Tiago Francisco Preto Xavier

Mestrado em Engenharia Electrotécnica e de Computadores

Desenvolvimento de uma Infraestrutura Computacional que Visa o Aumento da Eficiência Energética em Edifícios Através da Utilização de Redes *Ad Hoc*

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Celson Pantoja Lima, Professor,
Universidade Federal do Oeste
do Pará
Co-orientador: João Francisco Alves Martins,
Professor Auxiliar, Faculdade de
Ciências e Tecnologia da
Universidade Nova de Lisboa

Júri:

Presidente: Prof. Doutor Luis Gomes
Arguente: Prof. Doutor José Barata Oliveira

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Abril, 2012

COPYRIGHT

DESENVOLVIMENTO DE UMA INFRAESTRUTURA COMPUTACIONAL
QUE VISA O AUMENTO DA EFICIÊNCIA ENERGÉTICA
EM EDIFÍCIOS ATRAVÉS DA UTILIZAÇÃO DE REDES *AD HOC*

TIAGO FRANCISCO PRETO XAVIER

DEDICATÓRIA E AGRADECIMENTOS

Ao Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, pelo uso das intalações e pelo apoio financeiro e institucional na aquisição de hardware e software.

A todos os professores do Departamento de Engenharia Electrotécnica por me terem transmitido os conhecimentos fundamentais para que esta dissertação fosse possível de realizar e que estiveram sempre presentes quando foram necessários.

Ao João Santos, em primeiro lugar pela sua amizade e, em segundo lugar, por ter partilhado comigo grande parte do trabalho desenvolvido nesta dissertação e que contribuiu em muito para que ela fosse possível de concretizar.

Aos meus amigos por todos os momentos que partilhei com eles e que me deram sempre alento nos momentos mais difíceis.

À minha família pela paciência, compreensão, apoio incondicional, incentivo e motivação que foram fundamentais para a concretização desta dissertação.

Finalmente, ao Prof. Celson Pantoja Lima, orientador da dissertação, um sincero agradecimento por acreditar e confiar, pela sua vontade e dedicação na acção de potenciar sempre ao máximo os seus alunos e incentivá-los a inovar. Pelas suas avaliações, juízos e decisões que me influenciaram bastante no passado e que espero que continuem a ser uma referência no meu presente e futuro. Pela amizade.

RESUMO

Sinais identificados na literatura e no comportamento social apontam para uma acentuada diminuição da eficiência energética em edifícios, tornando-se este facto num aspecto social do qual nenhum de nós se poderá distanciar. Esta diminuição tem alcançado nos últimos anos números que nos devem alarmar para a criação de uma rápida solução para este problema global. Por outro lado, as tecnologias de redes sem fios constituem um enorme suporte para o desenvolvimento das mais variadas aplicações baseadas nas comunicações móveis. Mais, com o crescimento do mercado dos dispositivos móveis (como são exemplo os *Smartphones* e PDAs) essas tecnologias de redes sem fios estão agora e, com o avançar dos tempos, cada vez mais ao alcance de um elevado número de pessoas.

Nesta dissertação é desenvolvida uma solução computacional denominada *Building Sensors Manager* (BSM), que visa auxiliar a gestão de edifícios de forma energeticamente eficiente, baseada na utilização de tecnologias de redes sem fios (mais precisamente redes *ad hoc*). Nessa solução, as pessoas são transformadas em elementos activos no processo, contribuindo elas próprias para uma melhoria da sua qualidade de vida.

O objectivo do BSM é recolher valores relativos às constantes físicas verificadas no edifício para poder actuar nele de uma forma calculada. Desta forma, são aplicadas no edifício as medidas que maximizam a sua eficiência energética. Neste processo, o papel das pessoas consiste em difundir a informação relativa ao estado físico do edifício até a fazer chegar a um local deste que possua os meios necessários para poder actuar. Essa difusão é efectuada através da criação de redes *ad hoc* de forma a que a informação circule através de dispositivos móveis até chegar ao seu destino, ao mesmo tempo que informa as pessoas na rede acerca estado actual do edifício.

Quanto às funcionalidades do BSM, destacam-se: a capacidade de os dispositivos móveis na mesma rede *ad hoc* poderem comunicar livremente entre si, quer através de mensagens de texto, quer através de ficheiros; a possibilidade de os dispositivos móveis no edifício servirem de ponte entre dispositivos de recolha de dados e um sistema de gestão do edifício; e a capacidade para juntar num local pré-determinado no edifício toda a informação relevante acerca deste e que pode ser utilizada para se poder actuar nele de forma ponderada.

ABSTRACT

The literature and social behavior point to a sharp decline in energy efficiency in buildings, becoming a social aspect which neither of us could pull away. This reduction has achieved in recent years numbers that should alarm us to create a quick solution to this global problem. Thus, the wireless networks technologies are a huge support for the development of several applications based on mobile communications. Moreover, with the exploit of the market for mobile devices (such as Smartphones and PDAs) such wireless networks technologies are now and, with the advancing of time, are available for a larger group of people.

This thesis addresses a solution named Building Sensors Manager (BSM), that aims to help the management of buildings in an energy-efficient way, based on the use of wireless networks technologies (more precisely ad hoc networks). In this solution, people are transformed into active elements in the process, helping themselves to improve their quality of life.

The goal of BSM is to collect values for the physical constants observed in the building for it to act in a calculated manner. This way, the measures that maximize energy efficiency of the building are applied to it. In this process, the role of people is to broadcast the information of the physical condition of the building to make it reach a place that possesses the necessary means to act. This broadcast is made by creating ad hoc networks so that information navigates through mobile devices until it reaches its destination, the same time as the network informs people about the current state of the building.

The functionalities of BSM include: the ability that mobile devices in the same ad hoc network have to communicate freely with each other, either through text messages, either through files; the possibility of mobile devices in the building to serve as a bridge between data collection devices and a building management system; and the ability to join in a pre-determined location in the building all the relevant information about it and that can be used in order to act on it thoughtfully.

ÍNDICE DE MATÉRIAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

ANEE:	Ad Hoc Networks for Energy Efficiency
AP:	Access Point
APDC:	Associação Portuguesa para o Desenvolvimento das Comunicações
API:	Application Programming Interface
BMS:	Building Management System
BSM:	Building Sensors Manager
CBRNE:	Chemical, Biological, Radiological, Nuclear and Explosives
CM:	Conselho de Ministros
DEE:	Departamento de Engenharia Electrotécnica
DER:	Diagrama de Entidades e Relações
ECO.AP:	Programa de Eficiência Energética na Administração Pública
ESE:	Empresas de Serviços Energéticos
ESM:	Estação de Suporte à Mobilidade
FCT/UNL:	Faculdade de Ciências e Tecnlologia da Universidade Nova de Lisboa
GUI:	Graphic User Interface
GUID:	Group Unique Identifier
HA:	Hotspot Application
IEEE:	Institute of Electrical and Electronics Engineers
IMT:	International Mobile Telecommunications
IP:	Internet Protocol
LAN:	Local Area Network
MD:	Map Designer
MDA:	Mobile Device Application
MP3:	MPEG – ½ Audio Layer 3
MPEG:	Moving Picture Experts Group
SMS:	Short Message Service
SVA:	Sensor Values Analyzer
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
UE:	União Europeia
UML:	Unified Modeling Language
UTD:	Universal Tag Distributer
Wi-Fi:	Wireless Fidelity
WiMAX:	Worldwide Interoperability for Microwave Access
WLAN:	Wireless Local Area Network
XML:	Extensible Markup Language

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

1.1.1. EFICIÊNCIA ENERGÉTICA

A eficiência energética tem sido um tema sobre o qual muito se tem debatido nos últimos anos, quer devido à crescente sensibilização acerca deste tema por parte dos governos e das populações, quer devido à propagação de informação (por vezes até de forma alarmista) dos meios de comunicação social. A verdade é que cada vez mais as pessoas se preocupam com a problemática da preservação do meio ambiente e cada vez mais se mostram disponíveis para de alguma forma contribuírem para um melhoramento nesta área.

A questão da eficiência energética coloca-se tanto ao nível dos edifícios, nomeadamente edifícios públicos, tais como centros comerciais ou estabelecimentos de ensino, como em edifícios privados, como empresas, casas particulares e grandes condomínios. Neste tipo de edifícios é comum a utilização de sistemas de aquecimento e arrefecimento para melhorar o conforto no seu interior. No entanto, a literatura relevante refere que a grande maioria destes sistemas não possui qualquer tipo de controlo para verificar se a quantidade de energia que está a ser usada para o seu funcionamento é a correcta.

Em Março de 2007 [1], os líderes da União Europeia (UE) embarcaram numa política climática e energética com o objectivo de aumentar a segurança energética da UE e, ao mesmo tempo, fortalecer a sua competitividade. Estas entidades comprometeram-se em transformar a Europa numa economia altamente eficiente energeticamente.

Para iniciar este processo, os Chefes de Estado e de Governo da UE definiram uma série de objectivos climáticos e energéticos para serem cumpridos até 2020, denominados como objectivos "20-20-20", os quais caracterizam-se por:

- ✓ Uma redução das emissões de gases com efeito de estufa na UE de pelo menos 20% abaixo dos níveis de 1990;
- ✓ 20% de consumo de energia da UE terá de ser proveniente de recursos renováveis; e
- ✓ Uma redução de 20% na utilização de energia primária comparando com os níveis projectados, através da melhoria da eficiência energética.

Em Janeiro de 2008, a Comissão Europeia propôs uma legislação para implementar os objectivos "20-20-20". Este pacote climático e energético foi aprovado pelos Chefes de Estado e de Governo da UE em Dezembro de 2008 e tornou-se lei em Junho de 2009 [2].

Passando para o plano nacional, de acordo com um estudo [3] promovido pela Associação Portuguesa para o Desenvolvimento das Comunicações (APDC), mais de metade da factura energética do Estado corresponde aos edifícios públicos. Só em 2010 o consumo de energia nos edifícios da Administração Pública ascendeu aos 260 milhões de euros, contribuindo para a factura total de 500 milhões ligados ao consumo energético do estado. Os consumos nos edifícios da Administração Pública estão a crescer ao dobro da média nacional (77%), se tivermos como referência os anos de 1994 e 2008, indicam ainda as conclusões do mesmo estudo.

Como resposta a este estudo, o governo deciciu lançar um programa que pretende actuar nos edifícios públicos até 2013 visando aumentar a eficiência energética em 20% até 2020, de forma a cumprir os objectivos exigidos pela UE. Esta iniciativa denomina-se Programa de Eficiência Energética na Administração Pública – ECO.AP [4] e, citando a nota do Conselho de Ministros (CM), "traduz-se num conjunto de medidas de eficiência energética para a execução a curto, médio e longo prazo nos serviços, organismos e equipamentos públicos e que tem por objectivo alterar comportamentos e promover uma gestão racional dos serviços energéticos, nomeadamente através da contratação de Empresas de Serviços Energéticos (ESE)".

Entre as medidas aprovadas na resolução inclui-se uma "intervenção em todos os edifícios [da Administração Pública] até 2013", a "criação da figura do gestor local de energia, responsável pela dinamização e verificação das medidas comportamentais de eficiência energética em cada serviço ou organismo da Administração Pública" ou "a implementação de um barómetro da eficiência energética", destinado a divulgar os consumos energéticos de todos os edifícios e serviços.

Desta forma é compreensível que o tema da eficiência energética é um tema relevante e de muita importância o qual necessita da participação conjunta da sociedade como um todo.

1.1.2. TECNOLOGIAS DE REDES SEM FIOS

Nos últimos anos tem-se notado um crescimento exponencial das redes sem fios trazendo consigo uma proliferação de inúmeras tecnologias de redes wireless (WLAN – *Wireless Local Area Network*). São exemplos dessas tecnologias o Bluetooth, o IMT (*International Mobile Telecommunications*) – 2000 (mais conhecido como 3G) e equipamentos baseados nas normas IEEE (*Institute of Electrical and Electronics Engineers*) 802.11 (Wi-Fi – *Wireless Fidelity*) e IEEE 802.16 (WiMAX – *Worldwide Interoperability for Microwave Access*).

Hoje em dia a maioria das pessoas transporta consigo dispositivos portáteis tais como computadores pessoais, telemóveis, PDAs (*Personal Digital Assistant*) e leitores MP3 (MPEG (*Moving Picture Experts Group*) – *½ Audio Layer 3*), tanto para uso pessoal como profissional. Estima-se que em poucos anos a grande maioria da população mundial possuirá algum tipo de dispositivo portátil com capacidade de se comunicar quer com um ponto fixo numa rede, quer com outros dispositivos móveis.

Todos os equipamentos anteriormente mencionados partilham de um ponto comum oferecem aos seus utilizadores várias possibilidades ao nível da comunicação sem as desvantagens associadas à utilização de cabos. É aqui que entram as tecnologias de redes sem fios. As suas capacidades são ilimitadas e pode-se dizer que, apesar de já há bastante tempo que elas estão envolvidas na nossa vida quotidiana, ainda não extraímos delas todas as capacidades que elas nos podem oferecer.

As redes sem fios dividem-se basicamente em dois tipos: redes infra-estruturadas e redes *ad hoc*. As primeiras caracterizam-se principalmente pela existência de um ponto central na rede (ESM (Estação de Suporte à Mobilidade) ou AP (*Access Point*)) por onde cada *host* móvel sempre que pretende comunicar deve fazer passar a informação, mesmo que ao seu alcance se encontrem outros dispositivos móveis que eventualmente poderiam comunicar entre si. Este tipo de redes apresenta uma fiabilidade considerável mas ao mesmo tempo obriga à existência de pontos fixos na rede que retiram a possibilidade de os dispositivos móveis comunicarem entre si.

Por outro lado, nas redes *ad hoc* não existe nenhum ponto fixo na rede, comunicando os dispositivos móveis directamente entre si. Esta característica das redes *ad hoc* torna-as numa tecnologia com múltiplas aplicações móveis, uma vez que a comunicação envolvida não obriga à existência de componentes imóveis na rede. Desta forma, utilizando esta tecnologia, para que dois ou mais dispositivos se comuniquem basta que eles estejam presentes, seja num edifício, seja numa rua, numa floresta ou num deserto.

Tendo em conta as características particulares das redes *ad hoc*, enumeram-se várias aplicações tendo em conta as suas capacidades:

- ✓ Coordenação de equipas de resgate em situações de desastre, tais como sismos, inundações ou furacões;
- ✓ Partilha de informações em reuniões e aulas;
- ✓ Partilha automática de informação entre as pessoas quando se encontram;
- ✓ Comunicação entre automóveis para propagar informação sobre as condições das estradas;
- ✓ Melhoria da eficiência energética em edifícios através da partilha de informação sensorial no seu interior.

Todos estes exemplos de aplicabilidade das redes *ad hoc*, para serem tornados realidade, necessitam de uma camada de desenvolvimento "por cima" dos protocolos de comunicação utilizados por este tipo de redes. Ou seja, partindo do que já existe ao nível do roteamento de dispositivos numa rede *ad hoc*, o desafio

principal caracteriza-se por constuir uma camada de suporte à comunicação entre os dispositivos na rede, de forma a que essa comunicação seja espontânea entre eles.

Um dos focos desta dissertação tem como objectivo a construção dessa camada que torne transparente para o utilizador a comunicação numa rede *ad hoc* entre dispositivos móveis.

1.2. VISÃO

Para que um edifício seja eficiente é necessário que consuma o mínimo de energia possível tendo conforto e qualidade do ar como se exige. Todos os edifícios deveriam ter como objectivo aumentar a qualidade energética da construção e não somente a qualidade estética. Este comportamento eficiente contribuiria inclusivamente para que os edifícios reduzissem os seus custos financeiros associados aos sistemas dependentes da energia.

No entanto, para que um edifício consiga racionalizar os seus custos energéticos necessita primeiro saber qual o estado actual/usual das divisões que o compõem de forma a poder aplicar nesses pontos as medidas correctas. Assim, uma forma equilibrada de fornecer energia a um edifício baseia-se na avaliação das constantes físicas nele verificadas. Neste cenário, o edifício deve, através da análise de informação sensorial recolhida em várias partes deste, ajustar os seus gastos de energia. Para que isto seja possível é necessário que exista um conjunto de dispositivos (sensores e controladores) distribuídos pelo edifício que fiquem responsáveis pela recolha de informação sensorial e, consoante os valores obtidos, actuem sobre o próprio edifício em tempo real. Com um sistema computacional de suporte deste género a energia dispendida pelo edifício estaria sempre em consonância com a energia por ele requerida, terminando-se assim com gastos superiores aos necessários.

Adicionalmente à existência destes sensores e controladores, que são principalmente componentes estáticos no edifício, seria também interessante a presença no processo de aumento da eficiência energética de dispositivos móveis que ajudariam à propagação de informação relevante dentro do edifício. Estes dispositivos móveis podem ser, por exemplo, PDAs ou *Smartphones*, cujas capacidades intrínsecas permitem comunicar-se com qualquer tipo de dispositivo que possua as mesmas tecnologias com que eles próprios vêm equipados.

Um sistema computacional de apoio ao processo de eficiência energética com estas características, necessita ainda da existência de um "centro de comando" no edifício que armazene todos estes dados, ou seja, um *Building Management System* (BMS). Este componente ficaria responsável, principalmente, pela gestão de todos os dados relevantes para se obter um aumento da eficiência energética no edifício. Também este constituinte do sistema necessita de comunicar com os dispositivos por ele espalhados, por forma a trocar informação com estes acerca do estado actual do edifício.

1.3. OBJECTIVOS

O principal objectivo do trabalho desenvolvido é fornecer ao edifício, através de comunicação via redes *ad hoc* entre este, os dispositivos móveis e os dispositivos de recolha de dados, informação relevante sobre este que o possa auxiliar a tomar medidas que levem a um aumento da sua eficiência energética. O sistema computacional desenvolvido foi denominado de *Building Sensors Manager* (BSM) e para que ele cumpra o seu principal objectivo, outros objectivos específicios necessitam também de ser alcançados.

Um dos objectivos específicos, e talvez o mais importante, está relacionado com a comunicação entre os componentes envolvidos no processo descrito anteriormente. Esta comunicação, como já foi referido, será realizada usando o protocolo de comunicação *ad hoc*. Desta forma, é necessário desenvolver um protocolo baseado nas redes *ad hoc* que dê suporte a esta comunicação que possa ser utilizado por qualquer equipamento que cumpra os requisitos do BSM. Este protocolo pretende-se que seja transparente para o utilizador e que suporte a transferência de vários tipos de informação (mensagens de texto, classes XML (*Extensible Markup Language*), ficheiros multimédia, etc.), de forma a que todos e quaisquer dados possam ser trocados entre dois ou mais dispositivos.

O último objectivo específico que necessita de ser alcançado é a recolha da informação sensorial no edifício. Essa informação constitui um elemento fulcral no processo de aumento de eficiência energética no edifício pois é nela que se baseia a actuação neste. Este objectivo deve ser alcançado através da instalação nas divisões do edifício de dispositivos que realizem periodicamente uma avaliação das constantes físicas verificadas e enviem esses dados aos dispositivos móveis ao seu alcance.

Depois de alcançados estes objectivos, estão garantidas as condições para que se possa actuar de forma equilibrada e energeticamente eficiente no edifício.

1.4. CONTEXTO DE DESENVOLVIMENTO

O conteúdo desta dissertação encontra-se inserido num projecto de maiores dimensões denominado *Ad Hoc Networks for Energy Efficiency* (ANEE) que engloba não só a intenção de aumentar a eficiência energética em edifícios tendo como base as constantes físicas nele verificadas, mas também através do número de pessoas que o habitam. Ainda outro trabalho complementar a este usa os valores fornecidos pelo ANEE e serve-se deles para actuar fisicamente no edifício.

Nesta dissertação serão abordados dois temas principais: as redes *ad hoc* e a eficiência energética. De uma forma geral este documento tem como objectivo apresentar uma solução baseada nas redes *ad hoc* que permita aumentar a eficiência energética. Como já foi referido, essa acção será maioritariamente destinada a edifícios; no entanto, a solução criada é adaptável a vários outros ambientes. Uma das principais missões

é, portanto, desenvolver um protocolo baseado nas redes *ad hoc* destinado a dispositivos móveis, para que estes tenham a capacidade de se tornarem agentes activos no aumento da eficiência energética.

No desenvolvimento do protocolo baseado nas redes *ad hoc* foi necessário abordar algumas temáticas relacionadas com os sistemas de telecomunicações. Este trabalho envolve protocolos de mais baixo nível responsáveis pela regulação de uma rede *ad hoc* e protocolos de comunicação como o TCP/IP *Transmission Control Protocol/Internet Protocol* (TCP/IP) e *User Datagram Protocol* (UDP). Foi ainda necessário criar outros protocolos destinados a situações específicas do sistema e que ainda não se encontravam desenvolvidos até ao momento.

Como consequência do desenvolvimento destes protocolos, outras aplicações foram adicionalmente desenvolvidas num contexto à parte da temática da eficiência energética. Entre elas destaca-se uma aplicação que permite que os dispositivos móveis numa rede *ad hoc* possam comunicar entre si sem qualquer tipo de custos para o utilizador, ao contrário do que sucederia com a comunicação através da Internet, de uma chamada telefónica ou de um *Short Message Service* (SMS). Esta comunicação pode ser efectuada entre dois dispositivos móveis (conversa privada) ou dentro de um grupo criado dentro da própria rede *ad hoc* (conversa de grupo). Outra aplicação desenvolvida suporta a criação de eventos em grupos existentes dentro da rede *ad hoc*, sendo todos os dispositivos móveis pertencentes a um grupo avisados acerca da existência de todos os eventos criados para ele. Para a criação destas aplicações foi necessário criar o conceito de grupo numa rede *ad hoc* para que a privacidade possa ser mantida na rede, ou seja, um dispositivo móvel só se poderá comunicar com outro que pertença a um grupo seu.

1.5. ESTRUTURA DO DOCUMENTO

O capítulo 2 descreve as redes *ad hoc*. Nele são apresentadas e discutidas as suas vantagens e desvantagens, uma breve descrição do que são serviços móveis, bem como uma visão geral daquilo que já existe desenvolvido a nível de software até ao momento. Serão ainda abordadas as diferenças entre as redes sem fios infra-estruturadas e as redes *ad hoc*. Num contexto mais aproximado ao do sistema apresentado nesta dissertação, serão também neste capítulo discutidas as redes *ad hoc* móveis (MANETs) e as redes *ad hoc* de sensores.

O capítulo 3 explica de uma forma geral o funcionamento BSM. Aqui será abordado de uma forma mais específica o sistema responsável pela gestão do edifício, ou seja, reponsável por fazer a previsão dos gastos de energia nele. Será ainda explicada a forma como este elemento actua instantaneamente no edifício, como consegue antecipar os gastos de energia através de um registo histórico e como efectua o controlo sensorial.

No capítulo 4 é apresentado o modelo conceptual utilizado para o desenvolvimento do BSM. Essa modelação foi realizada através de *Unified Modeling Language* (UML). Numa primeira fase são apresentados

os requisitos que o sistema necessita de cumprir. De seguida é apresentada a visão funcional através de um diagrama de casos de uso. Segue-se a visão arquitectural que é apresentada através de diagramas de classes conceptuais e diagramas de blocos. Para finalizar é apresentado o modelo de dados utilizado através de um DER (Diagrama de Entidades e Relações).

No capítulo 5 é descrita a parte da implementação onde são apresentadas as tecnologias utilizadas neste trabalho, bem como os diagramas de classes representativos da implementação.

No capítulo 6 é apresentado um exemplo de utilização que mostra a operação do BSM acompanhada de algumas *screenshots* do sistema computacional e respectiva descrição.

Finalmente, no capítulo 7 encontram-se descritas as conclusões e possíveis trabalhos futuros resultantes do trabalho desenvolvido.

2. REDES DE COMUNICAÇÃO

2.1. INTRODUÇÃO

Uma rede contém um conjunto de dispositivos cuja finalidade é executar acções. Esses dispositivos são, em telecomunicações, denominados de *hosts*. Os *hosts* estão conectados por uma sub-rede de comunicação.

A sub-rede consiste em dois componentes distintos: linhas de transmissão e elementos de comutação. As linhas de transmissão transportam a informação entre as máquinas. Elas podem ser formadas por fios de cobre, fibra óptica ou mesmo enlaces de rádio. Os elementos de comutação são computadores especializados que conectam três ou mais linhas de transmissão. Quando os dados chegam a uma linha de entrada, o elemento de comutação deve escolher uma linha de saída para encaminhá-los. Esses computadores de comutação são denominados de *routers*.

No modelo apresentado na Figura 2.1 os *hosts* estão ligados a uma LAN (*Local Area Network*) em que há um *router*, embora em alguns casos um *host* possa estar ligado directamente a um *router*. O conjunto de linhas de comunicação forma a sub-rede.

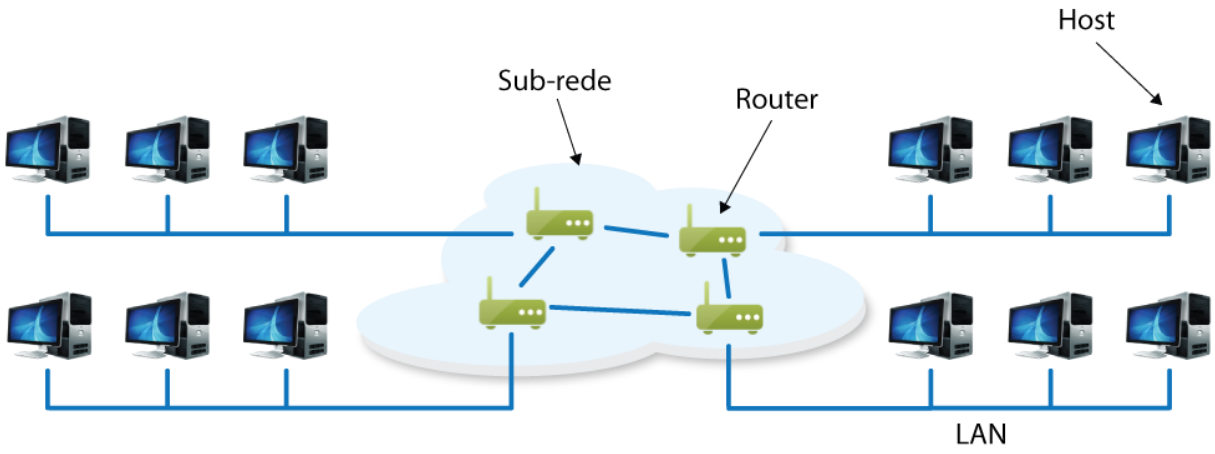

Figura 2.1 – Relação entre *hosts* em LANs e a sub-rede.

2.2. COMUNICAÇÃO MÓVEL

Computadores móveis, como *notebooks* e PDAs, constituem um dos segmentos de mais rápido crescimento da indústria da informática [5]. Muitos utilizadores desses computadores têm máquinas de *desktop* em casa ou no escritório e querem manter-se ligados a essa base mesmo quando estão longe.

As redes sem fios têm muitas utilidades. Um uso comum é o escritório portátil. Quando viajam, muitas vezes as pessoas querem usar o seu equipamento electrónico portátil para enviar e receber ligações telefónicas, fax e correio electrónico, navegar pela Web, aceder a arquivos remotos e ligar-se a máquinas distantes. Além do mais, elas querem fazer isso enquanto se encontram em qualquer lugar do planeta.

Embora as redes sem fios e a computação móvel frequentemente tenham uma estreita relação, elas não são idênticas, como mostra a Tabela 2.1. Aqui, observa-se uma distinção entre redes sem fios fixas e redes sem fios móveis. Algumas vezes, até mesmo os computadores portáteis podem estar ligados por fios. Por exemplo, se um turista liga o seu *notebook* à tomada de telefone num quarto de hotel, ele tem mobilidade sem precisar de utilizar uma rede sem fios.

Tabela 2.1 – Combinações de redes sem fios e de computação móvel [5].

REDE SEM FIOS	REDE MÓVEL	APLICAÇÕES
Não	Não	Computadores *desktop* em escritórios
Não	Sim	Um *notebook* usado num quarto de hotel
Sim	Não	Redes em edifícios mais antigos que não dispõem de fios
Sim	Sim	Escritório portátil; PDA para registar o *stock* de uma loja

Porém, é claro que existem as verdadeiras aplicações sem fios móveis, que variam desde o escritório portátil até pessoas a caminhar por uma loja com um PDA para fazer o levantamento do stock.

À medida que a tecnologia sem fios se torna cada vez mais difundida, numerosas outras aplicações vão surgindo de forma a aproveitar ao máximo as suas vantagens.

2.3. REDES SEM FIOS

Numa primeira aproximação, as redes sem fios podem ser divididas em duas categorias principais: interconexão de sistemas e WLANs.

Interconexão de sistemas significa interligar os componentes de um computador usando rádio de alcance limitado. Ainda não há muito tempo, quase todos os computadores tinham um monitor, um teclado, um rato e uma impressora, ligados por cabos à unidade principal. Em 1994, cinco empresas (Ericsson, IBM, Intel, Nokia e Toshiba) uniram-se para projectar uma rede sem fios de alcance limitado, denominada Bluetooth, a fim de conectar esses componentes sem a utilização de fios (Figura 2.2). A rede Bluetooth também permite a conexão de cameras digitais, auscultadores, scanners e outros dispositivos a um computador, simplesmente trazendo-os para dentro do alcance da rede [5].

Figura 2.2 – Exemplo de uma configuração Bluetooth.

As WLANs, ou LANs sem fios, são sistemas em que todo o computador tem um *modem* de rádio e uma antena através dos quais se pode comunicar com outros sistemas. Frequentemente, existe uma antena nos edifícios que permite a comunicação das máquinas, como mostra a Figura 2.3. Porém, se os sistemas estiverem próximos o suficiente eles poderão comunicar-se directamente um com o outro numa configuração não hierárquica. O padrão para as WLANs é chamado de IEEE 802.11.

Figura 2.3 – Esquema de uma WLAN.

2.4. REDES *AD HOC*

Uma rede *ad hoc* consiste num conjunto de nós ou terminais autónomos que comunicam entre si formando uma rede de rádio multi-hop (rede que faz uso de dois ou mais hops[1] sem fios para trocar informação desde uma origem até um destino) mantendo a conectividade de uma forma descentralizada. Uma vez que os nós comunicam através de um suporte sem fios, eles ficam reféns dos efeitos das comunicações por rádio, tais como ruído, perdas e interferência. Mais, as ligações numa rede *ad hoc* contém tipicamente menor largura de banda quando comparadas com redes com fios. Na Figura 2.4 encontra-se representada a forma como os dispositivos ficam dispostos no modelo *ad hoc*.

Figura 2.4 – Esquema representativo de uma rede *ad hoc*.

Cada nó numa rede *ad hoc* funciona como *host* e como *router*, sendo o controlo da rede distribuído pelos vários nós que a compõem. A topologia da rede é dinâmica pois a conectividade entre os nós pode variar com o tempo devido à partida e chegada de nós e à possibilidade de haver nós móveis. Desta forma, há a necessidade de existirem protocolos de comunicação bastante eficientes para permitir que os nós comuniquem através de caminhos multi-hop formando várias ligações, de uma forma que se faça uso apenas dos recursos da rede necessários para o efeito.

Algumas destas características das redes *ad hoc* fazem parte ainda daquilo que foi estudado intensivamente nos anos 1970 e 1980, quando este tipo de redes começou a ser abordado. Ainda assim, a pesquisa na área das redes *ad hoc* está a receber muita atenção por parte da investigação, da indústria e do governo. No entanto, uma vez que este tipo de redes possui muitas questões complexas existem muitas reservas na sua pesquisa e nas oportunidades para se fazerem avanços significativos [6].

[1] Hops são "saltos" dados por uma mensagem, entre um nó e outro de uma rede, antes de chegar ao seu destino.

As redes *ad hoc* possuem determinadas características que as tornam numa tecnologia de comunicação única que se destaca bastante das restantes. Elas apresentam muitos pontos positivos, mas também apresentam algumas características menos positivas que se não forem bem estudadas podem por em causa factores na rede, tais como a segurança, a fidelidade ou a própria comunicação [8].

Uma característica das redes *ad hoc* que convém destacar é a capacidade de os nós pertencentes à rede se poderem mover arbitrariamente. Isto obriga a que a rede seja capaz de se adaptar a variações constantes na sua topologia, ou seja, ela necessita de estar permanentemente a reconfigurar as tabelas de encaminhamento dos seus nós.

No entanto, a não existência de pontos fixos na rede traz consigo algumas limitações às redes *ad hoc*. A principal reside no facto de a distância entre os dispositivos numa rede *ad hoc* não poder ser superior àquela que é necessária para que as antenas Wi-Fi dos próprios dispositivos estejam em contacto. Quer isto dizer que, se um dispositivo possui uma antena Wi-Fi com um alcance de, por exemplo, 100 metros, qualquer outro dispositivo que pretenda comunicar com este primeiro terá sempre de estar situado num raio não superior a este valor. Isto, claro, não abordando as questões de possíveis interferências relacionadas com a existência de obstáculos físicos que podem sempre causar alguma redução do alcance original. Ainda assim estas limitações verificam-se não só para as redes *ad hoc*, mas para todas as tecnologias de redes sem fios, não se tratando portanto de uma desvantagem das primeiras em relação a qualquer outra das segundas.

Na Figura 2.5 encontram-se representados três nós numa rede *ad hoc*, onde o respectivo alcance de cada um é representado por uma circunferência centrada na sua posição.

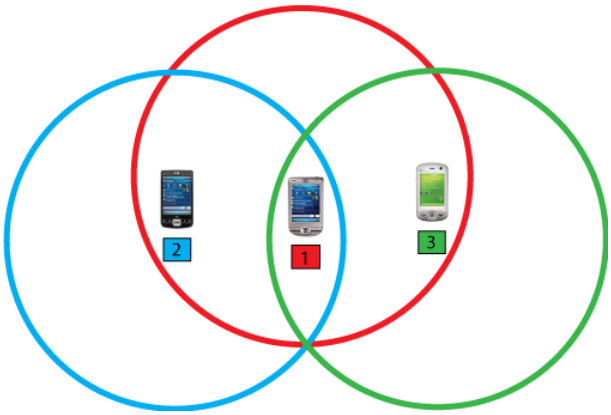

Figura 2.5 – Representação do alcance de três nós numa rede *ad hoc*.

Como se pode observar, o nó 1 está ao alcance dos nós 2 e 3, mas o nó 2 não está ao alcance do nó 3. No entanto, uma vez que estes têm um nó entre eles que está ao alcance dos dois, eles conseguem comunicar entre si. O que acontece neste caso é que quando o nó 2 quiser comunicar com o nó 3, a informação terá sempre que passar pelo nó 1 e vice-versa, ou seja, o nó 1 funciona como ponte entre os nós 2 e 3.

2.5. REDES *AD HOC* VS. REDES INFRA-ESTRUTURADAS

Na Tabela 2.2 encontram-se apresentadas as principais vantagens e desvantagens apresentadas pelas redes *ad hoc* quando comparadas com as redes infra-estruturadas [9].

Tabela 2.2 – Vantagens e desvantagens das redes *ad hoc* em relação às redes infra-estruturadas.

	PARÂMETRO	DESCRIÇÃO
VANTAGENS	Rápida instalação	As redes *ad hoc* podem ser estabelecidas dinamicamente em locais onde não haja previamente uma infra-estrutura de rede instalada.
	Tolerância a falhas	A permanente adaptação e reconfiguração das rotas em redes *ad hoc* permitem que perdas de conectividade entre os nós possam ser facilmente resolvidas desde que uma nova rota possa ser estabelecida.
	Conectividade	Dois nós móveis podem comunicar directamente entre si desde que cada nó esteja dentro da área de alcance do outro. Em redes infra-estruturadas ou em redes fixas, mesmo que dois nós estejam próximos, é necessário que a comunicação passe pela estação de suporte à mobilidade (no caso de redes infra-estruturadas) ou, no caso de redes fixas, haver uma ligação por meio de cabo entre os dois nós.
	Mobilidade	Esta é uma vantagem primordial principalmente em comparação com as redes fixas.
DESVANTAGENS	Encaminhamento	A mobilidade dos nós e uma topologia de rede dinâmica contribuem directamente para tornar a construção de algoritmos de encaminhamento um dos principais desafios em redes *ad hoc*.
	Localização	Uma questão importante em redes *ad hoc* é a localização de um nó pois além do endereço da máquina não ter relação com a posição actual do nó, também não existem informações geográficas que auxiliem na determinação do posicionamento do nó.
	Taxa de erros	A taxa de erros associada a enlaces sem-fio é consideravelmente mais elevada.
	Largura de banda	Enquanto que em meios com cabos a largura de banda pode chegar até 1 Gbps, os enlaces sem-fio suportam tipicamente taxas de até 2 Mbps.

2.6. *MOBILE AD HOC NETWORKS* (MANETS)

Na próxima geração de sistemas de comunicação sem fios, existirá a necessidade da implantação rápida de utilizadores móveis independentes [10]. Exemplos significativos incluem o estabelecimento de comunicações eficientes e dinâmicas para operações de emergência/resgate, desastres naturais e redes militares. Ainda segundo [10], o tipo de redes necessárias nestes cenários não se pode basear numa conectividade centralizada e organizada, podendo ser caracterizadas por redes *ad hoc* móveis.

Uma MANET consiste num conjunto autónomo de utilizadores móveis que comunicam através de ligações sem fios com uma relativamente limitada largura de banda. Uma vez que os nós são móveis, a topologia da rede pode alterar-se rapidamente e de uma forma inesperada ao longo do tempo. A rede é descentralizada, onde toda a actividade da rede, incluindo a descoberta da topologia e a entrega de mensagens, tem de ser executada pelos póprios nós, ou seja, as capacidades de encaminhamento na rede necessitam de estar integradas nos nós móveis. Na Figura 2.6 mostra-se como se procede numa MANET quando ocorre uma falha numa ligação.

Figura 2.6 – Reparação da rota em caso de falha numa ligação numa MANET.

O conjunto de aplicações para as MANETs é muito vasto, podendo ir desde redes pequenas e estáticas, limitadas na sua potência, até redes móveis de larga escala altamente dinâmicas. O desenho dos protocolos de comunicação para este tipo de redes é bastante complexo. Independentemente da aplicação, as MANETs necessitam de algoritmos eficientes e distribuídos de forma a determinar a organização da rede, as tabelas de encaminhamento e o próprio encaminhamento [10].

No entanto, determinar tabelas de encaminhamento e entregar mensagens de uma forma viável num ambiente descentralizado onde a topologia varia, é um processo difícil. Enquanto que o caminho mais curto (com base numa dada função de custo) de uma origem até um destino (numa rede estática) é normalmente a melhor escolha para realizar o encaminhamento, esta ideia não é facilmente extensível às MANETs. Factores tais como a variável qualidade da ligação sem fios, a perda de caminho de propagação, o desvanecimento, a interferência multi-utilizador, a potência dispendida e as alterações na topologia da rede tornam-se problemas relevantes. A rede deve estar apta a adpatar-se alterando as tabelas de encaminhamento constantemente, de forma a minimizar todos estes efeitos [10].

Tendo como base o exemplo de um ambiente militar, é perceptível que este tipo de redes necessitam de ser desenvolvidas de forma a preservar algumas características significativas, tais como a segurança, a latência, a fiabilidade ou a ocorrência e correcção de falhas. As redes militares são desenhadas de forma a manterem uma baixa probabilidade de intercepção e/ou uma baixa probabilidade de detecção. Daí que os nós prefiram irradiar a menor potência possível e transmitir o menor número de vezes possível, diminuindo assim a probabilidade de detecção ou intercepção. Um lapso em qualquer um destes requisitos pode por em causa a performance e a dependência da rede.

Desta forma, é perceptível que ao aplicarmos uma rede *ad hoc* a uma determinada situação necessitamos primeiro de ter em conta vários aspectos destas que possam por em causa o seu propósito. Este cuidado tem de ser especialmente aplicado na utilização de redes *ad hoc* devido às características já referidas destas, que as tornam numa tecnologia com inúmeras aplicações e com um futuro muito prometedor a vários níveis, mas também numa tecnologia com a qual é necessário conhecer as suas limitações antes de ser aplicada a alguma situação.

2.7. REDES *AD HOC* DE SENSORES

Uma rede *ad hoc* de sensores consiste num conjunto de sensores espalhados numa determinada área geográfica. Cada sensor possui capacidades de comunicação sem fios e algum nível de inteligência para efectuar processamento de sinais e encaminhamento de informação. Alguns exemplos de redes *ad hoc* de sensores apresentam-se de seguida:

- ✓ Redes de sensores para detecção e monitorização de alterações ambientais em planícies, florestas, oceanos, etc.;
- ✓ Redes de sensores de tráfego para monitorização do tráfego de veículos em auto-estradas ou em partes especialmente congestionadas numa cidade;
- ✓ Redes de sensores militares para detecção e ganho do maior número de informação possível acerca do movimento do inimigo, explosões e outros fenómenos de interesse;
- ✓ Redes de sensores para detectar e caracterizar material e ataques químicos, biológicos, radioactivos, nucleares e explosivos (CBRNE);

- ✓ Redes de sensores de vigilância para providenciar segurança em centros comerciais, parques de estacionamento e outras infraestruturas; e
- ✓ Redes de sensores de estacionamento para determinar quais lugares se encontram ocupados e quais se encontram livres.

A lista de exemplos apresentada sugere a ideia de que as redes *ad hoc* de sensores oferecem certas capacidades e melhorias na eficiência operacional em aplicações nas mais variadas áreas. Na Figura 2.7 está representada a forma como os sensores se encontram normalmente distribuídos numa rede deste tipo.

Figura 2.7 – Esquema representativo de uma rede *ad hoc* de sensores [15].

Existem duas formas de classificar uma rede *ad hoc* de sensores: dependendo se os nós são ou não individualmente endereçáveis ou se a informação na rede está agregada. Os sensores (nós) numa rede de estacionamento devem ser individualmente endereçáveis, de forma a que cada um possa determinar a localização de todos os lugares vazios. Esta aplicação mostra que pode ser interessante difundir uma mensagem para todos os nós na rede. Por outro lado, se pretendemos saber a temperatura de uma determinada divisão de um edifício, então o endereçamento talvez não seja tão importante. Qualquer nó numa dada região possui capacidade de resposta. A capacidade da rede de sensores para agregar a informação recolhida pode reduzir bastante o número de mensagens transmitidas pela rede.

O objectivo de uma rede *ad hoc* de sensores geralmente varia de acordo com a sua aplicação. A seguir apresentam-se as tarefas mais comuns para muitas redes:

- ✓ Determinar o valor de um determinado parâmetro numa determinada localização.
- ✓ Detectar a ocorrência de eventos de interesse e estimar os parâmetros de um ou múltiplos eventos.
- ✓ Classificar um determinado objecto.
- ✓ Determinar a localização de um objecto.

Em cada uma destas quatro tarefas, um importante requisito da rede de sensores é que a informação pretendida seja processada pelos utilizadores correctos. Em alguns casos existem requisitos temporais estritos na comunicação. Por exemplo, na detecção de um intruso numa rede de vigilância, a rede deve ser imediatamente comunicada à polícia para que esta tome as medidas necessárias. Na Tabela 2.3 estão descritos os requisitos que necessitam de ser cumpridos numa rede *ad hoc* de sensores.

Tabela 2.3 – Requisitos a serem cumpridos pelas redes *ad hoc* de sensores [10].

REQUISITO	DESCRIÇÃO
Elevado número de sensores (na maioria estáticos)	Para além da colocação de sensores na superfície dos oceanos ou do uso de sensores móveis, não manuseáveis ou robóticos em operações militares, a maioria dos nós numa rede de sensores inteligente são estáticos.
Baixo gasto de energia	Como na maioria das aplicações os sensores são colocados numa área distante, a actuação sobre o nó pode não ser possível. Neste caso, o tempo de vida de um nó pode ser determinado pela duração da bateria, requerendo assim a minimização do gasto de energia.
Auto-organização da rede	Dado o elevado número de nós e a sua potencial colocação em ambientes hostis, é essencial que a rede seja capaz de se auto-organizar, uma vez que a configuração manual não é uma opção. Para além disso, os nós podem falhar (seja por falta de energia ou pela sua destruição física) e novos nós podem-se juntar à rede. Desta forma, a rede deve ser capaz de periodicamente reconfigurar-se a si própria para que continue a funcionar correctamente. Nós individuais podem desligar-se do resto da rede, no entanto, um elevado nível de conectividade deve ser mantido.
Processamento de sinal de uma forma colaborativa	De forma a melhorar a performance de detecção/apreciação é muitas vezes útil unir a informação proveniente de múltiplos sensores. Esta união de informação requere a transmissão de informação e controlo de mensagens, podendo adicionar limitações à arquitectura da rede.
Capacidade de consulta	Um utilizador pode querer consultar um nó individual ou um grupo de nós

	para saber a informação recolhida na região. Dependendo da quantidade de união de informação realizada, pode não ser viável transmitir uma grande quantidade de informação através da rede. Em vez disso, vários nós locais sincronizados recolhem a informação numa determinada área e criam relatórios acerca do estado dela. Uma consulta pode ser depois direccionada para o nó sincronizado mais próximo do local pretendido.

Com o aparecimento de antenas de baixo custo e reduzido alcance, juntamente com os avanços que se estão a verificar nas redes sem fios, é esperado que as redes *ad hoc* de sensores venham a ser comummente utilizadas. Neste tipo de redes, cada nó pode ser equipado com uma variedade de sensores, tais como acústicos, sísmicos, infra-vermelhos, de videovigilância, etc.. Esses nós podem ser organizados em grupos, de forma a que a ocorrência de um evento local possa ser detectada pela maioria, senão por todos, os nós num grupo. Cada nó deve possuir capacidade de processamento suficiente para tomar uma decisão e deverá ser capaz de difundir essa decisão para outros nós no seu grupo. Nesta solução, um nó pode actuar como o líder do grupo e pode também possuir um maior alcance, recorrendo a protocolos tais como o IEEE 802.11 ou o Bluetooth.

3. O SISTEMA COMPUTACIONAL BSM – REQUISITOS E ASPECTOS OPERACIONAIS

Como já foi referido anteriormente, o BSM encontra-se inserido noutro sistema computacional com outras funcionalidades: o ANEE. No entanto, embora tenha existido uma participação activa no desenvolvimento de todos os componentes do ANEE, o BSM destacou-se em relação aos restantes.

3.1. REQUISITOS FUNCIONAIS

Como qualquer sistema computacional, o BSM possui requisitos funcionais. De uma forma geral, eles especificam quais os serviços que o sistema deverá fornecer.

Desta forma, os requisitos funcionais do BSM são:

- Obter informação sensorial do edifício através de dispositivos de recolha de dados espalhados pelas várias divisões de forma a obter um conhecimento global das constantes físicas nele verificadas;
- Encaminhar a informação sensorial desde os dispositivos de recolha de dados até aos dispositivos móveis através de comunicação via *ad hoc*;
- Permitir a comunicação entre os dispositivos móveis de forma a poderem trocar entre si os mais variados tipos de informação;
- Fazer a informação sensorial chegar até ao sistema central do edifício e aí armazená-los e processá-los de uma forma cuidada e, preferencialmente, direccionada a casos específicos que possam ajudar a melhorar o estado do edifício;
- Garantir a escalabilidade das classes responsáveis pela comunicação no sistema computacional, de forma garantir o dinamismo deste. Assim, caso seja necessário adicionar qualquer outro componente ao sistema inicialmente previsto, não é necessário realizar alterações de fundo às classes que servem de base ao seu funcionamento;
- Distinguir de forma correcta qual o dispositivo de recolha de dados que está a enviar informação sensorial para um dispositivo móvel para que quando essa informação seja processada, o seja de forma mais precisa possível;

- Tornar a comunicação entre os dispositivos móveis através de redes *ad hoc* é transparente para o utilizador e que oferece uma interface de uso simples;
- Armazenar toda a informação recolhida no edifício para que depois possa ser utilizada, quer em tempo real quer acedendo ao seu histórico.

3.2. BUILDING MANAGEMENT SYSTEM

Para que os dados possam ser processados é necessária a existência de um componente com uma considerável capacidade de inteligência e processamento de forma a que eles possam ser utilizados para a finalidade pretendida.

Um componente com estas características é denominado de Building Management System e esta secção será dedicada à abordagem das principais características dele quando inserido num sistema como o BSM.

Os edifícios necessitam de um "cérebro" de forma a controlar de forma inteligente os muitos sistemas e inúmeros focos de informação que podem gerar.

Um BMS fornece uma vasta gama de possibilidades que permitem ajudar os sistemas de um edifício, aumentando o seu nível de conforto e segurança e poupando recursos tais como a electricidade, o aquecimento, a água e outros mais.

Desta forma, as principais vantagens de um BMS são:

- Redução dos custos de manutenção e execução (aquecimento, iluminação, electricidade, etc.);
- Aplicação de uma política de sustentabilidade ambiental;
- Aumento do nível de conforto; e
- Aumento dos níveis de segunrança e controlo.

UM BMS é responsável por determinadas tarefas dentro do edifício. As principais funções que um BMS executa num edifício são sucintamente descritas de seguida.

Interface Homem/Máquina

Um BMS deve colocar à disposição uma interface que seja de fácil utilização tanto para utilizadores experientes como inexperientes para realizar a interacção com os sistemas que o compõem.

Segurança do Sistema

O sistema deve estar preparado para permitir apenas a pessoal autorizado aceder à suas funcionalidades, bem como definir que tarefas cada tipo de utilizador está autorizado a realizar.

Separação da Informação

O fluxo de informação que circula por todo o BMS deve estar separado por categorias bem definidas de forma a não existirem falhas na apresentação dos resultados.

Gestão de Alarmes

A ocorrência de situações fora do normal deve ser transmitida pelo próprio sistema aos responsáveis pela sua gestão. Este informe deve ter em conta factores como a importância da ocorrência ou a urgência da sua resolução.

Actuação no Edifício

O BMS deve disponibilizar formas de poder actuar no edifício de forma rápida e eficiente, através de comandos de simples execução.

Registo de Eventos

Todos os eventos relevantes que ocorram no edifício, tais como o disparo de alarmes ou alterações no edifício provocadas pelo operador, devem ser armazenados .

Relatórios

O BMS deve ter capacidade de realizar estatísticas com base tanto em dados actuais como históricos e de os apresentar ao utilizador de uma forma concisa e clara.

Exportação de Dados

O BMS deve estar preparado para enviar os seus dados para outrs sistemas computacionais de uma forma simples e rápida para que entidades exteriores possam ter conhecimento do estado do edifício.

3.3. MÓDULOS DO BSM

De forma a exercer as funções para que foi projectado, o BSM faz uso do seu próprio BMS. Neste estão inseridos vários módulos responsáveis então pela gestão e recolha e toda a informação relevante que exista no edifício.

Como se pode observar pela Figura 3.1, quatro sistemas computacionais fundamentais constituem o sistema BSM.

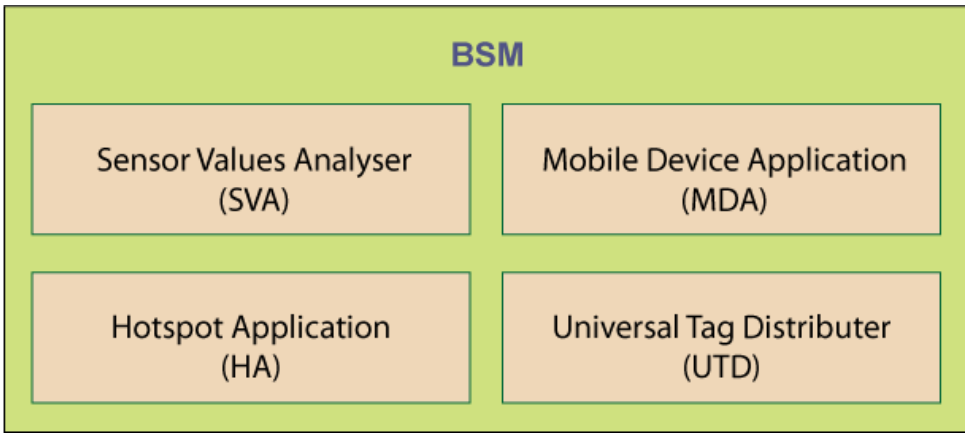

Figura 3.1 – Sistemas computacionais que compõem o BSM.

A seguir apresenta-se uma breve descrição de cada um dos sistemas apresentados:

- Sensor Values Analyser (SVA): aplicação responsável pelo armazenamento de todos os dados relativos ao edifício, bem como por todas as decisões a tomar relativamente à actuação nele.
- Mobile Device Application (MDA): aplicação destinada aos dispositivos móveis que circulam pelo edifício e que trocam informação entre si através das redes *ad hoc* por si criadas.
- Hotspot Application (HA): aplicação reservada aos dispositivos de recolha de dados (Hotspots) espalhados pelas várias divisões do edifício que permite que estes comuniquem com os dispositovos móveis.
- Universal Tag Distributer (UTD): aplicação localizada num servidor na Internet que armazena as informações gerais acerca de todos os SVAs existentes no mundo.

3.4. COMUNICAÇÃO

Cada edifício estará equipado com um SVA, que efectuará a gestão de todos os dados recolhidos pelos Hotspots por ele espalhados e que são transmitidos pelos dispositivos móveis através do MDA. O SVA consiste numa aplicação de software localizada no interior do edifício com capacidades de comunicação via *ad hoc* e via Internet. A comunicação via *ad hoc* é utilizada para a troca de dados entre o SVA e os dispositivos móveis, enquanto que a comunicação via Internet é necessária para situações de validação, envolvendo o UTD, que serão explicadas mais à frente. A disposição destes componentes e a forma como eles comunicam encontra-se representada na Figura 3.2.

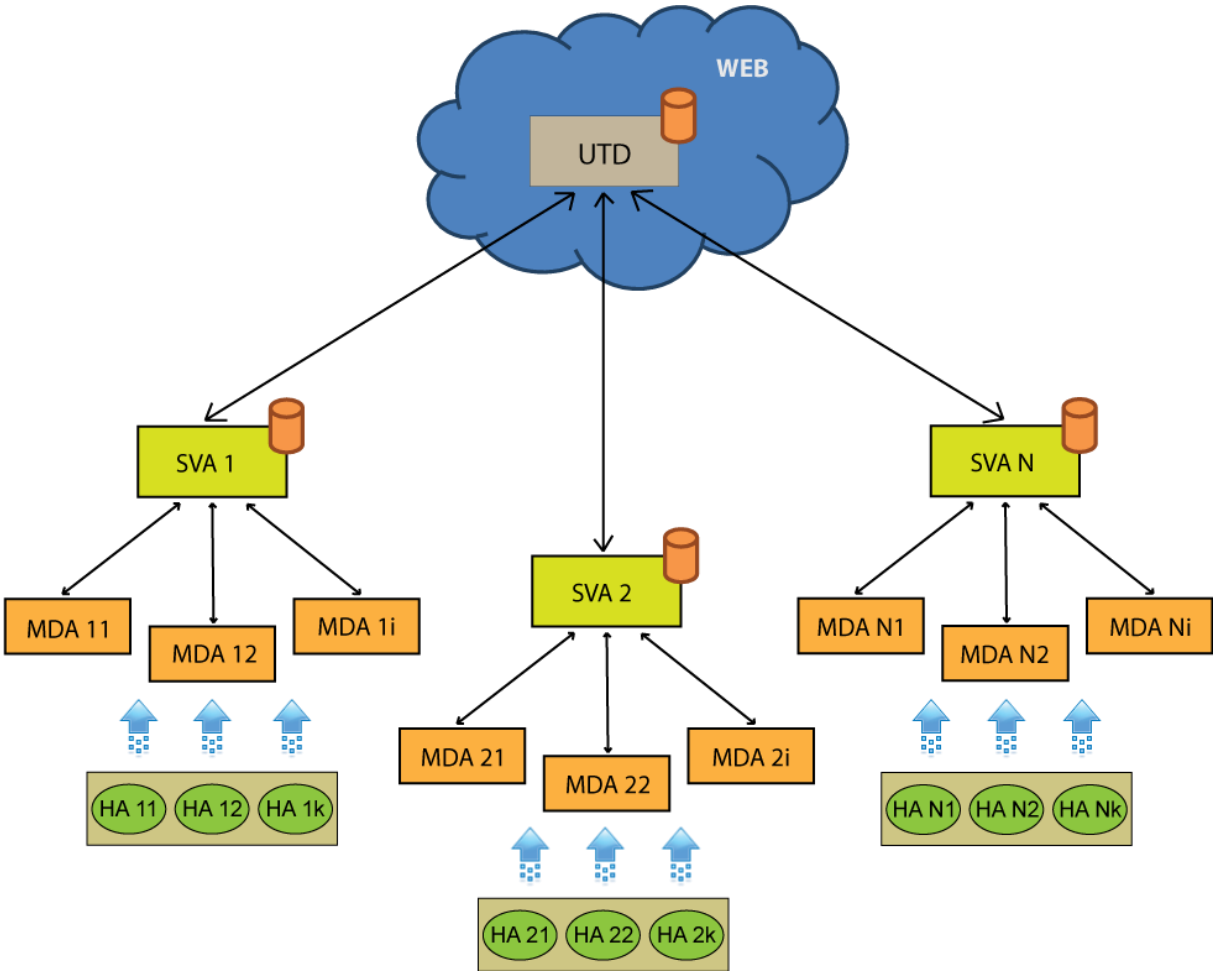

Figura 3.2 – Esquema geral de comunicação do BSM.

Apesar de o SVA suportar estes dois tipos de comunicação, a via *ad hoc* será a mais frequentemente utilizada, sendo a via Internet usada apenas em casos pontuais. Desta forma, este componente do sistema necessita de ter implementado, tal como MDA, um protocolo de comunicação *ad hoc*, embora com pequenas variâncias em relação a este.

3.5. GRUPOS

3.5.1. CONCEITO DE GRUPO

Anteriormente nesta dissertação foi abordado de uma forma muito superficial o conceito de grupo numa rede *ad hoc*. Esta noção de grupo foi criada principalmente por duas razões: em primeiro lugar, por uma questão de segurança/privacidade e, em segundo lugar, por uma questão de escalabilidade.

Em relação à problemática da segurança/privacidade, esta questão relaciona-se principalmente com duas situações: no caso em que um dispositivo móvel comunica com outro(s) na rede por iniciativa própria e no caso em que um dispositivo móvel visualiza no mapa do edifício outros dispositivos móveis. Quando um dispositivo móvel toma a iniciativa de comunicar com outro(s) na rede, ele está interessado em que apenas um grupo restrito de dispositivos tenha possibilidade de ter acesso a essa informação, ou seja, é pretendido que apenas os dispositivos que compartilhem grupos com ele sejam capazes de aceder a esses dados. O mesmo acontece com a posição que cada dispositivo móvel ocupa no edifício. A posição de um determinado dispositivo móvel no edifício só pode ser acessível por aqueles que compartilhem grupos com ele. Desta forma, ao ser filtrada a informação distribuída na rede, de forma a que esta só seja acessível por determinados dispositivos móveis, consegue-se manter a privacidade nela.

Quanto à questão da escalabilidade, esta está relacionada com o facto de que ao se separarem os dispositivos móveis em grupos, está-se a reduzir a complexidade na rede do ponto de vista do utilizador, tornando a comunicação dentro da rede muito mais cómoda. Desta forma, é possível filtrar os dispositivos móveis que pertencem a um determinado grupo, para depois escolher um deles para comunicar, ou mesmo partilhar uma informação com todo o grupo em questão.

Uma questão importante a referir aqui tem a ver com o facto de esta comunicação entre dispositivos móveis com grupos em comum, não estar limitada ao espaço físico do edifífio. Isto é, dois dispositivos móveis que partilhem um grupo pertencente a um determinado edifício podem comunicar livremente fora dele, desde que se encontrem a uma distância através da qual a antena Wi-Fi de um deles esteja ao alcance da do outro. Esta é uma das vantagens introduzidas pelo uso de redes *ad hoc* pois estejam onde estiverem, dois ou mais mais dispositivos móveis podem comunicar entre si desde que se encontrem ao alcance uns dos outros.

3.5.2. CRIAÇÃO DE GRUPOS

A criação dos grupos num edifício é da responsabilidade do seu SVA, no entanto, existem algumas regras que necessitam de ser respeitadas no processo de criação de um grupo. Cada grupo ficará sempre relacionado com o SVA responsável pela sua criação. Este facto, porém, não invalida que dois ou mais dispositivos móveis pertencentes a um grupo de um edifício possam comunicar entre si fora do espaço físico deste, como já foi referido.

Este facto gera já aqui uma situação que necessita de ser tratada – a possibilidade de existência de vários grupos iguais em edifícios diferentes. Para solucionar este problema, foi adicionado ao sistema um novo componente que contém todos os grupos existentes a nível mundial, ao qual foi atribuído o nome de Universal Tag Distributer (UTD). Este componente, basicamente é um webservice localizado num servidor na Internet que, no processo de criação de um grupo num SVA, gera um código único no mundo

(GUID – Group Unique Identifier) que será o identificador desse grupo a nível mundial. Desta forma, é possível existirem dois grupos com nomes iguais, mas que ao possuírem GUID's diferentes, é eliminada a possibilidade de qualquer confusão entre eles por parte do sistema.

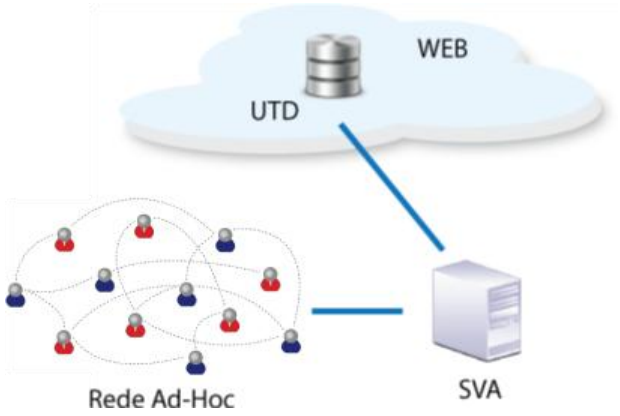

Figura 3.3 – Localização do UTD na visão geral do sistema BSM.

3.5.3. Partilha de Grupos

Depois de criado, um grupo necessita de ser partilhado pelos dispositivos móveis presentes no edifício. Para tal, o SVA encontra-se constantemente a enviar a todos os dispositivos móveis no edifício os vários grupos que contém. É depois da responsabilidade de cada dispositivo móvel que recebe o convite para integrar um grupo aceitá-lo ou não. Após a integração num grupo, um dispositivo móvel encontra-se capacitado para comunicar com todos os outros dispositivos móveis que também perteçam a esse grupo, tanto dentro como fora do edifício.

A partilha de grupos por parte do SVA é realizada via *ad hoc*, o que significa que nem todos os locais do edifício podem estar ao alcance deste convite. No entanto, como o SVA foi pensado como um elemento estático no edifício, este pode estar equipado com uma antena Wi-Fi com um alcance superior ao dos dispositivos móveis, podendo assim abranger um maior espaço físico no edifífio. Ainda assim, é possível que em alguns locais deste não seja possível a comunicação entre os dispositivos móveis e o SVA devido a interferências que sempre existem causadas, por exemplo, por obstáculos físicos, mas trata-se apenas de locais pontuais no espaço total do edifício.

3.6. O Mapa do Edifício

Para se poder fazer uma gestão correcta de cada espaço contido num edifício, é necessário um conhecimento a fundo de todos os locais por que este é constituído. Ora, a melhor forma de definir um

edifício quanto ao seu espaço físico é através do seu mapa. Com um mapa consegue-se criar um relacionamento entre os valores obtidos por um determinado sensor no edifício e a divisão em que estes valores foram lidos. Já para não falar que do ponto de vista do utilizador, poder visualizar num mapa a informação relativa a um edifício constitui uma opção muito mais elegante que apenas visualizar por exemplo um relatório apenas em texto.

Uma outra aplicação para o mapa de um edifício consiste na possibilidade de os dispositivos móveis poderem visualizar nele a posição que outros com grupos em comum ocupam. Isto obriga a que cada dispositivo móvel tenha em seu poder o mapa do edifício. Assim, para além, de distribuir os seus grupos, o SVA está encarregue também da partilha do mapa do edifício a que pertence aos dispositivos móveis que o ocupam.

3.6.1. Estrutura do Mapa

Uma vez que se pretende criar uma relação entre os dados recebidos pelos sensores espalhados pelo edifício e as suas divisões, o mapa do edifício necessita de respeitar uma determinada estrutura. Essa estrutura tem que ser ao mesmo tempo escalável e dinâmica. Por um lado, necessita de ser escalável pois um edifício pode conter vários pisos, cada piso pode conter várias divisões e, por fim, cada divisão é definida por uma coordenada relativa ao espaço ocupado pelo edifício. Por outro lado, a estrutura do mapa necessita também de ser dinâmica pois exige-se que ela seja capaz de se adaptar a qualquer tipo de edifício, quer na sua forma, quer na sua constituição.

Uma opção, talvez a mais simples, fosse utilizar uma planta do edifício já existente e aplicá-la directamente no SVA do edifício. No entanto, esta solução não se tornava aplicável pois, apesar de se possuir uma ou mais imagens da constituição do edifício, seria impossível para o SVA relacionar essas imagens com as divisões por que o edifício é constituído. Esta situação levou à conclusão que seria necessário implementar uma estrutura própria para o edifício com base naquilo que é realmente necessário para que o sistema realize as tarefas pretendidas de forma correcta. Essa estrutura deveria dar origem a uma ou mais imagens do edifício para que a informação recolhida pelos seus sensores pudesse ser traduzida de uma forma clara para o utilizador.

3.6.2. Criação do Mapa

Como foi referido, foi necessária a criação de uma estrutura própria para o edifício que pudesse servir de base para armazenar os dados recolhidos pelos seus sensores. Foi também referido que o mapa do edifício não podia consistir na utilização de uma planta já existente do edifício pois ela não iria representar a estrutura pretendida pelo sistema. No entanto, isso não invalida que ela sirva de base para a criação do

mapa pretendido para o sistema. Tornou-se então necessário desenvolver um suporte à criação de mapas que consigam ser reconhecidos pelo nosso sistema, de forma a obter dele o comportamento desejado.

Desta forma, foi tomada a decisão de implementar uma aplicação no SVA, denominada *Map Designer* (MD), que permita ao seu administrador criar um mapa para o edifício. Nesta aplicação o administrador pode definir todas as características físicas necessárias ter em conta no edifício para que o SVA consiga fazer um mapeamento correcto dos dados recolhidos pelos seus sensores. Estas características incluem, por exemplo, as dimensões do edifício e o número de pisos e divisões que contém. Estes dados inseridos pelo administrador do SVA darão depois origem várias imagens para o mapa do edifício (uma para cada piso) e a uma estrutura em formato XML, ambas a ser armazenadas na sua base de dados.

3.6.3. PARTILHA DO MAPA

Tal como para os grupos, o SVA necessita também de partilhar o mapa do seu edifício com os dispositivos móveis que o ocupam. Esta partilha processa-se da mesma forma que para a partilha dos grupos, ou seja, através da comunicação via *ad hoc* entre o SVA e os dispositivos móveis. O SVA vai então encontrar-se constantemente a enviar o seu mapa através da rede *ad hoc* para os dispositivos móveis presentes no edifício, podendo estes depois optar por aceitá-lo ou não.

Uma vez que o processo de transferência de informação utilizada na partilha de mapas é o mesmo que o utilizado para a partilha de grupos, as questões levantadas neste último caso também se verificam na partilha de mapas. Poderão existir na mesma locais pontuais no edifício onde a comunicação entre os dispositivos móveis e o SVA não será possível devido a limitações no alcance da antena Wi-Fi dos componentes envolvidos ou devido a obstáculos de natureza física presentes no edifício. No entanto, como já foi referido, estas pequenas limitações não colocam em causa o funcionamento correcto e desejado do sistema.

3.7. CONTROLO SENSORIAL

A missão principal do SVA consiste em actuar no edifício de forma a que se alcance um aumento significativo ao nível da eficiência energética nele verificada. Essa actuação tem de ser baseada em dados fidedignos verificados no edifício, dados esses obtidos através de controlo sensorial em todo o espaço que este abrange. Desta forma, se o SVA possuir estes dados que o auxiliem a tomar decisões dentro do edifício, consegue-se actuar nele da forma mais indicada.

O controlo sensorial implementado neste sistema aproxima-se bastante ao discutido no capítulo anterior quando foi abordado o tema das redes *ad hoc* de sensores. Um ponto porém diferencia as duas tecnologias:

no controlo sensorial desenvolvido, os sensores não comunicam entre si. No entanto, para o desenvolvimento dos sensores foram utilizados dispositivos com uma capacidade de processamento e de inteligência não muito elevada, limitando o sistema em alguns níveis. Ainda assim, este sistema desenvolvido tem também como objectivo mostrar que é possível implementá-lo, podendo mais tarde serem-lhe aplicados inúmeros melhoramentos para que o seu funcionamento se torne cada vez mais aproximado ao pretendido.

No controlo sensorial desenvolvido, cada divisão do edifício encontra-se equipada com um dispositivo, denominado por Hotspot, que pode possuir vários tipos de sensores. Esse dispositivo necessita de possuir inteligência e capacidade de processamento suficientes para que consiga transmitir os dados por si recolhidos. Necessita também de estar equipado com uma antena Wi-Fi que lhe permita comunicar com outros dispositivos presentes numa rede *ad hoc*. Ele precisa assim de "correr" uma aplicação de baixo nível denominada de *Hotspot Application* (HA) que lhe permita realizar as tarefas de que está incumbido. A forma como os Hospots se encontram distribuídos pelo edifício está representada na Figura 3.4.

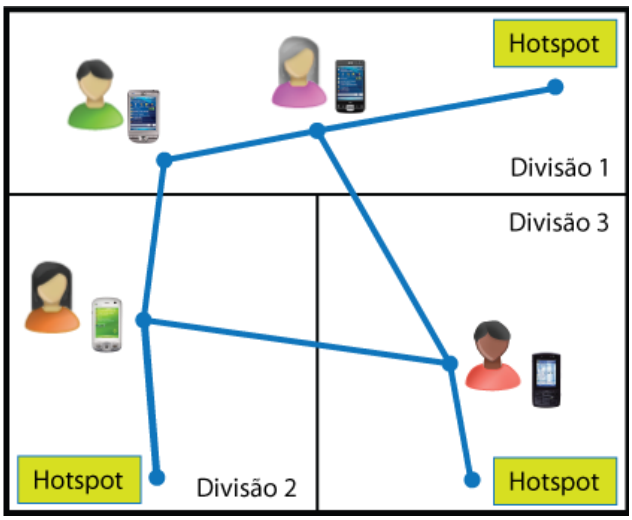

Figura 3.4 – Disposição dos Hotspots pelo edifício.

O SVA necessita de ter conhecimento de todos os sensores com que os Hotspots presentes no seu edifício estão equipados. Desta forma, uma das configurações necessárias a fazer também no SVA consiste na escolha de quais as constantes físicas que ele está capacitado para avaliar. Este conhecimento que o SVA necessita de ter tem a ver com o facto de ele precisar de saber quais as constantes físicas que terá que avaliar para depois poder actuar no edifício da forma mais correcta possível.

Depois de instalados nas várias divisões do edifício, os Hotspots irão recolher dados acerca das suas respectivas divisões. Cada sensor com que os Hotspots estão equipados irá realizar periodicamente leituras das respectivas constantes físicas para o qual está calibrado. Os Hotspots vão depois receber esses valores

lidos pelos seus sensores e difundi-los pela rede para que todos os dispositivos móveis que se encontrem ao seu alcance possam ter acesso a estes dados. Os dispositivos móveis ficam depois com a responsabilidade de enviar esses dados para o SVA para que este possa armazená-los e utilizá-los para actuar na divisão em causa.

3.8. ARMAZENAMENTO DOS DADOS

Antes de poder actuar sobre o edifício, o SVA necessita primeiro de possuir informação em que se possa basear para o fazer correctamente. Essa informação necessita de estar guardada em algum sítio. Para tal foi projectada uma base de dados para o SVA que desse suporte ao armazenamento de todos os dados que fossem relevantes para o processo.

Antes dos dados recolhidos pelos sensores, é necessário guardar algumas informações básicas, em primeiro lugar, as informações relativas ao próprio edifício. Essa informação inclui por exemplo o nome do edifício, o país a que pertence e o seu código postal. Outra informação que também é necessária armazenar na base de dados do SVA está relacionada com o mapa do edifício. Foi já referida a necessidade de o edifício ser caracterizado por uma estrutura que fosse acompanhada por uma visualização gráfica da constituição física dele. Desta forma, tanto o mapa do edifício como a sua estrutura necessitam também de ser armazenadas na base de dados do SVA.

Em segundo lugar e, porque o SVA foi desenvolvido no sentido que só poderia ser acessado por pessoal administrativo devidamente credenciado, é necessário ainda armazenar na base de dados do SVA as informações de todos os administradores que se encontram autorizados acedê-lo. Estas informações incluem obrigatoriamente os dados que são necessários para que alguém possa ter acesso a todas funcionalidades do SVA, ou seja, um nome de utilizador e uma palavra-passe, sendo depois acompanhados por outros dados relativos à identificação de cada administrador.

Em terceiro lugar é necessário armazenar os tipos de sensores para os quais o SVA está preparado para precessar os respectivos valores. Esta informação é necessária pois a forma de processar os dados varia consoante a constante física que se está a avaliar. Desta forma, o SVA necessita de estar preparado para processar pelo menos os dados para os quais se compromete a avaliar. Esses dados dependem dos vários tipos de sensores espalhados pelo edifício necessitando assim esta informação também de estar armazenada na base de dados do SVA.

Finalmente, em quarto lugar, é necessário armazenar os próprios dados lidos pelos sensores (Figura 3.5). Estes dados são armazenados "em bruto" na base de dados, não sofrendo qualquer tipo de processamento antes de serem guardados, de forma a obter estatísticas relativas ao estado do edifício mais precisas.

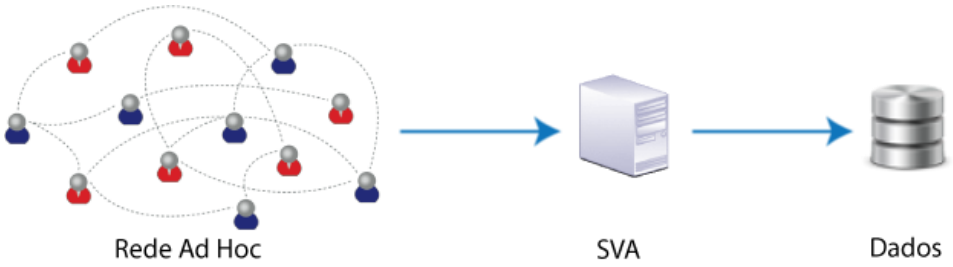

Figura 3.5 – Esquema de armazenamento dos dados por parte do SVA.

De referir que cada valor recebido pelo SVA lido por um sensor vem acompanhado com a respectiva divisão do edifício onde foi verificado, bem como com o tipo de sensor que realizou essa leitura. Facilmente se percebe que estes dados adicionais ao simples valor lido pelo sensor servem para o SVA saber onde esse valor foi verificado e o que ele significa. Com esta informação armazenada na base de dados, o SVA encontra-se assim com totais capacidades para poder tomar decisões acerca de como actuar sobre cada divisão do edifício.

3.9. ACTUAÇÃO NO EDIFÍCIO

Depois de ter os dados em sua posse, o SVA encontra-se preparado para actuar no edifício (Figura 3.6). Essa actuação pode variar de acordo com o tempo desde o qual os dados se encontram armazenados na base de dados do SVA.

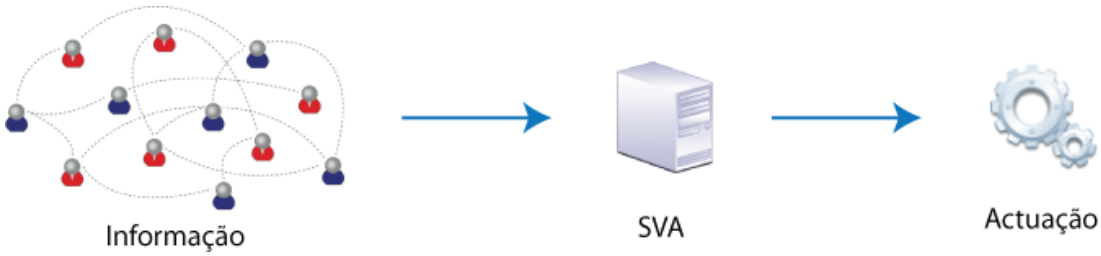

Figura 3.6 – Esquema de actuação no edifício por parte do SVA.

Desta forma, conseguem-se destacar dois tipos de actuação possível no edifício por parte do SVA: uma com base nos dados recolhidos num período temporal mais curto, ou seja, os dados recentemente inseridos na base de dados e outra com base em todos os dados presentes na base de dados. Facilmente se percebe que através do primeiro tipo de actuação esta será uma actuação instantânea enquanto que a segunda permitirá construir estatísticas acerca do comportamento habitual do edifício de forma a tentar

prever o que nele possa acontecer. De seguida apresenta-se mais em detalhe a descrição e o funcionamento de cada um dos tipos de actuação referidos.

3.9.1. ACTUAÇÃO INSTANTÂNEA

Como já foi referido o SVA pode apresentar dois comportamentos distintos no que se refere à sua actuação no edifício. Num desses comportamentos pretende-se que o SVA consiga actuar de uma forma instantânea. Para tal ser possível o SVA necessita de primeiro filtrar na sua base de dados a informação que mais se adeque aplicar para se cumprir tal tarefa. Ora, para se poder actuar em tempo real os dados mais relevantes são aqueles que foram recolhidos mais recentemente. O período de tempo escolhido para a filtragem dos dados pode variar, no entanto, pretende-se que não seja muito elevado pois o objectivo é actuar no edifício de forma a combater algo de menos correcto que esteja suceder neste no momento actual.

Este tipo de actuação está direccionado principalmente para situações em que o estado do edifício se encontre distante daquele em que habitualmente se encontra. Tais situações são caracterizadas pela constatação de que os valores recebidos por um determinado sensor se distanciem por uma margem considerável dos valores habitualmente por ele verificados. Exemplos dessas situações envolvem por exemplo casos de incêndios, fugas de gás ou qualquer outro tipo de situação de emergência. Em qualquer um destes casos pretende-se que o SVA tenha a capacidade de actuar sobre o edifício quer directamente, actuando fisicamente nele, quer indirectamente, alertando as entidades mais apropriadas para solucionar a situação verificada.

3.9.2. ACTUAÇÃO COM BASE NO REGISTO HISTÓRICO (PREVISÃO)

Uma segunda possibilidade de actuação no edifício pelo SVA é aquela baseada no registo histórico existente neste, ou seja, em toda a informação armazenada na sua base de dados. Desta forma, enquanto que no tipo de actuação descrito anteriormente em que apenas eram utilizados os dados recolhidos recentemente, neste a informação a aplicar abrange todos os valores contidos na base de dados do SVA. Estes dados são depois alvos de um processamento especializado de forma a obter um estudo estatístico acerca dos valores mais comuns verificados pelos sensores em cada divisão do edifício.

Com este tipo de actuação o objectivo pretendido é tentar prever o que possa acontecer no futuro no edifício. Uma aplicação deste tipo de actuação pode ser por exemplo fazer um estudo acerca dos valores verificados numa divisão do edifício, numa determinada altura no ano, a uma determinada hora do dia. De seguida, com base neste estudo constatar quais serão os prováveis valores que se irão verificar numa data

futura que corresponda ao período em causa e adoptar um comportamento de forma a antecipar essa situação.

Esta forma de actuação baseada no registo histórico constitui um método operacional bastante eficiente do ponto de vista energético, uma vez que o sistema não necessita de esperar pelo recebimento dos valores actuais para depois saber como actuar. Em vez disso, o SVA já tem conhecimento à partida daquilo que será mais provável de se verificar no edifício, podendo desde logo antecipar-se e actuar em conformidade com essa informação.

4. MODELO CONCEPTUAL DO BSM

4.1. INFRAESTRUTURA OPERACIONAL REQUERIDA

Para que o correcto funcionamento do sistema BSM seja obtido, existem certos requisitos que necessitam de ser cumpridos. Estes requisitos envolvem principalmente material de hardware e de software, mas também algumas características que esse material necessita de possuir.

Um dos requisitos é que as pessoas que habitam o edifício possuam um *Smartphone* ou PDA a correr Windows Mobile e que estejam equipados com uma antena Wi-Fi para poderem correr a aplicação MDA. É necessária também a existência de um computador no edifício a correr a aplicação SVA que tenha capacidade de ligação sem fios para se poder ligar à rede *ad hoc* e também à ligação por cabo para se poder ligar à Internet. Finalmente, é necessária também a existência de Hotspots equipados com sensores espalhados pelas várias divisões do edifício, com capacidade de comunicação via Wi-Fi, a correr a aplicação HA.

Partindo do princípio de que todos os requisitos mencionados sejam cumpridos, estão reunidas todas as condições para que o sistema BSM possa executar as suas funções sem problemas.

4.2. PROCESSO DE MODELAÇÃO

A modelação do sistema BSM foi realizada através da linguagem UML. A escolha desta linguagem de modelação tem a ver com o facto de permitir a criação de diagramas normalizados, facilitando a sua análise e compreensão pela maioria dos desenvolvedores. De uma forma geral, o principais objectivos do UML são: especificação, documentação, estruturação para sub-visualização e maior visualização lógica do desenvolvimento completo de um sistema de informação.

Para a modelação do sistema BSM foram utilizados três tipos de diagramas UML:

- Diagrama de casos de uso: documento narrativo que descreve as acções que um actor pode realizar num sistema computacional.
- Diagrama de classes: representação da estrutura e relações das classes que servem de modelo para objectos. É uma modelagem muito útil para o sistema, define todas as classes que o sistema necessita possuir e é a base para a construção dos diagramas de sequência.
- Diagrama de sequência: representa a sequência de processos (mais especificamente de mensagens passadas entre objectos) num programa de computador.

4.3. VISÃO FUNCIONAL

Nesta secção pretende-se apresentar as funcionalidades do sistema BSM. Desta forma, serão brevemente discutidas as acções que é possível efectuar em cada elemento que compõe o sistema BSM. Na Figura 4.1 mostra-se o diagrama de casos de uso do sistema BSM que servirá de base para as descrições mais pormenorizadas feitas de seguida.

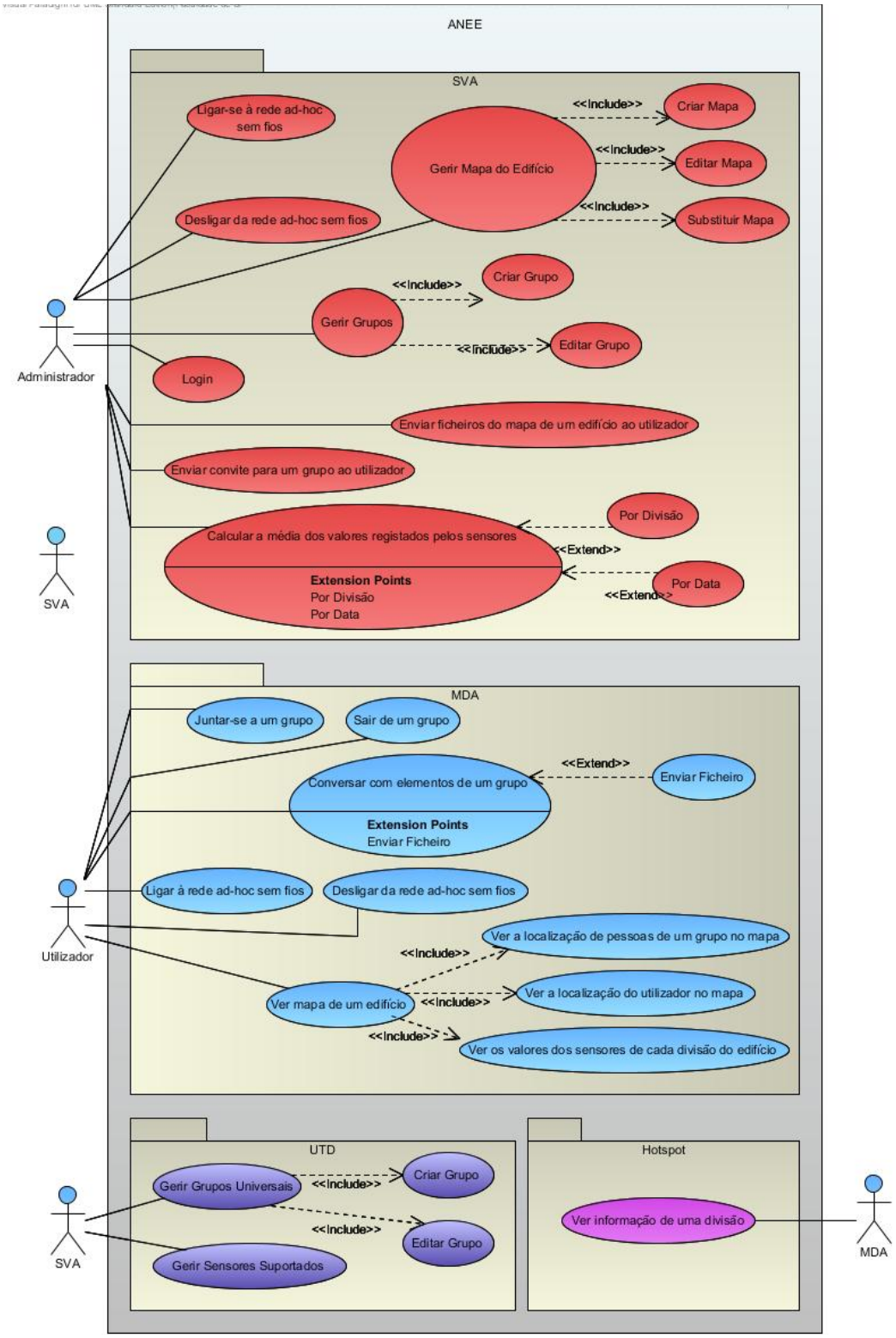

Figura 4.1 – Diagrama de casos de uso do sistema BSM.

4.3.1. MOBILE DEVICE APPLICATION (MDA)

No MDA o actor[2] representa cada pessoa que habita o edifício. Nesta aplicação, a principal funcionalidade consiste na possibilidade de o dispositivo móvel se poder ligar e desligar da rede *ad hoc*. Esta ligação ocorre da seguinte forma: caso já exista uma rede *ad hoc* criada e que esteja ao alcance do dispositivo móvel, ele liga-se a ela; caso não exista nenhuma rede *ad hoc* criada ao seu alcance, ele cria uma nova e liga-se a ela. De notar que o MDA a correr no dispositivo móvel efectua este processo de forma automática, não sendo necessário o utilizador efectuar alguma acção. A funcionalidade de desligar da rede ad hoc, por seu lado, envolve a acção por parte do utilizador, como é óbvio.

Outra funcionalidade oferecida pelo MDA ao seu utilizador é a capacidade para ele se juntar ou abandonar um grupo. Para se juntar a um grupo, primeiro o utilizador recebe um convite enviado pelo SVA do edifício, tomando depois ele a decisão se pretende ou não juntar-se a ele. Para abandonar um grupo, basta dirigir-se às definições do MDA e escolher de qual dos grupos em que ele se encontra inserido ele pretende sair.

Depois de pertencer a pelo menos um grupo do edifício, o utilizador pode comunicar com todos os elementos desse grupo que estejam ao seu alcance na rede *ad hoc*. Essa comunicação pode ser privada ou em grupo, na qual podem ser transmitidas não só mensagens de texto mas também ficheiros de qualquer tipo, tais como imagens, vídeos, ou músicas.

Finalmente, a última funcionalidade oferecida pelo MDA é a possibilidade de o utilizador poder ver um mapa de todo o edifício. Nesse mapa ele pode constatar qual a sua posição geográfica nele, bem como de todos os elementos que partilhem grupos com ele. Adicionalmente, ele pode ainda visualizar quais os valores verificados pelos sensores em cada divisão do edifício.

4.3.2. SENSOR VALUES ANALYZER (SVA)

O SVA será acessado apenas por um actor que será o seu administrador, apesar de algumas funcionalidades deste elemento do sistema BSM se poderem considerar que são manipuladas por um BMS, pois estão mais relacionadas com a gestão dos dados do edifício. Desta forma, como esta aplicação será acedida apenas por pessoal administrativo com privilégios especiais, uma das funcionalidades deste elemento é a execução de um *login*.

Tal como o MDA, o SVA também possui capacidades de comunicação via *ad hoc*, desta forma este elemento também possui a funcionalidade de se ligar à rede *ad hoc* e desligar-se dela, nos mesmos termos já definidos para o MDA.

[2] Denominação dada em modelação UML ao elemento que funciona como utilizador de uma aplicação.

Outra tarefa para que o SVA também está equipado é a gestão dos grupos do edifício. Desta forma, o administrador está habilitado também a criar novos grupos, bem como a editá-los. O processo de criação de grupos envolve ainda outro componente do sistema BSM, o UTD, que mais à frente já será explicado. Depois de possuir grupos, um edifício está assim capacitado para os distribuir pelos dispositivos móveis que o habitam. Para isso o administrador apenas necessita de definir o estado do grupo para este se tornar partilhável.

Ao ser instalado no edifício, o SVA necessita de ser definido um mapa para o edifício. Essa responsabilidade também está a cargo do administrador. Para tal, o SVA fornece ao administrador uma ferramenta (MD) que permite facilmente criar um mapa para o edifício que seja reconhecido por ela. Esta ferramenta permite ainda editar um mapa criado ou substituir o mapa actual por um novo. Após o SVA ter um mapa que defina correctamente o edifício, ele irá partilhá-lo com os dispositivos móveis presentes.

Finalmente, a última funcionalidade do SVA é analisar valores relativos a cada divisão do edifício, recebidos a partir dos dispositivos móveis. Essa funcionalidade não é realizada através de nenhuma acção por parte do administrador, mas sim de uma forma automática aquando da sua recepção. Desta forma, nesta funcionalidade o actor não é o administrador mas sim o BMS. Neste processo de análise dos valores verificados nas divisões do edifício, é calculada a sua média aritmética, podendo-se depois visualizar os resultados filtrando-os por divisão e por data.

4.3.3. HOTSPOTS

Os Hotspots, como já foi referido no capítulo anterior, constituem os elementos dispersos pelas várias divisões do edifício, equipados com sensores e com o HA e efectuam a recolha das constantes físicas nelas verificadas. Este componente do sistema BSM possui apenas uma funcionalidade que é a de transmitir aos dispositivos móveis a informação relativa à divisão onde se encontra instalada. Desta forma, o actor neste processo é o MDA que vai receber essa informação.

4.3.4. UNIVERSAL TAG DISTRIBUTER (UTD)

O UTD existe no sistema BSM por duas razões principais: primeiro porque é necessário existir um componente na Internet que possua a informação de todos os grupos exixtentes no mundo, para que não existam grupos exactamente iguais; e em segundo lugar, para se definirem todos os tipos de sensores que o sistema BSM está capacitado para suportar.

Em relação à questão dos grupos, sempre que o administrador do SVA pretende criar um grupo novo, o UTD recebe esse pedido e cria um novo grupo com um identificador único de forma a que não possam existir confusões entre grupos. O mesmo acontece quando se pretende editar um grupo já existente.

Quanto à questão dos sensores suportados pelo sistema BSM, eles necessitam de estar armazenados no UTD para o SVA poder escolher de entre eles quais aqueles que ele está capacitado para avaliar, ou seja, quais aqueles com que os seus Hotspots estão equipados.

4.4. VISÃO ARQUITECTURAL

No que diz respeito à arquitectura do sistema BSM, três componentes principais podem ser destacados: aplicação, *core* e API (*Application Programming Interface*).

A parte relativa à aplicação já foi abordada anteriormente. O *core* de um sistema consiste nas classes de software que o suportam e que permitem que ele realize todas as tarefas para as quais foi desenhado.

Quanto à API, consiste num conjunto de regras e especificações que os programas de software podem utilizar para comunicarem uns com os outros. Funciona como uma interface entre diferentes programas de software e facilita a sua interacção. Uma API pode ser desenvolvida para aplicações, bibliotecas, sistemas operativos, etc., como uma forma de definir as convenções utilizadas para os aceder aos seus recursos. Pode incluir especificações para rotinas, estruturas de dados, classes de objectos ou protocolos usados para comunicar entre o utilizador do programa e o desenvolvedor da API.

4.4.1. ARQUITECTURA DO BSM

A organização dos três componentes referidos anteriormente que compõem o BSM pode ser observada na Figura 4.2.

De uma forma geral, os componentes que se encontram na parte da aplicação fazem uso dos componentes localizados na parte do *core* de forma a poderem realizar as suas tarefas internas bem como para comunicarem com outros componentes exteriores.

Todos os componentes localizados nestes dois elementos (aplicação e *core*) fornecem serviços a entidades exteriores que pretendam fazer uso deles através da API. Esta fornece serviços tanto da parte aplicacional como da parte do *core*, serviços esses que podem ser extendidos caso uma entidade exterior assim o pretenda, podendo essa alteração obrigar ou não à realização de desenvolvimento da parte do sistema BSM.

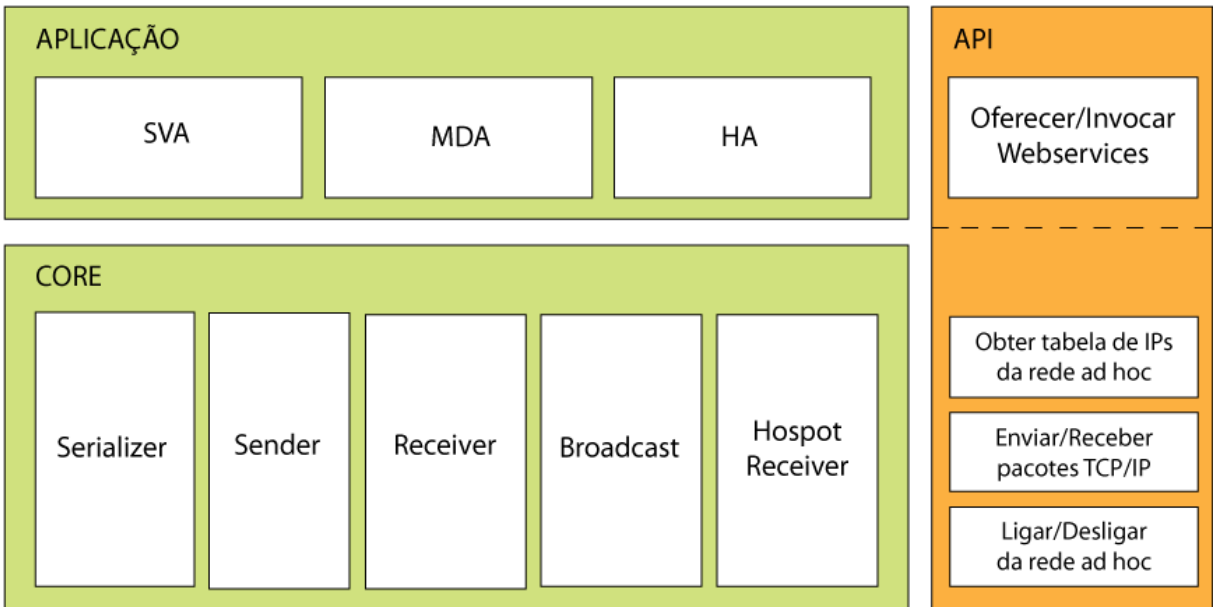

Figura 4.2 – Arquitectura do sistema BSM.

Em relação à parte de aplicação, existem três elementos, cada um correspondente a cada aplicação desenvolvida para o sistema BSM. Essas aplicações foram já brevemente descritas anteriormente e são o SVA, o MDA e o HA.

Quanto à parte do *core*, esta inclui todas as classes que dão suporte às aplicações. Elas realizam todos os processos que se pretende que sejam transparentes para o utilizador, mas de uma importância extrema para que o sistema BSM funcione correctamente. Tais classes serão explicadas de forma mais pormenorizadamente no próximo capítulo.

O último componente constitui uma API que se pretende que troque serviços com o exterior, de forma a poder expandir ao máximo o sistema BSM. Desta forma, o sistema BSM pode ser caracterizado por um sistema aberto que suporta e dá suporte a serviços externos.

4.5. MODELO DE DADOS

Nesta secção é apresentado o modelo de dados definido para a aplicação do SVA, uma vez que esta é a única que possui uma base de dados. Esse modelo de dados será apresentado na forma de um DER que representa a forma como as tabelas da base de dados do SVA se encontram dispostas e relacionadas. Esse diagrama escontra-se apresentado na Figura 4.3 e a descrição de cada tabela que o compõe será realizada com base nesse diagrama.

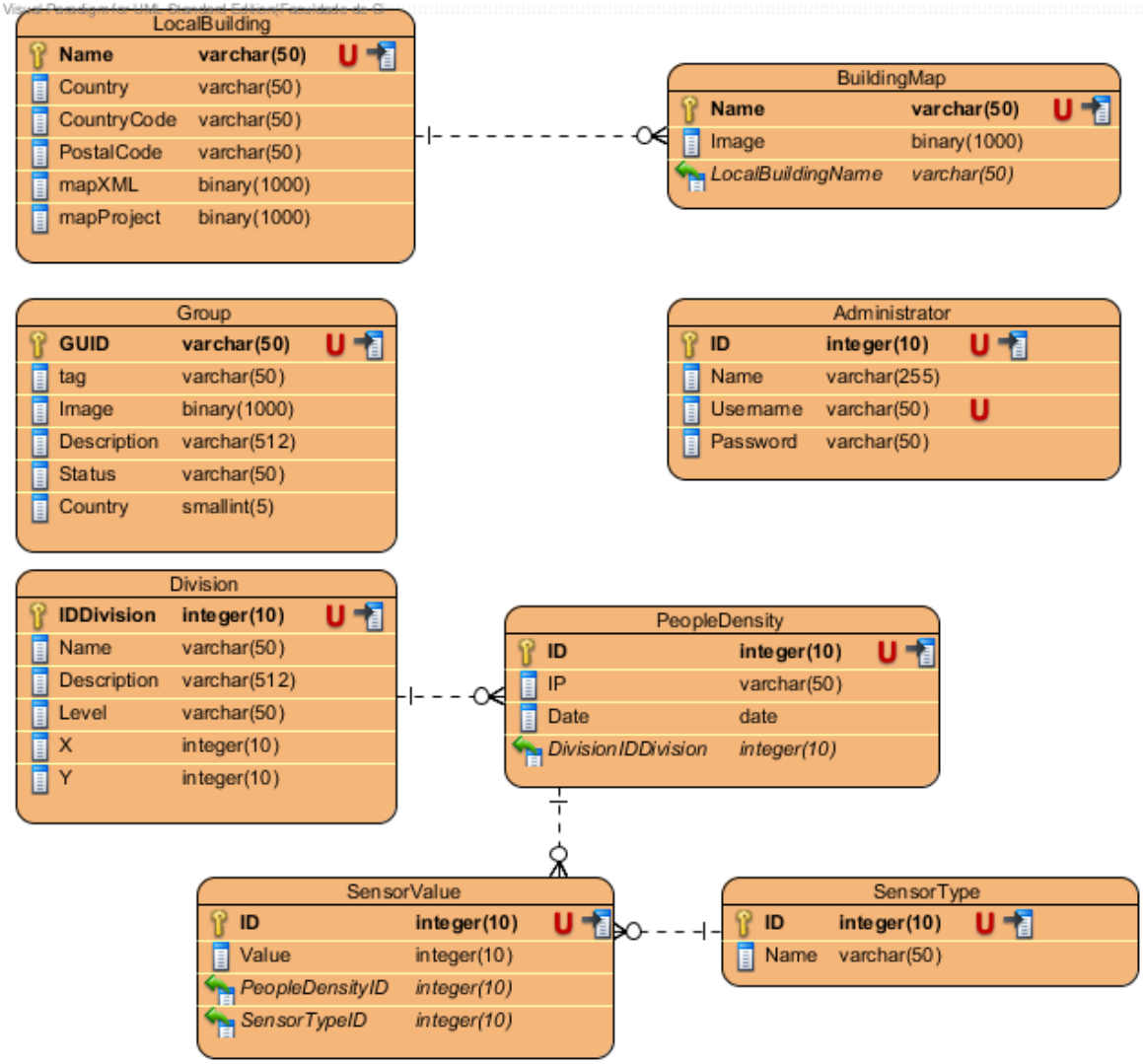

Figura 4.3 – Diagrama de entidades e relações do SVA.

A tabela *Administrator* armazena toda a informação necessária relativa ao(s) administrador(es) do SVA. A chave principal desta tabela é o ID e ela é composta ainda pelo nome do administrador, bem como pelo seu *username* e *password*.

Na tabela *Group* são armazenados todos os dados relativos aos grupos que o SVA contém. A sua chave principal é um GUID (Group Unique Identifier) que é gerado pelo UTD e onde estão armazenados ainda outros dados característicos de um grupo, como é o caso do seu nome (*tag*), imagem, descrição, estado (se se encontra a ser partilhado com os dispositivos móveis do edifício ou não) e país.

Existem depois duas tabelas que estão responsáveis pelo armazenamento da informação do edifício. A primeira denominada *LocalBuilding*, cuja chave principal é o nome do edifício (isto porque esta tabela só terá uma entrada, uma vez só existirá um SVA por edifício) está destinada a guardar dados do edifício tais como, para além do nome, o seu país, o código do seu país e o seu código postal. Esta tabela possui ainda

dois campos adicionais onde são armazenados dois ficheiros XML, um com o projecto do mapa criado na aplicação MDA (para esta aplicação poder realizar a edição do mapa actual) e outro semelhante a este, mas apenas com os dados relevantes para o SVA, ou seja, inclui a estrutura do edifício para que o SVA tenha informação acerca da composição física do seu edifício.

A segunda tabela denominada *BuildingMap* armazena os ficheiros de imagem de cada piso do edifício. A sua chave principal é o nome do piso (pois a aplicação MDA não permite a criação de dois pisos com o mesmo nome para um mesmo edifício)e os restantes campos são o ficheiro com a imagem do mapa do piso e uma chave estrangeira que relaciona esta tabela com a anterior.

As últimas quatro tabelas são talvez as mais importantes de todo o diagrama uma vez que são elas que estão incumbidas de armazenar a informação recolhida pelos Hotspots. A primeira dessas tabelas denominada *Division* armazena apenas os dados de cada divisão do edifício. A sua chave principal é um ID de divisão e os seus restantes elementos são o nome da divisão, a sua descrição, o piso do qual faz parte e a sua coordenada relativa ao edifício.

Outra tabela denominada *PeopleDensity* está incluída neste diagrama apesar não ser necessária para os objectivos do SVA. A razão para a existência dela no DER tem a ver com o facto de o SVA ter sido estruturado de forma a suportar não só o projecto desenvolvido nesta dissertação mas também o projecto desenvolvido pelo meu colega João Santos, cujo objectivo era calcular o número de pessoas que se encontra em cada divisão do edifício. A sua chave principal é um ID e os seus elementos são o IP de cada dispositivo móvel na rede *ad hoc*, a data em que a entrada foi criada e a divisão (chave estrangeira) na qual se encontrava o dispositivo móvel quando enviou essa informação ao SVA.

Finalmente, existe uma tabela encarregue de armazenar a informação relativa aos sensores espalhados pelo edifício e uma outra denominada *SensorType* que armazena apenas os tipos de sensores suportados pelo SVA. Desta forma, na tabela *SensorValue* a chave principal é um ID e aí é armazenado o valor verificado por cada sensor, qual o tipo de sensor a que corresponde esse valor (representado por uma chave estrangeira que relaciona esta tabela com a tabela *SensorType*) e qual a informação da divisão onde esse valor foi verificado (representado também por uma chave estrangeira que relaciona esta tabela com a tabela *PeopleDensity*).

5. IMPLEMENTAÇÃO DO BSM

5.1. TECNOLOGIAS E DISPOSITIVOS

No desenvolvimento do sistema BSM, foram utilizadas várias tecnologias que tornaram possível o seu correcto funcionamento. Essas tecnologias incluem material de software e material de hardware. Em relação ao primeiro, destacam-se as tecnologias de redes sem fios e os ambientes e linguagens de programação. Quanto ao hardware destaca-se a utilização de *Smartphones* com sistema operativo Windows Mobile, o Arduino e vários sensores de medição de constantes físicas.

5.1.1. *SMARTPHONES*

Os *Smartphones* constituem um elemento crucial no funcionamento do sistema BSM. Eles representam as pessoas que habitam o edifício, fazendo circular a informação relevante neste entre elas. Para além disso, ainda estão responsáveis pela entrega dessa informação proveniente dos dispositivos de recolha de dados ao SVA.

A aplicação de software MDA foi desenvolvida para o sistema operativo Windows Mobile, estando desta forma limitada a dispositivos que utilizem esta plataforma. No entanto, o sistema desenvolvido tem como objectivo alargar os horizontes e permitir que outros dispositivos com outros sistemas operativos possam também fazer uso da aplicação MDA.

No processo de desenvolvimento e teste da aplicação MDA foram utilizados quatro *Smartphones* disponibilizados pelo Departamento de Engenharia Electrotécnica (DEE) da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (FCT/UNL), mais precisamente o HTC HD Mini (Figura 5.1) com a versão 6.5.3 do sistema operativo Windows Mobile.

Figura 5.1 – HTC HD Mini, utilizado no desenvolvimento do MDA.

5.1.2. ARDUINO USB/BT COM MÓDULO WISHIELD

Para o desenvolvimento dos Hotspots era necessário recorrer a um dispositivo, preferencialmente de reduzidas dimensões, que pudesse facilmente ser transportado de uma divisão do edifício para outra, caso tal fosse necessário, de simples instalação e que tivesse capacidade para receber dados recolhidos por sensores e posteriormente difundi-los via Wi-Fi. A resposta para todos estes requisitos foi encontrada no Arduino. O Arduino consiste numa plataforma electrónica baseada em hardware e software relativamente simples de utilizar. Possui a capacidade de analisar o ambiente através da recepção de uma variedade de sensores, estando também preparado para actuar no mesmo ambiente que o rodeia caso tal seja necessário.

Existem vários tipos de Arduinos, divergindo estes na forma como comunicam com outros dispositivos. No processo de desenvolvimento do sistema BSM foram utilizados dois tipos de Arduinos: o Arduino Uno (Figura 5.2) e o Arduino BT (Figura 5.3). Em relação ao primeiro, este Arduino efectua a comunicação via USB, quanto ao segundo a comunicação é realizada via Bluetooth.

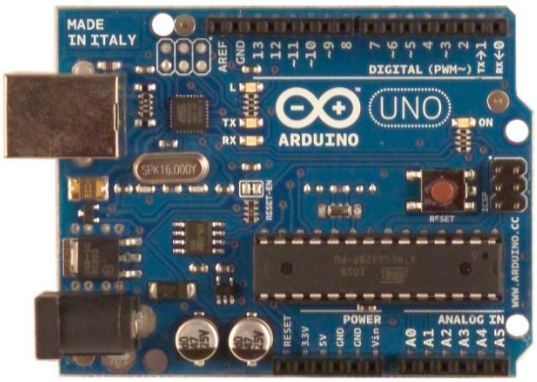

Figura 5.2 – Arduino Uno (USB).

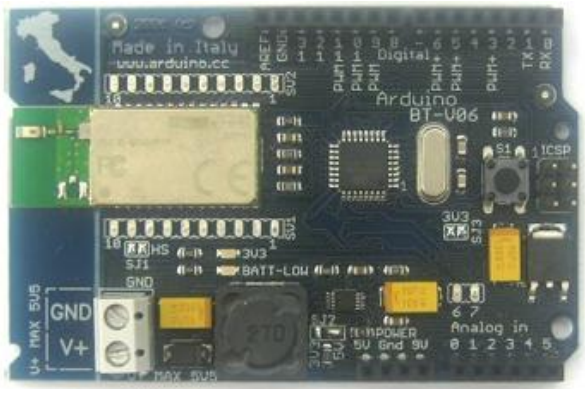

Figura 5.3 – Arduino BT (Bluetooth).

Ora, uma vez que um requisito obrigatório para o sistema a desenvolver era que o Arduino tivesse capacidade para comunicar via Wi-Fi e nenhum dos tipos de Arduinos referidos a possui, foi necessário incluir ainda um módulo adicional, o módulo WiShield. Este módulo pode ser instalado facilmente em qualquer tipo de Arduino, adicionando-lhe a capacidade de comunicar também via Wi-Fi. Desta forma, foram necessários incluir dois módulos Wi-Fi, um para cada Arduino referido anteriormente.

Todo este material foi obtido também com a preciosa colaboração do Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

5.1.3. SENSORES

Para finalizar o material de hardware utilizado, mas não menos importante, foi necessário recorrer a sensores que tivessem a capacidade de efectuar leituras de constantes físicas. Para tal, foi adquirido o sensor MQ135 (Figura 5.4), que possui a capacidade de medir várias de constantes físicas. Este sensor é instalado no Arduino através dos pinos de entrada deste, de forma a que os valores por ele verificados possam ser recebidos pelo Arduino.

Figura 5.4 – Sensor MQ135.

O MQ135 possui algumas características que fizeram com que a escolha de sensores a utilizar no desenvolvimento do sistema BSM recaísse sobre ele. Entre elas destacam-se a sua boa sensibilidade aos gases nocivos, um longo alcance, um elevado tempo de vida, um preço reduzido e uma simples instalação. Desta forma, este sensor é especialmente apropriado para detecção de poluição no ar, quer a nível doméstico, quer a nível industrial.

Este sensor foi instalado no Arduino de forma a poder comunicar com ele e enviar-lhe informação do ambiente que o rodeia. Na Figura 5.5 está apresentada a forma como o sensor MQ135 é instalado no Arduino BT.

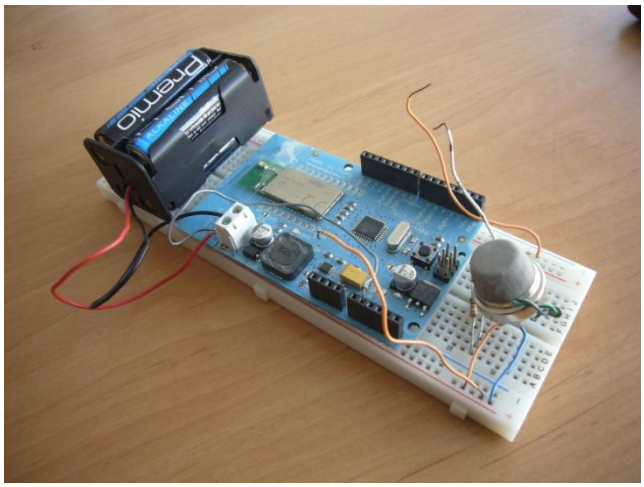

Figura 5.5 - Arduíno BT equipado com um sensor MQ135.

5.1.4. TECNOLOGIAS DE REDES SEM FIOS – IEEE 802.11 (WI-FI)

As tecnologias de redes sem fios constituem um ponto basilar no sistema BSM, mais precisamente a tecnologia IEEE 802.11, ou como é mais comummente conhecida, Wi-Fi. O protocolo desenvolvido quer para a aplicação do SVA, quer para a aplicação MDA necessita da camada já existente na tecnologia Wi-Fi para funcionar, isto é, sem esta tecnologia o sistema desenvolvido não seria possível de ser concebido.

Basicamente, toda a troca de pacotes existente na comunicação quer entre os dispositivos móveis e o SVA quer entre os dispositivos móveis entre si tem como base a tecnologia Wi-Fi. A própria criação da rede *ad hoc* é realizada com base neste protocolo. Em suma, o protocolo desenvolvido para a comunicação entre os diversos componentes do sistema encontra-se presente num nível acima do protocolo Wi-Fi, mas assente neste.

5.1.5. AMBIENTES E LINGUAGENS DE PROGRAMAÇÃO

No desenvolvimento de qualquer aplicação de software é necessária a utilização de linguagens de programação e, para a utilização destas, de ambientes de programação. Para o projecto desenvolvido recorreu-se a algumas linguagens e ambientes de programação consoante o objectivo pretendido em cada aplicação desenvolvida.

C# → Microsoft Visual Studio 2008

A linguagem de programação mais recorrente no desenvolvimento do sistema BSM foi o C# (C Sharp). Para a escolha desta linguagem foi determinante o facto de a maior parte dos protocolos pesquisados relacionados com a tecnologia Wi-Fi se encontrarem redigidos nesta linguagem, facilitando assim a junção destes com o protocolo a ser desenvolvido. Um outro factor para a escolha desta linguagem de programação reside nos recursos que ela proporciona a nível gráfico, possibilitando a criação de uma interface agradável para o utilizador, o que é sempre importante ter em conta.

Esta linguagem foi assim utilizada para as aplicações que necessitavam de ter implementado o protocolo de comunicação numa rede *ad hoc*, ou seja, a aplicação do SVA e o MDA. Para a aplicação do Arduino não foi necessária a implementação deste protocolo uma vez que este apenas realiza comunicação por difusão na rede, algo com que já vem preparado. No entanto, foi necessário o desenvolvimento de uma aplicação para este dispositivo mas não nesta linguagem de programação.

O ambiente de programação escolhido para o desenvolvimento de código na linguagem C# foi o mais óbvio, ou seja, o Microsoft Visual Studio 2008. Este ambiente de programação é o mais indicado para o

desenvolvimento de aplicações na linguagem apresentada. Adicionalmente, é também um ambiente que fornece um suporte considerável ao desenvolvimento de uma interface gráfica de forma simples.

Arduino Programming Language → Arduino Development Environment (Processing)

Como já foi referido, o Arduino não necessita de implementar o protocolo de comunicação em redes *ad hoc*, uma vez que este já vem por omissão com a capacidade de realizar difusão na rede ao qual se encontra ligado, cumprindo assim os requisitos para a tarefa para o qual está destinado no sistema BSM.

Ainda assim, o Arduino necessita de uma aplicação que ficará a ser executada constantemente, que implemente as regras dessa mesma execução, nomeadamente quais os sensores com que está equipado e, consequentemente, quais os valores que terá de difundir na rede. Esta informação é fundamental para o Arduino uma vez que ele realiza a leitura dos sensores através da leitura de pinos onde se encontram instalados esses mesmos sensores e que alteram a sua tensão de entrada. O Arduino necessita depois de saber de entre todos os pinos que possui quais aqueles que deverá ler o valor que está a ser verificado para posteriormente o partilhar com os dispositivos móveis na rede.

O Arduino possui um ambiente de programação próprio para placas de aquisição de dados, baseado na linguagem Processing. Desta forma, aqui não existiu sequer hipótese de escolha, tanto do ambiente de programação como da linguagem de programação.

5.2. DIAGRAMAS DE CLASSES

A implementação efectuada vai ser apresentada através de diagramas de classes, constituídos por três elementos principais: interface, controlo e entidades. Vão ser apresentados os diagramas de classes das duas principais aplicações desenvolvidas, ou seja, do MDA e do SVA. Na Figura 5.6 está representado o diagrama de classes do MDA que vai servir de base para a explicação das classes mais importantes utilizadas na implementação desta aplicação. O mesmo acontece para o diagrama de classes do SVA apresentado na Figura 5.7.

O MDA e o SVA partilham alguns serviços, principalmente na parte relativa à comunicação. As classes responsáveis pelo envio e recepção de informação (*Sender* e *Receiver*), pela difusão de dados (*Broadcast*) e pela conversão da informação de forma a que esta possa circular pela rede (*Serializer*) são o caso mais evidente. Estes componentes encontram-se inseridos nos diagramas de classes na área reservada aos serviços de Controlo e, pela sua importância no funcionamento global do BSM, serão descritas mais detalhadamente na próxima secção. Também as classes pertencentes à área de Interface serão descritas. As Entidades, uma vez que não possuem métodos, são auto-explicativas através do diagrama de classes.

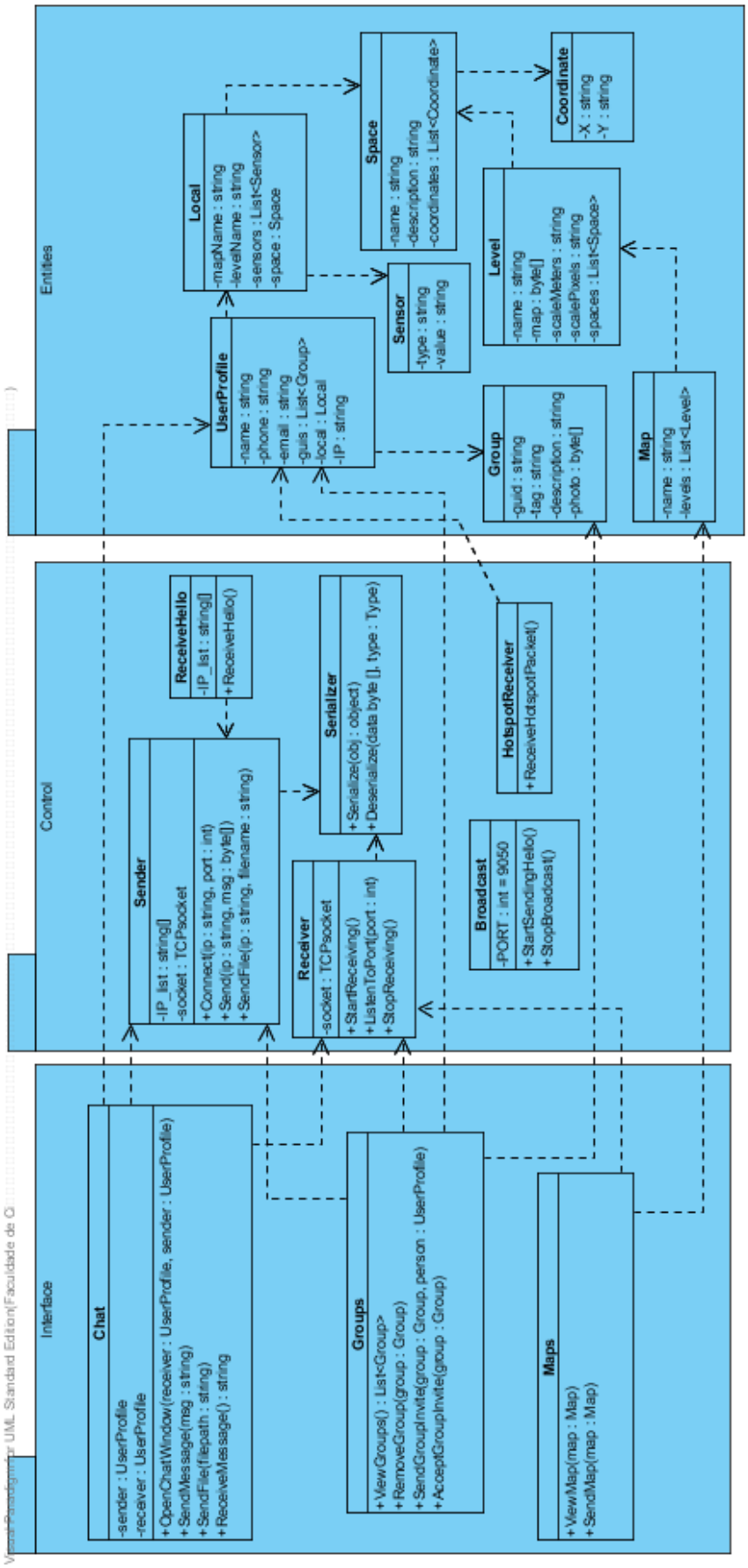

Figura 5.6 – Diagrama de classes do MDA.

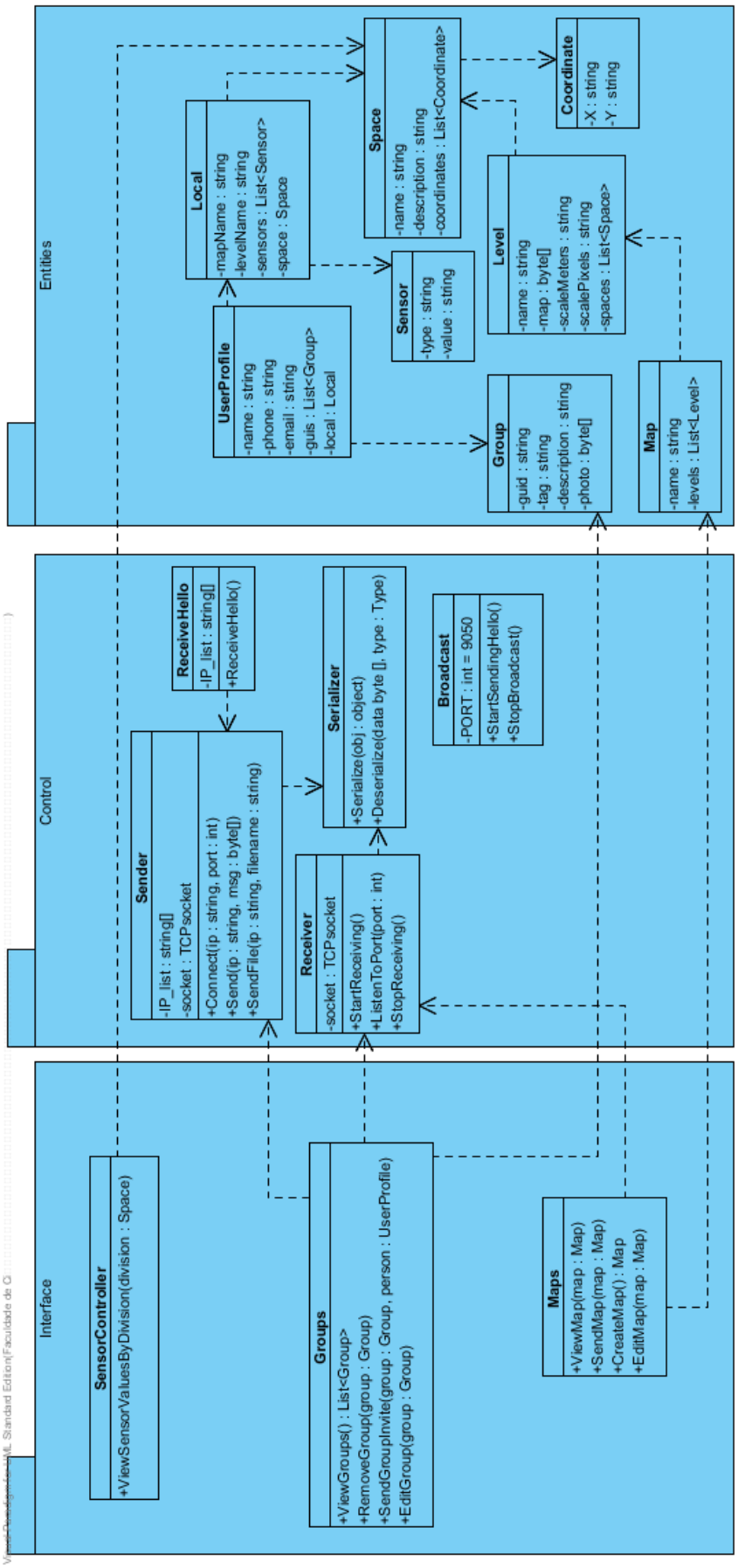

Figura 5.7 – Diagrama de classes do SVA.

5.3. CLASSES PRINCIPAIS DO CONTROLO

5.3.1. *SERIALIZER*

A classe *Serializer* efectua as conversões que são necessárias efectuar quando se pretende trocar informações numa rede. Esta classe não possui qualquer atributo e implementa dois métodos que efectuam a transformação de objectos[3] e XML *Schemas* para *strings* XML (*Serialize*) que depois podem ser convertidas num *array* de *bytes* quando se pretende enviar dados (Figura 5.8).

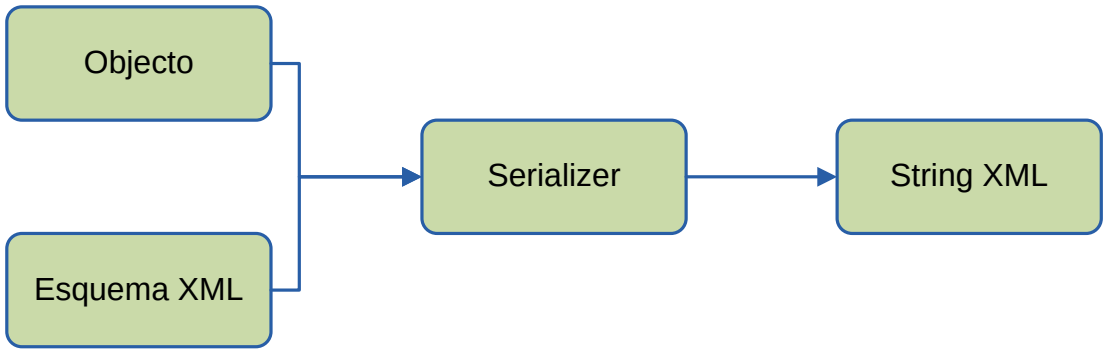

Figura 5.8 – Esquema representativo da função *Serialize*.

Ao receber efectua-se o processo inverso, ou seja, recebe-se o *array* de *bytes*, converte-se para uma *string* XML e efectua-se a conversão para objecto ou XML *Schema* (*Deserialize*) para se poderem trabalhar os dados recebidos (Figura 5.9).

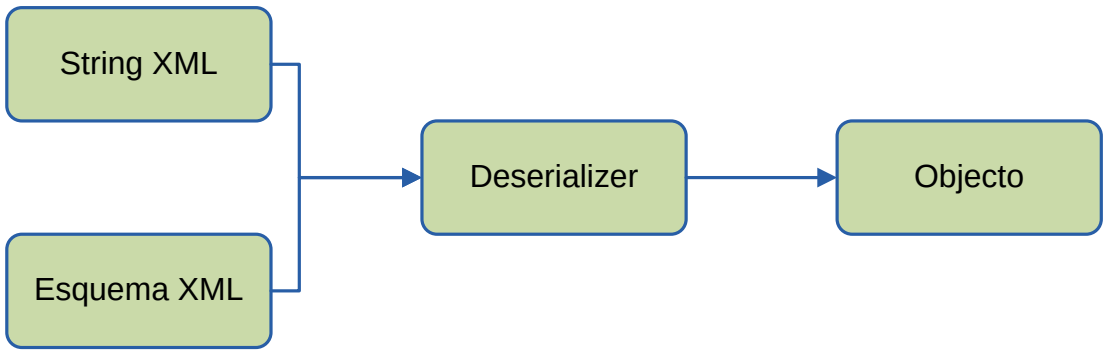

Figura 5.9 – Esquema representativo da função *Deserialize*.

[3] No contexto de Object-Oriented Programming (OOP), o termo objecto refere-se a uma estrutura de dados constituída por atributos e métodos, juntamente com todas as interacções entre si.

5.3.2. *SENDER*

A classe *Sender* faz uso de sockets TCP para transmitir dados pela rede *ad hoc*. Ela possui uma lista de IPs (*IPTable*) que corresponde à lista de todos os elementos que se encontram na sua rede, incluindo ele próprio. Esta é composta por três métodos principais: *Connect*, *Send* e *SendFile*. No método *Connect* vai ser efectuada a ligação ao IP e porto de destino para os quais se pretende enviar informação. Após estar estabelecida a ligação com o destinatário, é possível realizar uma de duas acções, quer se pretenda enviar uma mensagem de texto (*Send*) ou um ficheiro de dados (*SendFile*). O esquema básico de funcionamento da classe *Sender* encontra-se representado na Figura 5.10.

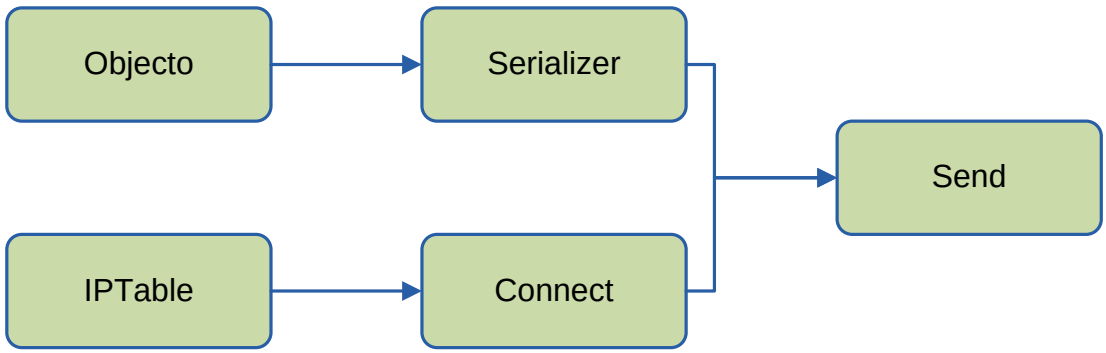

Figura 5.10 – Esquema representativo da classe *Sender*.

5.3.3. *RECEIVER*

A classe *Receiver* recorre também aos sockets TCP para receber dados da rede *ad hoc*. Ela começa por invocar o método *ListenToPort* para ficar pronta a receber dados. Sempre que alguém na rede lhe envia informação, ela invoca o método *StartReceiving* que inicia a transferência dos dados. Quando finalizada a tranferência é utilizado o método *StopReceiving* para fechar a ligação entre os dois intervenientes. O esquema da classe *Receiver* encontra-se apresentado na Figura 5.11.

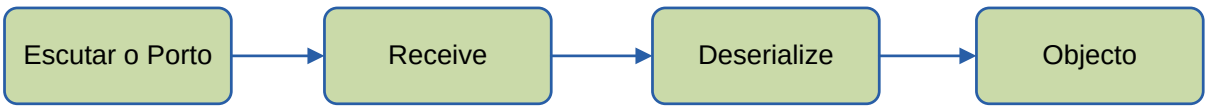

Figura 5.11 – Esquema representativo da classe *Receiver*.

5.3.4. *BROADCAST*

A classe *Broadcast* funciona de forma semelhante às classes *Sender* e *Receiver* apresentadas anteriormente. No entanto, estas duas apenas são úteis quando se pretende trocar dados com um endereço IP específico. Caso seja necessária realizar uma difusão por toda a rede de uma determinada informação, estas duas classes de nada servem.

Desta forma foi necessário criar uma outra classe, com o seu próprio *Sender* e *Receiver* que tenha a capacidade de efectuar difusão na rede. A grande diferença da classe *Broadcast* em relação às outras duas é que esta faz uso de sockets UDP e não TCP. Desta forma, ao realizar o envio de dados, não é especificado um IP para o qual se pretende enviar a informação pois ela já será enviada para todos os IPs na rede.

Esta necessidade de enviar informação para todos os IPs na rede está relacionada com o método desenvolvido para a contrução de uma tabela que contivesse todos os IPs existentes na rede. Esse método é explicado pormenorizadamente de seguida.

Suponha-se uma rede com topologia apresentada na Figura 5.12.

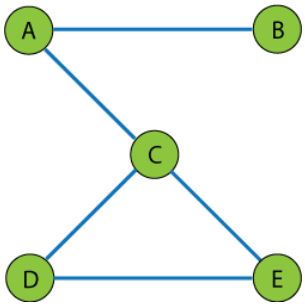

Figura 5.12 – Exemplo de topologia de uma rede *ad hoc*.

Tabela 5.1 – Pacote utilizado no processo de construção da tabela de IPs.

Pacote
IP
Última Actualização
Lista de Grupos

Cada nó na rede irá de *t* em *t* intervalos de tempo enviar o pacote apresentado na Tabela 5.1 via *broadcast* para todos os nós que se encontram ao seu alcance. Na fase inicial ($t = 0$), os nós apenas têm conhecimento dos seus próprios IPs, como se pode obervar pela Tabela 5.2.

Tabela 5.2 – IPs conhecidos na rede na fase inicial ($t = 0$).

Nó	IPs conhecidos na rede (Z_0)
A	IP_A
B	IP_B
C	IP_C
D	IP_D
E	IP_E

Na primeira iteração do processo, cada nó ao enviar o pacote a todos os seus vizinhos (nós que se encontram ao seu alcance), vai fazer com que a tabela de IPs conhecidos na rede passe a ser a apresentada na Tabela 5.3. Como se pode observar, por exemplo, para o nó A, ele ficará a conhecer, para além do seu IP, também os IPs dos seus vizinhos B e C.

Tabela 5.3 – IPs conhecidos na rede após a primeira iteração ($t = 1$).

Nó	IPs conhecidos na rede (Z_1)	
	Cálculo Auxiliar	**Resultado final**
A	$Z_{0_A} \cup Z_{0_B} \cup Z_{0_C}$	IP_A , IP_B , IP_C
B	$Z_{0_B} \cup Z_{0_A}$	IP_B , IP_A
C	$Z_{0_C} \cup Z_{0_A} \cup Z_{0_D} \cup Z_{0_E}$	IP_C , IP_A , IP_D , IP_E
D	$Z_{0_D} \cup Z_{0_C} \cup Z_{0_E}$	IP_D , IP_C , IP_E
E	$Z_{0_E} \cup Z_{0_C} \cup Z_{0_D}$	IP_E , IP_C , IP_D

Na segunda iteração, cada nó vai agora enviar o pacote novamente aos seus vizinhos, mas desta vez não apenas com os seus dados, mas também com os dados de todos os nós dos quais tem conhecimento do seu IP. Desta forma, o nó A, por exemplo, na segunda iteração vai enviar uma tabela com três entradas em que uma é a sua própria e as outras duas são dos nós dos quais ele tem conhecimento do IP, ou seja, os nós B e C. Efectuada esta troca de pacotes, após as segunda iteração os IPs conhecidos na rede passarão a ser os apresentados na Tabela 5.4.

Tabela 5.4 – IPs conhecidos na rede após a segunda iteração ($t = 2$).

Nó	IPs conhecidos na rede (Z_2)	
	Cálculo Auxiliar	**Resultado final**
A	$Z_{1_A} \cup Z_{1_B} \cup Z_{1_C}$	IP_A , IP_B , IP_C , IP_D , IP_E
B	$Z_{1_B} \cup Z_{1_A}$	IP_B , IP_A , IP_C
C	$Z_{1_C} \cup Z_{1_A} \cup Z_{1_D} \cup Z_{1_E}$	IP_C , IP_A , IP_D , IP_E , IP_B
D	$Z_{1_D} \cup Z_{1_C} \cup Z_{1_E}$	IP_D , IP_C , IP_E , IP_A
E	$Z_{1_E} \cup Z_{1_C} \cup Z_{1_D}$	IP_E , IP_C , IP_D , IP_A

Na terceira iteração, o mesmo processo é repetido, isto é, cada nó envia para os seus vizinhos toda a informação de que dispõe. Desta forma, a tabela após a terceira iteração terá o aspecto apresentado na Tabela 5.5.

Tabela 5.5 – IPs conhecidos na rede após a terceira iteração ($t = 3$).

Nó	IPs conhecidos na rede (Z_3)	
	Cálculo Auxiliar	**Resultado final**
A	$Z_{2_A} \cup Z_{2_B} \cup Z_{2_C}$	IP_A , IP_B , IP_C , IP_D , IP_E
B	$Z_{2_B} \cup Z_{2_A}$	IP_B , IP_A , IP_C , IP_D , IP_E
C	$Z_{2_C} \cup Z_{2_A} \cup Z_{2_D} \cup Z_{2_E}$	IP_C , IP_A , IP_D , IP_E , IP_B
D	$Z_{2_D} \cup Z_{2_C} \cup Z_{2_E}$	IP_D , IP_C , IP_E , IP_A , IP_B
E	$Z_{2_E} \cup Z_{2_C} \cup Z_{2_D}$	IP_E , IP_C , IP_D , IP_A , IP_B

Como se pode obervar pela Tabela 5.5 já todos os nós na rede têm conhecimento dos IPs de todos os nós que a constituem. No entanto, uma vez que no modelo *ad hoc*, a topologia das redes está em constante mudança, este processo não pode nunca parar, mesmo que se tenha chegado à situação verificada ao fim da terceira iteração.

Para verificar se um nó abandonou a rede, é consultado o campo *Última Actualização* do pacote de dados apresentado na Tabela 5.1. Desta forma, caso um nó não receba um pacote de um nó num determinado período de tempo, esse nó é removido da sua tabela de IPs.

A presença no pacote de dados da Tabela 5.1 do campo *Lista de Grupos* no pacote também tem uma explicação que reside no facto de um nó apenas poder comunicar com outro caso partilhem um ou mais grupos. Para além disso, quando é apresentado o mapa do edifício ao utilizador, apenas esses elementos com grupos em comum podem aparecer no mapa. Daí a necessidade de cada nó saber a que grupos cada um dos restantes pertence para poder determinar se tem possibilidades de se comunicar com ele e observar a sua posição no mapa.

5.3.5. *HotspotReceiver*

A classe *HotspotReceiver* está encarregue de receber e tratar os dados recebidos pelos Hotspots. O HA foi desenvolvido de forma a estar constantemente a fazer *broadcast* dos dados do seu respectivo Hotspot. Desta forma, o MDA necessita de incluir esta classe para conseguir descodificar a informação enviada pelos Hotspots.

A estrutura definida para o pacote que trará a informação acerca do Hotspot encontra-se apresentada na Tabela 5.6. Como se pode obervar, pacote está definido como um *array* de *bytes* uma vez que o ambiente de desenvolvimento utilizado para a programação do Arduino não suporta a classe *Serializer* anteriormente descrita. Para além disso, o Arduino por omissão apenas consegue armazenar até 512 *bytes*.

Tabela 5.6 – Disposição dos campos do pacote enviado pelos Hotspots ao MDA.

Campo	Nome do mapa	Nome do piso	Nome da divisão	X	Y	Sensores						
						Tipo 1	Valor 1	Tipo 2	Valor 2	...	Tipo *N*	Valor *N*
Tamanho em *bytes*	máx. 150	máx. 150	máx. 150	2	2	1	1	1	1	...	1	1

O MDA ao ter conhecimento desta estrutura do pacote, sempre que o recebe vai iniciar a análise deste até obter todos os campos que o compõem. Desta forma, ele fica a saber em que divisão do edifício se encontra, bem como os valores verificados por todos os sensores que lá existem.

5.4. CLASSES PRINCIPAIS DA INTERFACE

Nesta área dos diagramas de classes apresentados, o MDA e o SVA partilham duas classes: *Groups* e *Maps*. A forma como elas são aplicadas em cada um destes dois componentes varia ligeiramente, uma vez que um as aplica mais no sentido de utilizador (MDA) e outro mais no sentido de administrador (SVA). Para além destas duas classes o MDA e o SVA implementam ainda cada um outra classe: *Chat* e *SensorController*, respectivamente.

5.4.1. *GROUPS*

Esta classe está responsável pela gestão da informação de cada grupo existente dentro de uma rede *ad hoc*. Os seus métodos são os seguintes:

- *ViewGroups*: devolve toda a informação relativa a um grupo e permite a visualização dos seus detalhes. Do lado do MDA, este método é utilizado para visualizar apenas os grupos de um utilizador em particular, enquanto que do lado do SVA é utilizado para visualizar todos os grupos existentes num edifício.
- *RemoveGroup*: elimina um grupo de entre uma lista de grupos. No MDA eliminar um grupo equivale a um utilizador abandonar esse grupo. No SVA eliminar um grupo significa apagá-lo da lista de grupos de um edifício.
- *SendGroupInvite*: envia um convite de um grupo para um MDA. Este método é aplicado de forma igual quer no MDA como no SVA.

- *AcceptGroupInvite*: adiciona um grupo enviado por convite a um MDA. Este método apenas é utilizado no MDA, uma vez que o SVA não pode aceitar novos grupos pois já os contém a todos.
- *EditGroup*: edita os detalhes de um determinado grupo. Uma vez que os grupos são responsabilidade da administração, o MDA não tem privilégios para realizar esta acção, pelo que está método apenas se encontra disponível no SVA.

5.4.2. *MAPS*

A classe *Maps* dá o suporte à gestão dos mapas do edifício, bem como à sua consulta e partilha pela rede. Os métodos que implementa são descritos de seguida:

- *ViewMap*: dá suporte à visualização do mapa de um edifício. A sua aplicação no MDA e no SVA é semelhante, uma vez que devolve a mesma informação, variando apenas a forma como ela é apresentada ao utilizador.
- *SendMap*: envia a informação relativa a um mapa, bem como as suas imagens correspondentes. Este método é aplicado da mesma forma quer no MDA como no SVA, com a diferença de o SVA apenas poder partilhar o mapa do seu edifício enquanto que o MDA tem a possibilidade de partilhar todos os mapas que já possui.
- *CreateMap*: dá suporte à criação de um novo mapa para um edifício. Este método apenas se encontra disponível no SVA, pois este é que detém as permissões para a criação de novos mapas.
- *EditMap*: edita um mapa já existente. Tal como o método anterior, também a opção de editar mapas apenas se encontra disponível no SVA.

5.4.3. *CHAT*

Esta classe apenas é utilizada no MDA e serve de suporte à comunicação entre as pessoas que partilhem a mesma rede *ad hoc*. A troca de mensagens de texto bem como de ficheiros de dados é suportada por esta classe. Os seus métodos são então os seguintes:

- *OpenChatWindow*: abre uma nova janela de conversa entre duas pessoas na rede. Basicamente este método estabelece todas a ligações necessárias entre os dois intervenientes de forma a que estes estejam capacitados para trocar dados.
- *SendMessage*: envia uma mensagem de texto pela rede para uma determinada pessoa. Este método é aplicado quando se pretende trocar apenas texto entre duas pessoas.
- *SendFile*: envia um ficheiro de dados pela rede para uma determinada pessoa. Este método é semelhante ao anterior, com a diferença de aqui se enviar um ficheiro localizado numa directoria local.

- *ReceiveMessage*: recebe uma mensagem da rede. A recepção de dados, quer eles sejam mensagens de texto ou ficheiros de dados é toda implementada neste método. Ele trata de todas a verificações necessárias quando algo é recebido através da rede *ad hoc*.

5.4.4. *SENSORCONTROLLER*

Esta classe destina-se a fornecer a informação sensorial actual de todo o edifício, filtrada por divisão, de forma a que seja possível visualizar estes dados para que possam ser aplicadas medidas ao edifício para aumentar a sua eficiência energética. A sua utilização é feita exclusivamente pelo SVA e o seu único método é o seguinte:

- *ViewSensorValuesByDivision*: devolve a informação sensorial de um edifício. Este método é invocado quando se pretende mostrar qual o estado em que o edifício se encontra no que diz respeito às constantes físicas nele verificadas, bem como quando se pretende realizar algum tipo de processamento aos dados existentes.

5.5. DIAGRAMAS DE SEQUÊNCIA

Nesta secção serão apresentados três diagramas de sequência que pretendem servir de exemplo para a forma como o BSM foi desenvolvido. O primeiro caso corresponde ao processo de ligação e troca de dados entre dois utilizadores e encontra-se representado na Figura 5.13.

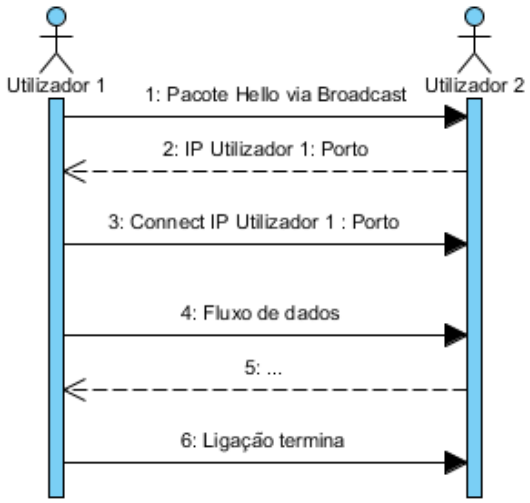

Figura 5.13 – Diagrama de sequência do processo de ligação e troca de dados entre dois MDAs.

O segundo caso escolhido foi a partilha de mapas entre dispositivos móveis e o processo de como eles são visualizados. Na Figura 5.14 encontra-se representado o diagrama de sequência relativo ao processo de envio do mapa do edifício entre dispositivos móveis através do MDA.

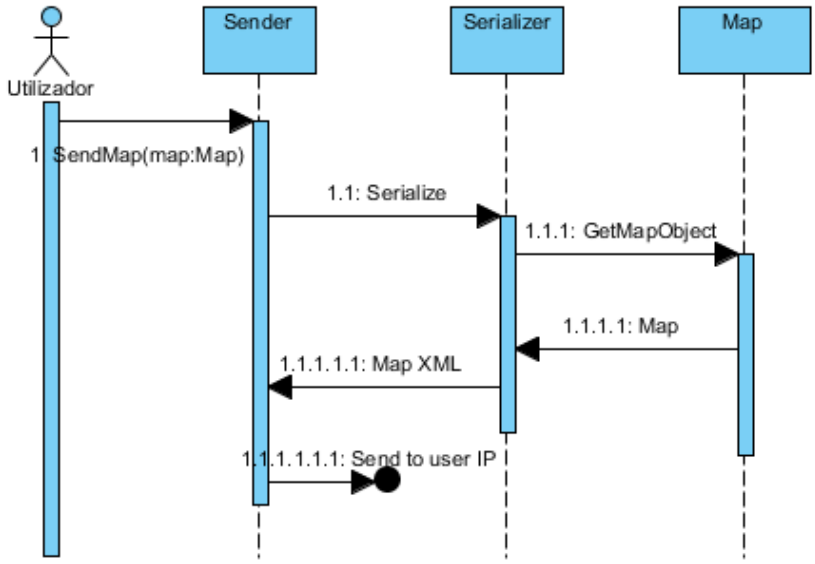

Figura 5.14 – Diagrama de sequência do processo de envio do mapa do edifício através do MDA.

Como se pode observar, antes de o mapa poder ser enviado através da classe *Sender*, ele necessita de ser obtido através da classe *Map* e passar pela classe *Serializer*.

O diagrama de sequência do processo de aceitação do mapa do edifício através do MDA pode ser observado na Figura 5.15. Neste diagrama pode verificar-se que quando recebe o mapa do edifício, o utilizador pode recusá-lo ou, pelo contrário, aceitá-lo, sendo-lhe devolvido o objecto *Map*.

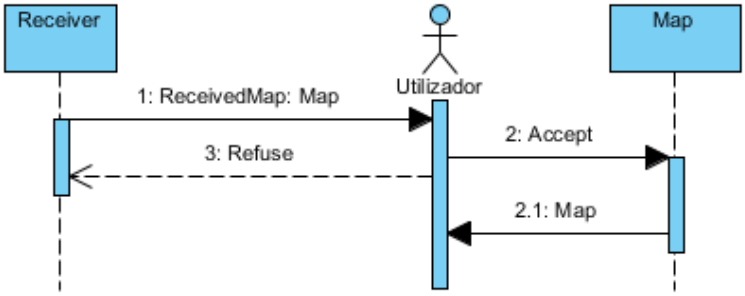

Figura 5.15 – Diagrama de sequência do processo de aceitação do mapa do edifício através do MDA.

6. EXEMPLO DE UTILIZAÇÃO

Para demonstrar a execução do sistema BSM vamos supor a entrada de três pessoas (Florencia de Jesus, Lourenço Marques e Armando Freitas), cada uma equipada com um *Smartphone* a correr o MDA, no DEE da FCT/UNL. Ao entrar no edifício, a interface na aplicação de cada um deles é a apresentada na Figura 6.1.

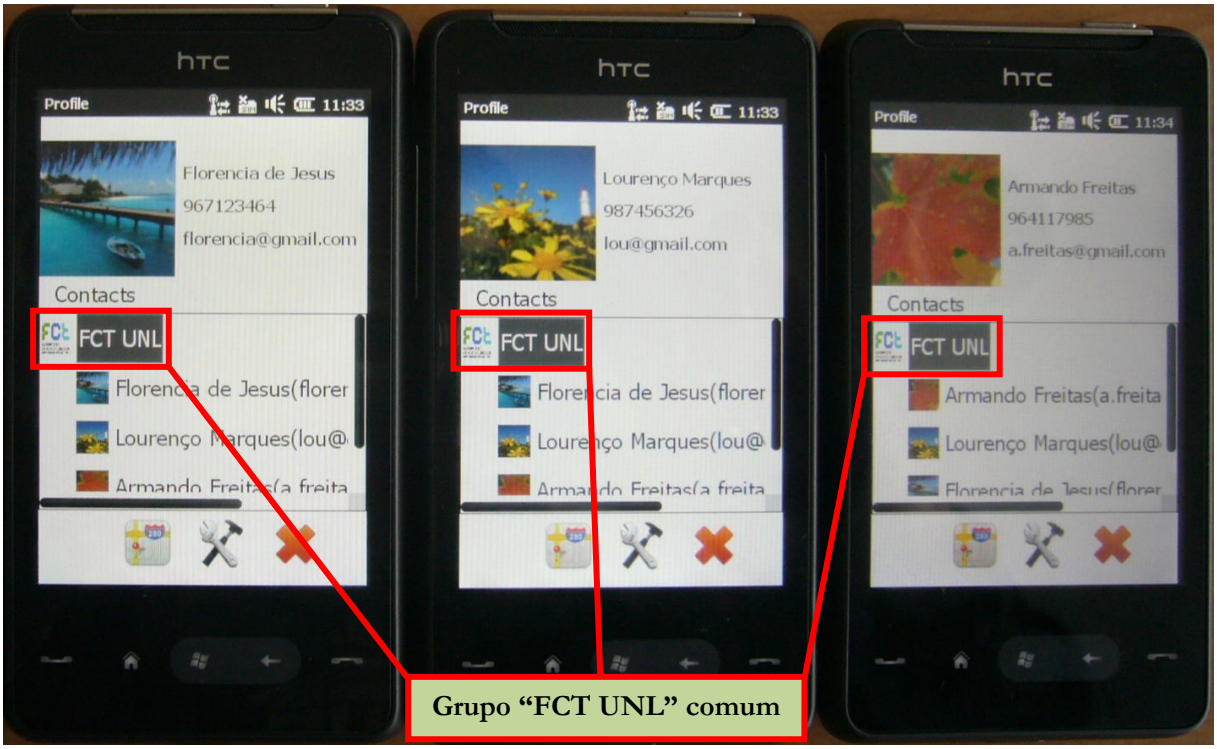

Figura 6.1 – Interface do MDA (entrada no DEE da FCT/UNL).

Como se pode observar pela Figura 6.1, todos os intervenientes pertencem ao grupo "FCT UNL", pelo que todos eles se conseguem visualizar uns aos outros no edifício. Também por esta razão eles podem ainda comunicar uns com os outros e ainda visualizar a posição no mapa de cada um.

De seguida, a Florencia de Jesus e o Lourenço Marques decidem iniciar uma conversa entre eles. Ao fazer isto, é aberta uma janela de *chat* onde eles podem trocar, quer mensagens de texto, quer ficheiros de dados. Na Figura 6.2 é possível observar o aspecto da interface nesta situação.

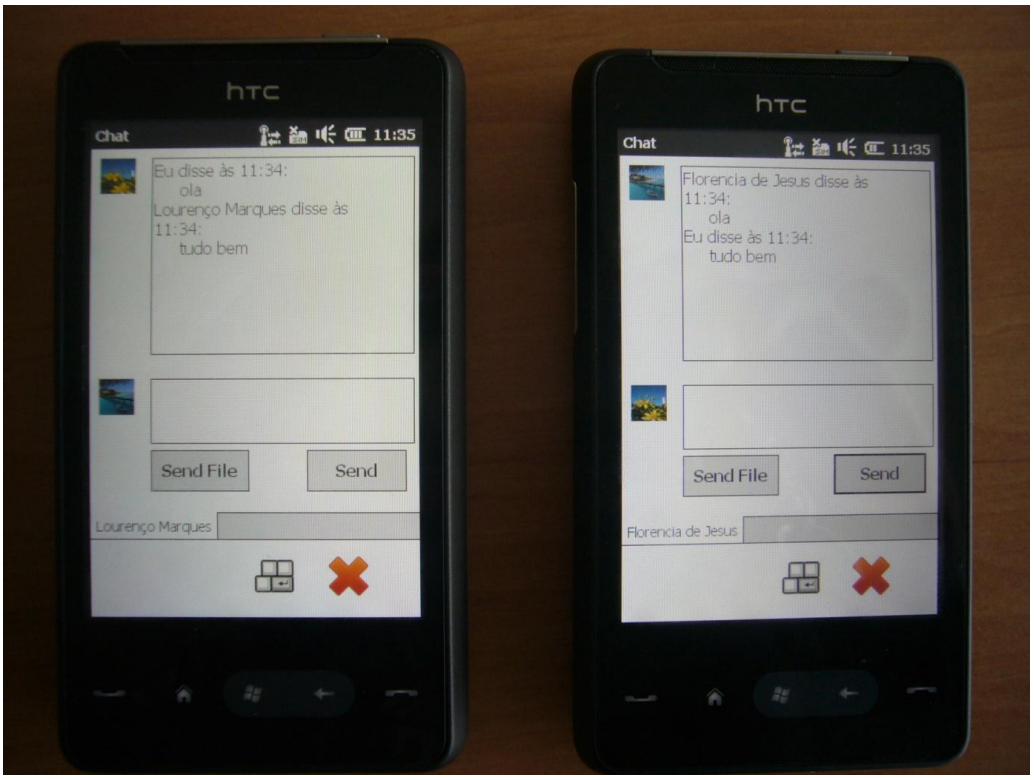

Figura 6.2 – Interface do MDA (sala de *chat*).

Como foi referido, para além de trocarem mensagens de texto, a Florencia de Jesus e o Lourenço Marques podem também trocar ficheiros de dados, como se encontra representado nas Figura 6.3 e Figura 6.4.

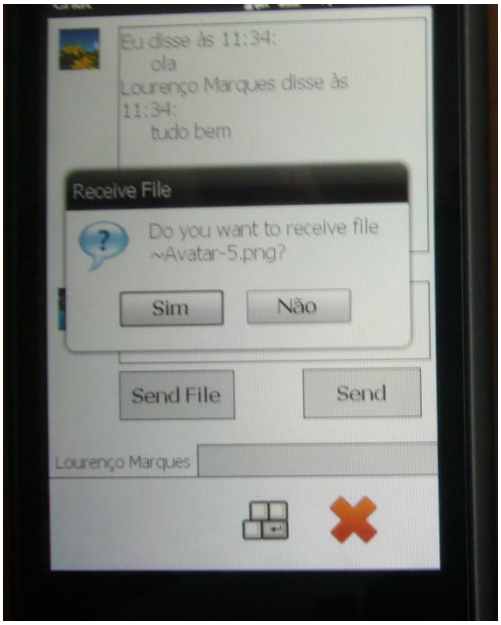

Figura 6.3 – Interface do MDA (pedido de recebimento de ficheiro).

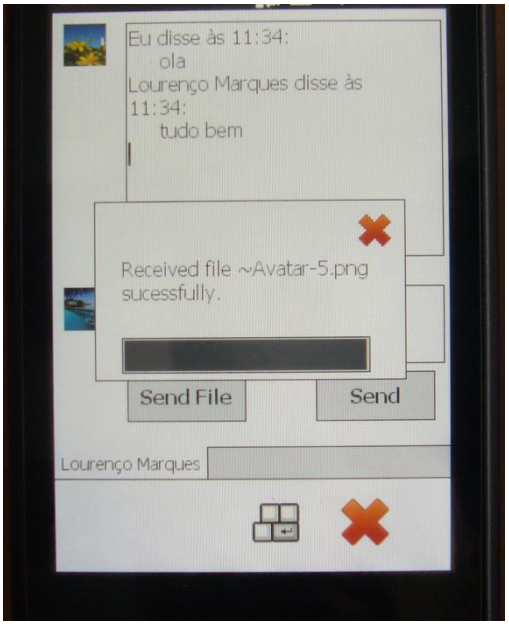

Figura 6.4 – Interface do MDA (recebimento de ficheiro concluído).

Para saber qual a sua posição no edifício, bem como os valores verificados pelos sensores, o cada uma destas pessoas necessita de estar ao alcance de um Hotspot. Dessa forma, o HA poderá enviar ao MDA os dados observados na divisão onde se encontra.

Elas podem assim ver qual a sua posição no edifício através do mapa disponível no MDA, como se pode obervar pela Figura 6.5. No mapa é possível visualizar qual o nome da divisão do edifício em que a pessoa se encontra, a sua descrição, bem como os valores aí recolhidos pelos sensores.

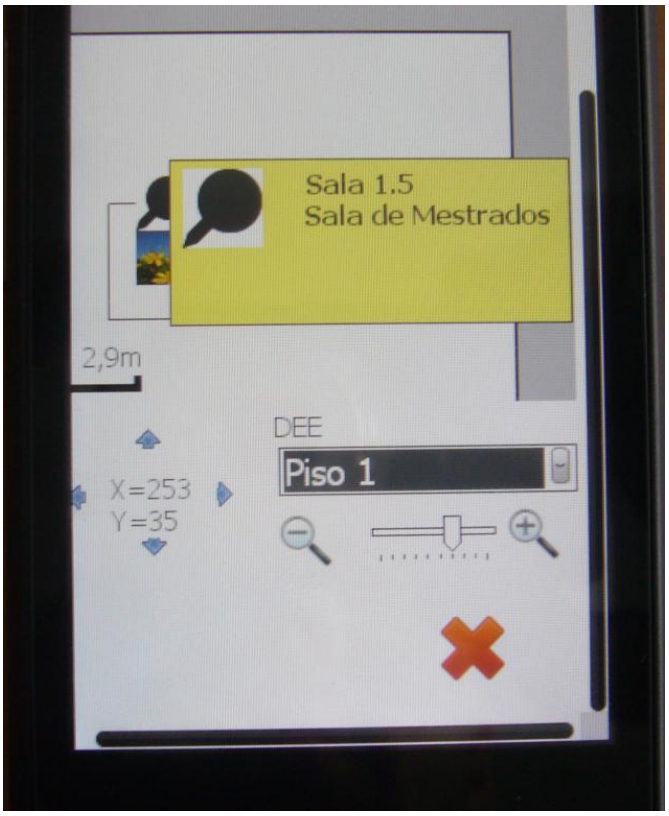

Figura 6.5 – Interface do MDA (posição actual no edifício).

Depois de receberem estes dados relativos a várias divisões do edifício eles vão enviar esta informação ao SVA para que possa depois ser cuidadosamente analisada. Aqui termina a actuação por parte dos três intervenientes e começa uma fase de análise dos dados recolhidos. Na Figura 6.6 encontra-se apresentado o sistema BSM completo, onde o SVA é executado num computador portátil.

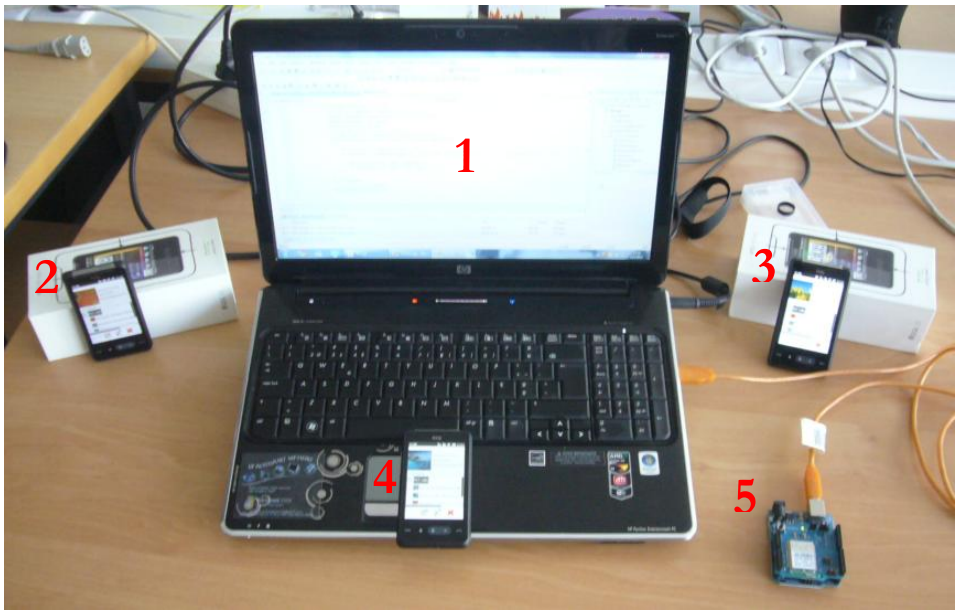

Figura 6.6 – Sistema BSM: SVA (1), MDA (2, 3 e 4) e Hotspot (5).

Na Figura 6.7 está apresentado o ecrã principal da aplicação SVA onde se podem observar quais os grupos criados no edifício do DEE da FCT/UNL. Como se pode observar este edifício é composto apenas por um grupo, que é precisamente aquele aos quais pertencem os utilizadores mencionados anteriormente a utilizar o MDA.

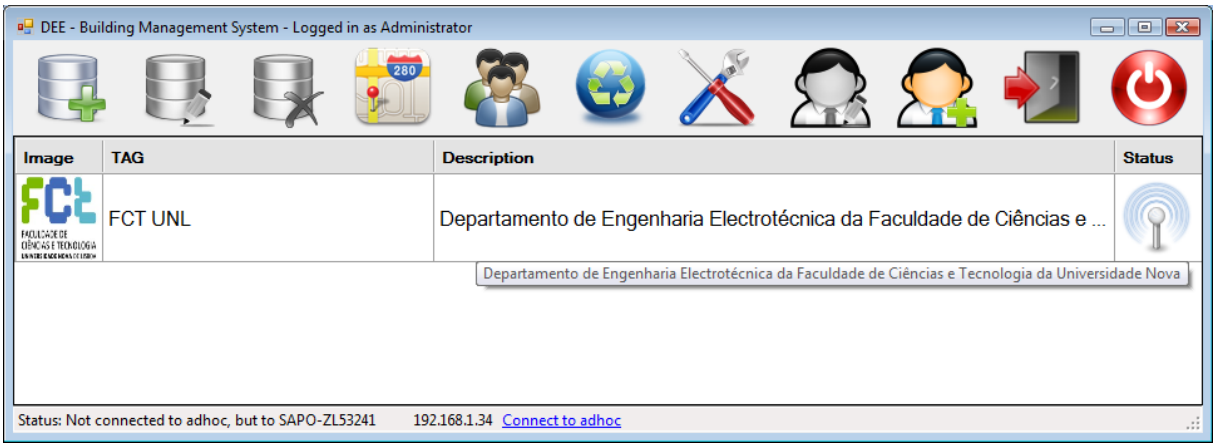

Figura 6.7 – Interface do SVA (ecrã principal).

Existem várias funcionalidade no SVA à disposição do seu administrador. Uma delas consiste em definir qual o tipo de sensores que o edifício suporta. Para tal, o administrador dirige-se às definições do SVA, onde lhe é apresentada uma lista de sensores (proveniente do UTD) de entre os quais ele pode escolher os que são suportados pelo edifício, como se pode obervar pela Figura 6.8.

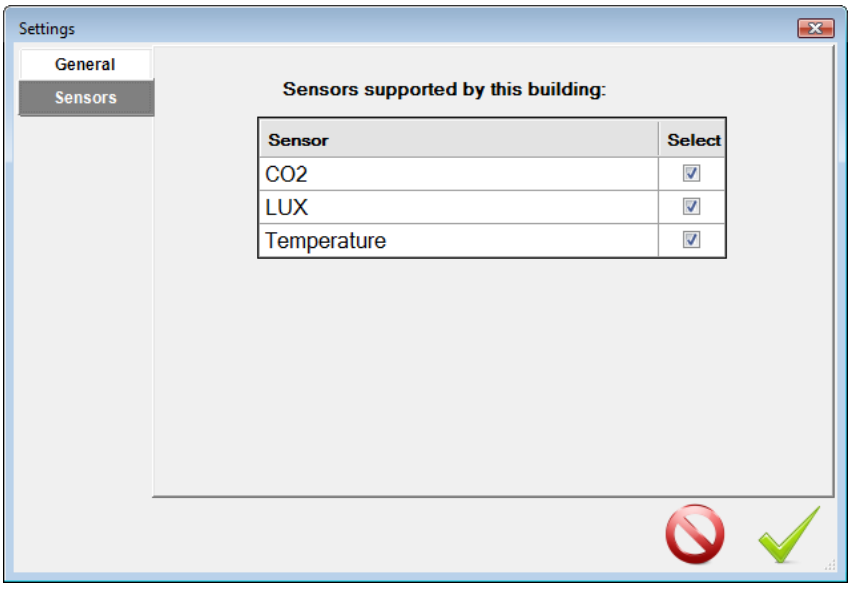

Figura 6.8 – Interface do SVA (definição dos sensores suportados pelo edifício).

Para além da definição dos sensores suportados, o administrador pode também visualizar o mapa do edifício, bem como editá-lo ou substituí-lo (Figura 6.9).

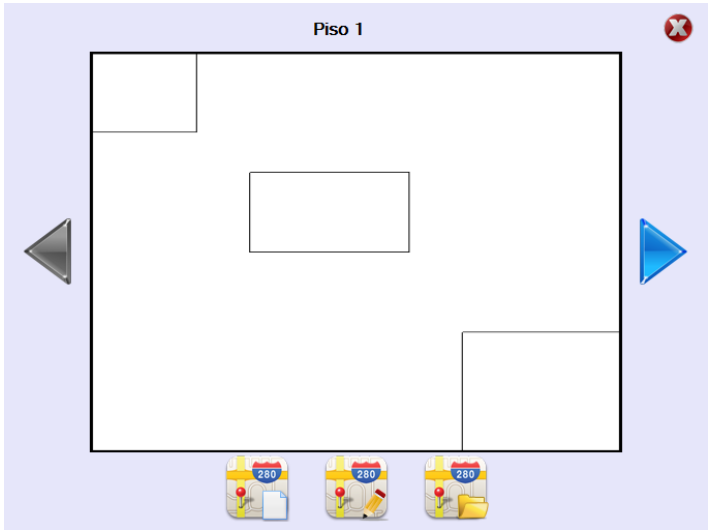

Figura 6.9 – Interface do SVA (mapa do edifício).

Para finalizar, a funcionalidade mais importante do SVA faz a análise dos valores recolhidos pelos Hotspots. O SVA, depois de processar essa informação, disponibiliza ao administrador a possibilidade de fazer estatísticas com os resultados das análises realizadas. Na Figura 6.10 pode-se verificar que o administrador pode filtrar os dados por data e por divisão de forma a tornar o resultado da pesquisa mais eficaz.

Os resultados dessas estatísticas são depois apresentados sob a forma de tabelas e gráficos de forma a simplificar a sua leitura por parte do administrador. Os valores apresentados para cada uma das constantes físicas correspondem ao valor médio verificado em cada uma divisões.

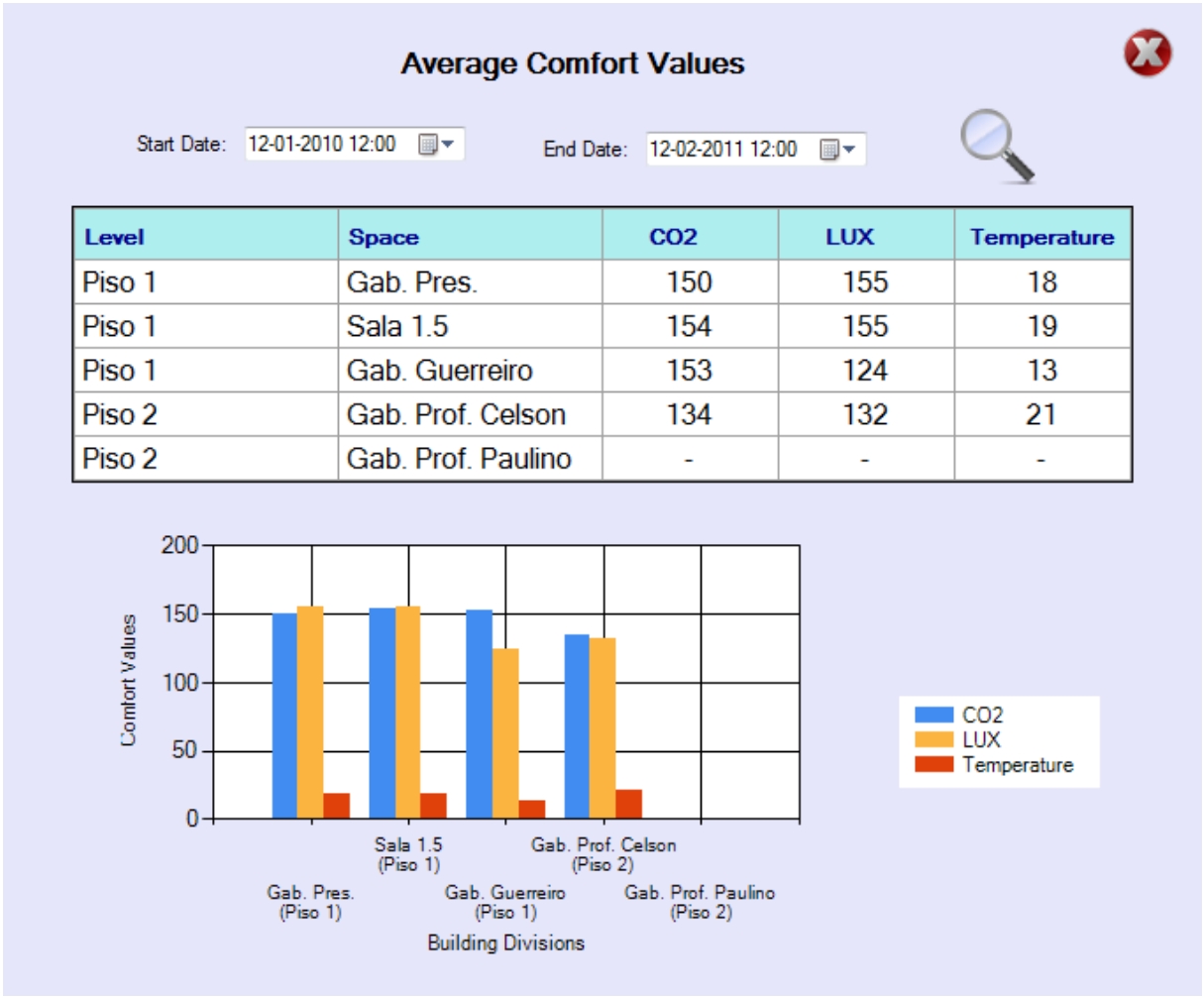

Figura 6.10 – Interface do SVA (resultados da análise aos valores obtidos através dos Hotspots).

7. CONCLUSÕES

A eficiência energética em edifícios é uma temática à qual deve ser dada muita importância, uma vez que ela pode resultar numa redução dos gastos de forma bastante significativa. A UE já demonstrou que está atenta a esta situação e inclusivamente tomou já medidas, pedindo aos países que a compõem que tenham em consideração esta problemática.

As tecnologias de redes sem fios podem ser um caminho viável para aplicar em diferentes situações do nosso quotidiano. Os avanços verificados nesta área têm sido bastante significativos e a cada dia que passa temos conhecimento de outras tecnologias e funcionalidades para elas. As redes *ad hoc*, especificamente, constituem uma tecnologia muito apetecível para aplicar em diversas situações.

Nesta dissertação as redes *ad hoc* foram aplicadas no desenvolvimento de um sistema computacional que visa melhorar a eficiência energética em edifícios. No entanto, como esta, muitas situações existem que poderiam também fazer uso deste trabalho desenvolvido, uma vez que um dos objectivos definidos foi o de criar um protocolo baseado em redes *ad hoc* que fosse totalmente independente do uso que lhe fosse atribuído.

Este protocolo que foi necessário desenvolver e que suporta toda a camada de comunicação envolvida no BSM foi talvez o ponto mais interessante e inovador em todo o trabalho realizado. Grande parte do tempo dispendido neste trabalho foi dedicado ao desenvolvimento deste protocolo. Esta opção deveu-se ao facto de que se o sistema estivesse bem definido e robusto logo à partida, isso iria trazer muitas vantagens quando se fosse desenvolver a camada seguinte. No final pode-se concluir que essa decisão foi a mais correcta uma vez que nunca existiram problemas em relação à parte do *core* na fase de desenvolvimento aplicacional.

Outro ponto fundamental no funcionamento do BSM consiste na recolha sensorial das constantes físicas do edifício. O conhecimento do estado do edifício é um requisito vital para o BSM e, portanto, a distribuição de sensores pelo espaço do edifício e a sua respectiva comunicação com os *Smartphones* das

pessoas que se encontram nele foram também grandes desafios. Ainda assim, foi possível superar todas as dificuldades que foram aparecendo no decorrer do trabalho de forma a atingir todos estes objectivos, fazendo com que o BSM se tornasse no que foi idealizado inicialmente.

De uma forma geral, todos os objectivos definidos no início da fase de desenvolvimento foram cumpridos e todos os componentes criados desempenham as funções para as quais foram projectados. Após a conclusão do desenvolvimento do sistema BSM, este foi testado de forma intensiva de forma a comprovar estes factos.

De entre todo o trabalho realizado no decorrer desta dissertação resultaram vários sistemas computacionais. Entre eles destacam-se o SVA, o MDA, o HA e o UTD. Cada um destes componentes exerce determinadas funções quando inseridos no BSM. À partida, todos eles desempenham funções que visam o aumento da eficiência energética em edifícios, no entanto, o seu desenvolvimento foi pensado de forma a poder ser adaptado a outros casos aplicacionais.

O desenvolvimento deste sistema computacional procura também ser parte da investigação que está a ser realizada acerca das redes *ad hoc*, sendo também ele um avanço em todo o estudo realizado acerca desta tecnologia. Este foi também um factor importante tido em conta aquando da escolha das tecnologias a utilizar neste trabalho.

Convém referir que o sistema computacional desenvolvido não constitui um sistema acabado mas sim um projecto ao qual podem ainda ser aplicados muitos mais melhoramentos e funcionalidades. Para além disso, este sistema encontra-se incluído num projecto de maiores dimensões, pelo que as funcionalidades globais do sistema não se encontram limitadas apenas ao que foi apresentado nesta dissertação.

Pode-se dizer que o passo seguinte ao BSM consiste em utilizar os dados recolhidos por ele para aplicar de forma activa e fisicamente no edifício. Esse passo já se encontra a ser dado e espera-se que dentro de um período de tempo não muito longo se possa obter o funcionamento completo do sistema.

BIBLIOGRAFIA

[1] The EU climate and energy package.
http://ec.europa.eu/clima/policies/package/ (última consulta: 29 de Abril de 2011).

[2] Council adopts climate-energy legislative package.
http://www.consilium.europa.eu/uedocs/cms_data/docs/pressdata/en/misc/107136.pdf
(última consulta: 29 de Abril de 2011).

[3] Eficiência Energética dos Edifícios e da Iluminação Pública na Administração Pública.
http://www.apdc.pt/filedownload.aspx?schema=f7664ca7-3a1a-4b25-9f46-2056eef44c33&channel=72F445D4-8E31-416A-BD01-D7B980134D0F&content_id=62E2FBDA-18B5-4C94-98A8-A2A68AD90843&field=storage_image&lang=pt&ver=1 (última consulta: 29 de Abril de 2011).

[4] Comunicado do Conselho de Ministros de 9 de Dezembro de 2010.
http://www.portugal.gov.pt/pt/GC18/Governo/ConselhoMinistros/ComunicadosCM/Pages/20101209.aspx (última consulta: 29 de Abril de 2011).

[5] Tanenbaum A. S. *"Computer Networks"*, Amsterdão, Holanda: Vrije Universiteit, 5ª ed., 2010.

[6] Wireless Ad Hoc Networks Background.
http://www.antd.nist.gov/wahn_bkgnd.shtml (última consulta: 24 de Junho de 2011).

[7] Li, J.; Blake, C; De Couto, D. S. J.; Lee, H. I. e Morris, R. *"Capacity of Ad Hoc Wireless Networks"*, M.I.T. Laboratory for Computer Science, 2001.

[8] Redes Móveis Ad Hoc.
http://www.projetoderedes.com.br/artigos/artigo_redes_moveis_ad_hoc.php (última consulta: 24 de Junho de 2011).

[9] Vantagens e Desvantagens das Redes Ad Hoc. http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/adhoc/vantagens.htm (última consulta: 24 de Junho de 2011).

[10] Mobile Ad Hoc Networks. http://www.antd.nist.gov/wahn_mahn.shtml (última consulta: 24 de Junho de 2011).

[11] Hogie, L.; Bouvry, P. e Guinand, F. *"An Overview of MANETs Simulation"*, Namur, Bélgica, Springer, 2005.

[12] Johnson, B. D.; Maltz, D. A. e Hu, Y. C. *"The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)"*, IETF MANET Working Group, 2004.

[13] Lang, D. *"A comprehensive overview about selected Ad Hoc Networking Routing Protocols"*, Munique, Alemanha: Technische Universitat Munchen, Department of Computer Science, 2003.

[14] Wireless Ad Hoc Sensor Networks. http://www.antd.nist.gov/wahn_ssn.shtml (última consulta: 24 de Junho de 2011).

[15] Oliveira, H. F. *"Seminário de Informação sobre Redes Ad Hoc de Sensores"*, Amazonas, Brasil: Universidade Federal do Amazonas, 2009.

[16] Ganesan, D.; Govidan, R.; Skenker, S. e Estrin, D. *"Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks"*, ACM Mobile Computing and Communications Review (MC2R, 2002).

[17] Panisson, A.; Barrat, A.; Cattuto, C.; Van den Broeck, W.; Ruffo, G. e Schifanella, R. *"On the Dynamics of Human Proximity for Data Diffusion in Ad Hoc Networks"*, Torino, Itália: Complex Networks and Systems Group, Institute for Scientific Interchange (ISI) Foundation, 2011.

[18] Singh, K.; Yadav, R. S. e Ranvijav *"A Review Paper on Ad Hoc Network Security"*, International Journal of Computer Science and Security, Volume (1): Issue (1), 2008.

[19] Kuosmanen, P. *"Classification of Ad Hoc Routing Protocols"*, Helsínquia, Finlândia: Finnish Defense Forces, Naval Academy, 2009.

[20] Jackson, D. *"MAC Protocols in Ad Hoc Networks"*, EE 360, Winter Quarter, 2010.

[21] Hodjat, A. e Verbauwhede, I. *"The Energy Cost of Secrets in Ad Hoc Networks"*, Los Angeles, United States of America: Department of Electrical Engineering, University of California, 2000.

[22] Zhou, L. e Haas, Z. J. *"Securing Ad Hoc Networks"*, Ithaca, United States of America: Cornell University, 2001.

[23] Meisel, M.; Pappas, V. e Zhang, L. *"Ad Hoc Networking via Named Data"*, United Kingdom: Ministry of Defense, 2011.

[24] Xu, Y.; Heidemann, J. e Estrin, D. *"Adaptative Energy-Conserving Routing for Multihop Ad Hoc Networks"*, USC/Information Sciences Institute, 2000.

[25] Burgess, M. e Canright, G. *"Scalability of Peer Configuration Management in Partially Reliable Ad Hoc Networks"*, Oslo, Noruega: Faculty of Engineering, Oslo University College, 2002.

[26] Bayya, A. K.; Gupte, S.; Shukla, Y. K. e Garikapati, A. *"Security in Ad Hoc Networks"*, Kentucky, United States of America: Computer Science Department, University of Kentucky, 2005.

[27] Gao, T.; Massey, T.; Selavo, L.; Welsh, M. e Sarrafzadeh, M. *"Participatory User Centered Design Techniques for a Large Scale Ad Hoc Health Information System"*, United States of America, 2007.

[28] Armuelles, I.; Robles, T.; O'droma, M.; Ganchev, I.; Chaouchi, H. e Siebert, M. *"On Ad Hoc Networks in the 4G Integration Process"*, Madrid, Spain: Department of Telematic Engineering, Technical University of Madrid, 2004.

[29] Li, C.; Chang, K. C. C. e Ilyas, I. F. "*Supporting Ad Hoc Ranking Aggregates"*, Illinois, United States of America: Department of Computer Science, University of Illinois at Urbana-Champaign, 2006.

[30] Engelstad, P. E.; Tonnesen, A.; Hafslund, A. e Egeland, G. *"Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks"*, Oslo, Noruega: Telenor R&D, 2005.

[31] Ramanathan, R.; Redi, J.; Santivanez, C.; Wiggins, D. e Polit, S. *"Ad Hoc Networking With Directional Antennas: A Complete System Solution"*, IEEE Journal on Selected Areas in Communications, Vol. 23, Nº 3, Março de 2005.

[32] Li, X. Y.; Tang, S. J. e Xu, X. H. *"Broadcast Capacity for Wireless Ad Hoc Networks"*, Chicago, United States of America: Department of Computer Science, Illinois Institute of Technology, 2008.

ANEXOS

ANEXO 1 – CLASSE AHPAM

```
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 está ligado, 0 - WiFi está 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 contrário.
        ///</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
compatível 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);
                return null;
            }
        }

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

```
        ///
        ///<summary>
        ///      Cria se não existir a rede com o nome SSID, ou liga-se a
ela caso já exista.
        ///</summary>
        ///<param name="ssid">O nome da rede adhoc.</param>
        ///<param name="password">A password de rede caso exista. Colocar
null caso não exista.</param>
        ///<param name="authentication">O método e autênticação (Aberto,
Partilhado, WPA, WPA-PSK, WPA2, WPA2-PSK).</param>
        ///<param name="wepStatus">A encriptação de dados (Desactivado,
WEP, AES, TKIP).</param>
        ///<param name="wzc">A instancia da interface de rede (Network
Interface) compatível com WZC.</param>
        ///<returns>
        ///  True se a ligação for bem sucedida, False caso contrário.
        /// </returns>
        public bool ConnectAdhoc(string ssid, string password,
AuthenticationMode authentication, WEPStatus wepStatus,
WirelessZeroConfigNetworkInterface wzc)
        {
            try
            {
                wzc.Refresh();
                AccessPointCollection apc = wzc.NearbyAccessPoints;
                AccessPoint ap = apc.FindBySSID(ssid);
                if (ap == null)
                {
                    EAPParameters eapParam = new EAPParameters();
                    wzc.AddPreferredNetwork(ssid, false, password, 1,
authentication, wepStatus, eapParam);
                    if (wzc.ConnectToPreferredNetwork(ssid))
                    {
                        Log.LogToFile("Connected to adhoc.");
                        return true;
                    }
                    else
                    {
                        Log.LogToFile("Failed to connect to adhoc.");
                        return false;
                    }
                }
                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 ligação à rede especificada em ssid associada à
interface de rede wzc.
        ///</summary>
        ///<param name="ssid">O nome da rede.</param>
        ///<param name="wzc">A instancia da interface de rede (Network
Interface) compatível com WZC.</param>
        ///<returns>
        ///  True se a desconexção for bem sucedida, False caso contrário.
        ///</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 máquina.
        ///</returns>
        public IPAddress GetLocalIP()
        {
            INetworkInterface[] nis =
NetworkInterface.GetAllNetworkInterfaces();
            return nis[0].CurrentIpAddress;
        }
    }
}
```

ANEXO 2 – CLASSE *IPTABLE*

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Xml.Serialization;

namespace Tables
{
    [Serializable]
    public class IPTable : List<IPTableEntry>
    {
        private string _type;
        public string type {
            get { return _type; }
            set { _type = value; }
        }

        public IPTable() {
            type = this.GetType().Name;
        }

        /// <summary>
        /// Procura na tabela de IPs um determinado IPEntry.
        /// </summary>
        /// <param name="ipEntry">O IPEntry a procurar.</param>
        /// <returns>A posição do IPEntry na tabela, ou -1 caso não
encontre.</returns>
        public int ExistsEntry(IPTableEntry ipEntry) {
            int pos = -1;
            int i = 0;
            foreach (IPTableEntry ipE in new List<IPTableEntry>(this)) {
                if (ipE.ip.Equals(ipEntry.ip))
                    pos = i;
                i++;
            }
            return pos;
        }

        /// <summary>
        /// Verifica se o lastUpdate de um determinado elemento da tabela é
maior ou menor que o novo IPEntry.
        /// </summary>
        /// <param name="ipEntry">O novo IPEntry.</param>
        /// <param name="pos">A posição do IPEntry actual.</param>
        /// <returns>True se o IPEntry actual for menor que o novo, false
caso contrário.</returns>
        public bool CompareLastUpdate(IPTableEntry ipEntry, int pos) {
            IPTableEntry ipEntryAux = this.ElementAt(pos);
            DateTime oldEntry = DateTime.ParseExact(ipEntryAux.lastUpdate,
"MM-dd-yyyy HH:mm:ss", null);
            DateTime newEntry = DateTime.ParseExact(ipEntry.lastUpdate,
"MM-dd-yyyy HH:mm:ss", null);
            if (oldEntry.CompareTo(newEntry) < 0)
                return true;
            return false;
        }
    }
}
```

ANEXO 3 – CLASSE *IPTABLEENTRY*

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Net;

namespace Tables
{
    [Serializable]
    public class IPTableEntry
    {
        public string ip;
        public string lastUpdate;
        public List<string> groupsList;

        /// <summary>
        /// Cria uma nova entrada para a tabela de IPs.
        /// </summary>
        public IPTableEntry() {
            ip = null;
            lastUpdate = DateTime.Now.ToString("MM-dd-yyyy HH:mm:ss");
            groupsList = null;
        }

        /// <summary>
        /// Cria uma nova entrada para a tabela de IPs com parametros.
        /// </summary>
        /// <param name="_ip">IP</param>
        /// <param name="_lastUpdate">Data e hora do último update.</param>
        public IPTableEntry(IPAddress _ip, DateTime _lastUpdate,
List<string> _groupsList) {
            ip = _ip.ToString();
            lastUpdate = _lastUpdate.ToString("MM-dd-yyyy HH:mm:ss");
            groupsList = _groupsList;
        }
    }
}
```

ANEXO 4 – CLASSE *SENDER*

```
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
contrário.</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 possível especificar se o socket
poderá ser reutilizado ou não.
        /// </summary>
        /// <param name="reusable">True se o socket for reutilizável,
false caso contrário.</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
contrário.</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]))
                {
                    found = true;
                    port = Convert.ToInt32(eps[1]);
                }
            }
            if (!found)
                return false;
            try
            {
                byte[] rawData = Serializer.Serialize(Data);
                byte[] sizeBytes = BitConverter.GetBytes(rawData.Length);
                byte[] data = new byte[9 + rawData.Length];
                sizeBytes.CopyTo(data, 0);
                data[8] = 0;
                rawData.CopyTo(data, 9);
                Socket socket = Connect(ipadress, port);
                arrayToSend = data;
                socket.Send(data);
                Log.LogToFile("Sent packet to " + ipadress + ":" + port);
                return true;
```

```
            }
            catch (SocketException)
            {
                Log.LogToFile("Error sending packet to " + ipadress + ":"
+ port);
                return false;
            }
            catch (Exception)
            {
                Log.LogToFile("Fatal Error on send.");
                return false;
            }
        }

        /// <summary>
        /// Envia um ficheiro.
        /// </summary>
        /// <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;
            foreach (string remoteEP in tcpClients)
            {
                string[] eps = remoteEP.Split(':');
                if (ipadress.Equals(eps[0]))
                {
                    found = true;
                    port = Convert.ToInt32(eps[1]);
                }
            }
            if (!found)
                return false;
            Socket socket = Connect(ipadress, port);
            if (socket == null)
                return false;
            try
            {
                string path = filePath + Filename;
                FileHandler fh = new FileHandler();
                FileInfo fi = new FileInfo(path);
                int bytesSent = 0;
                byte[] data;
                byte[] sizeBytes;
                byte[] filenameBytes;
                byte[] filenameSize;
                byte[] rawData = fh.GetFile(path);
                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);
                //Get file type
                data[8] = (byte)fileType;
                //Get filename size
                filenameSize =
BitConverter.GetBytes(Encoding.UTF8.GetByteCount(file));
                filenameSize.CopyTo(data, 9);
                //Get filename in bytes
```

```
                filenameBytes = Encoding.UTF8.GetBytes(file);
                filenameBytes.CopyTo(data, 13);
                //Get file data
                rawData.CopyTo(data, 13 +
Encoding.UTF8.GetByteCount(file));
                do {
                    bytesSent += socket.Send(data, bytesSent, data.Length-
bytesSent, SocketFlags.None);
                }
                while(bytesSent < rawData.Length);
                Log.LogToFile("Sent file to " + ipadress + ":" + port);
                return true;
            }
            catch (SocketException) {
                Disconnect(socket);
                Log.LogToFile("Error sending file to " + ipadress + ":" +
port);
                return false;
            }
            catch (Exception)
            {
                Log.LogToFile("Fatal Error on send file.");
                return false;
            }
        }

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

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

        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
    }
}
```

ANEXO 5 – CLASSE *RECEIVER*

```
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 recepção de pacotes TCP.
        /// </summary>
        /// <param name="port">Porta que vai ouvir a recepção 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 recepção de pacotes IP e respectivas threads e
sockets.
        /// </summary>
        public void Disconnect()
        {
            listenThread.Abort();
            listener.Close();
            if (clientThread != null)
            {
                clientThread.Abort();
                client.Close();
            }
        }

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

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

        /// <summary>
        /// Função que vai ser corrida por uma thread para receber pacotes
TCP de clientes já 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 até 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) {
                    break;
                }
                catch (Exception)
                {
                    //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();
        }

        // END CLASS
    }
}
```

ANEXO 6 – CLASSE *BROADCAST*

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using System.Reflection;
using Tables;
using Packets;
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>
        /// Função 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();
        }


        /// <summary>
        /// Cria um pacote broadcast com a tabela de IPs.
        /// </summary>
        private void CreatePacket() {
            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))
                gList.Add(gui.guid);
            IPTableEntry ipTableEntry = new
IPTableEntry(ahpam.GetLocalIP(), DateTime.Now, gList);
            int pos = ipTable.ExistsEntry(ipTableEntry);
            if (pos != -1)
            {
                ipTable.RemoveAt(pos);
```

```
                ipTable.Add(ipTableEntry);
            }
            else
                ipTable.Add(ipTableEntry);
            data = Serializer.Serialize(ipTable);
        }

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

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

        /// <summary>
        /// Verifica a tabela de IPs periodicamente de modo a verificar
IPs da rede que morreram, i.e. que já não respondem
        /// à algum tempo.
        /// </summary>
        private void CheckTable()
        {
            while (true)
            {
                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);
```

```
                    // Se não ouvir nenhum HELLO à mais de 30 segundos,
remove IP
                    if (difference.Hours > 0 || difference.Minutes > 5)
                    {
                        ipTable.Remove(ipTableEntry);
                        mainFormAction.Invoke(ipTable);
                    }
                }
                Thread.Sleep(5000);
            }
        }

        //END CLASS
    }
}
```

ANEXO 7 – XML SCHEMA *LOCAL*

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="TAG"
    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"
>
  <xs:include schemaLocation="map.xsd"/>
  <xs:complexType name="Sensor">
    <xs:sequence>
      <xs:element name="Type" type="xs:string"/>
      <xs:element name="Value" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="Sensors">
    <xs:sequence>
      <xs:element name="sensor" type="Sensor" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="Local">
    <xs:sequence>
      <xs:element name="MapName" type="xs:string"/>
      <xs:element name="Level" type="xs:string"/>
      <xs:element name="Space" type="tSpace"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Arduino">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="local" type="Local" />
        <xs:element name="sensors" type="Sensors"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

ANEXO 8 – EXEMPLO DE FICHEIRO XML *LOCAL*

```
<?xml version="1.0" encoding="utf-8" ?>
<Arduino xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://ahpam.org/maps/">
  <local>
    <MapName>35121FCTUNLEDX</MapName>
    <Level>Piso 1</Level>
    <Space>
      <Name>Entrada</Name>
      <Description>Entrada do DEE</Description>
      <Coordinates>
        <Coordinate>
          <x>220</x>
          <y>660</y>
        </Coordinate>
      </Coordinates>
    </Space>
  </local>
  <sensors>
    <sensor>
      <Type>1</Type>
      <Value>980</Value>
    </sensor>
  </sensors>
```