Actuadores

As válvulas e os actuadores são os equipamentos em que o SGE actua por forma a controlar o edifício. Muitas das vezes são necessárias diversas válvulas em série são necessárias para controlar eficazmente o edifício.

3.1.3 Características de Software

Os SGE são sistemas que dependem fortemente dos equipamentos de *hardware* que o compõem, mas sem uma base de *software* não é possível configurar o sistema de uma forma simples por parte do administrador, nem ver relatórios do que se está a passar no edifício.

User Interface

Para sistemas de grandes proporções, i.e. para grandes edifícios, tipicamente são utilizados programas que correm em Windows e Unix. Estes programas incluem todas as interacções com o utilizador, nomeadamente: estado de pontos monitorizados *on/off*, entradas analógicas, identificação de cada ponto, estado de controlo do equipamento (automático, manual e alarme), gráfico simbólico do equipamento, etc.. Em alguns sistemas o utilizador pode carregar num mapa no ecrã e planear, por exemplo, a temperatura nos diferentes espaços de um piso do edifício.

Programação e Acesso

Dependendo do SGE, existem alguns sistemas que permitem o acesso de utilizadores ao nível de programação, ou de configuração de todos os equipamentos do sistema, contudo esse acesso pode ser limitado. Alguns SGE permitem que os utilizadores criem novos programas, testem-nos e simulem-nos antes de os activarem.

Alarmes

Um SGE tem diferentes níveis de alarme, cada um deles com diferentes opções de alerta e podem incluir relatórios remotos ou locais.

Relatórios

Os SGE são capazes de produzir automáticamente relatórios sobre aquilo que está a acontecer no sistema e incluem: listagem de todos os pontos; todos os pontos em alarme, desactivados ou fechados; registo histórico de todos os comandos enviados aos equipamentos e razão do envio desses comandos; programação semanal; limites e problemas.

Segurança

Especifica todos os níveis de segurança que o sistema apresenta.

Estratégias de Controlo

As estratégias de controlo passam por conservar energia e optimizar o conforto.

3.1.4 Capacidades básicas de um SGE

As capacidades de um SGE podem variar de fabricante para fabricante mas no global têm algumas caracteríticas universais, nomeadamente: Agendamento, Pontos de Controlo, Alarmes, Segurança e Monitorização.

Agendamento

Os SGE oferecem não só tempos de *on/off* mas também diferentes pontos de controlo. Com vários cenários de agendamento é possível conseguir poupanças siginificativas [Stum et al., 1997]. O primeiro passo no agendamento é desligar todos os equipamentos que não são necessários e o segundo é optimizar o tempo de uso dos equipamentos, mantendo-os ligados o mínimo tempo possível.

Pontos de Controlo

Pontos de controlo podem ir desde lógica interna dos equipamentos, que não necessitam de

muita manutenção, até pontos de controlo de temperatura de divisões, que requerem manutenção e calibração constante. Alguns pontos de controlo são definidos por um operador e associados a um cenário de agendamento, outros podem ser ajustados por cálculos internos do programa que controla o sistema (e.g. restabelecer valores de temperatura ou pressão).

Alarmes

Em adição às funções básicas de alarme de um SGE, o sistema fornece algumas opções acerca de como os alarmes são monitorizados, reportados, encaminhados e finalmente resolvidos. Os sistemas podem ser configurados para reagir às seguintes situações: Falhas de equipamento, Falhas nos sensores, Valor elevado (temperatura, pressão, etc.), Valor reduzido (Temperatura, pressão, etc.), Valores inválidos, Substituição manual de máquinas em locais remotos e falhas de comunicação.

Segurança

Por forma a garantir a segurança dos equipamentos e da propriedade existem sequências de segurança programadas num SGE. A condição que inicia um processo de segurança também pode activar um alarme, pelo que a utilização de rotinas de segurança pode evitar danos nos equipamentos e nos edifícios bem como evitar alarmes que poderiam exigir a intervenção de serviços externos ao edifício. Este tipo de rotinas de segurança não devem ser dependentes de rotinas de *software* mas sim estar embutidas directamente no *hardware*.

Monitorização

A monitorização num SGE consiste em manter um registo histórico dos diferentes paramêtros da operação de um equipamento. A monitorização pode ser efectuada em quase todos os pontos que controlam equipamento e para algum *software* e pontos de controlo virtuais. Desta forma é possível fazer monitorização dos dados medidos pelos sensores sem ter de fazer as medidas no local onde o sensor está instalado.

3.1.5 Vantagens do SGE

Algumas das vantagens da utilização de um SGE são as seguintes:

- Possibilidade de controlo índividual de divisões para condições de conforto internas.
- Monitorização e utilização eficiente de energia.
- Poupança de tempo e dinheiro.
- Monitorização automática e central do edifício.
- Horário de manutenção controlado por computador.
- Detecção rápida de problemas.
- Facilidade de acesso à informação do edifício.

3.2 O ANEE como um Sistema de Gestão de Edifício

3.2.1 Visão Global do ANEE

O ANEE é um sistema de suporte à monitorização dos espaços de um edifício no sentido em que fornece informação ao SGE de modo a que este controle eficazmente os sistemas mecânicos e electrónicos do edifício, tendo em conta a eficiência energética. Esta informação pode vir de diferentes sistemas sensoriais que medem a temperatura, a qualidade do ar e a luminosidade, mas poderá também vir directamente das pessoas que estão num determinado momento no edifício. Ao se determinar a densidade de pessoas em diferentes espaços de um edifício é possível adequar as condições desses mesmos espaços de acordo com essa densidade de pessoas, sendo mais fácil detectar desperdícios de energia e consequentemente atingir um consumo óptimo de energia.

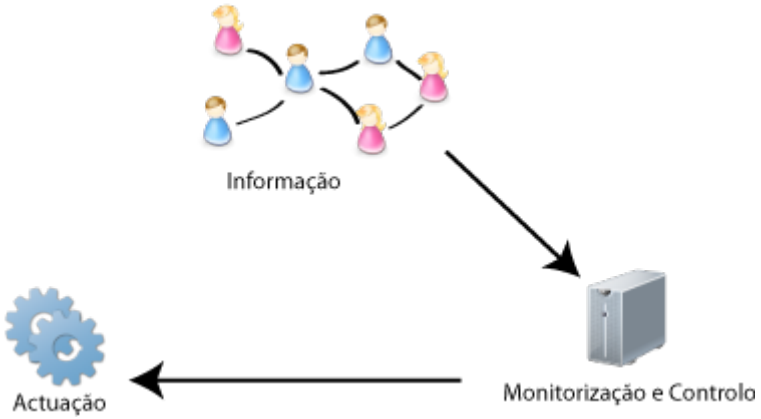


Figura 3.1: Visão Global do ANEE.

Tendo em conta os dados fornecidos, o SGE poderá actuar de duas formas: actuação instantânea e previsão. No primeiro caso assim que o SGE recebe novos dados analisa-os e faz os ajustes necessários (e.g. se num determinado momento não existe pessoas num espaço desliga-se o ar condicionado e reduz-se a iluminação). No segundo caso através de registos históricos da densidade de pessoas no edifício (e.g. se o registo histórico indica que às 12h00 haverá um aumento exponencial na densidade de pessoas nas zonas de restauração de um edifício, então actua previamente no ar condicionado e iluminação para que a essa hora as condições sejam as ideais).

Poderíamos pensar noutras aplicações para o SGE, mas o foco desta dissertação é o fornecimento de dados ao SGE, mais concretamente na densidade de pessoas no edifício, pelo que é nesse aspecto que este documento se vai centrar.

3.2.2 Pré Requisitos Operacionais

Por forma a se conseguir uma contabilização correcta da quantidade de pessoas num edifício, o sistema desenvolvido deve obedecer a alguns requisitos sem os quais os seus objectivos não poderiam ser cumpridos.

Em primeiro lugar é necessário que cada pessoa tenha um *smartphone* que seja compatível com a norma IEEE 802.11. Este requisito é fundamental uma vez que se pretende criar uma rede ad-hoc sem fios. A razão pela qual foi escolhido este tipo de equipamento é pelo facto de hoje em dia grande parte da população utilizar telemóvel, levando-o consigo para todo o lado, fazendo com que se possa capitalizar desta grande densidade de utilizadores para obter informação mais fiável.

Em segundo lugar, é necessário que os edifícios estejam equipados com *hotspots* nas diferentes divisões. Sem estes não é possível determinar a localização exacta das pessoas, apenas o número de pessoas que estão no edifício, pelo que a informação fica limitada.

Uma vez que o micro-controlador utilizado para implementar os *hotspots* é limitado em termos de memória e de capacidades de processamento é necessário garantir que a aplicação seja o mais simples possível.

Capítulo 4

ANEE: Modelo Conceptual

O modelo conceptual apresenta a ideia que suporta a solução desenvolvida e permite que a mesma seja estruturada em partes mais pequenas. Tem como objectivo uma representação de alto nível do sistema do ponto de vista do utilizador, em termos de funcionalidades e estrutura arquitectural. Não se pretende neste capítulo entrar em detalhes de implementação, detalhes esses serão vistos mais tarde.

O ANEE é um sistema composto por diversos módulos. Este capítulo ajuda a estruturar melhor o problema, separando-o em partes mais pequenas contribuindo para um melhor entendimento do que se realizou e implementou. A modelação foi efectuada recorrendo à linguagem Unified Modeling Language (UML) para se representar o ANEE na linguagem padrão de desenvolvimento de sistemas computacionais.

A base do ANEE é uma rede ad-hoc de pessoas dentro de um determinado edifício, pessoas que chamaremos utilizadores. A rede é construída a partir dos telemóveis desses utilizadores através de uma aplicação chamada Mobile Device Application (MDA). Para além dos utilizadores, existem *hotspots* espalhados pelo edifício (um por cada divisão) que informam através de difusão os telemóveis da sua localização dentro do edifício. Sempre que um utilizador passa perto de um desses *hotspots* a sua localização é actualizada. Este sistema de localização permite saber em que divisão do edifício é que o utilizador se encontra de modo a se poder fazer um mapeamento dos utilizadores ao longo de todo o edifício.

Por forma a se realizar a contagem de pessoas dentro do edifício criou-se um controlador que está ligado à rede ad-hoc (que pode correr dentro do BMS) chamado People Density Controller (PDC). O PDC monitoriza constantemente a rede, perguntando a cada elemento da rede (os telemóveis dos utilizadores) a sua localização dentro do edifício, efectuando assim a contabilização do número de pessoas nos diversos espaços do edifício (figura 4.1). O PDC é também responsável por manter um registo histórico da contabilização efectuada através de uma base de dados. Esta base de dados pode depois ser acedida por um BMS por forma a utilizar esta informação.

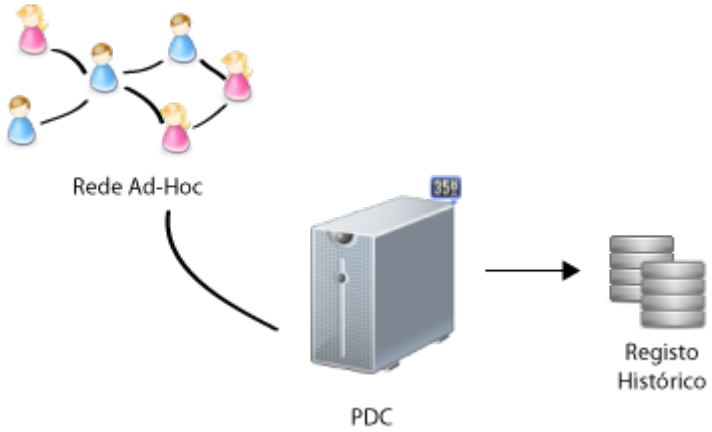


Figura 4.1: Contabilização do número de pessoas por parte do PDC.

Para não se colocar o utilizador num papel puramente passivo, onde bastaria que este utilizasse o seu telemóvel e se mantivesse ligado à rede ad-hoc, foi pensado outras funcionalidades que tirassem partido da informação que é trocada entre os elementos da rede. Nesse sentido pensou-se num sistema de grupos, onde diversos utilizadores se poderiam juntar permitindo que comunicassem entre si através da rede, ou até mesmo verificar a sua localização num mapa dentro de um edifício. A gestão de grupos e criação de grupos é feita pelo PDC e permite que cada edifício tenha os seus próprios grupos. Para evitar a repetição de grupos no mundo, cada PDC tem de pedir autorização a uma entidade que controla todos os grupos do mundo, chamada Universal Tag Distributer (UTD). Se for possível criar o grupo, o UTD dá permissão ao PDC para criar o novo grupo e regista-o numa base de dados universal (figura 4.2). Todos os grupos criados pelo PDC são depois enviados para os utilizadores na rede podendo estes aceitar ou recusar o grupo. Apenas utilizadores do mesmo grupo podem comunicar entre si.

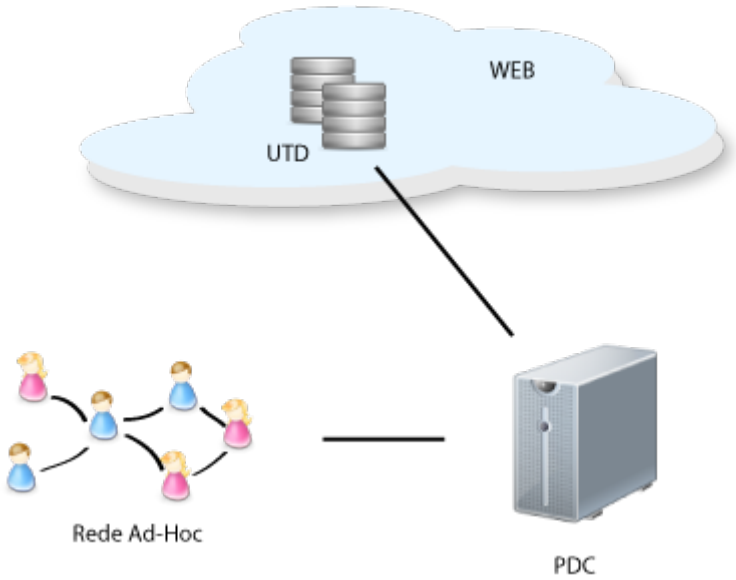


Figura 4.2: Sistema de distribuição de grupos.

4.1 Visão Funcional

A visão funcional representa todas as funcionalidades que o ANEE oferece ao utilizador. Este é o ponto de partida para o desenvolvimento do nosso sistema, uma vez que representa de uma forma clara e objectiva todos os casos de uso da infra-estrutura desenvolvida.

O ANEE é composto por quatro aplicações principais (figura 4.3): Mobile Device Application (MDA), People Density Controller (PDC), Hotspot Application (HA) e Universal Tag Distributer (UTD). No primeiro caso, o MDA é a aplicação que está a ser executada nos telemóveis dos utilizadores e é o único meio de interacção do utilizador com o ANEE. A segunda aplicação, o PDC, está instalado em cada edifício e é responsável por efectuar a contabilização do número de pessoas em cada divisão, bem como pela criação dos mapas e grupos do edifício e pela sua difusão pelos utilizadores da rede ad-hoc. A terceira aplicação, o HA é a que corre nos *Hotspots* presentes em cada divisão e é responsável por informar o MDA em que divisão do edifício é que o utilizador se encontra. Por último, o UTD é responsável pela gestão de todos os grupos que existem no mundo de modo a não haver repetições de grupos.

O MDA tem apenas um actor, i.e, a entidade que utiliza a aplicação, que é o utilizador que tem o MDA instalado no seu telemóvel. Assim, o MDA fornece-lhe as seguintes funcionalidades:

1. **Ligar-se à rede ad-hoc sem fios:** O utilizador informa o MDA para se ligar à rede ad-hoc sem fios;
2. **Desligar-se da rede ad-hoc sem fios:** O utilizador informa o MDA para se desligar da rede ad-hoc;
3. **Juntar-se a um grupo:** O utilizador aceita um convite para um novo grupo, adicionando-o à sua lista de grupos;
4. **Sair de um grupo:** O utilizador sai de um grupo apagando-o da sua lista de grupos impedindo-o de continuar a ver os restantes utilizadores desse grupo;
5. **Conversar com elementos de um grupo:** O utilizador inicia janelas de conversação com um elemento de um grupo, ou com todos os elementos de um grupo, podendo neste caso aceder a outra funcionalidade:
   a) **Enviar Ficheiro:** O utilizador pode enviar ficheiros (imagens, documentos de texto, etc.) para as pessoas com quem está a conversar;
6. **Ver mapa de um edifício:** O utilizador vê a planta do edifício e do piso onde está, podendo também:
   a) **Ver a localização do utilizador no mapa:** O utilizador vê a sua localização na planta do mapa;
   b) **Ver a localização de pessoas de um grupo no mapa:** O utilizador vê a localização dos elementos de um grupo na planta do mapa.

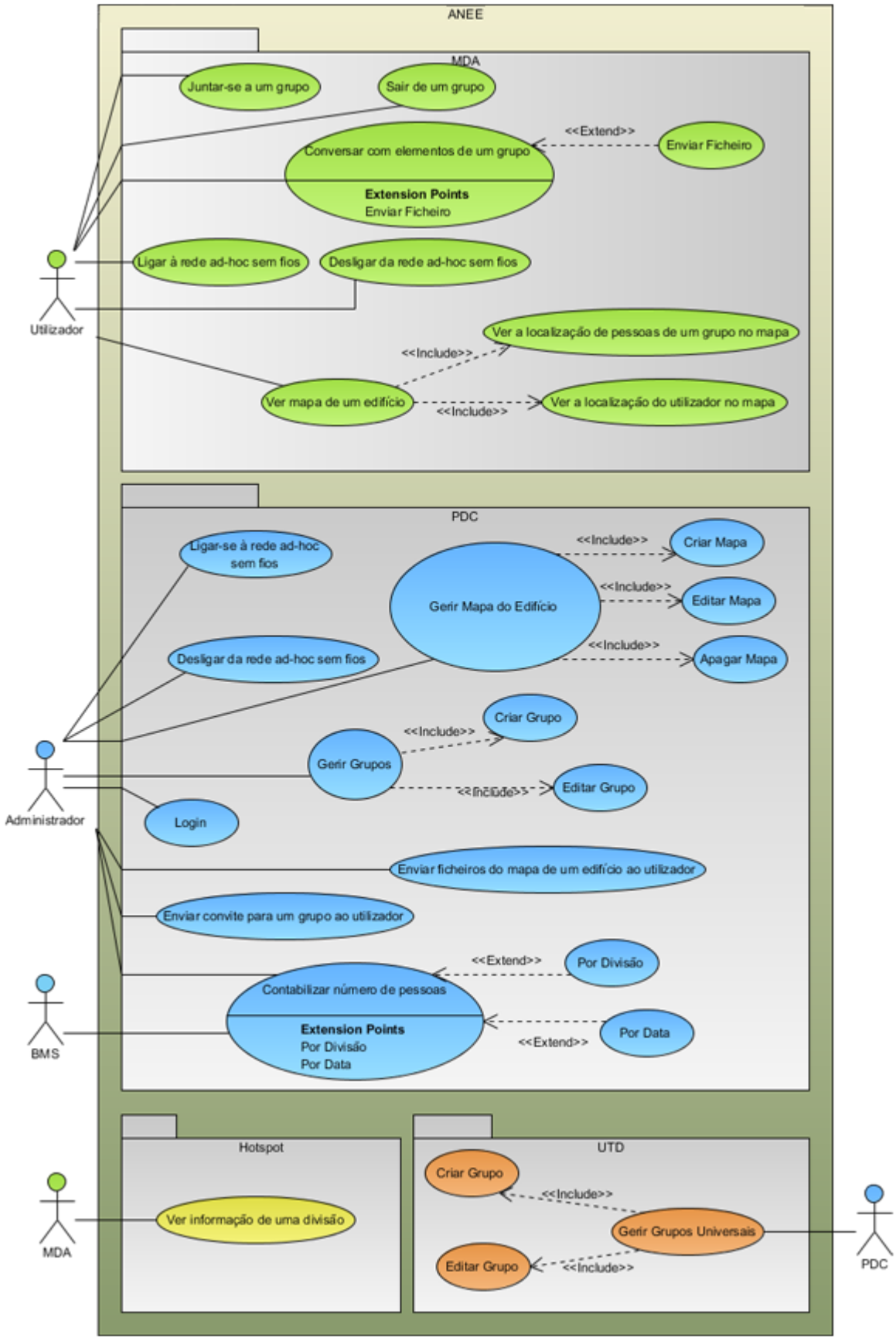


Figura 4.3: Diagrama de casos de uso do ANEE.

O PDC tem dois actores possíveis: Administrador e BMS. No primeiro caso, o administrador utiliza todas as funcionalidades abaixo, enquanto que o BMS apenas utiliza o último ponto.

1. **Login:** O administrador efectua a autenticação das suas credenciais de acesso no PDC, de modo a poder administrá-lo;
2. **Ligar-se à rede ad-hoc sem fios:** O administrador informa o PDC para se ligar à rede ad-hoc sem fios;
3. **Desligar-se da rede ad-hoc sem fios:** O administrador informa o PDC para se desligar da rede ad-hoc;
4. **Gerir mapa do edifício:** O administrador entra no modo de administração do mapa de um edifício, que lhe permite as seguintes funcionalidades funcionalidades:
   a) **Criar mapa:** Permite ao administrador criar um novo mapa do edifício;
   b) **Editar mapa:** Permite ao administrador editar os detalhes de um mapa previamente criado;
   c) **Apagar mapa:** Permite ao administrador apagar o mapa do edifício;
5. **Gerir Grupos:** O administrador entra no modo de administração dos grupos do PDC que está a administrar, podendo:
   a) **Criar grupo:** Permite ao administrador criar um novo grupo;
   b) **Editar grupo:** Permite ao administrador editar os detalhes de um grupo;
6. **Enviar ficheiros de um mapa de um edifício ao utilizador:** O administrador informa o PDC que pode começar a distribuir os ficheiros relativos ao mapa do edifício aos utilizadores da rede;
7. **Enviar convite para um grupo ao utilizador:** O administrador informa o PDC que pode começar a enviar convites de grupos criados nesse PDC para os utilizadores da rede;
8. **Contabilizar o número de pessoas:** Mostra ao administrador do PDC a contabilização do número de pessoas, i.e., a densidade de pessoas no edifício de acordo com os seguintes filtros:
   a) **Por Divisão:** Mostra o número de pessoas por divisão do edifício;
   b) **Por Data:** Mostra o número de pessoas no edifício por data;

O HA tem um único actor, a aplicação MDA executada pelos telemóveis, oferecendo-lhe a seguinte funcionalidade:

1. **Ver informação de uma divisão:** Permite ao MDA descobrir a localização do utilizador dentro de um edifício ao saber a informação do *hostspot* mais próximo de si.

O UTD tem como actor o PDC e oferece-lhe as seguintes funcionalidades:

1. **Gerir grupos universais:** Permite ao PDC gerir os seus grupos na base de dados global (que guarda todas os grupos criados por todos os PDC do mundo), onde pode:
   a) **Criar grupo:** Cria um novo grupo na base de dados universal, caso ainda não exista.
   b) **Editar grupo:** Edita os detalhes de um grupo na base de dados global.

4.2 Diagrama Arquitectural

É desejável que o ANEE seja um sistema modulável, pois futuramente este pode ser aprofundado e melhorado acrescentando-lhe valor. A arquitectura do ANEE é estruturalmente simples, quer em termos das funcionalidades da rede ad-hoc, quer em termos das aplicações que corressem por cima da rede.

O ANEE está estruturado em três camadas, ou níveis, nomeadamente: CORE, APPLICATION e API. O primeiro nível trata de todas as funcionalidades de baixo nível do ANEE, i.e. do acesso dos DM e do PDC à rede ad-hoc, da troca de pacotes de dados e da construção da tabela de IPs da rede necessárias para a comunicação. O segundo nível trata de todas as aplicações que utilizam os recursos do nível CORE. O último nível do ANEE é transversal aos dois primeiros níveis no sentido em que usa recursos de ambos os níveis, *CORE* e *APPLICATION*, permitindo a utilização do ANEE por agentes externos.

A grande vantagem de uma divisão por níveis, como a mostrada na figura 4.4, é o facto de se conseguir abstracção inter-níveis. Todas as aplicações do nível superior *APPLICATION* não têm de saber como é que o nível inferior *CORE* trabalha, simplesmente diz-lhe “Envia esta informação para aquele destino”. O mesmo acontece no sentido contrário, onde o nível *CORE* pode dizer ao nível *APPLICATION* “Recebi novos dados do dispositivo X. Trata deles”. Assim, mesmo que se altere completamente um dos níveis, os outros níveis mantém-se exactamente iguais.

O primeiro nível, CORE, implementa todos os métodos necessários para um DM se ligar à rede ad-hoc e trocar pacotes de dados. Ao conjunto desses métodos dá-se o nome de Ad-hoc Managment System (AMS).

O segundo nível utiliza os recursos do CORE. É neste nível que todas as infraestruturas de software são desenvolvidas utilizando as funcionalidades que o CORE oferece. No caso do ANEE, existem três aplicações que já foram mencionadas: MDA, PDC e HA. O MDA é executado nos *smartphones* (i.e. nos DM), o PDC contabiliza o número de pessoas e gere os grupos de um edifício, e o HA implementa os *hotspots* para localização dos utilizadores nos edifícios. Todas estas aplicações usam os recursos da rede ad-hoc directamente.

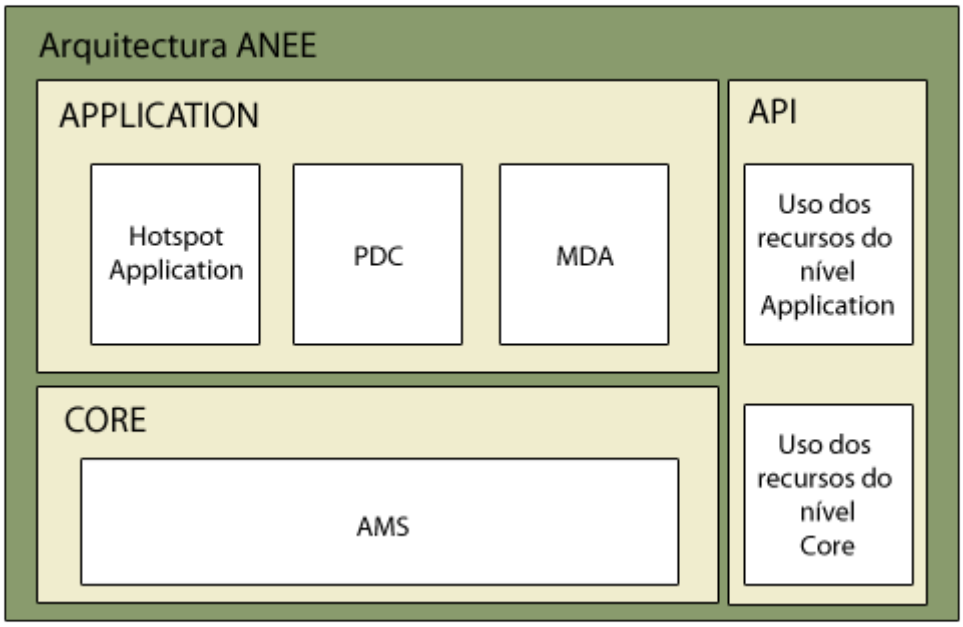


Figura 4.4: Arquitectura do ANEE.

Os DM são responsáveis pela criação da rede ad-hoc, sendo cada um deles um nó da rede. Por seu lado o PDC utiliza a rede para enviar convites de grupos, enviar mapas dos edifícios aos utilizadores e mais importante de tudo, fazer a contabilização e mapeamento dos utilizadores pelo edifício. Os hotspots utilizam os recursos da rede no sentido em que necessitam de fazer broadcast da sua localização para que os DM que passem no seu raio de alcance.

Por fim, e por forma a fornecer todas as funcionalidades desenvolvidas ao mundo, foi desenvolvida uma API que permite simplificar a utilização do ANEE, sem preocupações de baixo nível.

4.2.1 Nível *Core*

O nível CORE é constituído por diversas classes que compões o AMS. Uma classe define o comportamento dos seus objectos através de métodos e os estados possíveis desses objectos através de atributos. As principais classes que compõem o nível CORE são: *ListIP*, *Serializer*, *Sender*, *Receiver*, *BroadcastHello* e *BroadcastRcv*, como mostrado na figura 4.5.

O nível CORE tem os seguintes objectivos:

- Manter uma lista de IPs e de portos de todos os elementos da rede.
- Serielizar todos os objectos para serem enviados e recebidos como bytes.
- Enviar pacotes de dados para um endereço IP específico.
- Receber pacotes de dados.
- Enviar pacotes *Hello* via broadcast.

- Receber pacotes broadcast dos hotspots e pacotes *Hello*.

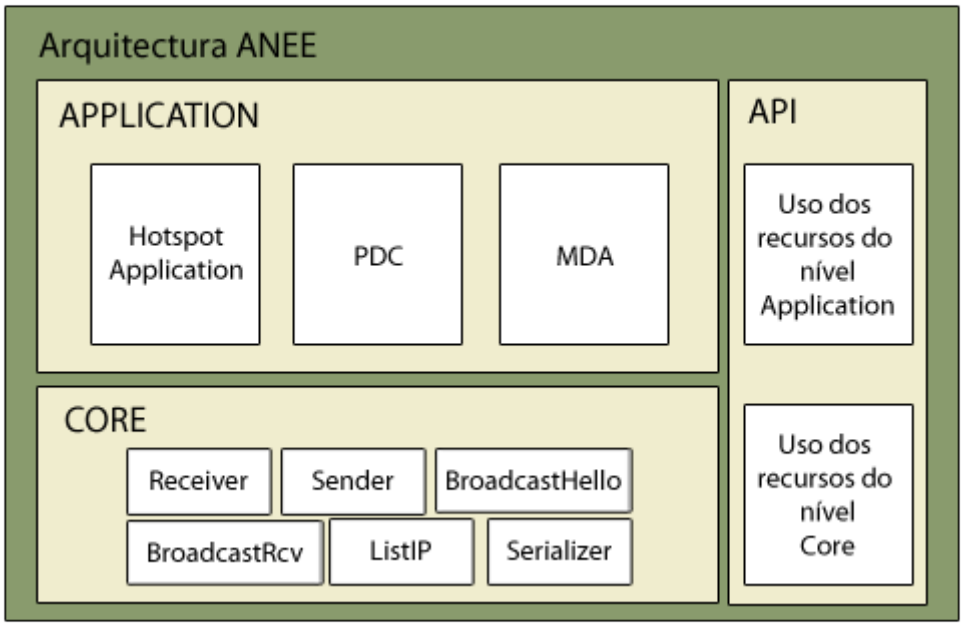


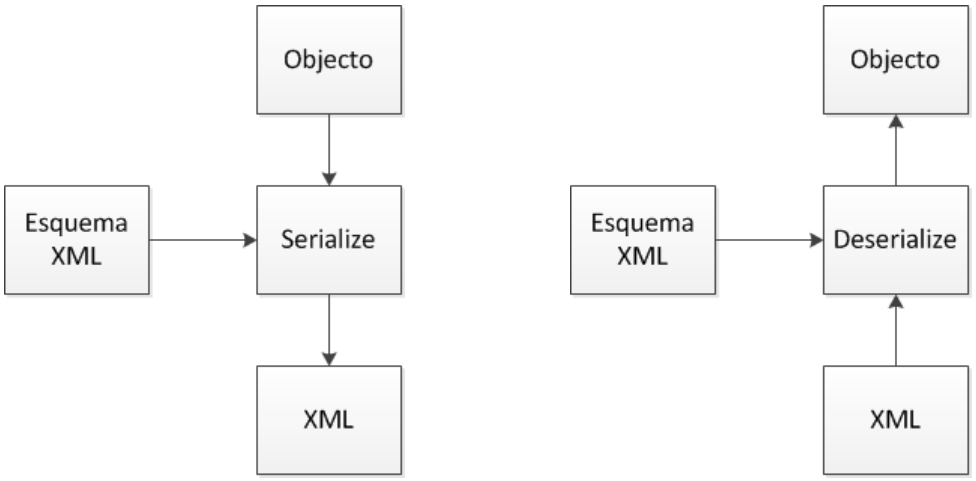
Figura 4.5: Detalhe da arquitectura do ANEE.

A nível de implementação cada objectivo será implementado numa classe. O primeiro caso é implementado pela classe ListIP e é responsável por manter uma lista de IPs da rede e respectivos portos. Esta lista funciona como uma tabela de encaminhamento de modo a que cada elemento da rede saiba a que porto se deve ligar quando quer enviar pacotes para um determinado IP. Esta tabela está representada na tabela 4.1.

Tabela 4.1: Exemplo de uma tabela de IPs e portos, mantida por cada elemento da rede ad-hoc.

Endereço IP	**Porto**
169.254.50.33	1958
169.254.184.70	2854
169.254.22.90	3589
169.254.20.189	1450
169.254.50.225	7895

O segundo ponto é implementado pela classe *Serializer*. Esta classe permite transformar qualquer objecto num *array* de bytes e vice-versa tendo em conta esquemas XML (figura 4.6), que serão explicadas mais à frente. Este processo garante-nos duas vantagens, a primeira transparência em termos de sistemas operativos dos DM uma vez que o XML é uma linguagem universal e standard, e em segundo temos um *array* de bytes completamente preparados para enviar como stream de dados através do protocolo Transmition Control Protocol (TCP).



(a) Serializer (b) Deserializer

Figura 4.6: Processo de conversão de (a) objecto em XML e de (b) XML em objecto.

O terceiro ponto é implementado pela classe Sender (figura 4.7). É nesta classe que o envio de dados é efectuado, utilizando o protocolo TCP bem como as duas classes descritas anteriormente. Utiliza a classe ListIP para saber a que porto é que se deve ligar quando quer enviar dados para um determinado IP (e.g. na tabela 4.1 um DM que quisesse enviar para o IP 169.254.22.90 teria de se ligar ao porto 3589). Para além disso utiliza a classe Serializer de modo a transformar o objecto a enviar numa *string* XML e consequentemente num *array* de bytes que seja reconhecido pelo protocolo TCP.

O processo de receber pacotes de dados é feito pela classe *Receiver* (figura 4.8) e apenas tem de efectuar o *deserialize* da informação recebida, i.e. converter o conjunto de bytes recebidos e convertê-los em XML e entregá-los a um processo que trate dessa informação.

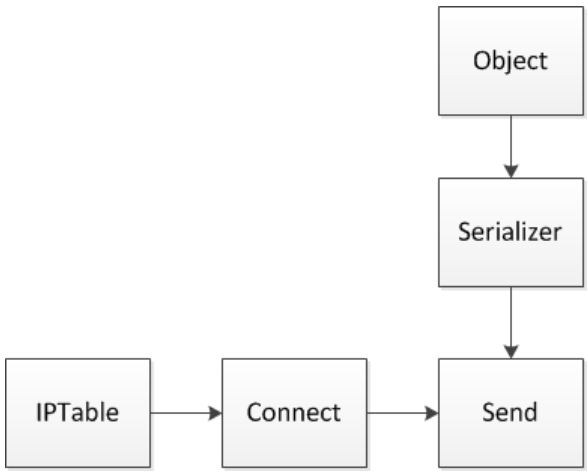


Figura 4.7: Estrutura da classe sender.

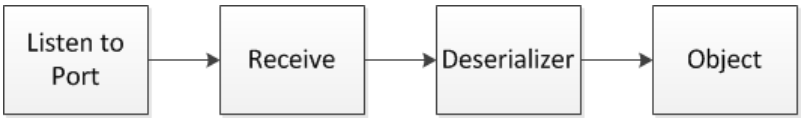


Figura 4.8: Estrutura da classe receiver.

O envio de pacotes *Hello* e consequente recepção é essencial para se poder construir a tabela de IPs da rede. Quando um DM se liga à rede ad-hoc envia um pacote *Hello* via broadcast. Os restantes DM ao receber estes pacotes actualizam a sua lista de IPs. Se não receberem um pacote *Hello* durante algum tempo assumem que o DM já não está na rede e retira-o da lista. O envio e recepção de pacotes *Hello* serão apresentados com mais detalhe posteriormente.

Os *hotspots* espalhados pelo edifício são configurados para ter informação acerca da divisão onde se encontram. Esta informação é transmitida via *broadcast* para os DM. Para esta troca de dados foi também criada a classe ArduinoBroadcast[1]que trata do envio e recepção deste tipo de pacotes. Esta última funcionalidade não é essencial para o funcionamento da rede, mas é essencial quando se necessita de fazer um mapeamento dos utilizadores pelo edifício.

4.2.2 Nível *Application*

O nível APPLICATION (ou Aplicação) é onde se implementa toda a lógica de processamento dos pacotes trocados na rede, pelo que corre por cima do nível CORE. É de notar que o número de aplicações neste nível não está limitado às três aplicações mostradas na figura 4.4. Se uma determinada aplicação necessitar utilizar os recursos do nível CORE e for uma componente que se enquadre dentro dos objectivos do ANEE então é neste nível que essa aplicação se integrará. No contexto desta dissertação existe três aplicações diferentes:

- Mobile Device Application (MDA).
- People Density Controller (PDC).
- Hotspot Application (HA).

MDA - Mobile Device Application

O MDA é executado nos telemóveis dos utilizadores. É aqui que a rede começa a ser montada, pois são os utilizadores que criam a rede ad-hoc através dos seus telemóveis, sendo cada utilizador visto como um DM. Esta aplicação tem as funcionalidades descritas na figura 4.3 com o actor utilizador.

Quando um utilizador entra num edifício e liga o seu telemóvel juntamente com o MDA, a aplicação começa a descobrir todos os outros utilizadores que também se encontram na rede. Ao descobri-los mostra ao utilizador todos aqueles que pertencem aos seus grupos de interesse e que também se encontram naquele momento na rede, permitindo que o utilizador interaja com essas pessoas. O utilizador também pode ver um mapa com a planta do edifício e identificar a sua localização no mesmo, bem como a localização dos outros membros dos seus grupos.

[1]O nome Arduino vem do micro controlador utilizado para implementar os hotspots. Informação adicional pode ser vista em http://www.arduino.cc

PDC - People Density Controller

O PDC pode ou não correr na mesma máquina que controla o BMS, mas tem de ser um ponto central no edifício para onde todos os dispositivos móveis possam enviar a sua localização por forma a que o PDC possa fazer a contagem dos utilizadores. Por essa razão tem de ter acesso aos recursos da rede ad-hoc da mesma maneira que o MDA pelo que também pertence a este nível.

Esta aplicação faz todo o seu trabalho de um forma automática, sendo necessário apenas um administrador para configurá-lo. O administrador é apenas responsável por criar os grupos específicos daquele PDC, bem como os mapas que correspondem à planta do edifício onde o PDC se encontra. Após esta configuração inicial o PDC envia automaticamente esses grupos e os ficheiros dos mapas do edifício aos dispositivos móveis. Para além desta função o PDC contabiliza a quantidade de pessoas no edifício mantendo um registo histórico dessa informação.

HA - Hotspot Application

O HA é a aplicação executada nos hotspots, que são micro controladores com capacidade comunicação sem fios. Nesse sentido, como estes são responsáveis por informar os diferentes DM da sua localização dentro do edifício, também têm de utilizar os recursos do CORE.

Os detalhes de implementação do HA serão vistos mais à frente, contudo cada *hotspot* é configurado inicialmente para guardar as informações da divisão onde se encontra, como por exemplo, o nome da divisão, o piso do edifício onde a divisão está e as suas coordenadas. O *hotspot* pode controlar também até 6 sensores, pelo que a informação desses sensores também tem de ser guardada e depois enviada para os *smartphones*. Esses dados são enviados via broadcast para todos os *smartphones* que passem dentro do raio de alcance do hotspot.

Quando um DM de um utilizador está no alcance do *hotspot* começa automaticamente a receber as informações desse *hotspot*, que contém todas as informações da divisão onde se encontra. Neste momento o telemóvel do utilizador já tem as coordenadas do local onde se encontra e pode mostrar num mapa do edifício a sua localização. Sempre que o utilizador passar por um novo *hotspot* a sua localização será actualizada.

Como é que saber a localização dos utilizadores ajuda em termos de eficiência energética? Se todos os dispositivos móveis souberem a sua localização, então o PDC consegue saber exactamente em que divisão é que cada dispositivo móvel se encontra e consequentemente a posição de cada pessoa, fazendo assim a contagem dessas mesmas pessoas. Se a divisão onde o utilizador se encontra estiver vazia, então o BMS pode utilizar essa informação para diminuir a circulação de ar nessa divisão, contribuindo para um melhor bem estar das pessoas que lá se encontram e para uma melhor utilização dos recursos energéticos e consequentemente para a poupança de energia.

4.2.3 Application Programing Interface (API)

A API é um conjunto de métodos estabelecidos pelo ANEE (e por qualquer outro sistema em geral) para a utilização das suas funcionalidades por terceiros sem que estes necessitem de se envolver com os detalhes internos desse sistema. O objectivo principal é o de reduzir o nível de complexidade que um utilizador que queira utilizar o sistema tem de enfrentar.

A API desenvolvida para o ANEE oferece essencialmente uma forma transparente de utilizar as funcionalidades do nível CORE e do nível APPLICATION. No caso do CORE é possível utilizar métodos para um dispositivo se ligar e desligar da rede adhoc, enviar e receber pacotes de dados. Em relação ao nível de aplicação oferece-se um conjunto de *webservices* que permitem o acesso à informação do PDC, como os grupos criados por esse PDC, o mapa do edifício que o PDC monitoriza, o número de pessoas em cada divisão do edifício num determinado dia e hora.

4.2.4 Universal Tag Deliver - UTD

Para se impedir que os diversos PDC espalhados pelos edifícios do mundo criem grupos repetidos, existe uma entidade que controla a criação destes grupos, o UTD. O UTD é um *webservice* que é acedido pelo PDC sempre que este quer criar ou editar um grupo. No caso da criação de grupos o UTD atribui um Globally Unique Identifier (GUID) para aquele grupo e retorna-o ao PDC para que este possa usar quando distribuir o grupo pela rede ad-hoc. O UTD apoia-se fortemente numa base de dados para guardar e gerir todos os grupos do mundo.

4.3 Diagrama de Entidades e Relacionamentos

O Diagrama de Entidades e Relações (DER) define o modelo de dados de um sistema. Neste caso o modelo de dados do ANEE é composto por dois DER que descrevem o conjunto de dados do MDA e do PDC.

O modelo de dados do MDA (figura 4.9) é composto por 8 entidades, nomeadamente: *UserProfile*, *Group*, *Local*, *Sensor*, *Map*, *Level*, *Space* e *Coordinate*. A entidade *UserProfile* representa as informações do utilizador com atributos tais como o seu nome (*name*), o seu email, o seu número de telefone (*phone*) e o IP do seu telemóvel. A entidade *Group* representa um grupo e é composto por um GUID, um nome (*tag*), uma imagem (*image*), uma descrição (*Description*), um estado (*Status*) e um país (*Country*).

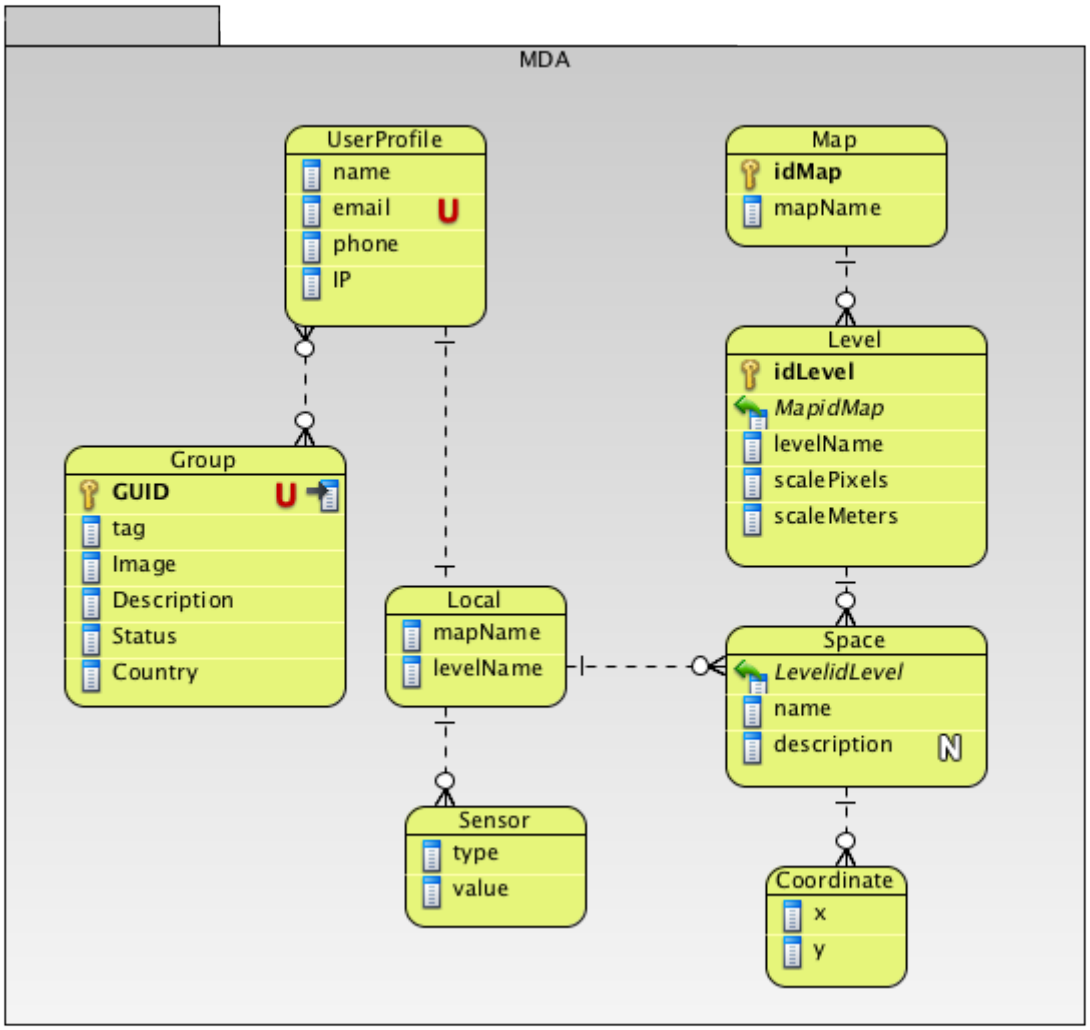


Figura 4.9: Diagrama de entidades e relacionamentos do MDA.

As últimas 6 entidades são todas utilizadas para descrever o mapa de um edifício. Assim a entidade *Map* apresenta o nome do edifício, a entidade *Level* representa um piso do edifício tendo como atributos um identificador de piso (*idLevel*), um nome (*levelName*), uma escala em metros (*scaleMeters*) e uma escala em pixels (*scalePixels*) que são utilizadas para desenhar o mapa. A entidade *Space* representa uma divisão do edifício e é composta por um nome (*name*) e uma descrição (*description*). A classe *Coordinate* é composta pela coordenada X e coordenada Y que representam a localização da divisão no mapa. A classe *Local* é composta pelo nome do mapa (*mapName*), o nome do piso (*levelName*) e representa a localização do utilizador no mapa. A entidade *Sensor* dá suporte às leituras dos sensores de uma determinada divisão.

Um utilizador pode estar associado a vários grupos, razão pela qual a entidade *UserProfile* apresenta uma relação de *n* para *n* com a entidade *Group*. Um utilizador apenas pode estar num local de cada vez daí a entidade *UserProfile* ter uma relação de 1 para 1 com a entidade *Local*. Um edifício pode ter vários pisos, logo a entidade *Map* tem uma relação de 1 para *n* com a entidade Level e o mesmo acontece com as divisões, em que cada piso pode ter várias divisões pelo que a entidade *Level* tem uma relação de 1 para *n* com a entidade *Space*. As divisões de um edifício podem ter àreas muito dispares e uma divisão pode ter mais do que uma coordenada pelo que a entidade *Space* tem uma relação de 1 para *n* com a entidade *Coordinate*.

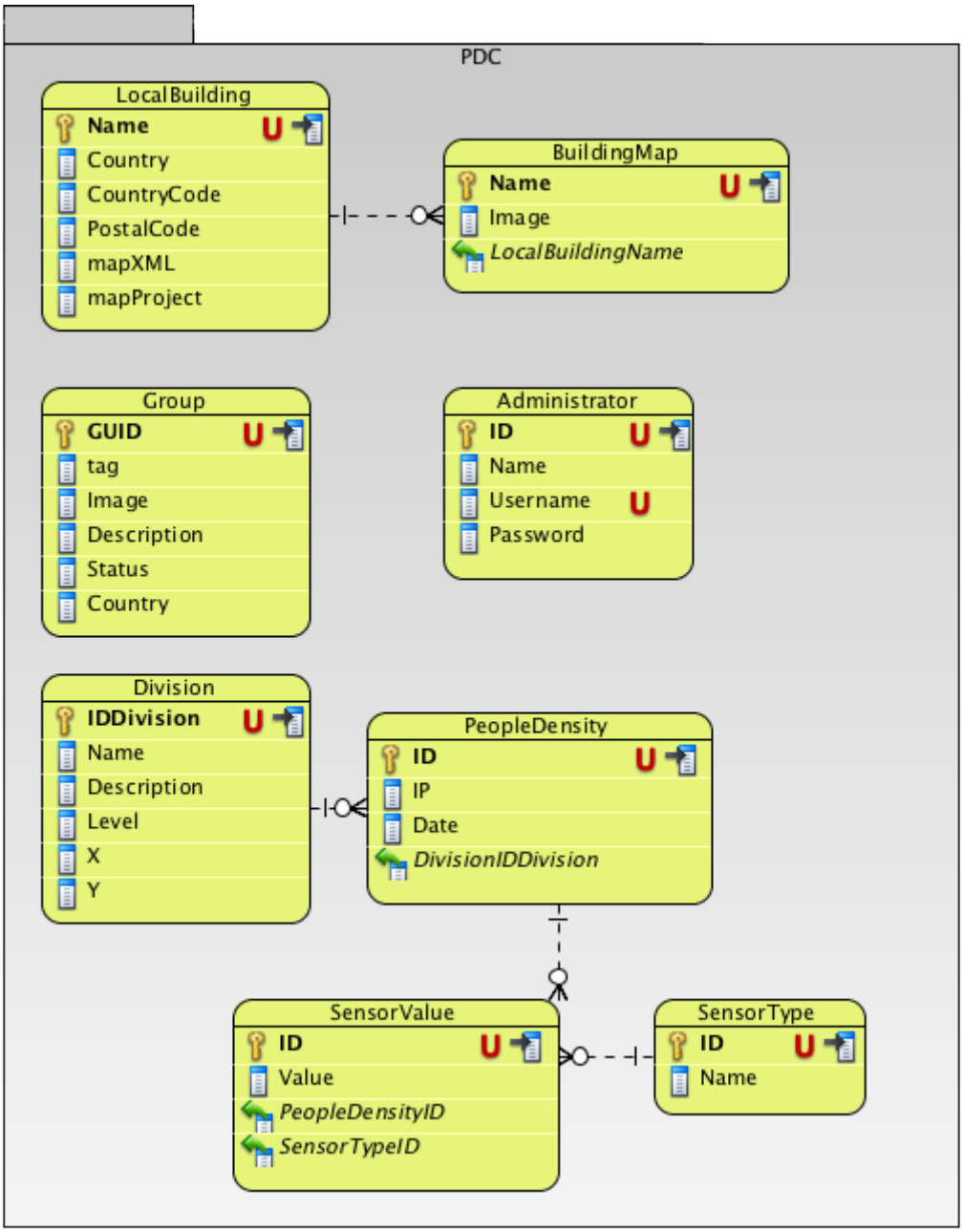


Figura 4.10: Diagrama de entidades e relacionamentos do PDC.

O modelo de dados do PDC, mostrado na figura 4.10 é composto por 8 entidades, nomeadamente: *LocalBuilding*, *BuildingMap*, *Group*, *Administrator*, *Division*, *PeopleDensity*, *SensorValue* e *SensorType*. A entidade *LocalBuilding* guarda todas as informações do edifício, o seu nome, o país, código do país e código postal (necessários para criar novos grupos), dois ficheiros XML que guardam todos as informações do mapa desse edifício, ficheiros estes que serão vistos mais à frente.

Um edifício poderá ter vários mapas (i.e. várias imagens das plantas do edifício), tipicamente um mapa por piso, sendo com esse propósito que a entidade *BuildingMap* existe. Guarda-se o nome dessa imagem e a imagem em si. Como o PDC é também responsável por criar novos grupos, a entidade *Group* representa estes grupos, com todas as suas informações, o GUID, uma tag que é o nome do grupo, uma imagem, uma descrição, um status (para definir se está a transmitir ou não o

grupo pela rede ad-hoc) e um país, sendo exactamente à entidade *Group* do MDA.

A entidade *Administrator*, define todos os administradores que têm acesso ao sistema e que controlam o PDC. Tipicamente basta um administrador, mas é possível haver mais. A entidade *Division* representa uma divisão do edifício e guarda o nome da divisão, a sua descrição, o piso em que se encontra e as coordenadas X e Y dessa divisão. É de notar que a informação desta entidade é a que é inserida nos respectivos *hotspots* do edifício. Daí a importância da informação inserida nos *hostspots* ser concordante com a informação da base de dados do PDC.

A entidade *PeopleDensity* é responsável por guardar as informações relativas aos perfis de utilizador recebidos da rede, sendo portanto nesta tabela que se faz a contabilização das pessoas. Cada entrada desta entidade tem um ID único, um IP associado (cada IP diferente representa uma pessoa diferente num determinado tempo), uma chave estrangeira que aponta para a divisão onde o utilizador se encontra e uma data correspondente à altura em que o perfil de utilizador foi recebido e inserido na base de dados. Com esta entidade pode-se portanto, com uma simples pesquisa à base de dados, saber o número de pessoas que estiveram, numa determinada divisão do edifício num determinado intervalo de tempo. Pode-se saber também as pessoas que estão no momento actual em cada divisão fazendo para isso uma pesquisa em que a data de inicio e fim é a data actual. As entidades seguintes, *SensorValue* e *SensorType* dão suporte ao outro tipo de monitorização, monitorização sensorial, que não será discutida aqui.

Capítulo 5

ANEE: Implementação

5.1 Tecnologias

Toda a implementação do ANEE foi baseado em plataformas .NET da Microsoft. Assim sendo utilizaram-se as seguintes tecnologias nas diferentes aplicações desenvolvidas, MDA, PDC, UTD e HA.

MDA

Para o MDA, aplicação que corre nos dispositivos móveis utilizou-se smartphones com sistema operativo Microsoft Windows Mobile 6.5 e com suporte para Wi-Fi. Nesse sentido desenvolveu-se a aplicação com recurso a biblioteca .NET Compact Framework (.NET CF) da Microsoft, específica para desenvolvimento de software para a plataforma Windows Mobile, baseado na linguagem C#.

Ao longo do desenvolvimento, notou-se algumas limitações que a biblioteca .NET CF tinha em termos de funcionalidades de mais baixo nível. Para colmatar essas limitações recorreu-se também a uma biblioteca *open-source* para Windows Mobile, baseada também em C#, chamada OpenNETCF[1], que permite o controlar facilmente a interface de rede dos DM com uma linguagem de mais alto nível como é o caso do C#.

PDC

O desenvolvimento do PDC foi optimizado para correr numa plataforma Windows Vista / Windows 7. Mais uma vez utilizou-se a biblioteca .NET da Microsoft e a linguagem C# para o desenvolvimento

[1] A bibilioteca OpenNETCF pode ser descarregada em http://www.opennetcf.com/.

desta aplicação. O Sistema de Gestão de Base de Dados (SGBD) utilizado foi o Microsoft SQL Server 2008 por permitir um integração quase transparente com o .NET.

UTD

O UTD é um *webservice* alojado num servidor web. Como é um *webservice* totalmente desenvolvido com recurso à biblioteca .NET da Microsoft e com o uso de um SGBD SQL Server 2008, alojou-se o *webservice* no servidor Internet Information Services (IIS) da Microsoft.

HA

A HA é executado numa plataforma Arduino UNO. O Arduino UNO tem um microcontrolador ATMEGA 328 de 16 MHz, memória flash de 32KB, Static Random Access Memory (SRAM) de 2KB e Electrically-Erasable Programmable Read-Only Memory (EEPROM) de 1KB, 14 pinos digitais de Entrada / Saída (E/S) e 6 pinos analógicos de entrada. Como o Arduino UNO por si só não suporta Wi-Fi, foi adicionado um módulo WiShield que lhe conferisse capacidades Wi-Fi.

O desenvolvimento desta aplicação foi feito com recurso ao ambiente de desenvolvimento Arduino na linguagem C.

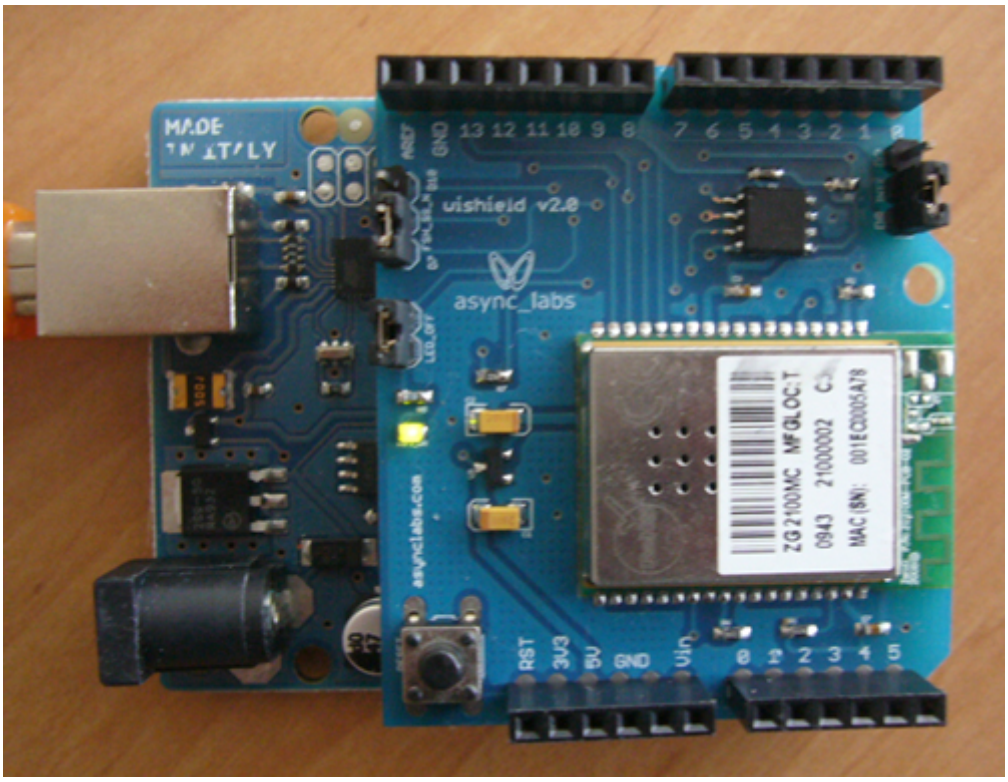


Figura 5.1: Arduino UNO com WiShield.

5.2 Diagramas de Classes

Para se construir os diagramas de classes do ANEE foi adoptada uma estratégia de desenvolvimento em camadas, nomeadamente Interface, Controlo e Entidade, i.e. seguindo o modelo Model-View-Controller (MVC). Nos próximos capítulos serão apresentados os diagramas de classes para as três principais aplicações apresentadas no capítulo 4.

O nível de Interface contém todas as classes necessárias à interacção com o utilizador, como por exemplo o toque num botão ou a apresentação de uma tabela de dados ao utilizador, especificando o que acontece a partir dessa interacção. O nível de Controlo, trata de toda a lógica que é necessária efectuar desde que o utilizador interage com o sistema até que o resultado final é mostrado novamente ao utilizador. Por fim, o nível de Entidades trata da implementação de todas as classes necessárias ao armazenamento de dados que serão alvo de processamento por parte dos dois níveis anteriores, Interface e Controlo. Note-se que este armazenamento de dados pode ou não ser permanente, i.e. pode ser memória temporária onde os dados são destruídos quando a aplicação encerra, ou memória permanente onde os dados são preservados mesmo depois do programa ser encerrado, tipicamente em bases de dados ou ficheiros XML.

5.2.1 Mobile Device Application - MDA

Para se perceber melhor o diagrama de classes é mais fácil começar pelo fim, ou seja pelo nível de entidades. Neste nível, como se pode ver pela figura 5.2, temos três entidades (ou classes) principais, a classe *UserProfile*, a classe *Group* e a classe *Map*.

A classe *UserProfile* especifica um utilizador e todas as suas informações pessoais como o seu nome, o seu número de telefone, o seu e-mail, uma lista de grupos ao qual está associado e a sua localização dentro de um edifício (Tabela 5.1).

Tabela 5.1: Detalhe da entidade *UserProfile*.

Atributo	**Tipo**	**Descrição**
name	string	Nome do utilizador
phone	string	Número de telefone do utilizador
email	string	Email do utilizador
guis	List<Group>	Lista de grupos a que o utilizador está associado
local	Local	Localização do utilizador dentro de um edifício

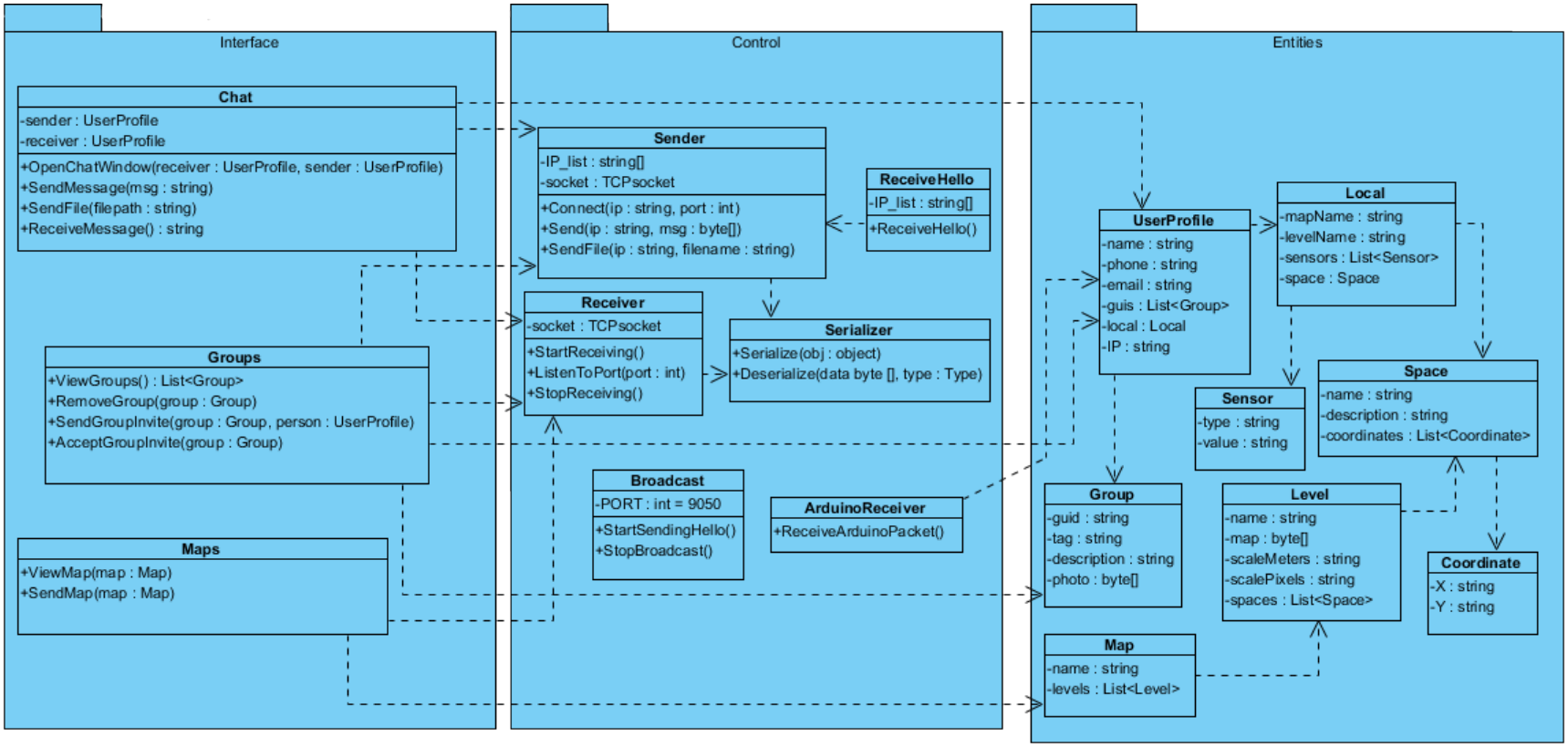


Figura 5.2: Diagrama de classes do MDA.

A classe *Group* especifica um grupo ao qual múltiplos utilizadores podem estar associados. Estes grupos têm um GUID, que é uma chave única para cada grupo, uma tag que representa o nome do grupo, uma descrição e uma fotografia ou imagem representativa desse grupo (Tabela 5.2).

Tabela 5.2: Detalhe da entidade *Group*.

Atributo	**Tipo**	**Descrição**
guid	string	Chave única de identificação
tag	string	Nome do grupo
description	string	Descrição do grupo
photo	byte[]	Fotografia ou imagem do grupo

A classe *Map* especifica um mapa de um edifício. É composta por um nome do mapa e uma lista de pisos que esse edifício terá (Tabela 5.3).

Tabela 5.3: Detalhe da entidade *Map*.

Atributo	**Tipo**	**Descrição**
name	string	Nome do mapa
levels	List<Level>	Lista de pisos do edifício

Estas três entidades principais utilizam outras entidades auxiliares, por relação. A entidade *Level* por exemplo é utilizada pela entidade *Map* e específica um piso de um edifício e nesse sentido tem um nome, um mapa (a imagem da planta do piso), uma escala em metros, uma escala em pixeis e uma lista de divisões que esse piso tem (Tabela 5.4).

Tabela 5.4: Detalhe da entidade *Level*.

Atributo	**Tipo**	**Descrição**
name	string	Nome do piso
map	byte[]	Imagem da planta do piso
scaleMeters	string	Escala do mapa em metros
scalePixels	string	Escala do mapa em pixels
spaces	List<Space>	Lista de divisões do piso

A classe *Level* por sua vez também tem as suas relações, nomeadamente com a classe *Space*, que específica uma divisão do edifício, tendo um nome, uma descrição e uma lista de coordenadas relativas à planta do piso (Tabela 5.6).

Tabela 5.5: Detalhe da entidade *Space*.

Atributo	**Tipo**	**Descrição**
name	string	Nome da divisão
description	string	Descrição da divisão
coordinates	List<Coordinate>	Lista de coordenadas da divisão relativas à planta do piso

No caso da entidade *UserProfile* na tabela 5.1, pode-se verificar a relação com a classe *Local* que indica onde é que um determinado utilizador está, dentro de um edifício. Assim, a classe *Local*

é composta por um nome do mapa onde o utilizador está, o piso onde o utilizador está, a divisão onde está e uma lista que contém os valores dos sensores dessa divisão.

Tabela 5.6: Detalhe da entidade *Space*.

Atributo	**Tipo**	**Descrição**
mapName	string	Nome do mapa
levelName	string	Nome do piso
sensors	List<Sensor>	Lista de sensores e respectivos valores
space	Space	Divisão onde o utilizador está

O nível de Controlo apresenta todas as classes necessárias à comunicação. A classe *Broadcast* é responsável por fazer difusão de pacotes *Hello*, fazendo uso de dois métodos, um para começar o envio dos pacotes *Hello* e um para parar a difusão. Faz uso ainda de um atributo PORT, onde se define o porto de envio dos pacotes *Hello*, neste caso o porto 9050.

Tabela 5.7: Detalhe da entidade *Broadcast*.

Atributo	**Tipo**	**Descrição**
PORT	int	Porto de envio
StartSendingHello	Método	Inicia o envio de pacotes Hello via broadcast
StopBroadcast	Método	Termina o envio de pacotes Hello

A classe *ArduinoReceiver* é a classe responsável pela recepção dos dados enviados pelos *hotspots*, nesse sentido apenas apresenta apenas um método para esse efeito.

A classe *Sender* é uma das principais classes no nível de controlo, é esta classe a responsável por todo o envio de dados para um destinatário específico. Apresenta uma lista de IPs dos dispositivos da rede (Tabela 4.1) e um socket TCP que será responsável pela ligação. Para além destes atributos, apresenta todos os métodos necessárias à ligação a um endereço remoto, bem como o envio de dados ou de ficheiros.

Tabela 5.8: Detalhe da entidade *Sender*.

Atributo	**Tipo**	**Descrição**
IP_list	List<string>	Lista de IPs da rede
socket	TCP Socket	Socket onde a ligação e troca de dados ocorre
Connect(ip:string, port:string)	Método	Inicia uma nova ligação ao endereço IP e porto especificados
Send(ip:string, msg:byte[])	Método	Envia para o endereço IP um conjunto de dados
SendFile(ip:string, filename:string)	Método	Envia para o endereço IP um ficheiro

A classe *Receiver* é outra das principais classes do nível de controlo, até porque é ela que vai tratar todos os dados enviados pela classe Sender. Assim, tem um socket onde vai ouvir todas as novas ligações feitas para si e receber os dados.

Para se receber os pacotes *Hello*, foi criada uma classe *ReceiveHello* que é responsável por esta

Tabela 5.9: Detalhe da entidade *Receiver*.

Atributo	Tipo	Descrição
socket	TCP Socket	Socket onde a ligação e troca de dados ocorre
StartReceiving	Método	Começa a receber dados num determinado porto
ListenToPort(port:int)	Método	Começa a ouvir um determinado porto à espera de novas ligações
StopReceiving	Método	Termina a recepção de dados

acção. Uma vez que é na classe *ReceiveHello* que se começa a construir a lista de IPs da rede então é necessário que esta esteja ligada a classe *Sender* por forma a informá-la das alterações na lista (ou porque foram adicionados novos IPs ou porque foram removidos). Esta classe à semelhança da classe *Sender* mantém uma lista de endereços IP da rede bem como um método para começar a recepção de pacotes *Hello*.

É de notar ainda, a classe *Serializer* que é utilizada quer pelo *Sender* quer pelo *Receiver*, e que é fulcral para haver uma normalização em termos de comunicação. A classe *Serializer* apresenta dois métodos, um para converter um objecto numa *string* XML (utilizada pelo *Sender*) e um para converter uma *string* XML num objecto (utilizada pelo *Receiver*).

Tabela 5.10: Detalhe da entidade *Serializer*.

Atributo	Tipo	Descrição
Serializer	Método	Converte um objecto C# numa string XML
Deserialize	Método	Converte uma string XML num objecto C#

Focando agora no primeiro nível verificamos três classes *Chat*, *Maps* e *Groups*. A classe *Chat* apresenta quatro métodos, o método *OpenChatWindow* abre uma nova janela de chat, com um emissor e um receptor, sendo estes guardados como atributos dentro da classe. O método *SendMessage* permite ao emissor enviar uma mensagem de texto, enquanto que o método *SendFile* permite enviar um ficheiro. Nada disto faria sentido se não se pudesse receber respostas às mensagens enviadas, e é para isso que o método*ReceiveMessage* serve. Percebe-se portando a relação necessária com as classes *Sender* e *Receiver* do nível de controlo.

Tabela 5.11: Detalhe da entidade *Chat*.

Atributo	Tipo	Descrição
receiver	UserProfile	Perfil de utilizador do receptor
sender	UserProfile	Perfil de utilizador do emissor
OpenChatWindow	Método	Inicia uma nova janela de chat
SendMessage(msg:string)	Método	Envia uma mensagem de texto
SendFile(filepath:string)	Método	Envia um ficheiro
ReceiveMessage():string	Método	Processo notificado pela classe Receiver aquando da recepção de uma nova mensagem.

A classe *Groups* é responsável por mostrar ao utilizador todos os grupos em que este está associado, bem como as pessoas que estão na rede e que pertencem aos mesmos grupos que o

utilizador. Para além disso permite ao utilizador aceitar novos convites para grupos, remover grupos em que está registado ou enviar convites para grupos para pessoas próximas de si.

Tabela 5.12: Detalhe da entidade *Groups*.

Atributo	**Tipo**	**Descrição**
ViewMyGroups	Método	Mostra os grupos do utilizador
AcceptGroupInvite(group:Group)	Método	Aceita um novo convite para um grupo
RemoveGroup(group:Group)	Método	Remove um grupo da lista de grupos
ViewPeopleInGroup(group:Group)	Método	Ver as pessoas que estão na rede e que pertencem a um determinado grupo
SendGroupInvite(group, person)	Método	Envia um grupo para uma determinada pessoa

A classe *Maps* trata de mostrar ao utilizador um mapa do edifício onde este se encontra, bem como mostrar a sua posição e a posição das pessoas que estão nos seus grupos. Se o utilizador ainda não tiver os ficheiros necessários à amostragem do mapa então também existe a hipótese de aceitar estes ficheiros.

Tabela 5.13: Detalhe da entidade *Maps*.

Atributo	**Tipo**	**Descrição**
ViewMap	Método	Visualiza mapa do edifício onde o utilizador se encontra
AcceptMap	Método	Aceita um novo mapa
ViewPersonInMap(group:Group)	Método	Ver as pessoas que estão na rede e que pertencem a um determinado grupo no mapa

5.2.2 Person Density Controller - PDC

O PDC partilha muitas das classes com o MDA, principalmente nos níveis de Controlo e de Entidades. No último nível esta partilha faz sentido uma vez que ambas as aplicações têm de conhecer as mesmas entidades de modo a que possam comunicar eficazmente. No nível de controlo, como o PDC também necessita de aceder à rede ad-hoc então as classes também são reaproveitadas em relação ao MDA.

No nível de Interface existem grandes modificações essencialmente porque aquilo que se mostra ao utilizador é completamente diferente em relação ao que se mostra no MDA. O diagrama de classes do PDC é mostrado na figura 5.3.

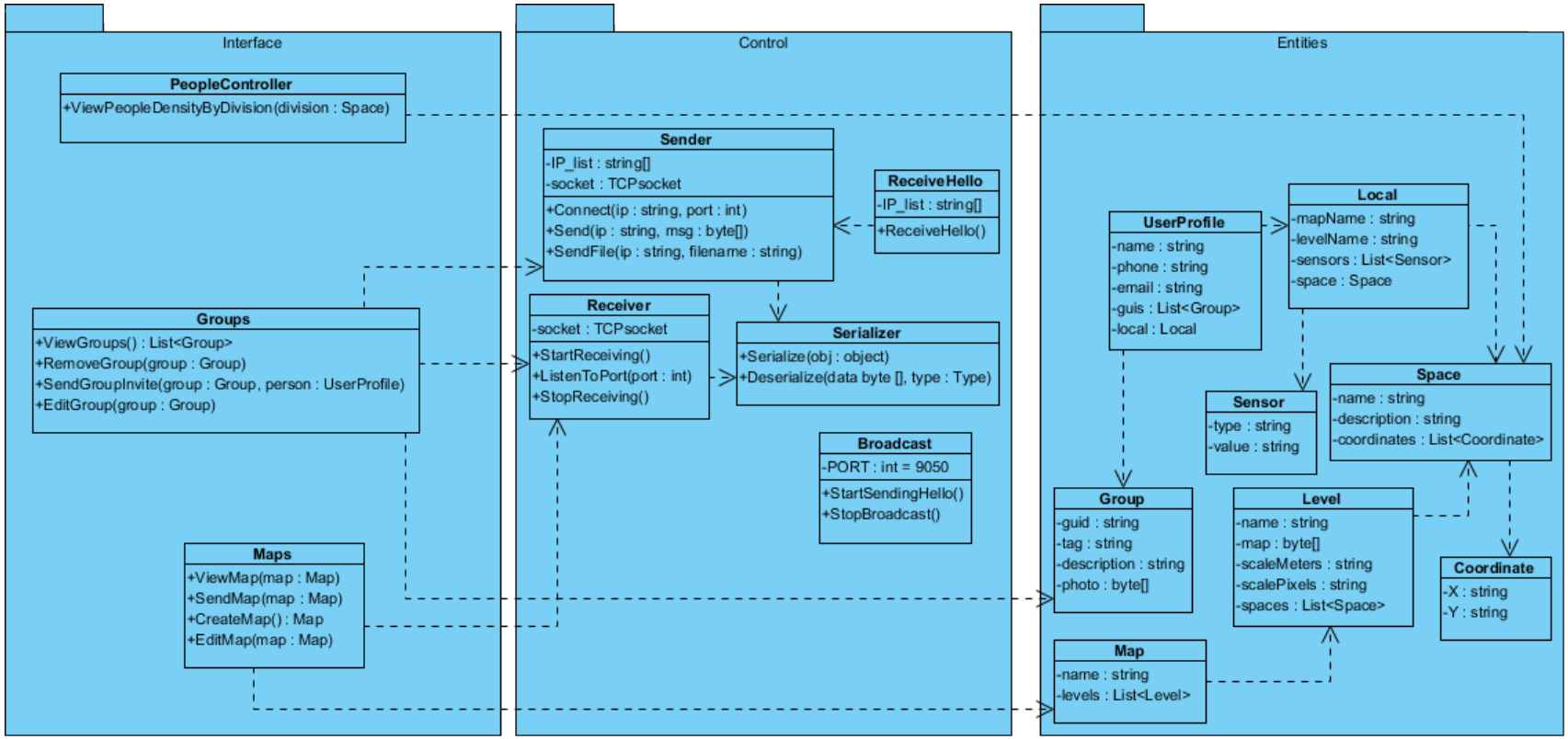


Figura 5.3: Diagrama de classes do PDC.

Como é o PDC o responsável pela criação de grupos, então a classe *Groups* permite ao administrados do sistema fazer uma gestão desses mesmos grupos. Um administrador, depois de autenticado no PDC pode criar novos grupos, apagar grupos ou editá-los.

Tabela 5.14: Detalhe da entidade *Groups* do PDC.

Atributo	**Tipo**	**Descrição**
ViewGroups	Método	Visualiza todos os grupos criados
RemoveGroup	Método	Remove um grupo
EditGroup	Método	Edita os detalhes de um grupo
SendGroupInvite	Método	Envia um convite para um grupo para um determinado dispositivo móvel

O administrador do PDC também é responsável pela criação dos mapas do edifício onde o PDC se encontra. A classe *Maps* dá o suporte necessário à gestão desses mapas bem como ao envio desses mesmos mapas a todos os dispositivos móveis do edifício.

Tabela 5.15: Detalhe da entidade *Maps* do PDC.

Atributo	**Tipo**	**Descrição**
CreateMap	Método	Cria um novo mapa
SendMap	Método	Envia o mapa do edifício para um determinado dispositivo móvel
ViewMap	Método	Visualiza o mapa do edifício
EditMap	Método	Edita o mapa do edifício

Sendo a contabilização das pessoas efectuada pelo PDC, então é possível ver os resultados dessa contabilização no próprio PDC. A classe *PeopleController* dá suporte a esta acção, sendo possível filtrar os resultados por divisão ou por data.

5.2.3 Hotspot Application - HA

Uma vez que o hardware dos *hotspots* é limitado em termos de recursos, não se desenvolveu nenhum diagrama de classes para esta aplicação, até porque ela não apresenta nenhuma classe na sua programação. Ao invés disso, existe uma estrutura que guarda os dados necessários no hotspot e que os envia por broadcast sempre que necessário. Como a memória está limitada a uma EEPROM de 512 bytes, essa memória foi organizada como o mostrado na tabela 5.16.

O *hotspot* guarda toda a informação relativa à divisão onde se encontra. O nome do mapa (ou o nome do edifício), o nome do piso onde se encontra, as coordenadas relativas à divisão onde está e o nome dessa divisão. Note-se que estas informações têm de ser inseridas no *hotspot* aquando do seu processo de configuração e têm de ser consistentes com o mapa criado no PDC.

Tabela 5.16: Organização da memória EEPROM do Hotspot

Tamanho do Nome do Mapa	Nome do Mapa	Tamanho do Nome do Piso	Nome do Piso	Tamanho do Nome da divisão	Nome da Divisão	Coordenada X	Coordenada Y
1 byte		1 byte		1 byte		2 bytes	2 bytes

O primeiro byte da EEPROM especifica quantos bytes é que o nome do mapa irá ocupar, seguido do nome do mapa. De seguida vem mais um byte que especifica quantos bytes é que o nome do Piso irá ocupar, seguido do nome do piso. A mesma lógica é aplicada para o nome da divisão, com 1 byte para o tamanho que o nome da divisão irá ocupar. No final guarda-se 4 bytes, 2 para a coordenada X e outros 2 para a coordenada Y. Podemos ver um exemplo na tabela 5.17.

Tabela 5.17: Exemplo da organização da memória EEPROM do Hotspot

3	DEE	6	Piso 1	8	Sala 1.4	440	300
1 byte	3 bytes	1 byte	6 bytes	1 byte	8 bytes	2 bytes	2 bytes

O exemplo da tabela 5.17 mostra um *hotspot* que está situado na Sala 1.4 do Piso 1 do DEE (Departamento de Engenharia Electrotécnica). Ao receber a informação especificada na tabela 5.17 e com as coordenadas especificadas (440,300), o MDA consegue desenhar facilmente no mapa deste edifício um marcador nessa localização, que corresponde à localização da pessoa.

5.3 Diagramas de Sequência

As aplicações desenvolvidas têm diversas funcionalidades em que o utilizador pode interagir. Essas funcionalidades foram apresentadas no diagrama de casos de uso da figura 4.3 e os diagramas de sequência apresentados de seguida mostram os passos que a aplicação executa sempre que uma dessas funcionalidades é activada.

Um caso prático de como o ANEE pode ser usado, é o caso em que uma pessoa, vamos chamá-la de Marco, vai às compras no maior centro comercial da sua cidade. Sendo uma pessoa preocupada com o ambiente e um apaixonado pelas novas tecnologias liga o seu smartphone e arranca o MDA assim que entra no centro comercial. O Marco não tem de se preocupar com mais nada pois o MDA começa a trabalhar sozinho e de forma automática, ligando-se à rede ad-hoc caso ela já exista, ou então cria ele próprio a rede. O MDA tem agora de descobrir os smartphones das outras pessoas igualmente preocupadas com a escassez dos recursos energéticos no mundo e para isso envia um pacote Hello via broadcast com o seu endereço IP.

A Rita com os seus desejos consumistas também está a passear pelo centro comercial e, ainda que não conheça o Marco gostam de beber o seu café no mesmo lugar. A Rita está suficientemente próxima do Marco, i.e. no raio de alcance do *smartphone* do Marco e, como tal, vai receber o pacote Hello e saber imediatamente o seu endereço IP. Ao receber o pacote Hello, o smartphone da Rita cria um porto onde irá ouvir todas as ligações TCP do smartphone do Marco, e responde-lhe num

pacote User Datagram Protocol (UDP) com essa informação. O smartphone do Marco ao receber o porto TCP do dispositivo da Rita está pronto para lhe enviar dados. Neste momento o smartphone do Marco guarda o endereço IP e o porto TCP a que se deve ligar na sua lista de IPs, para que sempre que lhe queira enviar dados saiba exactamente para onde se deve ligar.

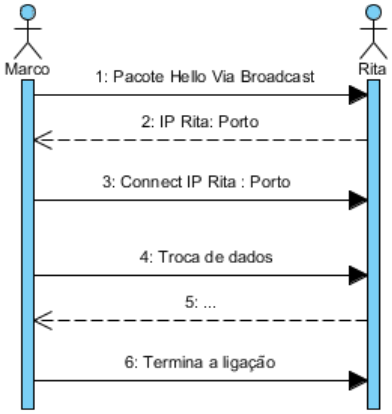


Figura 5.4: Diagrama de Sequência - Troca de mensagens entre o Marco e a Rita.

O Marco tem todas as informações que necessita para enviar dados para a Rita. Mas e a Rita como irá enviar dados para o Marco? O telemóvel do Marco não atribuí-o nenhum porto ao telemóvel da Rita para que este se pudesse ligar ao telemóvel do Marco. Bem, basta pensar que o smartphone da Rita também envia os seus próprios pacotes Hello e como tal o processo repete-se no caso da Rita também.

Quando o utilizador inicia a aplicação dos dispositivos móveis, o MDA, a ligação à rede ad-hoc é efectuada automaticamente, e logo de seguida é mostrado ao utilizador todos uma lista de todos os grupos a que este pertence. O diagrama de sequência da figura 5.5 ilustra esta situação.

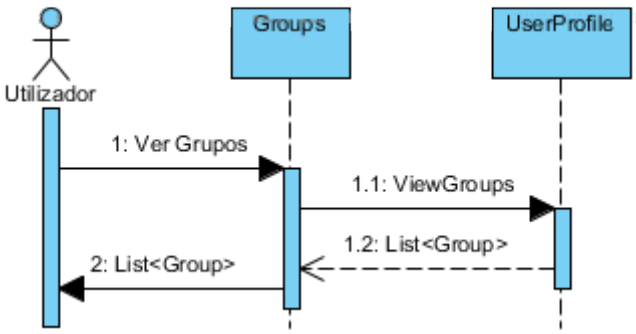


Figura 5.5: Diagrama de Sequência - Visualização dos grupos do utilizador.

Quando o utilizador inicia a aplicação, esta verifica no perfil de utilizador quais são os grupos que este está associado, mostrando-lhe de seguida esses grupos na interface gráfica do programa.

A lista de grupos que é apresentada ao utilizador permite diversas interacções. O utilizador pode ver todas as pessoas que estão num determinado momento na rede e que pertencem aos mesmos grupos que ele, pode iniciar conversações com essas pessoas, pode ver a posição dessas pessoas no mapa do edifício e pode convidar convites para grupos se essas pessoas tiverem pelo menos um grupo em comum com o utilizador. Suponhamos um utilizador, chamado Manuel, que pertence aos grupos A, B e C, e um outro chamado António, que pertence apenas ao grupo A. A aplicação do

Manuel mostrar-lhe-á que o António está no grupo A, mas não no grupo B e C, pelo que o Manuel poderá enviar convites ao António para este aderir aos grupos B e C. O processo de convite é explicado na figura 5.6.

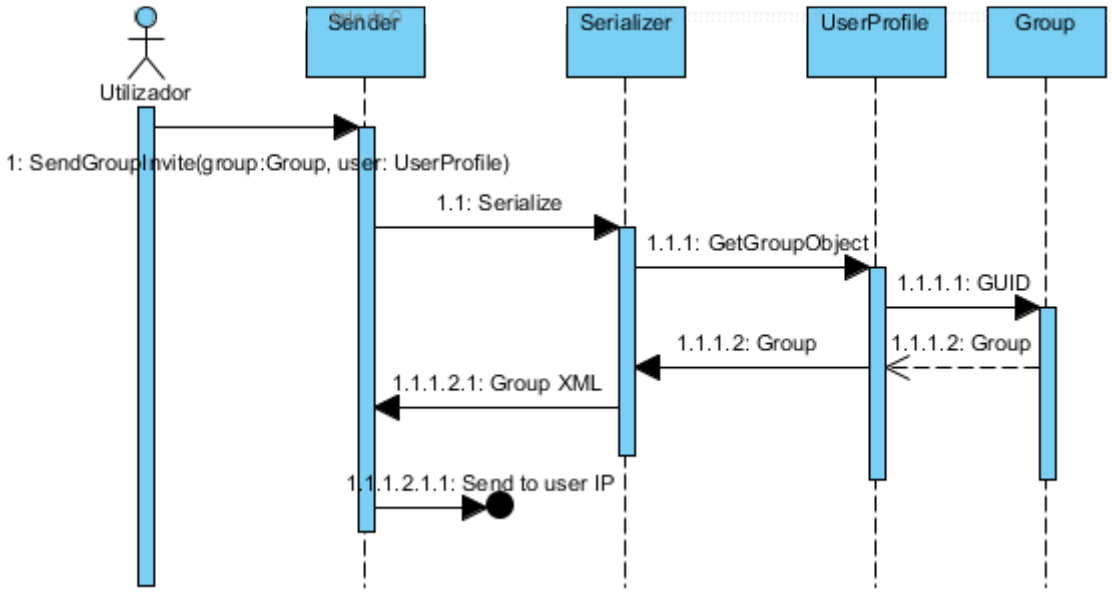


Figura 5.6: Diagrama de Sequência - Envio de convites para um grupo.

O Manuel inicia o processo ao carregar no botão de enviar um convite para um grupo B para o António. Em termos práticos o que a aplicação faz é enviar todas as informações referentes ao grupo B e o perfil de utilizador do António (o UserProfile) para o destino. Estas informações são então convertidas num conjunto de bytes através do processo *serializer* e enviadas para a aplicação que está a ser executada no telemóvel do António. A classe sender trata de descobrir por si só a que porto é que se deve ligar quando vai enviar para o IP do António que é especificado no seu perfil de utilizador.

Do outro lado o António pode aceitar ou recusar o convite do Manuel, como mostrado na figura 5.7. Se recusar, o processo termina imediatamente, caso contrário, o grupo é adicionado à lista de grupos do perfil de utilizador do António. Esta alteração é depois enviada novamente à interface para que seja mostrada uma lista actualizada de grupos ao utilizador.

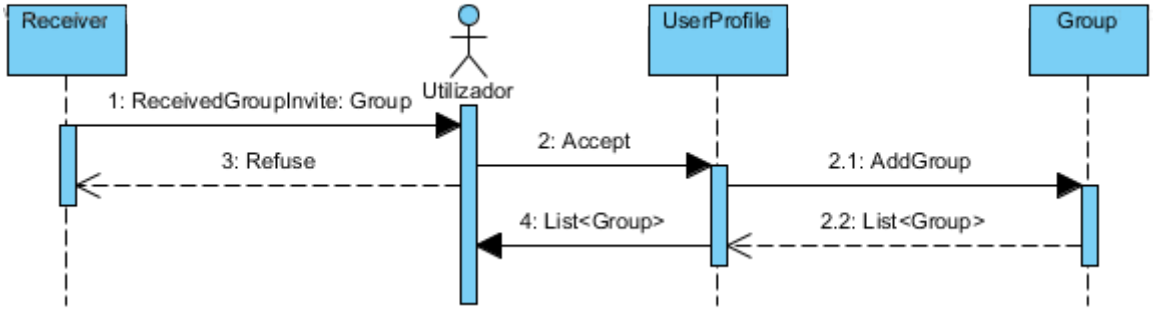


Figura 5.7: Diagrama de Sequência - Recepção de convites de um grupo.

5.4 Contabilização do número de pessoas em cada divisão

Todos os dispositivos móveis a correr o MDA enviam periodicamente os seus perfis de utilizador para todos os endereços IP presentes nas suas listas de IPs. Nessa lista de IP está também o endereço IP do PDC, pelo que este também recebe todos os perfis de utilizador dos dispositivos que estão na rede ad-hoc.

Como foi visto no diagrama de classes do MDA, na figura 5.2, e pela tabela 5.1, o perfil de utilizador para além de guardar os dados pessoais do utilizador, guarda a sua localização dentro do edifício. O PDC ao receber os perfis de utilizador dos dispositivos móveis, sabe automaticamente em que divisão do edifício em que estes se encontram e pode ir guardando esta informação numa base de dados.

Poderia se levantar algumas questões relacionadas com a privacidade dos utilizadores em relação a guardar a localização das pessoas, talvez por alguma invasão de privacidade, pelo que o PDC na sua base de dados nunca guarda dados pessoais dos utilizadores, mas apenas IPs de máquinas, uma vez que nunca poderá haver IPs repetidos ao mesmo tempo na rede. Assim o PDC consegue diferenciar as pessoas apenas pelo endereço IP dos seus telemóveis e fazer uma contabilização correcta das mesmas.

5.5 Diagrama de Entidades e Relações

O DER do ANEE já foi visto, contudo apenas para mostrar os conceitos por trás do modelo de dados e sem os detalhes de implementação. Aqui mostra-se o DER com o tipo de dados de cada atributo de cada uma das entidades que compõem o DER do MDA e do PDC nas figuras 5.8 e 5.9, respectivamente.

Os atributos de cada entidade em cada um dos DER foram explicados no capítulo 4 e o tipo de dados de cada atributo foram escolhidos de acordo com aquilo que cada atributo vai guardar. Caso um atributo tenha de guardar caracteres alfanuméricos então o tipo de dados tem de ser *varchar(n)*, onde n representa o número máximo de bytes que se pode guardar nesse atributo, como por exemplo o atributo *name* da entidade *UserProfile*. Caso o atributo apenas guarde números então o seu tipo de dados é *integer*, como é o caso das coordenadas X e Y na entidade *Division*. Se por outro lado o atributo vai guardar dados binários, como é por exemplo o caso de imagens, então o tipo desse atributo é *binary*, como é o caso do atributo *Image* da entidade *Group*.

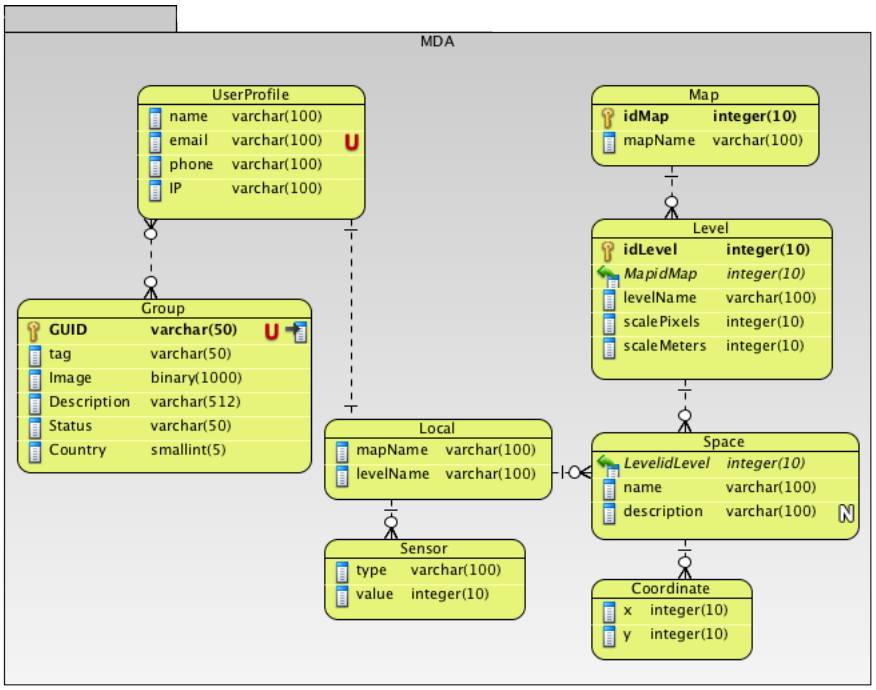


Figura 5.8: Detalhes de implementação do DER do MDA.

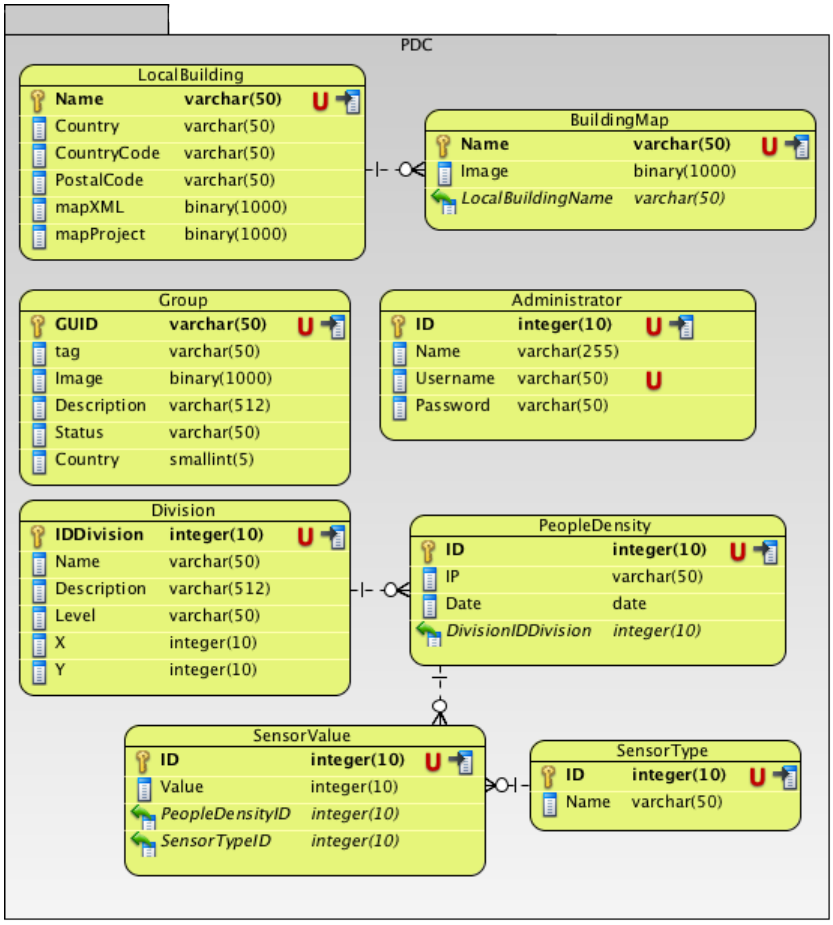


Figura 5.9: Detalhes de implementação do DER do PDC.

5.6 Map Designer: Criação de mapas de edifícios

O Map Designer é uma ferramenta utilizada pelo PDC para a criação de novos mapas de edifícios. Apresenta uma interface gráfica baseada em matrizes que permite ao administrador do PDC criar mapas do edifício de uma forma simples e rápida. As plantas dos diferentes pisos do edifício são guardados em ficheiros Portable Network Graphics (PNG) enquanto que as informações relativas a esses mapas (e.g. nome e descrições das divisões, coordenadas, etc.) são guardadas em ficheiros XML, que por ser uma linguagem universal pode ser aberto por qualquer outro programa que queira vir a aumentar as funcionalidades do Map Designer.

O administrador pode criar uma planta por cada piso do edifício e cada piso pode ter diversas divisões que podem ocupar uma ou mais posições da matriz (ver tabela 5.18). A dimensão da matriz de trabalho pode ser variável e definida pelo administrador, não havendo limites para o número de divisões que se pode criar. Todas as posições da matriz que não sejam preenchidas são assumidas como corredores, ou espaços de ligação entre divisões.

Tabela 5.18: Esquema matricial em que se baseia a construção dos mapas do *Map Designer*.

	0	1	2	3	4	5
0	Div. 1	Div. 1	Div. 1			
1	Div. 1	Div. 1	Div. 1			
2						
3						
4			Div. 2	Div. 2	Div. 2	Div. 2
5			Div. 2	Div. 2	Div. 2	Div. 2

A imagem da planta gerada por um mapa como o da figura 5.18 seria o da figura 5.10. Todas as posições da matriz que pertencem a uma divisão são juntadas na imagem final como um único espaço. Por exemplo, todas as posições marcadas com *Div. 1* na tabela 5.18 são juntadas numa única àrea a verde na figura 5.10. O mesmo acontece na divisão *Div. 2* na àrea a azul.

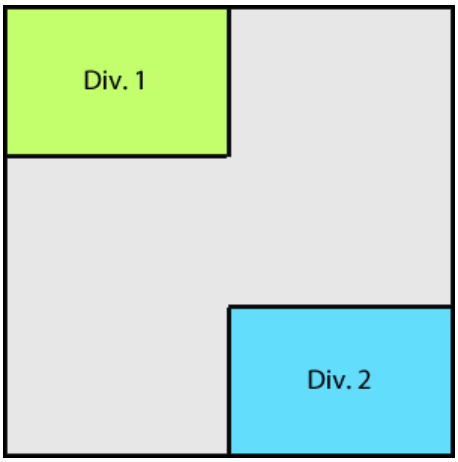


Figura 5.10: Exemplo de um mapa gerado pelo *Map Designer*.

O Administrador pode definir o nome das divisões e as respectivas descrições, mas não tem de se preocupar em dar coordenadas às divisões; essas coordenadas são calculadas automaticamente quando se gera o ficheiro XML que representa o mapa. Esse ficheiro XML tem a estrutura mostrada no apêndice H. No caso do mapa da figura 5.18, o XML gerado é apresentado no apêndice I. O ficheiro XML apresentado no apêndice I é reconhecido pelos DM na sua aplicação MDA e juntamente com as imagens das plantas também geradas pelo *Map Designer*, conseguem informar o utilizador da sua localização.

Capítulo 6

ANEE: Exemplo Ilustrativo de Aplicação

Este capítulo apresenta os principais ecrãs referentes às funcionalidades que o ANEE oferece. Começa por mostrar as funcionalidades do MDA, nomeadamente o sistema de grupos, de chat e de mapas, mostrando depois o PDC e algumas das suas principais funções, nomeadamente a criação de grupos, a partilha de grupos, a criação de mapas e a contabilização da quantidade de pessoas num edifício.

Figura 6.1: Listagem de grupos de um utilizador no MDA.

Um utilizador pode falar com qualquer um outro, desde que pertençam aos mesmos grupos, por exemplo o utilizador João Santos representado na figura 6.1, pode falar com o utilizador Armando Freitas pois pertencem ao mesmo grupo “FCT UNL”. Na janela de conversação, mostrada na figura 6.2, o utilizador pode trocar mensagens de texto e ficheiros de música, de texto e de vídeo.

Figura 6.2: Janela de chat do MDA.

Para além destas funcionalidades, sempre que o utilizador passa por um *hotspot*, o MDA fica a saber a posição do utilizador e pode mostrá-la na planta do edifício, como mostrado na figura 6.3.

Figura 6.3: Posição do utilizador mostrada no mapa no MDA.

Em relação ao PDC, depois do administrador se autenticar e caso não haja nenhum mapa ainda associado ao PDC, então é pedido ao administrador que o crie através do *Map Designer* apresentado na figura 6.4. Neste ecrã é possível criar vários pisos e várias divisões por piso, editando todos os detalhes tais como nomes e descrições. O administrador pode definir o tamanho da planta de cada piso, definindo as dimensões da matriz. A matriz é composta por células de 50x50 pixeis e cada planta de um piso não pode exceder um tamanho máximo de 800x600 pixeis. Esta limitação

foi imposta pelo facto de estas plantas serem depois mostradas nos *smartphones* dos utilizadores, que têm capacidade de processamento limitada, pelo que imagens de grandes resoluções não são recomendadas.

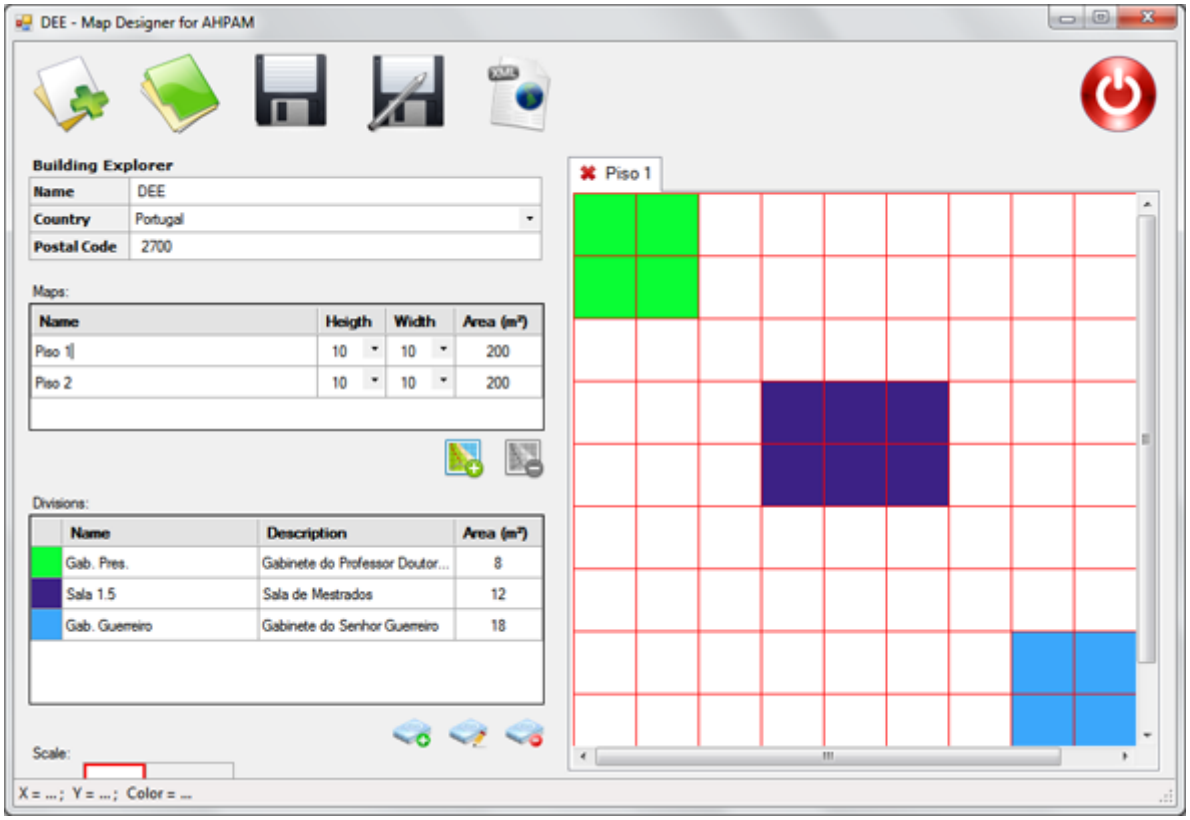


Figura 6.4: *Map Designer* para a criação de mapas de um edifício.

Após a criação do mapa é mostrado ao administrador a interface principal (Figura 6.5) do PDC onde pode aceder a todas as funcionalidades da aplicação. Essas funcionalidades incluem criação, edição e remoção de grupos, edição de mapas existentes, visualização da densidade de pessoas por divisão, visualização do ambiente de uma divisão (temperatura, luz, etc.) e criação e edição de administradores.

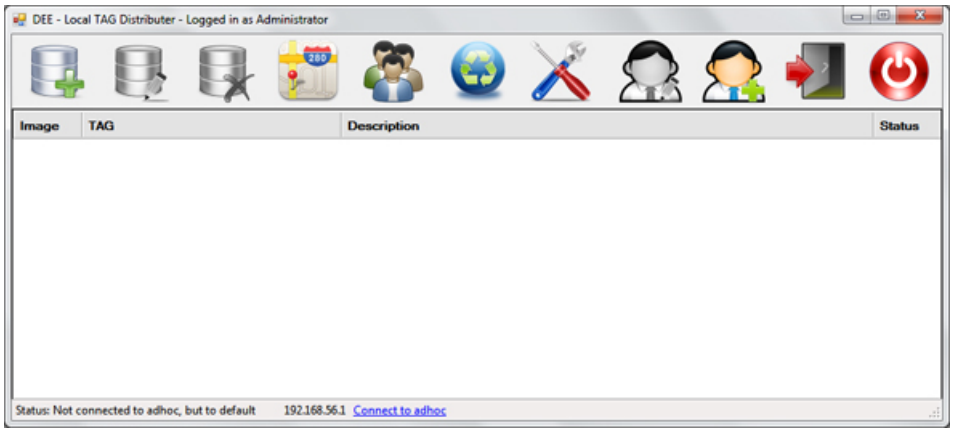


Figura 6.5: Janela principal do PDC sem grupos criados.

Para a criação de novos grupos, como mostrado na figura 6.6, o administrador tem de inserir uma Tag do grupo (i.e. o nome), uma descrição, o país do grupo e uma imagem. O país é importante

para o UTD poder gerar o GUID único para esse grupo. Quando o administrador carrega no botão *Create Group* (Criar Grupo), o PDC faz um pedido via webservice ao UTD a pedir a criação de um novo grupo e o respectivo GUID. O UTD gera um GUID e devolve-o ao PDC, garantindo assim a não repetição de grupos no mundo.

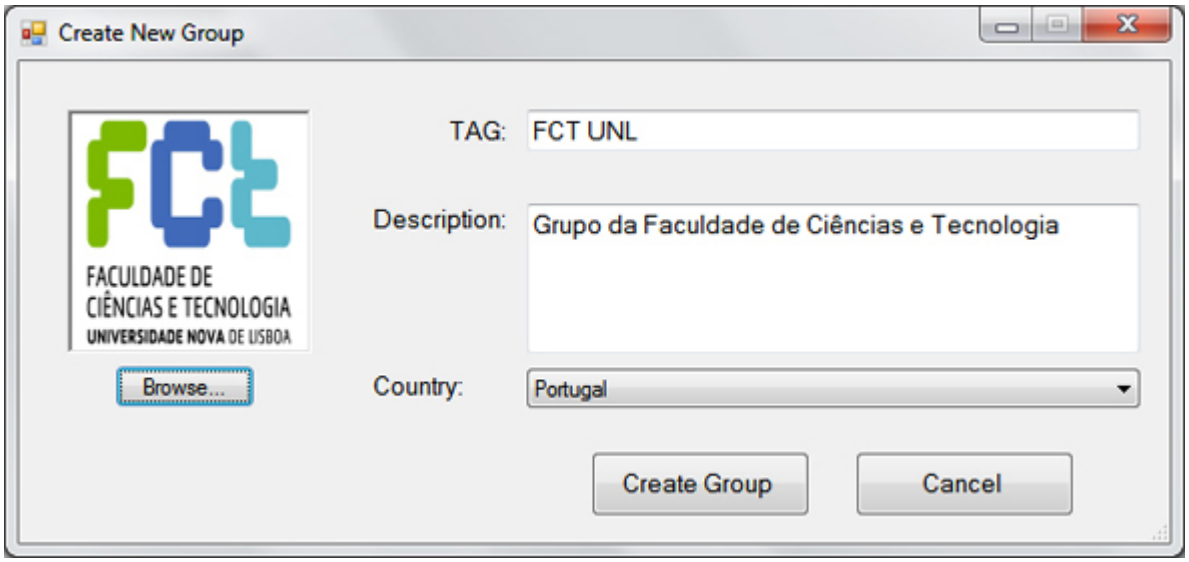


Figura 6.6: Janela de criação de grupos.

Aqui foi exemplificado a criação de um grupo apenas, mas quaisquer grupos subsequentes são criados da mesma forma e apresentados na interface principal numa forma de lista. Para o grupo criado na figura 6.6 a interface principal do PDC seria como a mostrada na figura 6.7. A coluna *Status* permite ao administrador definir se quer ou não difundir esse grupo pela rede ad-hoc. Para além desta acção o administrador pode sempre editar os detalhes de um grupo carregando na coluna correspondente.

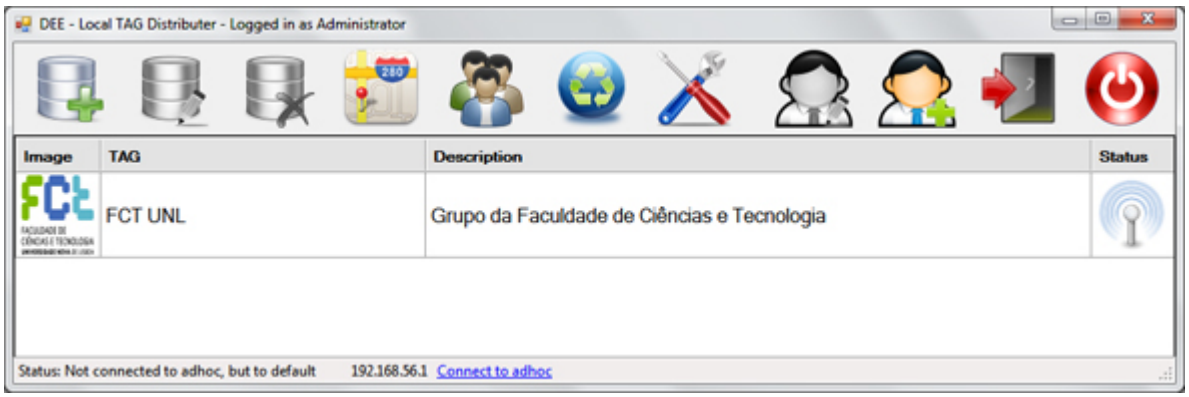


Figura 6.7: Janela principal do PDC com um grupo criado.

Depois de o PDC começar a recolher os dados da localização dos utilizadores, é possível mostrar uma tabela com a quantidade de utilizadores por divisão, como mostrado na figura 6.8, onde para além de uma tabela com os detalhes de cada divisão também se mostra um gráfico para uma visualização mais fácil. Esta é a tabela a que o BMS poderá aceder para verificar os registos históricos da densidade de pessoas em cada divisão, podendo depois actuar no edifício tendo em vista a optimização do uso da energia num determinado edifício.

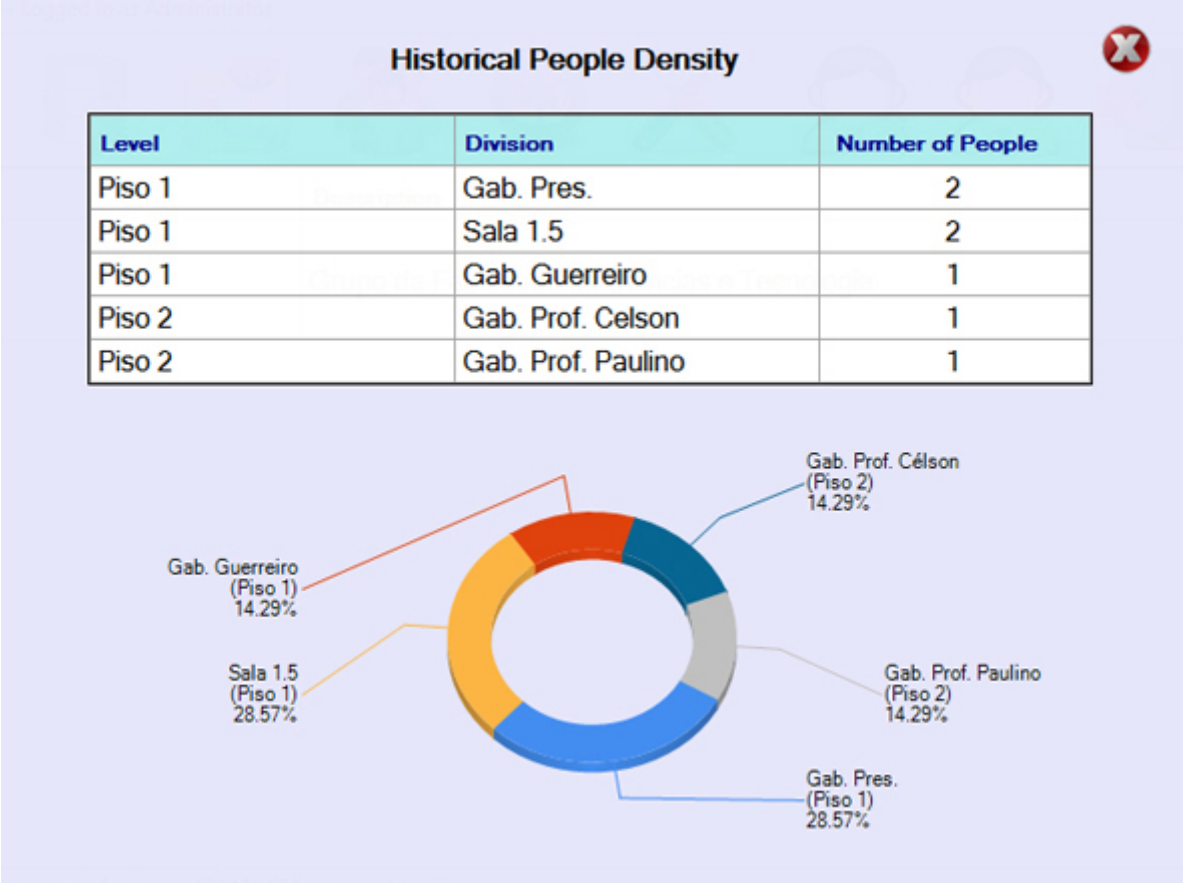


Figura 6.8: Contabilização do número de pessoas por divisão.

Capítulo 7

Conclusões e Perspectivas Futuras

Este capítulo apresenta uma síntese do trabalho desenvolvido, comentários aos contributos do projecto e aos possíveis trabalhos futuros.

7.1 Síntese Geral

Começou-se este trabalho com o objectivo de desenvolver um sistema capaz de contribuir eficazmente para a optimização do consumo de energia de um edifício, uma vez que é nestes ambientes que se podem verificar grandes desperdícios energéticos. Esta situação é tão ou mais verdade em edifícios públicos onde por não se saber quando e como as pessoas se vão movimentar no edifício é necessário garantir que alguns sistemas como a iluminação e climatização estejam constantemente ligados, para a eventualidade de uma pessoa entrar num determinado espaço e não encontra-lo completamente escuro e sem ar fresco. O entrave principal ao controlo das condições de um determinado espaço é mesmo esse, o não saber quando e onde é que uma pessoa vai entrar e como se vai movimentar pelo edifício pelo que é necessário efectuar algum tipo de monitorização dos comportamentos das pessoas.

A solução concentra-se na monitorização das movimentações das pessoas, i.e. saber exactamente onde estão dentro do edifício e adequar as condições dos espaços de acordo com essas mesmas pessoas. Como um dos objectos que mais acompanham as pessoas nos dias de hoje é o telemóvel, e com o crescimento que o mercado dos *smartphones* está a evidenciar, então é correcto identificar a posição das pessoas como sendo a posição dos seus telemóveis.

Com o Global Positioning System (GPS), descobrir geograficamente a localização do telemóvel (ou outro equipamento GPS) e consequentemente a localização da pessoa é relativamente simples de implementar, contudo de uma perspectiva de um edifício (ambiente fechado) o GPS não oferece a fiabilidade necessária para se atingir os objectivos planeados. É necessário saber algo mais sobre o edifício, nomeadamente a sua topologia, o número de pisos, o número de divisões e as áreas

restritas ao público de modo a que se possa indicar com exactidão a localização da pessoa dentro do edifício.

A utilização de tecnologias sem fios, como o IEEE 802.11 (Wi-Fi), permitiu a criação de *hotspots* espalhados por todas as divisões de um edifício e que comunicassem com os telemóveis das pessoas informando-os da sua localização sempre que a pessoa se aproximava com o seu telemóvel de um hotspot. Com este mecanismo todos os telemóveis de todas as pessoas que se encontrassem dentro do edifício sabiam a sua localização e poderiam, juntos, constituir uma rede ad-hoc sem fios, dinâmica, capaz de fazer um mapeamento da localização das pessoas. Um sistema autónomo, o PDC faz a monitorização da rede e contabiliza a quantidade de pessoas em cada divisão, mantendo um registo histórico dessa contabilização. É com base neste registo histórico que se consegue depois agir nos diferentes sistemas que compõem o edifício de modo a adequar as condições dos espaços de acordo com a quantidade de pessoas que se encontram nesse mesmo espaço, contribuído assim para uma optimização do consumo energético do edifício.

7.2 Contribuição da Investigação

A infra-estrutura de software desenvolvida permite a criação de redes adhoc pró-activas que permitem uma troca de informação que pode ocorrer em qualquer lugar. Esta troca de informação permite criar inúmeras aplicações que necessitem de comunicação persistente entre todos os agentes presentes num determinado espaço.

O software desenvolvido apresenta-se como suporte à componente de monitorização de um sistema de gestão de energia de um edifício, fornecendo informação acerca do comportamento das pessoas nesse mesmo edifício de modo a que o sistema de gestão possa actuar de uma forma mais eficaz nos equipamentos do edifício.

A utilização do standard IEEE 802.11 permite que a infra-estrutura de software desenvolvida funcione sobre uma base de redes ad hoc sem fios, onde os dispositivos da rede são os telemóveis das pessoas que circulam dentro do edifício. A infra-estrutura abrange todo o processo de criação da rede até à contabilização do número de pessoas na rede, a determinação das diferentes divisões em que as pessoas estão e o manutenção de um registo histórico dessa informação.

7.3 Trabalhos Futuros

As principais dificuldades deveram-se ao facto de os *smartphones*, apesar de já serem dispositivos com alguma capacidade de processamento, continuarem a ser um pouco limitados em termos de memória e como a aplicação desenvolvida consome alguns recursos importantes do sistema, nomeadamente os recursos Wi-Fi, de processamento e de memória, principalmente quando se está a processar os mapas dos edifícios é possível verificar alguma lentidão na aplicação móvel. Seria in-

teressante, no futuro, verificar como a aplicação se comportaria noutros sistemas operativos móveis mais recentes como o Android, iOS e Windows Phone 7 ao invés do Windows Mobile 6, uma vez que estes apresentam uma melhor gestão de memória, tornando os equipamentos mais eficientes.

Outro problema encontrado foi o facto de o utilizador poder estar ao alcance de dois ou mais *hotspots* ao mesmo tempo, pelo que é necessário de alguma forma escolher o *hotspot* correcto em relação à divisão em que o utilizador se encontra. Uma possível solução seria utilizar um método de detecção de potência do sinal recebido, verificando qual dos *hotspots* tinha o sinal mais forte e assim se assumir que esse seria o *hotspot* que corresponde à localização do utilizador. Esta possível solução poderia, no entanto, trazer outros problemas não sendo completamente fiável, pois pode acontecer que o utilizador esteja na divisão A e o sinal do *hotspot* da divisão B chegue com mais potência ao telemóvel do utilizador do que o sinal do *hotspot* da divisão A. Seria também interessante, estudar no futuro, um mecanismo que garantisse a não ocorrência de conflitos no broadcast de vários *hotspots* que estejam no raio de alcance uns dos outros.

Bibliografia

[Alzate,] Alzate, M. A. Bandwidth and available bandwidth concepts for wireless ad hoc networks. *Information Sciences*.

[Arduino, 2010] Arduino (2010). Arduino - Reference. Website. http://arduino.cc/en/Reference/HomePage.

[Binod, 2011] Binod, S. (2011). Concepts of energy management. Website. http://www.pdhonline.org/courses/e102/e102.htm.

[Burchfiel et al., 1975] Burchfiel, J., Tomlinson, R., and Beeler, M. (1975). Functions and structure of a packet radio station. *AFIPS 75 Proceedings of the May 19-22, 1975, national computer conference and exposition*.

[Daly, 2010] Daly, V. (2010). Smartphone market drives 600% growth in mobile web usage. *Bango*. http://news.bango.com/2010/02/16/600-percent-growth-in-mobile-web-usage/.

[Gauthier, 2009] Gauthier, M. (2009). Wireless networking in the developing world.

[Gilaberte, 2007] Gilaberte, R. L. (2007). Automatic configuration of ad-hoc networks : Establishing unique ip link local addresses. *Ad Hoc Networks*.

[InfoWester, 2011] InfoWester (2011). Tecnologia Wi-Fi (IEEE 802.11). Website. http://www.infowester.com/wifi.php.

[Jubin and Tornow, 1987] Jubin, J. and Tornow, J. D. (1987). The darpa packet radio network protocols. *Proceedings of the IEEE*.

[Kamm, 2007] Kamm, K. (2007). Achieving energy savings with building automation systems. Website. http://www.automatedbuildings.com/news/apr07/articles/esource/070322105430kamm.htm.

[Kunz, 2006] Kunz, T. (2006). AD-Hoc, Mobile, and Wireless Networks. In Ravi, S. S., editor, *5th International Conference, Adhoc-Now 2006, Ottawa, Canada, August 17-19, 2006 Proceedings*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG.

[Lewis, 2007] Lewis, J. (2007). Wireless internet usage continues to increase. Website. http://www.webpronews.com/wireless-internet-usage-continues-to-increase-2007-02.

[Marwaha et al., 2008] Marwaha, S., Indulska, J., and Portmann, M. (2008). Challenges and recent advances in qos provisioning signaling routing and mac protocols for manets. *IEEE*.

[Mauve et al., 2001] Mauve, M., Widmer, J., and Hartenstein, H. (2001). A survey on position-based routing in mobile ad hoc networks. *IEEE Network*.

[Moraes et al., 2011] Moraes, A., Xaud, A., and Xaud, M. (2011). Redes Ad Hoc - Protocolos. Website. http://www.gta.ufrj.br/grad/09_1/versao-final/adhoc/redesadhoc.html.

[MSDN, 2010] MSDN (2010). MSDN: Microsoft Development, MSDN Subscriptions, Resources, and More. Website. http://msdn.microsoft.com/pt-pt/.

[Oxer and Blemings, 2010] Oxer, J. and Blemings, H. (2010). *Practical Arduino: Cool Projects for Open Source Hardware*. APress.

[Ozan et al., 2006] Ozan, K., Tonguz, and Ferrari, G. (2006). *Ad Hoc Wireless Networks: A Communication-Theoretic Perspective*. John Wiley & Sons.

[Rahman, 2010] Rahman, M. (2010). Building management system : a case study at Albukhary foundation head office. Master's thesis, Faculty of Civil Engineering - Universiti Teknologi Malaysia.

[Roth and McKenney, 2007] Roth, K. W. and McKenney, K. (2007). *Energy Consumption by Consumer Electronics in U.S. Residences*. Tiax.

[Royer and Toh, 1999] Royer, E. and Toh, C. (1999). A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*.

[Saching, 2011] Saching (2011). MANET (Mobile Adhoc NETwork). Website. http://www.saching.com/Article/MANET—Mobile-Adhoc-NETwork–/334.

[Schacham and Westcott, 1987] Schacham, N. and Westcott, J. (1987). Future directions in packet radio architectures and protocols. *Proceedings of the IEEE*.

[Singh et al.,] Singh, K., Yadav, R. S., and Ranvijay. A review paper on ad hoc network security. *International Journal of Computer Science and Security*.

[Stum et al., 1997] Stum, K., Mosier, R., and Haasl, T. (1997). *Energy Managment Systems - A Pratical Guide*. PECI.

[Tanenbaum and Wetherall, 2010] Tanenbaum, A. S. and Wetherall, D. J. (2010). *Computer Networks*. Pearson Education (US), 5 edition.

[Toh, 2002] Toh, C. K. (2002). *Ad Hoc Mobile Wireless Networks: Protocols & Systems*. Prentice Hall Publishers.

[URVIL, 2011] URVIL (2011). Building Managment Systems. http://urvil.wordpress.com/bms-building-management-system/.

[Vilarigues, 2007] Vilarigues, S. (2007). Entrevista a António Sá da Costa sobre a energia em Portugal. *Jornal Quercus Ambiente n.º23*.

[W3CC, 2011] W3CC (2011). Extensible Markup Language (XML). Website. http://www.w3.org/XML/.

Apêndice A

Classe *Sender*

Listagem A.1: Código C# da classe Sender.

```
ïż£using System;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using Packets;
using System.IO;
using Helper;
using System.Threading;
using Helpers;

namespace Communication
{
    public class Sender
    {
        private List<Socket> sockets;
        private List<string> tcpClients;
        private static byte[] arrayToSend;
        public Action<List<string>> refreshTCP;

        /// <summary>
        /// Cria uma nova instancia de uma classe Sender que envia pacotes TCP.
        /// </summary>
        public Sender() {
            refreshTCP = new Action<List<string>>(RefreshTCPClients);
            sockets = new List<Socket>();
            tcpClients = new List<string>();
        }

        /// <summary>
        /// Liga-se a um listener local ou remoto.
        /// </summary>
        /// <param name="ipadress">IP do listener.</param>
        /// <param name="port">Porto do listener.</param>
        /// <returns>True se conseguir se ligar, false caso contrÃąrio.</returns>
        private Socket Connect(string ipadress, int port) {
```

```
            try
            {
                IPAddress ip = IPAddress.Parse(ipadress);
                IPEndPoint remoteEP = new IPEndPoint(ip, port);

                Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                    ProtocolType.Tcp);
                //sock.BeginConnect(remoteEP, new AsyncCallback(ConnectCallback), sock);
                sock.Connect(remoteEP);
                Log.LogToFile("Connected to " + remoteEP);
                return sock;
            }
            catch(Exception){
                Log.LogToFile("Fatal error on connecting.");
                return null;
            }
        }

        /// <summary>
        /// Destroi o socket TCP, sendo possÃ■vel especificar se o socket poderÃą ser
            reutilizado ou nÃčo.
        /// </summary>
        /// <param name="reusable">True se o socket for reutilizÃąvel, false caso
            contrÃąrio.</param>
        public void Disconnect() {
            try
            {
                foreach (Socket socket in sockets)
                {
                    socket.Shutdown(SocketShutdown.Both);
                    socket.Close();
                }
                sockets.Clear();
            }
            catch { }
        }

        private void Disconnect(Socket socket) {
            try
            {
                socket.Shutdown(SocketShutdown.Both);
                socket.Close();
                sockets.Remove(socket);
            }
            catch { }
        }

        /// <summary>
        /// Sends an object of Type Pack.
        /// </summary>
        /// <param name="Data">The object of type Pack to send.</param>
        /// <returns>True se conseguir enviar, false caso contrÃąrio.</returns>
        public bool Send(string ipadress, int port, object Data)
        {
            bool found = false;
            foreach (string remoteEP in tcpClients) {
                string[] eps = remoteEP.Split(':');
                if (ipadress.Equals(eps[0]))
                {
```

```
95                      found = true;
                        port = Convert.ToInt32(eps[1]);
                    }
                }
                if (!found)
100                 return false;
                try
                {
                    byte[] rawData = Serializer.Serialize(Data);
                    byte[] sizeBytes = BitConverter.GetBytes(rawData.Length);
105                 byte[] data = new byte[9 + rawData.Length];
                    sizeBytes.CopyTo(data, 0);
                    data[8] = 0;
                    rawData.CopyTo(data, 9);
                    Socket socket = Connect(ipadress, port);
110                 arrayToSend = data;
                    socket.Send(data);
                    Log.LogToFile("Sent packet to " + ipadress + ":" + port);
                    return true;
                }
115             catch (SocketException)
                {
                    Log.LogToFile("Error sending packet to " + ipadress + ":" + port);
                    return false;
                }
120             catch (Exception)
                {
                    Log.LogToFile("Fatal Error on send.");
                    return false;
                }
125         }

            /// <summary>
            /// Envia um ficheiro.
            /// </summary>
130         /// <param name="rawData"></param>
            /// <param name="Filename"></param>
            /// <returns></returns>
            public bool SendFile(string ipadress, int port, string Filename, string filePath,
                int fileType) {
                bool found = false;
135             foreach (string remoteEP in tcpClients)
                {
                    string[] eps = remoteEP.Split(':');
                    if (ipadress.Equals(eps[0]))
                    {
140                     found = true;
                        port = Convert.ToInt32(eps[1]);
                    }
                }
                if (!found)
145                 return false;
                Socket socket = Connect(ipadress, port);
                if (socket == null)
                    return false;
                try
150             {
                    string path = filePath + Filename;
                    FileHandler fh = new FileHandler();
```

```
            FileInfo fi = new FileInfo(path);
            int bytesSent = 0;
155         byte[] data;
            byte[] sizeBytes;
            byte[] filenameBytes;
            byte[] filenameSize;
            byte[] rawData = fh.GetFile(path);
160         string file = fi.Directory.Name + "\\" + Filename;
            data = new byte[13 + Encoding.UTF8.GetByteCount(file) + fi.Length];
            //Get file size in bytes
            sizeBytes = BitConverter.GetBytes(fi.Length);
            sizeBytes.CopyTo(data, 0);
165         //Get file type
            data[8] = (byte)fileType;
            //Get filename size
            filenameSize = BitConverter.GetBytes(Encoding.UTF8.GetByteCount(file));
            filenameSize.CopyTo(data, 9);
170         //Get filename in bytes
            filenameBytes = Encoding.UTF8.GetBytes(file);
            filenameBytes.CopyTo(data, 13);
            //Get file data
            rawData.CopyTo(data, 13 + Encoding.UTF8.GetByteCount(file));
175         do {
                bytesSent += socket.Send(data, bytesSent, data.Length-bytesSent,
                    SocketFlags.None);
            }
            while(bytesSent < rawData.Length);
            Log.LogToFile("Sent file to " + ipadress + ":" + port);
180         return true;
        }
        catch (SocketException) {
            Disconnect(socket);
            Log.LogToFile("Error sending file to " + ipadress + ":" + port);
185         return false;
        }
        catch (Exception)
        {
            Log.LogToFile("Fatal Error on send file.");
190         return false;
        }
    }

    public static void ConnectCallback(IAsyncResult ar) {
195     try
        {
            Socket s = (Socket)ar.AsyncState;
            s.Send(arrayToSend);
            s.EndConnect(ar);
200     }
        catch (Exception ex) {

        }
    }
205
    public void RefreshTCPClients(List<string> clients) {
        tcpClients = clients;
    }

210 private void removeIPFromLists(string ip) {
```

```
            int pos = 0;
            foreach (string client in new List<string>(tcpClients))
            {
                string[] ips = client.Split(':');
                if (ips[0].Equals(ip))
                {
                    tcpClients.RemoveAt(pos);
                    break;
                }
                pos++;
            }
        }
        // END CLASS
    }
}
```

Apêndice B

Classe *Receiver*

Listagem B.1: Código C# da classe Receiver.

```
ï»¿using System;
using System.Net.Sockets;
using System.Threading;
using System.Net;
using System.Linq;
using Packets;
using System.Text;
using System.IO;
using System.Windows.Forms;
using System.Reflection;
using Helper;
using Helpers;
using Tables;

namespace Communication
{
    class Receiver
    {
        private int port;

        private static Pack pk = new Pack();

        private Socket listener;
        private Socket client;
        private Thread listenThread;
        private Thread clientThread;
        private string baseDir = Path.GetDirectoryName(Assembly.GetExecutingAssembly().
            GetName().CodeBase);
        private Sender ss;
        private Broadcast broadcast;

        private Action<Pack, byte[], string> action;

        /// <summary>
        /// Cria uma nova instancia da classe Receiver para receber pacotes TCP.
        /// </summary>
        /// <param name="_action">Action para actualizar a GUI.</param>
```

```
        public Receiver(Action<Pack, byte[], string> _action, Sender _sender, Broadcast
            _broadcast)
        {
            broadcast = _broadcast;
            action = _action;
            ss = _sender;
        }

        /// <summary>
        /// Incia a recepÃğÃčo de pacotes TCP.
        /// </summary>
        /// <param name="port">Porta que vai ouvir a recepÃğÃčo de novos pacotes.</param>
        public int StartReceiving()
        {
            IPEndPoint iep = new IPEndPoint(IPAddress.Any, 0);
            listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                ProtocolType.IP);
            listener.Bind(iep);
            listener.Listen(100);
            listenThread = new Thread(new ThreadStart(ListenForClients));
            listenThread.Start();
            //GET PORT
            string[] ports = listener.LocalEndPoint.ToString().Split(':');
            return Convert.ToInt32(ports[1]);
        }

        /// <summary>
        /// Desliga a recepÃğÃčo de pacotes IP e respectivas threads e sockets.
        /// </summary>
        public void Disconnect()
        {
            listenThread.Abort();
            listener.Close();
            if (clientThread != null)
            {
                clientThread.Abort();
                client.Close();
            }
        }

        /// <summary>
        /// FunÃğÃčo que vai ser corrida por uma thread para estar constantemente a ouvir
            novos pedidos de ligaÃğÃčo de
        /// clientes.
        /// </summary>
        private void ListenForClients()
        {
            while (true)
            {
                //bloqueia atÃľ que um cliente se ligue
                client = listener.Accept();

                //cria uma thread para tratar da comunicaÃğÃčo com o cliente ligado
                clientThread = new Thread(new ThreadStart(ReceiveMessage));
                clientThread.IsBackground = true;
                clientThread.Start();
            }
        }
```

```
        /// <summary>
        /// FunÃğÃčo que vai ser corrida por uma thread para receber pacotes TCP de
            clientes jÃą ligados.
        /// </summary>
        private void ReceiveMessage()
        {
            Socket tcpClient = (Socket)client;
            AHPAM ahpam = new AHPAM();
            byte[] message = new byte[9];
            byte packetType = 0;
            int bytesRead;
            string fileName = "nada";
            bool verify;
            int size;
            while (true)
            {
                bytesRead = 0;
                verify = true;
                size = 0;
                try
                {
                    do
                    {
                        //bloqueia atÃ' que um cliente envie uma mensagem
                        if (verify == true)
                        {
                            //Read Packet Size
                            message = new byte[8];
                            bytesRead += tcpClient.Receive(message, 0, 8, SocketFlags.None)
                                ;
                            byte[] sizeByte = new byte[8];
                            message.CopyTo(sizeByte, 0);
                            size = BitConverter.ToInt32(sizeByte, 0);
                            //Read Packet Type
                            message = new byte[1];
                            bytesRead += tcpClient.Receive(message, 0, 1, SocketFlags.None)
                                ;
                            packetType = message[0];
                            if (packetType == Pack.FILE || packetType == Pack.CUSTOM_FILE
                                || packetType == Pack.MAP_XML || packetType == Pack.
                                MAP_IMAGE)
                            {
                                //Get filename size
                                message = new byte[4];
                                bytesRead += tcpClient.Receive(message, 0, 4, SocketFlags.
                                    None);
                                int filenameSize = BitConverter.ToInt32(message, 0);
                                //Read File Name
                                message = new byte[filenameSize];
                                bytesRead += tcpClient.Receive(message, 0, filenameSize,
                                    SocketFlags.None);
                                fileName = Encoding.UTF8.GetString(message, 0, filenameSize
                                    );
                                fileName = fileName.Trim('\0');
                            }
                            message = new byte[size];
                            verify = false;
                            bytesRead = 0;
                        }
```

```
                    else
                    {
                        int missing = size - bytesRead;
                        int lastBytesRead = bytesRead;
                        bytesRead += tcpClient.Receive(message, lastBytesRead, missing,
                            SocketFlags.None);
                    }
                } while (bytesRead < size);
            }
            catch (ThreadAbortException tae) {
                break;
            }
            catch (Exception ex)
            {
                //ocorreu um erro no socket
                break;
            }
            if (bytesRead == 0)
            {
                //o cliente desligou-se
                break;
            }
            //a mensagem foi recebida com sucesso
            switch(packetType) {
                case Pack.FILE:
                    try
                    {
                        FileHandler fh = new FileHandler();
                        string dir = Path.GetDirectoryName(fileName);
                        if (!Directory.Exists(baseDir + "\\" + dir))
                            Directory.CreateDirectory(baseDir + "\\" + dir);
                        if (File.Exists(baseDir + "\\" + fileName))
                            File.Delete(baseDir + "\\" + fileName);
                        fh.WriteFile(baseDir + "\\" + fileName, message);
                        string[] ip = tcpClient.RemoteEndPoint.ToString().Split(':');
                        Ack ack = new Ack();
                        ack.ip = ahpam.GetLocalIP().ToString();
                        ack.filename = Path.GetFileName(fileName);
                        ack.packetType = Pack.FILE;
                        ack.type = ack.GetType().Name;
                        ss.Send(ip[0], port,ack);
                        Log.LogToFile("Received file.");
                    }
                    catch{}
                    break;
                case Pack.CUSTOM_FILE:
                    string[] ip2 = tcpClient.RemoteEndPoint.ToString().Split(':');
                    Ack ack2 = new Ack();
                    ack2.ip = ahpam.GetLocalIP().ToString();
                    ack2.filename = Path.GetFileName(fileName);
                    ack2.packetType = Pack.CUSTOM_FILE;
                    ack2.type = ack2.GetType().Name;
                    ss.Send(ip2[0], port,ack2);
                    Pack pack = new Pack();
                    pack.type = "Custom_File";
                    action.Invoke(pack, message, fileName);
                    Log.LogToFile("Received custom file.");
                    break;
                case Pack.MAP_XML:
```

```
                    string[] ip3 = tcpClient.RemoteEndPoint.ToString().Split(':');
                    Ack ack3 = new Ack();
                    ack3.ip = ahpam.GetLocalIP().ToString();
                    ack3.filename = Path.GetFileName(fileName);
                    ack3.packetType = Pack.MAP_XML;
                    ack3.type = ack3.GetType().Name;
                    ss.Send(ip3[0], port,ack3);
                    Pack pack2 = new Pack();
                    pack2.type = "Map_Xml";
                    action.Invoke(pack2, message, fileName);
                    Log.LogToFile("Received map xml file.");
                    break;
                case Pack.MAP_IMAGE:
                    string[] ip4 = tcpClient.RemoteEndPoint.ToString().Split(':');
                    Ack ack4 = new Ack();
                    ack4.ip = ahpam.GetLocalIP().ToString();
                    ack4.filename = Path.GetFileName(fileName);
                    ack4.packetType = Pack.MAP_IMAGE;
                    ack4.type = ack4.GetType().Name;
                    ss.Send(ip4[0], port,ack4);
                    Pack pack3 = new Pack();
                    pack3.type = "Map_Image";
                    action.Invoke(pack3, message, fileName);
                    Log.LogToFile("Received map image file.");
                    break;
                default:
                    Pack pk = new Pack();
                    pk = (Pack)Serializer.Deserialize(message, pk.GetType());
                    if (pk != null)
                    {
                        Log.LogToFile("Received packet " + pk.type + ".");
                        action.Invoke(pk, message, null);
                    }
                    else { //ITS IPTABLE!!
                        IPTable ipT = new IPTable();
                        ipT = (IPTable)Serializer.Deserialize(message, ipT.GetType());
                        broadcast.refIPTableFromReceiv.Invoke(ipT);
                    }
                    break;
            }
        }
        tcpClient.Close();
        clientThread.Abort();
    }

}
}
```

Apêndice C

Classe *Serializer*

Listagem C.1: Código C# da classe Serializer.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using System.Windows;
using System.Windows.Forms;
using OpenNETCF.Reflection;
using Packets;

namespace Communication
{
    public class Serializer
    {
        /// <summary>
        /// Converte um array de bytes num objecto.
        /// </summary>
        /// <param name="rawdata">O array de bytes a ser convertido.</param>
        /// <param name="type">O tipo do objecto.</param>
        /// <returns>O objecto deserializado.</returns>
        public static object Deserialize(byte[] rawdata, Type type)
        {
            try
            {
                string data = Encoding.UTF8.GetString(rawdata, 0, rawdata.Length);
                XmlSerializer deserializer = new XmlSerializer(type);
                TextReader textReader = new StringReader(data);
                object obj;
                obj = (object)deserializer.Deserialize(textReader);
                textReader.Close();
                return obj;
            }
            catch (Exception ex) {
                return null;
            }
        }
```

```
        /// <summary>
        /// Converte um objecto num array de bytes.
        /// </summary>
        /// <param name="obj">O objecto a ser serializado.</param>
        /// <returns>O array de bytes correspondente ao objecto.</returns>
        static public byte[] Serialize(object obj)
        {
            try
            {
                Type type = obj.GetType();
                XmlSerializer serializer = new XmlSerializer(type);
                TextWriter textWriter = new StringWriter();
                serializer.Serialize(textWriter, obj);
                string data = textWriter.ToString();
                textWriter.Close();
                return Encoding.UTF8.GetBytes(data);
            }
            catch(Exception) {
                return null;
            }
        }

    }
}
```

Apêndice D

Classe *Broadcast*

Listagem D.1: Código C# da classe Broadcast.

```
ïż£using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using Packets;
using System.Threading;
using System.Windows.Forms;
using Tables;
using System.IO;
using System.Reflection;
using UserProfile;
using UserProfile.Groups;
using Helpers;

namespace Communication
{
    public class Broadcast
    {
        private const int BROADCAST_PORT = 9050;

        private byte[] data;
        private Socket socket;
        private IPEndPoint iep;
        private Thread bcThread;
        private Thread ipTableThread;
        private static IPTable ipTable;
        public Action<IPTable> actionIPTable = new Action<IPTable>(RefreshIPTable);
        private List<string> gList = new List<string>();
        private BroadcastRcv broadcastReceiver;
        public Action<IPTable> refIPTableFromReceiv;
        private Action<IPTable> mainFormAction;

        private static string profileXML = Path.GetDirectoryName(Assembly.
            GetExecutingAssembly().GetName().CodeBase) + "\\userProfile.xml";

        /// <summary>
```

```
        /// Cria uma nova instancia da classe Broadcast que periodicamente envia a sua
            tabela de IPs da rede.
        /// </summary>
        public Broadcast(BroadcastRcv _broadcastReceiver, Action<IPTable> _action)
        {
            mainFormAction = _action;
            refIPTableFromReceiv = new Action<IPTable>(RefreshIPTableFromReceiver);
            broadcastReceiver = _broadcastReceiver;
            socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.
                Udp);
            iep = new IPEndPoint(IPAddress.Broadcast, BROADCAST_PORT);
            ipTable = new IPTable();
        }

        /// <summary>
        /// Inicia o broadcast da tabela de IPs da rede.
        /// </summary>
        public void Start() {
            bcThread = new Thread(new ThreadStart(Send));
            bcThread.Start();

            ipTableThread = new Thread(new ThreadStart(CheckTable));
            ipTableThread.IsBackground = true;
            ipTableThread.Start();
        }

        /// <summary>
        /// FunÃğÃčo executada pela thread e que envia periodicamente a tabela de IPs.
        /// </summary>
        private void Send() {
            AHPAM ahpam = new AHPAM();
            while(true) {
                try
                {
                    string localIP = ahpam.GetLocalIP().ToString();
                    if (!localIP.Equals("0.0.0.0") && !localIP.Equals("127.0.0.1"))
                    {
                        CreatePacket();
                        broadcastReceiver.refreshIpTable.Invoke(ipTable);
                        string ip = ahpam.GetLocalIP().ToString() + "\n";
                        byte[] arr = Encoding.UTF8.GetBytes(ip);
                        socket.SendTo(arr, iep);
                    }
                    Thread.Sleep(5000);
                }
                catch {}
            }
        }

        /// <summary>
        /// Para a thread que envia o broadcast.
        /// </summary>
        public void Stop() {
            bcThread.Abort();
            socket.Close();
            if (ipTableThread != null)
                ipTableThread.Abort();
        }
```

```
95
        /// <summary>
        /// Cria um pacote broadcast com a tabela de IPs.
        /// </summary>
        private void CreatePacket() {
100         AHPAM ahpam = new AHPAM();
            UP up = new UP();
            up = (UP)SerializerXML.Deserialize(profileXML, up.GetType());
            gList.Clear();
            foreach (GUI gui in new List<GUI>(up.guis))
105             gList.Add(gui.guid);
            IPTableEntry ipTableEntry = new IPTableEntry(ahpam.GetLocalIP(), DateTime.Now,
                gList);
            int pos = ipTable.ExistsEntry(ipTableEntry);
            if (pos != -1)
            {
110             ipTable.RemoveAt(pos);
                ipTable.Add(ipTableEntry);
            }
            else
                ipTable.Add(ipTableEntry);
115         data = Serializer.Serialize(ipTable);
        }

        /// <summary>
        /// Actualiza a tabela de IPs com uma tabela recebida num pacote broadcast.
120     /// </summary>
        /// <param name="ipT">Tabela de ips recebida.</param>
        private void RefreshIPTableFromReceiver(IPTable ipT)
        {
            if (ipT == null)
125             return;
            foreach (IPTableEntry ipEntry in ipT)
            {
                int pos = ipTable.ExistsEntry(ipEntry);
                if (pos != -1)
130             {
                    if (ipTable.CompareLastUpdate(ipEntry, pos))
                    {
                        ipTable.RemoveAt(pos);
                        ipTable.Add(ipEntry);
135                 }
                }
                else
                    ipTable.Add(ipEntry);
            }
140         mainFormAction.Invoke(ipTable);
        }

        /// <summary>
        /// Actualiza a tabela de IPs. Esta funÃğÃčo Ã© invocada por outras threads para
            alertar o Broacast que alterou a
145     /// tabela de IPs.
        /// </summary>
        /// <param name="ipT">Tabela de IPs actualizada.</param>
        private static void RefreshIPTable(IPTable ipT) {
            ipTable = ipT;
150     }
```

```
        /// <summary>
        /// Verifica a tabela de IPs periodicamente de modo a verificar IPs da rede que
            morreram, i.e. que jÃą nÃčo respondem
        /// Ãă algum tempo.
155     /// </summary>
        private void CheckTable()
        {
            while (true)
            {
160             foreach (IPTableEntry ipTableEntry in new List<IPTableEntry>(ipTable))
                {
                    DateTime now = DateTime.ParseExact(DateTime.Now.ToString("MM-dd-yyyy HH
                        :mm:ss"), "MM-dd-yyyy HH:mm:ss", null);
                    DateTime entryTime = DateTime.ParseExact(ipTableEntry.lastUpdate, "MM-
                        dd-yyyy HH:mm:ss", null);
                    TimeSpan difference = now.Subtract(entryTime);
165                 // Se nÃčo ouvir nenhum HELLO Ãă mais de 30 segundos, remove IP
                    if (difference.Hours > 0 || difference.Minutes > 5)
                    {
                        ipTable.Remove(ipTableEntry);
                        mainFormAction.Invoke(ipTable);
170                 }
                }
                Thread.Sleep(5000);
            }
        }
175
        //END CLASS
    }
}
```

Apêndice E

Classe *AHPAM*

Listagem E.1: Código C# da classe AHPAM.

```
ï»¿using System;
using System.Linq;
using OpenNETCF.Net.NetworkInformation;
using OpenNETCF.WindowsMobile;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Collections.Generic;
using Helpers;

namespace Communication
{
    public class AHPAM
    {
        ///
        ///<summary>
        ///  Retorna o estado do WiFi do PPC
        ///</summary>
        ///<returns>
        ///  1 - WiFi estÃ¡ ligado, 0 - WiFi estÃ¡ desligado, -1 - Erro
        ///</returns>
        public int GetWifiState()
        {
            try
            {
                var wifi = from radio in Radios.GetRadios() where radio.RadioType ==
                    RadioType.WiFi select radio;
                foreach (IRadio radio in wifi)
                {
                    if (radio.RadioState == RadioState.On)
                        return 1;
                    else if(radio.RadioState == RadioState.Off)
                        return 0;
                }
                return -1;
            }
            catch (Exception)
            {
```

```
            return -1;
        }
    }

    ///
    ///<summary>
    ///  Altera o estado do WiFi do PPC
    ///</summary>
    ///<param name="on_off">1 - Para ligar o WiFi, 0 - Para desligar o WiFi.</param>
    ///<returns>
    ///  True se o WiFi foi ligado com sucesso, False caso contrÃąrio.
    ///</returns>
    public bool ChangeWifiState(int on_off)
    {
        try
        {
            var wifi = from radio in Radios.GetRadios() where radio.RadioType ==
                RadioType.WiFi select radio;
            foreach (IRadio radio in wifi)
            {
                if (on_off == 1)
                {
                    radio.RadioState = RadioState.On;
                }
                else if (on_off == 0)
                {
                    radio.RadioState = RadioState.Off;
                }
            }
            return true;
        }
        catch (Exception)
        {
            return false;
        }
    }

    ///
    ///<summary>
    ///  Retorna uma Network Interface (placa de rede) Wireless e compatÃ■vel com Zero
        Config.
    ///</summary>
    ///<returns>
    ///  A interface de rede wireless.
    ///</returns>
    public WirelessZeroConfigNetworkInterface GetWirelessNI() {
        try
        {
            foreach (object ni in NetworkInterface.GetAllNetworkInterfaces())
            {
                if (ni is WirelessZeroConfigNetworkInterface)
                {
                    return (WirelessZeroConfigNetworkInterface)ni;
                }
            }
            return null;
        }
        catch (Exception ex) {
            MessageBox.Show(ex.Message);
```

```
95                  return null;
                }
            }

            ///
100         ///<summary>
            ///  Retorna todas os acess points de uma determinada WZC Network Interface.
            ///</summary>
            ///<param name="wzc">A instancia da interface de rede (Network Interface)
                compatÃ■vel com WZC.</param>
            ///<returns>
105         ///  Uma lista de Access Points associada Ãă interface de rede especificada.
            ///</returns>
            public AccessPointCollection GetAllAps(WirelessZeroConfigNetworkInterface wzc)
            {
                wzc.Refresh();
110             return wzc.NearbyAccessPoints;
            }

            ///
            ///<summary>
115         ///      Cria se nÃčo existir a rede com o nome SSID, ou liga-se a ela caso jÃą
                exista.
            ///</summary>
            ///<param name="ssid">O nome da rede adhoc.</param>
            ///<param name="password">A password de rede caso exista. Colocar null caso nÃčo
                exista.</param>
            ///<param name="authentication">O mÃ'todo e autÃłnticaÃğÃčo (Aberto, Partilhado,
                WPA, WPA-PSK, WPA2, WPA2-PSK).</param>
120         ///<param name="wepStatus">A encriptaÃğÃčo de dados (Desactivado, WEP, AES, TKIP)
                .</param>
            ///<param name="wzc">A instancia da interface de rede (Network Interface)
                compatÃ■vel com WZC.</param>
            ///<returns>
            ///  True se a ligaÃğÃčo for bem sucedida, False caso contrÃąrio.
            /// </returns>
125         public bool ConnectAdhoc(string ssid, string password, AuthenticationMode
                authentication, WEPStatus wepStatus, WirelessZeroConfigNetworkInterface wzc)
            {
                try
                {
                    wzc.Refresh();
130                 AccessPointCollection apc = wzc.NearbyAccessPoints;
                    AccessPoint ap = apc.FindBySSID(ssid);
                    if (ap == null)
                    {
                        EAPParameters eapParam = new EAPParameters();
135                     wzc.AddPreferredNetwork(ssid, false, password, 1, authentication,
                            wepStatus, eapParam);
                        if (wzc.ConnectToPreferredNetwork(ssid))
                        {
                            Log.LogToFile("Connected to adhoc.");
                            return true;
140                     }
                        else
                        {
                            Log.LogToFile("Failed to connect to adhoc.");
                            return false;
145                     }
```

```
            }
            else
            {
                AccessPoint ap_preff = wzc.PreferredAccessPoints.FindBySSID(ap.Name);
                if (ap_preff == null)
                {
                    wzc.AddPreferredNetwork(ap);
                }
                if (wzc.ConnectToPreferredNetwork(ap.Name))
                {
                    Log.LogToFile("Connected to adhoc.");
                    return true;
                }
                else
                {
                    Log.LogToFile("Failed to connect to adhoc.");
                    return false;
                }
            }
        }
        catch (Exception)
        {
            Log.LogToFile("Failed to connect to adhoc.");
            return false;
        }
    }

    ///
    ///<summary>
    ///  Desliga a ligaÃğÃčo Ãă rede especificada em ssid associada Ãă interface de rede wzc.
    ///</summary>
    ///<param name="ssid">O nome da rede.</param>
    ///<param name="wzc">A instancia da interface de rede (Network Interface) compatÃ■vel com WZC.</param>
    ///<returns>
    ///  True se a desconexÃğÃčo for bem sucedida, False caso contrÃąrio.
    ///</returns>
    public bool DisconnectAdhoc(string ssid, WirelessZeroConfigNetworkInterface wzc)
    {
        try
        {
            return wzc.RemovePreferredNetwork(ssid);
        }
        catch (Exception) {
            return false;
        }
    }

    ///
    ///<summary>
    ///  Retorna o IP local.
    ///</summary>
    ///<returns>
    ///  O IP local da mÃąquina.
    ///</returns>
    public IPAddress GetLocalIP()
    {
        INetworkInterface[] nis = NetworkInterface.GetAllNetworkInterfaces();
```

```
                return nis[0].CurrentIpAddress;
            }
205         }
    }
```

Apêndice F

Classe *BroadcastArduino*

Listagem F.1: Código C# da classe BroadcastArduino.

```
ï»¿using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using Packets;
using System.Net.Sockets;
using System.Net;
using System.Threading;

namespace Communication
{
    public class BroadcastArduino
    {
        private const int BROADCAST_PORT = 2500;
        private Action<ArduinoPack> action;
        private Socket socket;
        private Thread thread;
        private IPEndPoint iep;

        public BroadcastArduino(Action<ArduinoPack> _action) {
            action = _action;
            socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.
                Udp);
        }

        public void Start() {
            iep = new IPEndPoint(IPAddress.Any, 2500);
            socket.Bind(iep);
            //EndPoint ep = (EndPoint)iep;
            thread = new Thread(new ThreadStart(ReceiveThreadBody));
            thread.Start();
        }

        public void Stop() {
            if (thread != null)
                thread.Abort();
            if (socket != null)
                socket.Close();
```

```
        }

        private void ReceiveThreadBody() {
            while (true)
            {
                byte[] data = new byte[1024];
                int recv = socket.Receive(data);
                ArduinoPack ap = new ArduinoPack();
                ap = ProccesPacket(data);
                if(ap != null)
                    action.Invoke(ap);
                //socket.Bind(iep);
            }
        }

        private ArduinoPack ProccesPacket(byte[] data)
        {
            string raw = Encoding.UTF8.GetString(data, 0, data.Length);
            raw = raw.Trim('\0');
            string[] pieces = raw.Split('\n');
            if (pieces.Length == 1)
                return null;
            ArduinoPack ap = new ArduinoPack();
            ap.MapName = pieces[0];
            ap.LevelName = pieces[1];
            ap.DivisionName = pieces[2];
            ap.X = Convert.ToInt32(pieces[3]);
            ap.Y = Convert.ToInt32(pieces[4]);
            List<Sensor> sensors = new List<Sensor>();
            for (int i = 5; i < pieces.Length - 1; i = i + 2)
            {
                Sensor sensor = new Sensor();
                sensor.Type = pieces[i];
                sensor.Value = pieces[i + 1];
                sensors.Add(sensor);
            }
            ap.Sensors = sensors.ToArray();
            return ap;
        }
    }
}
```

Apêndice G

Classe *UserProfile*

Listagem G.1: Código C# da classe UserProfile.

```
ïż£using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using UserProfile.Groups;

namespace UserProfile
{
    [Serializable]
        public class UP
        {
        public string name;
        public string phone;
        public string email;
        public Photo photo;
        public GUIs guis;
        public Local local;
        public Sensor[] sensors;

        public UP(string _name, string _phone, string _email, string _filename) {
            name = _name;
            phone = _phone;
            email = _email;
            photo = new Photo(_filename);
            guis = new GUIs();
            local = new Local();
            sensors = new List<Sensor>().ToArray();
        }

        public UP() {
            guis = new GUIs();
            local = new Local();
            sensors = new List<Sensor>().ToArray();
        }

        }
}
```

Apêndice H

Esquema XML da classe *Map*

Listagem H.1: Esquema XML dos mapas

```
ïż£<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="map"
    targetNamespace="http://ahpam.org/maps"
    elementFormDefault="qualified"
    xmlns="http://ahpam.org/maps"
    xmlns:mstns="http://ahpam.org/maps"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <!-- COORDINATE -->
  <xs:complexType name="tCoordinate">
    <xs:sequence>
      <xs:element name="x" type="xs:string"/>
      <xs:element name="y" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <!-- COORDINATES -->
  <xs:complexType name="tCoordinates">
    <xs:sequence>
      <xs:element name="Coordinate" type="tCoordinate" minOccurs="0" maxOccurs="unbounded"/
          >
    </xs:sequence>
  </xs:complexType>

  <!-- SPACE -->
  <xs:complexType name="tSpace">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
      <xs:element name="Coordinates" type="tCoordinates"/>
    </xs:sequence>
  </xs:complexType>

  <!-- SPACES -->
  <xs:complexType name="tSpaces">
    <xs:sequence>
      <xs:element name="Space" type="tSpace" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
```

```
  </xs:complexType>

  <!-- LEVEL -->
  <xs:complexType name="tLevel">
    <xs:sequence>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Map" type="xs:string" />
      <xs:element name="ScaleMeters" type="xs:string"/>
      <xs:element name="ScalePixels" type="xs:string"/>
      <xs:element name="Spaces" type="tSpaces"/>
    </xs:sequence>
  </xs:complexType>

  <!-- BUILDING -->
  <xs:element name="Building">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Levels" type="tLevel" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Apêndice I

Ficheiro XML da classe *Map*

Listagem I.1: Ficheiro XML gerado pelo *Map Designer*

```
<?xml version="1.0"?>
<Building xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.
    org/2001/XMLSchema" xmlns="http://ahpam.org/maps">
  <Name>Map</Name>
  <Levels>
    <Name>map1</Name>
    <Map>351Mapmap1.png</Map>
    <Spaces>
      <Space>
        <Name>Div. 1</Name>
        <Description>DescriÃğÃčo da divisÃčo 1</Description>
        <Coordinates>
          <Coordinate>
            <x>175</x>
            <y>225</y>
          </Coordinate>
        </Coordinates>
      </Space>
      <Space>
        <Name>Div. 2</Name>
        <Description>DescriÃğÃčo da divisÃčo 2</Description>
        <Coordinates>
          <Coordinate>
            <x>175</x>
            <y>225</y>
          </Coordinate>
        </Coordinates>
      </Space>
    </Spaces>
  </Levels>
</Building>
```