Faculdade de Engenharia da Universidade do Porto

Smart Home Device Application

Fernando Silva Lopes

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientadores:
Prof.ª Dr.ª Maria Teresa Andrade (INESC Porto, FEUP)
Eng.º André Oliveira (Ubiwhere)

Junho de 2010

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

A Dissertação intitulada

"SMART HOME DEVICE APPLICATION"

foi aprovada em provas realizadas em 22/Julho/2010

o júri

Presidente Professor Doutor Sérgio Reis Cunha
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Pedro Miguel do Vale Moreira
Professor Adjunto da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Viana do Castelo

Professora Doutora Maria Teresa Magalhães da Silva Pinto de Andrade
Professora Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto (Orientador).

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - **Fernando da Silva Lopes**

Faculdade de Engenharia da Universidade do Porto

Resumo

Na presente tese é apresentado um sistema de monitorização e controle de temperatura com recurso à tecnologia OSGi. A plataforma OSGi foi desenvolvida pela organização OSGi *Alliance*, com o objectivo de especificar um sistema de componentes dinâmicos e orientado a serviços, para aplicações modulares em Java. O foco inicial da especificação foi o mercado dos *gateways* residenciais, dispositivos aos quais estariam conectados todos os equipamentos presentes na habitação e dessa forma acessíveis através da Internet.

Um sistema embutido é utilizado como *gateway* e integra todo o *hardware* necessário, que inclui sensores de temperatura, um emissor e descodificador infra-vermelhos. Para cumprir os requisitos propostos, vários serviços OSGi foram desenvolvidos. Os resultados obtidos demonstram que a plataforma OSGi disponibiliza um ambiente dinâmico e fiável para aplicações residenciais.

Abstract

This thesis presents a system for temperature monitoring and control in a residence using OSGi technology. The OSGi platform was developed by the OSGi Alliance with the purpose of specifying a dynamic component system and service oriented, for modular Java applications. The initial focus of OSGi specification was the market of residential gateways, devices where all the household devices would be connected and available from the Internet.

An embedded system is used as home gateway and incorporates all the required hardware, which includes temperature sensors, an infrared emitter and decoder. Several OSGi services were developed in order to fulfill the requirements of the system. The results obtained show that OSGi provides a dynamic and reliable framework for home applications.

Agradecimentos

Gostaria de agradecer aos meus orientadores Prof.ª Dr.ª Maria Teresa Andrade e Eng.º André Oliveira, pelo apoio no desenvolvimento do projecto.

Agradeço também ao colaborador da Ubiwhere, Eng.º Flávio Sá pelo suporte diverso no desenrolar do projecto.

A nível pessoal, gostaria de agradecer à minha família pelo apoio demonstrado ao longo da minha vida.

Índice

Lista de figuras

Lista de tabelas

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

AGC	Automatic Gain Control
API	Application Programming Interface
BPSK	Binary Phase Shift Keying
CMM	Context-Aware Multimedia Middleware
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
CWMP	CPE Wan Management Protocol
DSSS	Direct Sequence Spread Spectrum
FDM	Frequency Division Multiplexing
FEUP	Faculdade de Engenharia da Universidade do Porto
FFD	Full Function Device
FHSS	Frequency Hopping Spread Spectrum
FSK	Frequency Shift Keying
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
IPTV	Internet Protocol Television
IR	Infrared
ITU	International Telecommunication Union
I2C	Inter Integrated Circuit
JAR	Java Archive File
JSR	Java Specification Request
JTAG	Joint Test Action Group
JVM	Java Virtual Machine

LAN	Local Area Network
LCD	Liquid Crystal Display
MAC	Media Access Control
MAS	Multi Agent System
MIB	Management Information Base
MIPS	Microprocessor Interlocked Pipeline Stages
OSGi	Open Services Gateway Initiative
PPM	Pulse Position Modulation
PSK	Phase Shift Keying
P2P	Peer to Peer
RAM	Random Access Memory
RDS	Remote Diagnostic System
RF	Radio Frequency
RFD	Reduced Function Device
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface Bus
SSH	Secure Shell
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
WAN	Wide Area Network

Capítulo 1

Introdução

Hoje em dia, uma habitação não é apenas um compartimento repleto de equipamentos electrónicos, com funções repetitivas e específicas, mas sim, um sistema com enorme potencial colaborativo. A tendência é integrar todos os sistemas de forma a interagirem entre si, promovendo o conforto e a segurança aos seus habitantes. Mediante a proliferação do acesso à banda larga e o desenvolvimento de soluções de automação residencial, os assinantes do serviço têm a possibilidade de explorar novas formas de interacção com a sua habitação.

A presente tese pretende explorar as capacidades da plataforma OSGi [1] e a sua aplicação no mercado domótico. Salienta-se o contributo desta dissertação para a divulgação e expansão de aplicações que exploram o conceito de casa inteligente em conjunto com as potencialidades de uma plataforma modular orientada a serviços.

1.1 - Contexto

Num futuro próximo, a incorporação de sistemas domóticos numa habitação será tão vulgar como a existência de electrodomésticos. No entanto, o custo associado à implementação deste tipo de sistemas é ainda extremamente elevado para a condição económica actual. Este será porventura o principal factor limitador da utilização em larga escala. Além deste, a complexidade de implementação dos sistemas não é um factor abonatório.

Apesar de este tipo de soluções não ser uma prioridade para a maioria das pessoas, diversos fabricantes têm investido no desenvolvimento de novas tecnologias para este mercado. O mercado mais presente da domótica é com certeza o da indústria, com grande parte das implementações existentes a focarem-se na eficiência energética de grandes e médios edifícios. Em ambientes residenciais, o objectivo dos sistemas domóticos prende-se com a melhoria das condições de habitabilidade e segurança. No entanto, o conceito de eficiência energética tem vindo a ser interiorizado e aplicado mesmo em ambientes de pequeno consumo quando comparados com ambientes industriais e comerciais. O consumo energético no mercado residencial apresentado no estudo [2], aumentou na maioria dos países europeus desde 1999.

Neste contexto, a presente dissertação pretende desenvolver um sistema domótico que procura colmatar os pontos menos positivos de soluções actuais.

1.2 - Motivação

Os sistemas extremamente custosos e complexos inibem potenciais clientes de investirem em inovações tecnológicas. Tal como outros mercados, o potencial de crescimento da domótica é enorme.

O mercado da domótica vem evoluindo de forma semelhante à informática com a tendência natural para a unificação das tecnologias utilizadas. Os sistemas de automação residencial ambicionam uma fácil integração com os equipamentos electrónicos presentes nas habitações, como sistemas de iluminação e climatização. No entanto, as soluções existentes mais disseminadas implementam protocolos proprietários e a compatibilidade entre diversos fabricantes está longe de estar assegurada.

Actualmente, Portugal encontra-se na 12ª posição [3] no que se refere à penetração da banda larga (fixa+móvel), com 25 em cada 100 habitantes a ter acesso ao serviço. Tal crescimento não é ignorado pelos operadores, que procuram oferecer novos serviços aos clientes, investindo em novas infra-estruturas e equipamentos mais evoluídos.

Prevendo tal situação, um consórcio de empresas criou a OSGi *Alliance*, como forma de definir uma plataforma que proporcionasse aos utilizadores o acesso aos mais variados serviços. Um dos mercados de actuação da especificação OSGi, é a área da domótica, em que a modularidade oferecida permite a implementação de serviços baseados em diferentes tecnologias mas cooperantes entre si. Com a evolução do mercado de acesso à banda larga e a disponibilização de equipamentos com maiores capacidades de processamento por parte dos operadores, passa a ser possível desenvolver uma diversidade de serviços.

Na área da climatização, em específico, a indústria de sistemas de ar condicionado tem tido um crescimento acentuado, com cerca de 6.8 milhões de unidades disponibilizadas no mercado europeu em 2008 [4]. Este valor representa um aumento de 10% face ao ano anterior. Estes sistemas são largamente utilizados em grandes edifícios e cada vez mais em habitações particulares.

A conjugação de todos estes factos, motivam o desenvolvimento do sistema baseado nos componentes modulares definidos pela plataforma OSGi, apresentado na presente dissertação.

1.3 - Objectivos

Os objectivos propostos para esta dissertação focam o desenvolvimento de um sistema de controlo e monitorização de temperatura num ambiente residencial. Nesse sentido, pretende-se atingir os seguintes objectivos:

- Estudar pormenorizadamente a plataforma de serviços OSGi.
- Estudar de tecnologias relevantes no contexto de automação residencial.
- Desenvolver um sistema domótico modular capaz de interagir com o sistema de climatização e sensores de temperatura.
- Desenvolver de um sistema de baixo custo.

1.4 - Estrutura da dissertação

A tese é composta por sete capítulos e três anexos.

O primeiro capítulo apresenta uma introdução ao projecto desenvolvido. Um capitulo curto de contextualização, que refere a motivação e os objectivos da dissertação. Por fim, é delineada a estrutura do documento.

Os três capítulos seguintes referem-se ao conhecimento inerente ao trabalho realizado ao longo da tese. No segundo capítulo, é abordado o suporte de modularidade na linguagem Java, contextualizando o leitor para o capítulo seguinte.

No terceiro capítulo, é analisada em detalhe a plataforma de serviços OSGi. Focam-se as características e funcionalidades de importância relevante para o serviço desenvolvido, nomeadamente a definição e gestão dos componentes modulares.

O quarto capítulo, expõe as tecnologias mais relevantes disponíveis comercialmente no contexto de automação residencial. Apresentam-se as suas potencialidades e suas aplicações no contexto de casa inteligente.

O quinto capítulo, diz respeito ao sistema desenvolvido, com a exposição da arquitectura, a descrição funcional e as tecnologias utilizadas. Na parte final, são abordados os serviços desenvolvidos com mais detalhe.

No sexto capítulo, analisam-se aspectos de desempenho de ambientes de execução alternativos com a discriminação dos testes efectuados e suas conclusões.

O último capítulo, apresenta os resultados mais significativos, com uma análise crítica do projecto e discussão de futuros desenvolvimentos. A aquisição de conhecimentos e experiência é também realçada.

Capítulo 2

Plataforma OSGi

2.1 – Modularidade Java

Modularidade é o conceito geral de dividir um sistema em módulos que funcionam de forma independente, podendo no entanto, esses componentes interagir com outros módulos de forma a executar determinadas funções. As principais motivações de modularizar um sistema são, reduzir a sua complexidade e controlar de forma mais eficaz a interacção entre subsistemas. Este conceito aplica-se no desenvolvimento de *software*, ao possibilitar a divisão de um sistema em módulos desenvolvidos individualmente, mas que comunicam entre si seguindo um modelo definido. Este tipo de separação de processos é também aplicado nas linguagens de programação orientadas a objectos, mas numa escala menor.

Nesta secção analisa-se a aplicação deste conceito na linguagem Java.

2.1.1 – Ficheiros JAR

Um ficheiro JAR [5], é no seu aspecto mais geral um ficheiro único que engloba uma série de outros ficheiros, tal como um ficheiro zip. Este ficheiro contém o directório opcional META-INF. Na maioria dos casos, este directório é de especial importância, nomeadamente para a plataforma OSGi. O directório META-INF é utilizado para configurar aplicações, extensões e serviços. Os ficheiros normalmente presentes são:

- MANIFEST.MF: Ficheiro utilizado para descrever conteúdo do ficheiro JAR.
- INDEX.LIST: Contém informação sobre pacotes definidos nas aplicações ou extensões.
- x.SF: Ficheiro de assinatura para o JAR a que se refere.
- x.DSA: Este ficheiro contém a assinatura digital do ficheiro de assinatura a que se refere.
- Services/: Directório que contém informações sobre o fornecedor do serviço.

Dos itens referidos acima, destaca-se o ficheiro *manifest*. Este contém uma série de entradas com regras e restrições especificadas. Os principais atributos são apresentados de seguida:

- Manifest-Version: Entrada que define a versão do ficheiro *manifest*.
- Created-By: Define a versão e o fabricante da implementação utilizada na compilação do ficheiro.
- Signature-Version: Especifica a versão da assinatura do JAR.
- Class-Path: Este valor especifica a localização das extensões ou bibliotecas necessárias pela aplicação ou extensão.
- Main-Class: Define a classe a ser invocada mediante a execução do ficheiro JAR.

Os atributos acima referidos são complementados por outros específicos das aplicações contidas no JAR. O arquivo JAR e o ficheiro *manifest* proporcionam um mecanismo útil para agregar ficheiros e informação relacionada.

2.1.2 - Pacotes

No desenvolvimento em Java, os programas são organizados em pacotes [6]. Estes, são conjuntos de classes relacionadas e normalmente cooperantes. A primeira entrada de uma classe é a definição da *package* a que pertence, como por exemplo:

- package pt.ubi.service:

O facto de se agrupar as classes em pacotes permite uma maior facilidade na administração dos componentes, tornando acessível a reutilização de código ao permitir o acesso de outros programas a classes de determinado pacote e permite a definição de nomes de classes único para evitar conflitos. Estes conflitos são evitados, através da convenção de nomes especificada pela Sun, em que o início do nome do pacote é dado pela ordem inversa do nome do domínio da instituição da Internet. Os pacotes desempenham um papel importante na aplicação de modularidade no Java.

Os pacotes também interferem na partilha de informação. As classes e interfaces contidas numa mesma *package* têm acesso a atributos e métodos públicos. Este facto, permite agrupar classes com funcionalidades relacionadas.

2.1.3 – Carregamento de classes

Ao contrário do que acontece com as linguagens C e C++, em que depois do código ser compilado é necessário executar o processo de associar a funções de biblioteca, em Java, esta tarefa é realizada pela JVM. Esta operação é efectuada aquando do carregamento das classes na memória. As regras utilizadas no carregamento de classes por parte da JVM, está especificada em detalhe em [7]. Na linguagem Java, os *class loaders* são os componentes responsáveis pela ligação dinâmica e pelo carregamento do *bytecode* das classes.

Existem 2 tipos de *class loaders*: os definidos pelo utilizador e o *bootstrap class loader* fornecido pela JVM. Todos os que são definidos pelo utilizador são subclasses de java.lang.ClassLoader. Esta classe possui uma série de métodos que permitem encontrar, executar e disponibilizar para ligação as classes necessárias. Os *class loaders* organizam-se de forma hierárquica, em que o *bootstrap class loader* é a raiz da árvore, como apresentado na figura 2.1.

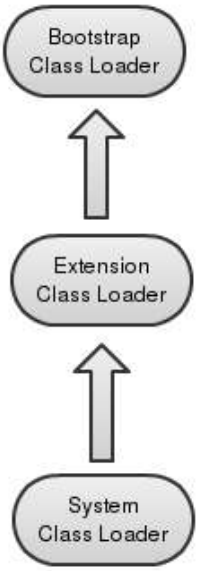

Figura 2.1 – Estrutura típica do *class loader*. Adaptado de [7].

O *bootstrap class loader* é responsável por carregar todas as classes presentes no directório de onde são obtidas as classes, o *Bootclasspath*. Estas classes *core* do Java, são as classes confiáveis do sistema e não são alvo do processo de validação típico das classes não confiáveis. No entanto, este não é o único *class loader* definido. A JVM, define também, o *extension class loader* e o *system class loader*. O *extension class loader* é responsável por incluir as classes das extensões *standard* do Java e o *system class loader* disponibiliza todas as classes incluídas no *classpath*. Cada *class loader* mantém uma referência do *parent class loader*, como definido pelo modelo de delegação do Java 2.

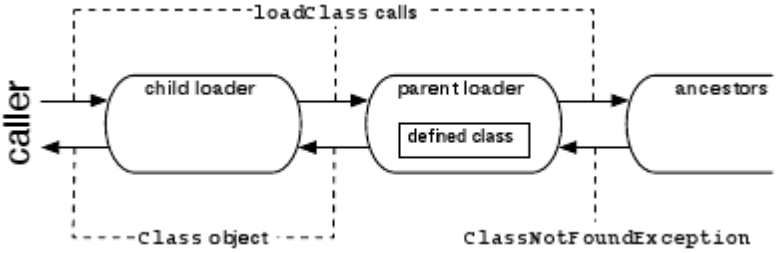

Figura 2.2 – Modelo de delegação do *class loader* [8].

O modelo apresentado na Figura 2.2, refere-se a um modelo hierárquico. No entanto, também existem modelos de delegação não hierárquicos, como por exemplo, o modelo de delegação do OSGi. Resumidamente, o ficheiro JAR, contém meta data referente às dependências e contribuições entre os vários componentes.

2.1.4 – Estado da Arte

A comunidade de desenvolvimento do Java tem-se esforçado no sentido de incorporar outros mecanismos de modularidade para além dos referidos acima. Neste contexto, foram propostos 2 Java *Specification Requests* para integração com o Java 7, JSR 277 e JSR 294.

2.1.4.1 – JSR 277: *Java Module System*

JSR 277 procura melhorar certos aspectos dos mecanismos de *packaging*, *versioning* e *deployment*. Resumidamente tem os seguintes objectivos:

- Um novo formato de distribuição de código Java, recursos e meta data numa unidade apelidado Java Module. A informação adicional inclui informações sobre o módulo, os seus recursos e as suas dependências.
- Um esquema de versões que define como um módulo deve declarar a sua versão e as dependências relativas a versões de outro módulos.
- Um repositório para armazenamento e recuperação de módulos.

2.1.4.2 – JSR 294: *Improved Modularity for Support in the Java Programming Language*

Esta especificação foca-se no conceito de superpacotes, um conjunto de pacotes membro que permitem o acesso a métodos membro de pacotes membro. O conceito procura evitar a declaração de elementos necessários a outras sub-partes do programa a ser desenvolvido como públicos, estando assim acessíveis globalmente.

Ambas as especificações oferecem um nível de modularidade baixo em relação ao OSGi, sendo este o sistema modular mais maduro para Java, que conta já com 11 anos de desenvolvimento e que irá ser descrito com detalhe no capítulo seguinte. O mecanismo evoluído de resolução de dependências e a meta data mais completa por parte do OSGi, em comparação com o JSR 277, provocou uma certa polémica ao apelidar este último, de sistema modular para Java. Em relação ao JSR 294, a funcionalidade proposta funciona de forma semelhante à disponibilização de pacotes públicos por parte dos *bundles*.

2.2 – Descrição Plataforma OSGi

Inicialmente a plataforma OSGi foi projectada para utilização num ambiente residencial, sendo que a primeira versão da especificação se focava na implementação num *gateway* residencial. Este primeiro conjunto de especificações baseadas em Java permitiam interligar equipamentos existentes na residência do utilizador, efectuando a ligação à Internet com recurso a sistemas de segurança. No entanto, nesta primeira *release* já se encontravam implementados os principais predicados desta *framework*.

Um modelo de componentes simples, um registo de serviços e suporte para desenvolvimento de aplicações são algumas das características gerais, que tornaram possível a migração desta plataforma para outros sectores. A plataforma foi gradualmente evoluindo no sentido de um modelo mais genérico orientado aos serviços com o objectivo de viabilizar, interligar e gerir uma série de serviços nos mais diversos ambientes. A sua recente aprovação como JSR 291 na Java Community Process e a sua implementação no ambiente de desenvolvimento Eclipse, tornaram inevitável a crescente procura de desenvolvimento de soluções que usufruam das enormes potencialidades da plataforma. Actualmente, é utilizada por um grande número de operadores e fornecedores de serviços, permitindo efectuar a gestão remota de serviços dos mais diversos tipos tendo em conta a natureza do utilizador. Estes serviços não são apenas executados em computadores mas também em dispositivos móveis e equipamentos electrónicos domésticos.

A plataforma OSGi é também denominada de sistema modular dinâmico para Java. A modularidade será a sua principal característica e que a torna particularmente útil. Os componentes, apelidados de *bundles*, nada mais são que ficheiros JAR que possuem determinadas características:

- *Self-Contained*: constituído por pequenas partes e possui a capacidade de se instalar, remover e deslocar.
- *Highly Cohesive*: cumpre a função para a qual é destinado.
- *Loosely Coupled*: independente da implementação interna.

A modularidade implementada pela plataforma, traz inúmeras vantagens:

- **Redução complexidade**: Cada *bundle* é desenvolvido de forma independente e apesar de existir a possibilidade de interacção entre *bundles*, esta comunicação segue regras definidas pela plataforma.
- **Manutenção**: Devido à independência dos *bundles*, estes podem ser modificados, instalados, eliminados sem que isso afecte o sistema ou outros *bundles*.
- **Reutilização**: É possível reutilizar *bundles* sem alterações significativas e integrar *bundles* de outras entidades.
- **Fácil desenvolvimento**: A plataforma define métodos de instalação e gestão dos diferentes módulos.
- **Adaptativo**: Os *bundles* utilizam o serviço de registo dinâmico para se registarem. É através deste registo que os *bundles* tomam conhecimento dos *bundles* e serviços presentes no sistema.

Nas secções seguintes, a plataforma é apresentada em detalhe com a descrição das suas principais funcionalidades e vantagens subsequentes. Por fim são apresentadas aplicações desenvolvidas sobre a *framework*.

2.2.1 – Arquitectura OSGi

A plataforma OSGi apresenta o modelo de camadas representado na figura 2.3:

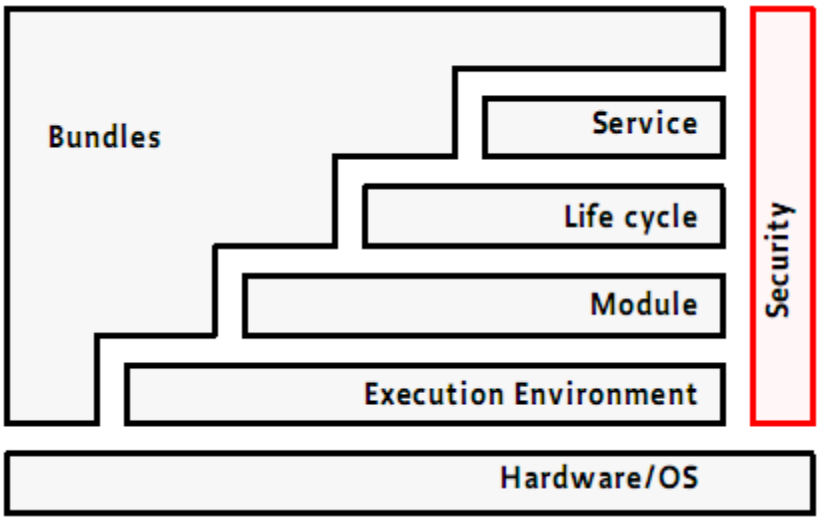

Figura 2.3 – Modelo de camadas da especificação OSGi [9].

- **Bundles**: *Bundles* são os componentes da plataforma OSGi. O *bundle* pode usufruir de diversos serviços das várias camadas, como por exemplo, serviços de segurança e gestão do ciclo de vida.
- **Serviços**: A camada de serviços define um modelo dinâmico de programação Java para os fornecedores de *bundles*, que procura facilitar o desenvolvimento de *bundles* de serviços ao separar a especificação do serviço, da sua implementação.
- **Ciclo de vida**: Esta camada fornece serviços de *runtime* aos *bundles*. Estes serviços permitem parar, iniciar, instalar, actualizar e remover *bundles* da *framework*.
- **Módulos**: Define o modelo de modularização. São estabelecidas as regras que permitem a partilha de Java *packages* entre *bundles*. As funções de *export* e *import* de código são mantidas nesta camada.
- **Segurança**: Define um formato seguro de *packaging* e interage com a camada de segurança do Java 2.
- **Ambiente de execução**: Define quais os métodos e classes disponíveis numa plataforma específica.

A figura 2.4 apresenta a interacção entre as diferentes camadas.

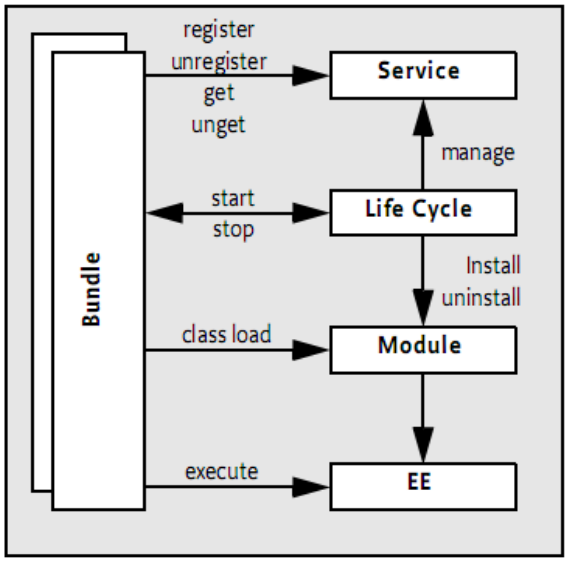

Figura 2.4 – Interacção entre camadas da especificação OSGi [9].

2.2.2 – Camadas da plataforma OSGi

Neste ponto são apresentadas as diferentes camadas que constituem a plataforma OSGi com mais detalhe.

2.2.2.1 – Camada de segurança

Como referido anteriormente esta camada é fortemente baseada na arquitectura de segurança do Java 2. Neste contexto, a plataforma Java deverá fornecer as interfaces de segurança para as permissões do Java 2. As classes *permission* têm como objectivo adicionar segurança aos recursos, protegendo-os de acções indevidas. A implementação OSGi separa as classes *permission* em:

- *Package Permission*: As permissões são atribuídas aos *bundles* e definem o controlo de acesso ao código (*import* e *export* de *bundles*).
- *Service Permission*: Os serviços são protegidos de acessos indevidos por parte de certos *bundles*.

2.2.2.2 – Camada de modularização

A plataforma Java tem sérias limitações em relação à execução de diferentes aplicações, pois estas requerem um processo diferente na JVM. Assim a partilha de classes entre as aplicações torna-se impossível. A *framework* OSGi é executada através de uma única instanciação da JVM, o que permite aos *bundles* aceder a classes do ambiente de execução e até de outros *bundles*. No entanto, estas regras de *class loading* estão bem definidas na especificação. Os *bundles* podem partilhar ou esconder *packages* de outros *bundles*. O mecanismo *class loader* de cada *bundle* é responsável por essa funcionalidade. O modelo de delegação entre *class loaders* é apresentado na figura 2.5.

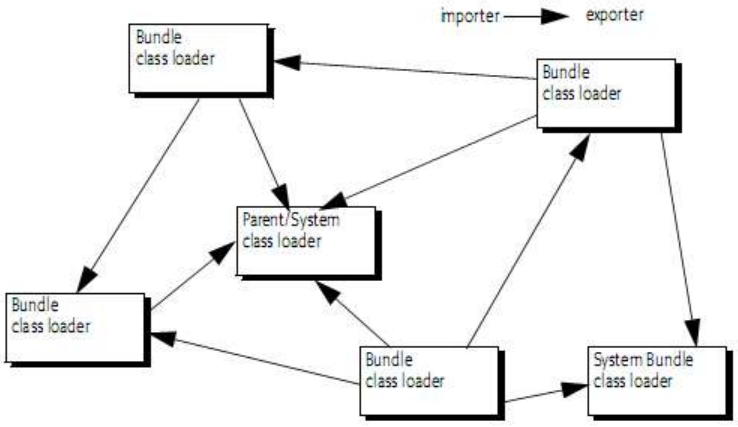

Figura 2.5 – Modelo de delegação entre *class loaders* da especificação OSGi [9].

Um *bundle*, pode então, aceder a:

- Classes do próprio *bundle*: Conjunto de classes presentes no Bundle-Classpath.
- Classes de outros *bundles*: Conjunto de classes partilhadas.
- Classes da *framework*: Conjunto de classes implementadas da *framework*, do tipo *org.osgi.framework*.
- Classes da plataforma Java: Conjunto de *java.* packages* e suas *packages* de implementação.

Cada *bundle* passa então a ter um número de classes ao seu dispor, conjunto esse denominado de *class space*, como representado na figura 2.6. A *framework* tem então que criar as ligações entre os componentes importadores e exportadores, sendo que este processo tem de ser efectuado antes da execução do *bundle*.

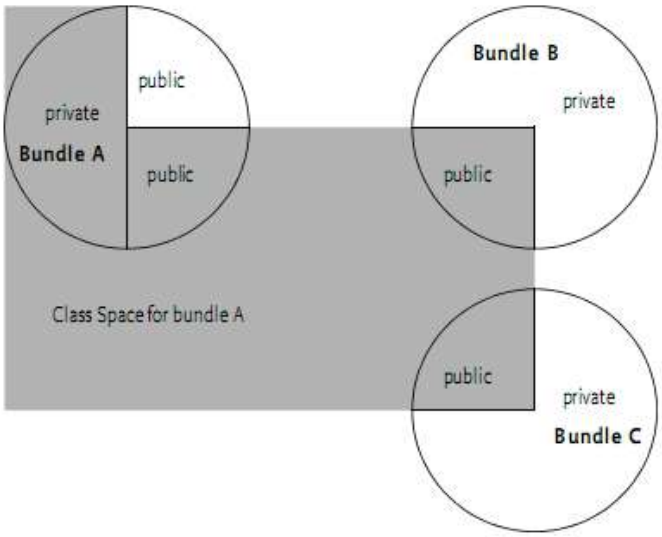

Figura 2.6 – Espaço de classes de um *bundle* [9].

Um *bundle* não tem acesso no momento da sua execução às classes *do system class loader*. Sendo assim, todas as packages não pertencentes a java.* têm de ser importadas explicitamente. Por exemplo, a package Java.xml.parsers está disponível em alguns ambientes sendo que o *bundle* de sistema exporta a mesma para a framework. No entanto, os *bundles* continuam a ter de importar os *packages*.

2.2.2.2.1 - Resolução do *bundle*

Um *bundle* tem um ciclo de vida descrito na secção 2.2.2.3, onde se verifica a transição de estados a que está sujeito. Depois de instalado, o módulo passa por um processo de resolução, que implica que todas as classes e recursos necessários à sua correcta execução estejam disponíveis na plataforma. A *framework* permite que o *bundle* defina os seguintes tipos de dependências:

- Dependências de Classpath: o *bundle* utiliza um conjunto de *packages* disponibilizado como ficheiros JAR.
- Dependências de código nativo: o *bundle* utiliza métodos de bibliotecas nativas. Estas devem estar contidas nos recursos do *bundle*.
- Dependências de *packages*: o *bundle* requer classes de outros *bundles*. Estas dependências só são resolvidas se as *packages* necessárias forem exportadas para o ambiente de execução da *framework*.

2.2.2.2.1.1 - Dependências de Classpath

O parâmetro Bundle-ClassPath permite ao *bundle* expandir o *Classpath* disponibilizado com um ou mais ficheiros JAR, separados por vírgula. Se não especificado, o *Classpath* é o directório *root* do ficheiro JAR. Para ser definido correctamente, deve ser utilizada a seguinte sintaxe:

*Bundle-ClassPath: path (, path)**
path: string <"/"-caminho que identifica o ficheiro JAR>

Um exemplo de utilização:

Bundle-ClassPath: com/sun/jes/impl/http/servlet.jar

2.2.2.2.1.2 - Dependências de código nativo

A entrada *Bundle-NativeCode* permite definir a localização de uma biblioteca de código nativo dentro do *bundle*. A sintaxe é a seguinte:

```
Bundle-NativeCode ::= nativecode
    ( ',' nativecode )* ( ',' optional) ?
nativecode ::= path ( ';' path )*
                ( ';' parameter )+
optional ::= '*'
```

Um exemplo de utilização:

```
Bundle-NativeCode: lib/http.dll ; lib/zlib.dll ;
    osname = Windows95 ;
    osname = Windows98 ;
    osname = WindowsNT ;
    processor = x86 ;
    selection-filter=
      "(org.OSGi.framework.windowing.system=win32)";
    language = en ;
    language = se ,
  lib/solaris/libhttp.so ;
    osname = Solaris ;
    osname = SunOS ;
    processor = sparc,
  lib/linux/libhttp.so ;
    osname = Linux ;
    processor = mips;
    selection-filter
      = "(org.OSGi.framework.windowing.system = gtk)"
```

No exemplo acima, observa-se a definição de diversos parâmetros:

- Osname: Sistema operativo onde a biblioteca é executável.
- Processor: Arquitectura do processador que executa a plataforma.
- Language: Idioma.
- Osversion: Versão do sistema operativo.
- Selection Filter: Filtro de selecção que indica a entrada deve ser seleccionada ou não.

A *framework* utiliza o seguinte algoritmo para a correcta instalação do *bundle*:

1. Escolhe a entrada que coincide com todas as propriedades referidas acima, através da análise das propriedades do sistema:
   - org.osgi.framework.processor
   - org.osgi.framework.os.name

- org.osgi.framework.osversion
- org.osgi.framework.language

Em relação ao *Selection-Filter*, este é verdadeiro se a entrada contiver as propriedades de sistema ou não for especificada. Se nenhuma opção coincidir com ambas as propriedades, a instalação do *bundle* falha e ocorre uma excepção do tipo *BundleException*.

2. As entradas seleccionadas são organizadas pela seguinte ordem de prioridade:
   - *Osversion*: versões em ordem descendente até versão não especificada.
   - *Language*: idioma especificado até idioma não definido.
   - Posição na entrada *Bundle-NativeCode*: da esquerda para a direita.
3. A primeira entrada disponível depois de efectuado o ponto 2.

Se nenhuma biblioteca de código nativo for encontrada de acordo com as entradas disponíveis no *manifest* ocorre uma excepção de *bundle*, o que implica uma instalação sem sucesso.

2.2.2.2.1.3 - Dependências de pacotes

Um *bundle* declara *packages* necessárias ao seu correcto funcionamento em *Import-Package* e disponibiliza classes para a plataforma através de *Export-Package* no ficheiro *manifest*.

2.2.2.2.1.3.1 - Exportação de pacotes

Um *bundle* tem a possibilidade de tornar acessíveis classes para outros *bundles*. A plataforma garante que o *bundle* exportador carrega as classes e que as *packages* são importadas pelo *bundle* solicitador. No caso de existir mais que um *bundle* a requerer as *packages* exportadas, a *framework* deve escolher o módulo com a versão mais recente da *package*. O campo *Export-Package* cumpre a seguinte sintaxe:

```
Export-Package ::= export ( ',' export )*
export         ::= package-names ( ';' parameter )*
package-names  ::= package-name
                   ( ';' package-name )*
```

Em exemplo, tem-se:

Export-Package: com.acme.foo; com.acme.bar; version=1.23

De referir que para o *bundle* ter a capacidade de exportar classes, deve definir a propriedade *PackagePermission* para *export*.

2.2.2.2.1.3.2 - Importação de pacotes

Esta entrada deve seguir a seguinte sintaxe:

*Import-Package ::= import (',' import)**
*import ::= package-names (';' parameter)**
package-names ::= package-name
*(';' package-name)**

Esta propriedade permite ao *bundle* aceder a classes presentes no *Classpath* da plataforma. No entanto as classes importadas têm de ser anteriormente exportadas por outro *bundle*. Caso isso não se concretize, o *bundle* instalado não poderá ser iniciado.

2.2.2.2.2 – Carregamento de classes em tempo de execução

Após o *bundle* transitar para o estado RESOLVED, a plataforma é responsável por criar um *class loader*. Este componente permite o acesso a todos os recursos no pacote por parte de objectos que pertencem à mesma *package* e permite a partilha de classes entre *bundles*. Quando o *bundle* requer o carregamento de uma classe, este é feito por uma ordem específica, de acordo com a figura 3.5.

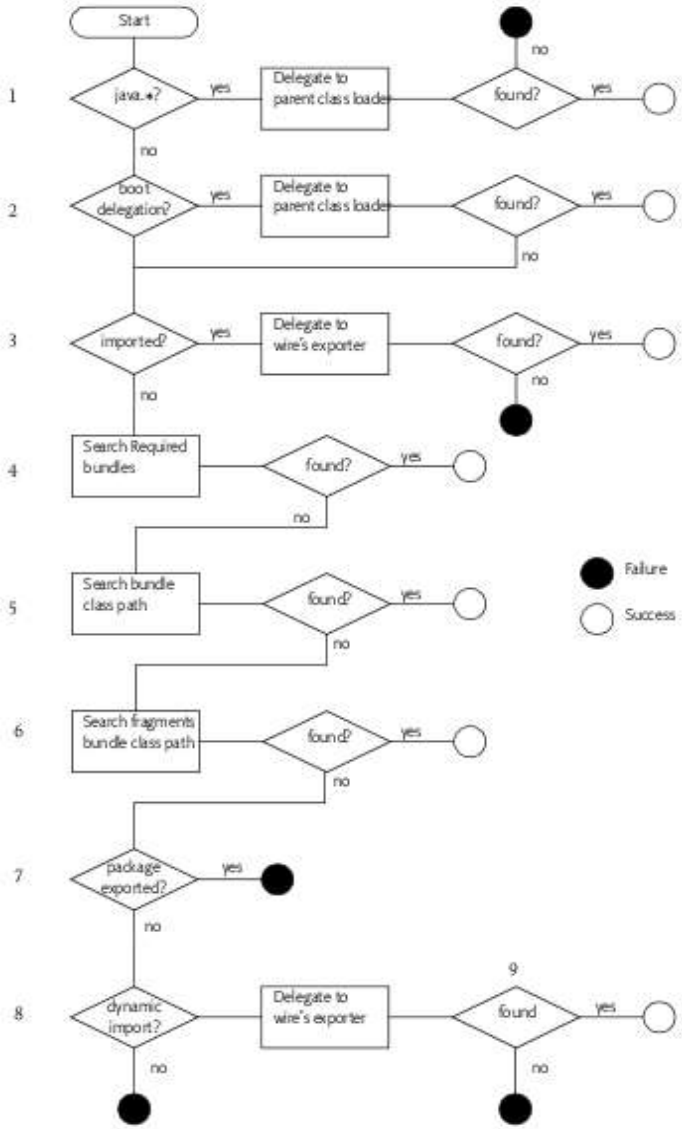

Figura 2.7 – Fluxograma do carregamento de classes em tempo de execução [9].

Existem dois conceitos necessários à compreensão do diagrama:

- *Parent Class Loader*: A plataforma deve definir obrigatoriamente um *parent class loader* para carregar as classes Java.*
- *Boot Delegation*: A plataforma permite definir um conjunto de classes a serem delegadas ao *parent class loader*. Por omissão, a propriedade org.osgi.framework.bootdelegation não está definida. No entanto, pode ser especificada de acordo com a seguinte sintaxe:

  org.osgi.framework.bootdelegation ::= boot-description
  *(',' boot-description)**
  boot-description::= package-name
  | (package-name '.')*
  | ''*

2.2.2.2.3 – Carregamento de recursos

A plataforma permite para além do acesso ao *class loader* do *bundle*, fornece uma API para acesso de recursos disponíveis no *bundle*. Os métodos da API retornam um objecto URL ou uma enumeração de objectos URL. Estes objectos URL são criados pela plataforma, mas por vezes utilizam esquemas próprios dos *bundles*. No entanto, são especificados de forma a permitirem acesso relativo ao conteúdo do *bundle*.

2.2.2.3 – Camada do ciclo de vida

Um ficheiro JAR é um conjunto de pacotes e outros recursos. Esse ficheiro JAR contém meta data no directório META-INF, de onde se destaca o ficheiro MANIFEST.MF. Este agrega informação sobre os pacotes e recursos contidos, bem como sobre as classes usadas e, caso se aplique, a classe que contém o método principal. Tal como um ficheiro JAR, um *bundle* também possui estes componentes, embora com alguns recursos adicionais. O ficheiro MANIFEST.MF, especifico para inclusão nos *bundles* OSGi, pode ser composto por:

- **Bundle-Name**: É atribuído um nome ao *bundle*, preferencialmente curto. Este nome será utilizado nas consolas de gestão das *frameworks* utilizadas.
- **Bundle-SymbolicName**: Um identificador único obrigatório, baseado na convenção de nomes dos *Java packages*, que especifica que o nome é atribuído pela ordem inversa do nome do domínio da instituição da Internet. Em conjunto com o *Bundle-Version*, permite identificar de forma única o *bundle* na *framework*.
- **Bundle-Description**: Breve descrição da função do *bundle*.
- **Bundle-ManifestVersion:** Define a especificação OSGi a ser usada para a leitura do *bundle*. O valor 1 é utilizado na versão 3 e anteriores, e o valor 2 na versão 4. A versão mais recente da plataforma OSGi é a 4.2.
- **Bundle-Version**: Especifica a versão do *bundle*. As versões podem conter 4 partes, sendo que 3 delas são dígitos e a restante pode incluir caracteres. Se o parâmetro não está definido, a versão 0.0.0 é assumida por omissão.
- **Bundle-Activator**: Classe a ser invocada aquando da execução do *bundle* de acordo com o ciclo de vida do *bundle*.
- **Bundle-Vendor**: Fabricante ou entidade responsável pela criação do *bundle*.
- **Bundle-Classpath**: Parâmetro que define o local no *bundle* onde se encontram os *resources*. Caso não seja especificado, o directório raiz do *bundle* é assumido.

- **Bundle-RequiredExecutionEnvironment**: Propriedade para definição dos ambientes de execução necessários para a operação do *bundle*.
- **Bundle-NativeCode**: Parâmetro para definir a localização de bibliotecas nativas para integração *Java Native Interface*. Outros parâmetros podem ser definidos relativamente à descrição das plataformas de execução.
- **Export-Package**: Especifica o conjunto de classes a serem disponibilizadas para a *framework*, separadas por vírgulas. Todos os pacotes possuem versão, se não definida, assume 0.0.0.
- **Import-Package**: Expressa as dependências do *bundle*. Os pacotes definidos neste parâmetro devem coincidir com as *packages* definidas no *Export-Package* de outro *bundle*. Permite definir intervalos de versões para as *packages* necessárias. Permite especificar como opcional determinada dependência.
- **Dynamic-ImportPackage**: Especifica pacotes extern0s implicitamente opcionais. Apenas são importadas mediante solicitação de utilização. Permite a utilização de *wildcards* "*" para importação de um conjunto de classes.
- **Required-Bundle**: Semelhante à utilização do *Import-Package* mas refere-se ao nome simbólico de outro *bundle*.

Toda esta informação é necessária, para que a plataforma efectue a gestão do ciclo de vida de um *bundle*. Um *bundle*, pode então passar por vários estados:

- **INSTALLED**: O *bundle* foi instalado correctamente.
- **RESOLVED**: As classes Java necessárias estão disponíveis, podendo o *bundle* ser parado ou iniciado.
- **STARTING**: O *bundle* encontra-se a iniciar, sendo que a função a ser chamada ainda não retornou.
- **ACTIVE**: A função presente no *bundle.activator* retornou, pelo que o *bundle* se encontra em execução.
- **STOPPING**: O *bundle* está a ser desactivado.
- **UNINSTALLED**: O *bundle* foi removido.

O ciclo de vida de um *bundle*, é caracterizado pelo diagrama de estados representado na figura 2.8:

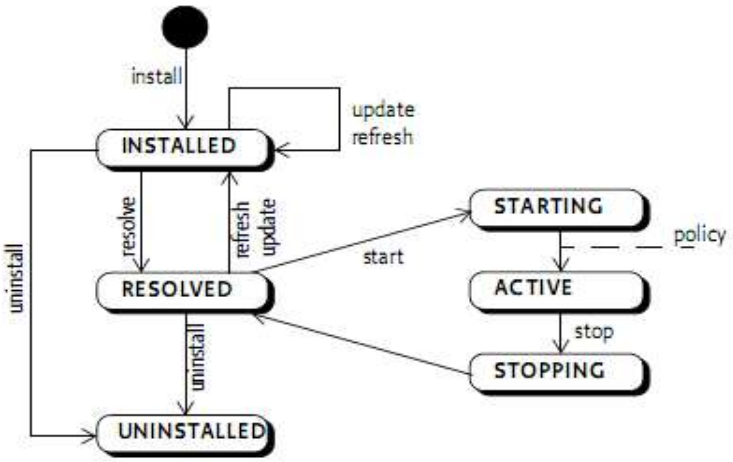

Figura 2.8 – Diagrama de estados de um *bundle* [9].

2.2.2.3.1 - Instalação

Instalação é o processo de disponibilizar o *bundle* na plataforma. O *bundle* é validado de acordo com os requisitos da plataforma e é identificado de forma única através de um identificador.

2.2.2.3.2 - Resolução

Este processo foi descrito com mais detalhe na secção 2.1.2.2.1. O *bundle* encontra-se no estado RESOLVED depois de todas as suas dependências terem sido satisfeitas.

2.2.2.3.3 - Inicialização

A entrada *Bundle-Activator* presente no *manifest* determina a instanciação da classe responsável por iniciar o *bundle*. Esta classe que implementa a interface *BundleActivator* contém o método start(), que executa as acções necessárias ao funcionamento do *bundle*, como por exemplo, alocar recursos, registar serviços, etc. Após a execução correcta do método, o *bundle* transita para o estado ACTIVE.

2.2.2.3.4 - Terminação

A interface *BundleActivator* implementa também o método stop(). Neste processo, o *bundle* deve libertar recursos e terminar threads activas, caso se aplique. A plataforma garante a desactivação de serviços registados na activação do *bundle* e permitir a correcta inicialização caso seja necessária. Após a execução correcta do método, o módulo encontra-se no estado RESOLVED.

2.2.2.3.5 - Actualização

A plataforma permite a actualização do *bundle* sem ser necessário reiniciar ou instalar novamente o *bundle*. Os serviços ou recursos previamente disponíveis são acedidos através da cache da plataforma, até que o método refreshPackages() seja invocado.

2.2.2.3.6 - Desinstalação

Este processo provoca a propagação de notificações para os *bundles* que o *bundle* alvo do processo está a ser desinstalado. A plataforma remove todas as ligações com o *bundle* e desactiva todos os seus recursos disponibilizados. No caso em que o *bundle* exporta classes, a plataforma garante a continuidade da disponibilização do serviço até que uma das seguintes condições se verifique:

- O método refreshPackages seja invocado.
- A plataforma seja reiniciada.

2.2.2.4 – Camada de serviços

A plataforma OSGi é caracterizada pela sua natureza dinâmica. Os *bundles* e serviços instalados na *framework* podem variar muito rapidamente, portanto, a plataforma tem de oferecer mecanismos que permitam que estas constantes alterações sejam efectuadas de forma correcta.

Um serviço é simplesmente um objecto Java registado sob uma ou mais interfaces Java através do *Service Registry*. Desta forma, um *bundle* pode registar, procurar e utilizar serviços ao invocar métodos do seu *BundleContext*.

2.2.2.4.1 – Registo de serviços

Um *bundle* difunde um serviço através do registo no *Service Registry*. A plataforma cria um objecto *service* que fica exposto a outros *bundles*, que contém um *ServiceRegistration* único e um ou mais *ServiceReference*. Um *ServiceReference* é um objecto que contém meta data e propriedades do serviço a que se refere. Um *ServiceRegistration* pode ser comparado a uma factura, em que para modificar as propriedades ou remover o serviço é necessário a utilização do objecto, tal como, em caso de troca ou devolução a factura tem de ser apresentada. A plataforma permite que um *bundle* registe serviços de forma dinâmica, isto é, nos estados STARTING, ACTIVE e STOPPING.

Resumidamente, um *bundle* regista um serviço ao aceder aos métodos BundleContext.registerService(), que corresponde à acção de *publish* na figura 2.9. Um *bundle* encontra um serviço ao solicitar um *ServiceReference* que coincida com as propriedades publicadas, ou seja, procura o serviço. Por fim, a interacção acontece quando o *bundle* requer o serviço através de BundleContext.getService().

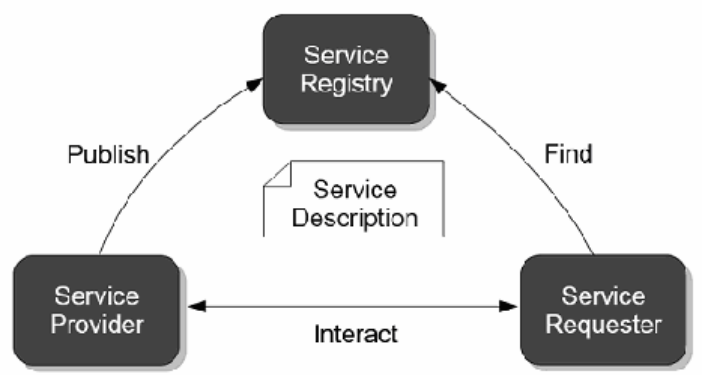

Figura 2.9 – Modelo de interacção do registo de serviços [10].

2.2.2.4.2 – Eventos de serviços

A plataforma gera eventos quando um serviço é registado. Eventos que podem ser escutados por *bundles* que pretendam usufruir do serviço e que forcem informações sobre o estado de disponibilidade do serviço. Os objectos fornecidos pela plataforma designam-se por *ServiceListeners*.

2.2.3 – Gestão remota

A proposta inicial da OSGi *Alliance* previa a especificação de uma plataforma de serviços a ser implementada num *gateway*, onde todas as operações de gestão e manutenção seriam efectuadas de forma remota por um fornecedor de serviços. Apesar da OSGi *Alliance* não definir um modelo de gestão remota rígido, com protocolos de comunicação bem especificados, outras entidades como o BroadBand Forum e HGI, desenvolveram o standard TR-069, descrito em detalhe na secção 2.4. De encontro a esse objectivo, foi proposto o seguinte modelo de gestão remota:

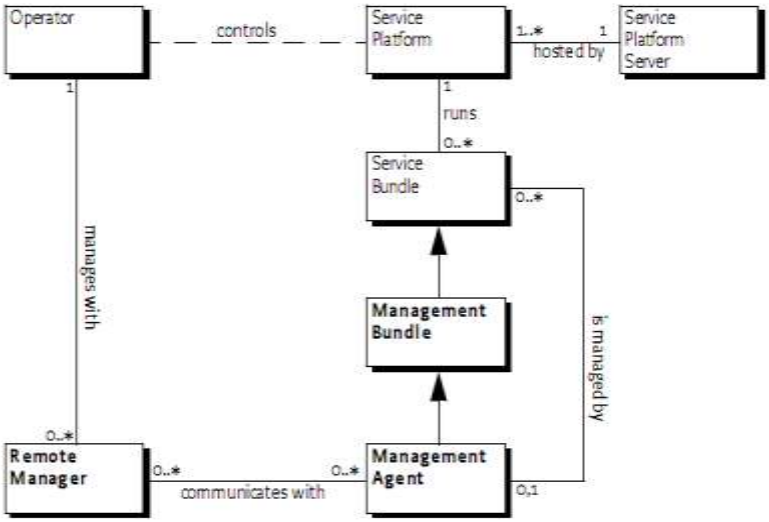

Figura 2.10 – Modelo de gestão remota proposto pela OSGi *Alliance* [11].

De acordo com a figura 2.10, as entidades envolvidas são:

- *Remote Manager*: Sistema gerido pelo operador para gestão remota da plataforma de serviços.
- *Management Agent*: Um ou mais *bundles* disponíveis na plataforma de serviços e que comunicam com o gestor remoto. *Bundles* esses que oferecem mecanismos de gestão da plataforma de serviços.
- *Management Bundle*: *Bundle* com permissões de administração, com capacidade de controlo do ciclo de vida e configurações dos restantes *bundles*.
- *Initial Provisioning*: Processo que disponibiliza um agente de gestão à plataforma de serviços.

2.2.3.1 – Gestor remoto

O gestor remoto é o sistema que expõe a interface de gestão remota da plataforma de serviços ao operador. Sendo assim, estão especificadas as suas funções:

- **Gestão do ciclo de vida**: Instalar, iniciar, actualizar, parar e remover *bundles*.
- **Gestão de segurança**: Definição das permissões para os *bundles* e tratamento de dados dos utilizadores na plataforma.
- **Gestão de configurações**: Definição das configurações da própria *framework* e dos *bundles*.
- **Gestão de falhas**: Execução de ferramentas de diagnóstico e correcção de problemas.
- **Gestão de contas de utilizadores**: Recolha de informações e envio de informações para a entidade responsável pela cobrança de serviços.
- **Gestão de performance**: Optimizar a utilização de recursos na plataforma de serviços.

2.2.3.2 – Agente de gestão

O agente de gestão, é implementado através de um ou mais módulos de gestão. O agente de gestão deve comunicar com um ou mais gestores remotos e suportar diversos protocolos de comunicação. Para executar as funções de gestão, os *bundles* responsáveis pela gestão

devem ter permissões de administração. No modelo ideal e como forma de assegurar segurança na administração da plataforma, todas as operações de manutenção e de modificação devem ser executadas pelo agente de gestão.

2.2.3.3 - Comunicações

Um aspecto fulcral no modelo de gestão acima descrito é a comunicação entre o gestor remoto e o agente de gestão. Apesar do modelo apresentado apenas definir entidades, significa que a gestão seja independente do operador e do fornecedor da plataforma de serviços.

2.2.3.3.1 - Conectividade

As plataformas de serviços são desenvolvidas em vários ambientes. Apesar de ser expectável a existência de conectividade IP, esse facto nem sempre pode ser assumido. Neste contexto, os protocolos de gestão devem ser capazes de se adaptar a uma conectividade intermitente e baixa largura de banda.

2.2.3.3.2 - Protocolos

Os agentes de gestão devem ser compatíveis com protocolos de comunicação de acesso remoto (GPRS, GSM) e a sua aplicação deverá ser independente do protocolo utilizado.

2.2.3.3.3 - Comunicações seguras

Como em qualquer tipo de comunicação, a segurança é um aspecto a ter em conta. É recomendável que os sistemas de comunicação providenciem no mínimo:

- Autenticação mutua.
- Verificação integridade de mensagens.
- Confidencialidade.

No acesso remoto por parte do gestor remoto, podem existir diversas restrições no acesso. Firewalls e endereços privados são dois exemplos. A comunicação no sentido contrário é normalmente possível e este é um aspecto a ter em conta nas implementações de gestão remota.

2.2.4 – Modelo *gateway* residencial

Dada a ideia inicial de implementação da plataforma de serviços no mercado residencial, surge na versão 3 da especificação OSGi, um modelo de referência para um *Service Gateway* residencial. O modelo da figura 2.11, distingue claramente 2 áreas: a zona do *Gateway Operator* e do *Service Gateway*.

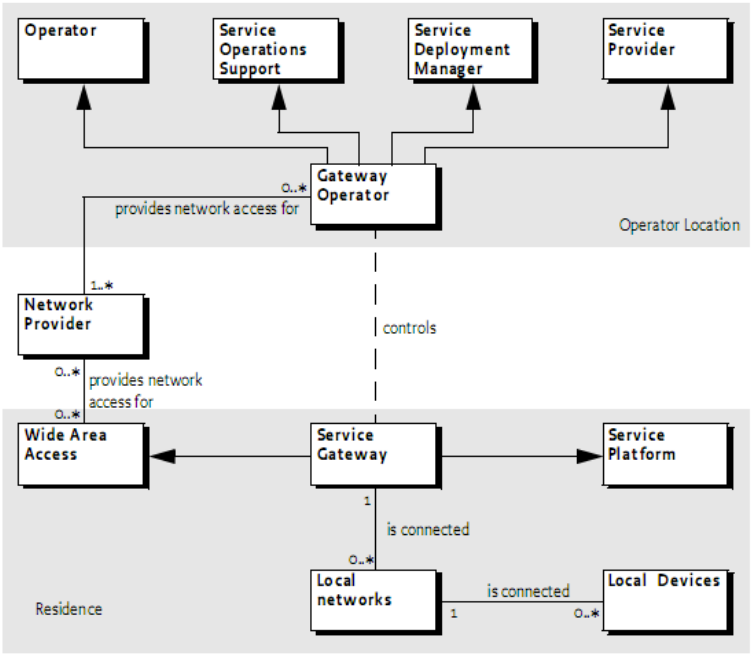

Figura 2.11 – Possível modelo de *gateway* residencial [11].

As diversas entidades presentes no modelo são detalhadas a seguir:

- **Service Gateway**: Equipamento que implementa uma plataforma de serviços OSGi. No ambiente residencial, pode desempenhar funções de *IP Gateway*. Nesse caso, o *Service Gateway* poderá interligar uma ou mais redes locais a uma rede WAN, sendo assim possível aceder a dispositivos locais. A rede WAN é disponibilizada por um operador de telecomunicações.
- **Service Platform**: Implementação de componentes de software executados sobre uma *framework* OSGi. A execução da plataforma de serviços pressupõe a instanciação de uma JVM.
- **Gateway Operator**: Entidade com a qual a plataforma de serviços comunica.
  - **Service Provider**: Fornece serviços a serem instalados e executados na plataforma de serviços.
  - **Service Deployment Manager**: Gestor de serviços provenientes de vários fornecedores.
  - **Service Operations Support**: Executa funções de suporte de serviços.
  - **Operator**: Responsável pela plataforma de serviços.

2.3 – Serviços OSGi

As especificações OSGi definem um conjunto de serviços standard (como interfaces Java), que as implementações podem disponibilizar opcionalmente como *bundles*. Em algumas implementações, determinados serviços são obrigatórios para a execução da plataforma. Esses serviços podem ser utilizados por outros *bundles* a serem executados na plataforma. A lista de serviços tem sido alargada ao longo das especificações e na versão 4.2, são cerca de 25. Os serviços disponibilizados são agrupados em diferentes categorias: serviços de sistema, serviços protocolares e serviços variados. De acordo com o contexto desta tese, apenas serão apresentados os serviços com importância no seu desenvolvimento.

2.3.1 – Serviço de Log

O serviço de registo tem como objectivo guardar eventos e erros para mais tarde serem analisados. Em sistemas como *gateways* residenciais, os dispositivos de input/output usuais, como por exemplo, teclado e monitor não estão disponíveis. Sendo assim, em caso de erro do sistema o acesso ao registo torna-se essencial para o restabelecimento do sistema. O serviço consiste em dois serviços:

- LogService: Permite aos *bundles* registar erros, informações, eventos, etc.
- LogReaderService: Permite o acesso aos registos criados pelo serviço anterior.

Este serviço é implementado segundo o diagrama de classes representado na figura 2.12:

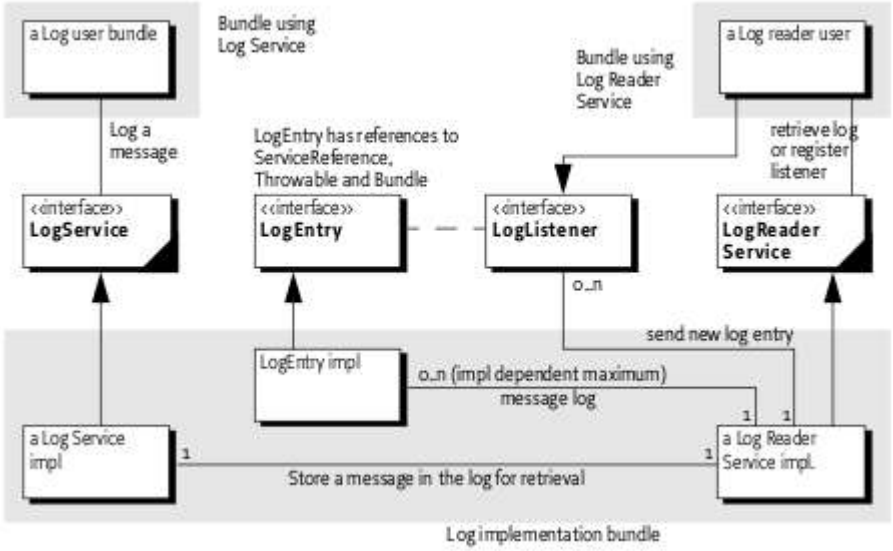

Figura 2.12 – Diagrama de classes do serviço de Log [11].

As entidades presentes são:

- LogService: A interface do serviço que permite a um *bundle* registar informações.
- LogEntry: Interface que permite o acesso uma entrada do registo. O registo inclui toda a informação gerada pelo LogService e um *timestamp*.
- LogReaderService: A interface do serviço que permite o acesso às entradas mais recentes dos registos e a implementação de um *listener* que recebe as entradas dos registos no momento em que são criadas.
- LogListener: Interface para o *listener* para objectos LogEntry. Tem de ser registado com o LogReaderService.

2.3.2 – Serviço de configuração de administração

Este serviço permite o envio de informações para *bundles* activos na plataforma, através da utilização do Java *Access Control Context* e do *Security Manager*. Na figura 2.13, observa-se que determinadas configurações do *bundle* não são especificadas no seu desenvolvimento, sendo necessária uma posterior configuração através deste serviço.

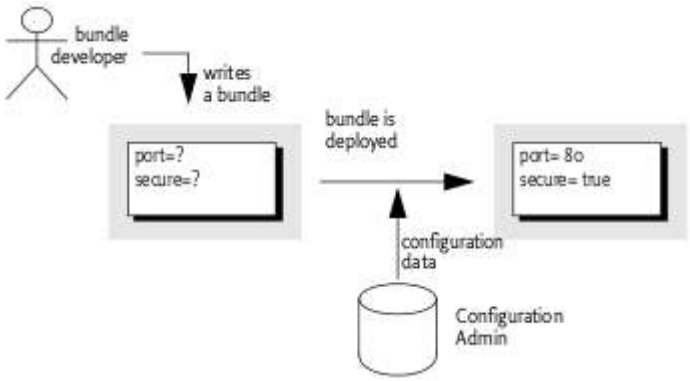

Figura 2.13 – Exemplo de utilização do serviço de configuração de administração [11].

2.3.3 – Serviço de Tracker

Tal como referido anteriormente, a plataforma fornece um ambiente de execução bastante dinâmico, com mudanças constantes no estado dos *bundles*. Apesar da intervenção da plataforma na resolução de dependências, os *bundles* são os principais responsáveis pelo correcto tratamento das mesmas. Neste contexto, a plataforma permite uma verificação constante do estado dos *bundles* e dos serviços registados no *Service Registry*, para prevenir problemas de execução. A especificação define 2 serviços: *ServiceTracker* e *BundleTracker*.

As operações realizadas por um *tracker*, seja de que tipo for são:

- Definição dos *targets*: objectos (*bundle* ou serviço) escutados pelo serviço.
- Escuta dos eventos relativos ao estado dos *targets*, de forma a proporcionar um *tracking* correcto.
- Permitir a configuração da escuta por parte do *bundle* cliente dos serviços/*bundles* e executar acções caso alterações ocorram nos serviços/*bundles*.

Um *ServiceTracker* escuta os *ServiceEvents* relativos ao serviço que coincide com os critérios definidos e pode ser personalizado através do *ServiceTrackerCustomizer*. O *BundleTracker* escuta *BundleEvents* e gera notificações caso o estado dos *bundles* se modifique.

2.3.4 – Serviço Http

Desde as primeiras especificações, que a plataforma de serviços OSGi foi desenvolvida para permitir o acesso a serviços através da Internet. Esse acesso executado geralmente através de um *browser*, permite solicitar informação, executar acções de controlo sobre serviços, etc. Dessa forma, o serviço HTTP permite a comunicação empregando diversos standards como HTML, XML e *Servlets*. O HttpService suporta 2 formas de suporte para os standards referidos:

- Registo de *Servlets*: A plataforma regista *servlets*, que permitem gerar conteúdo dinâmico de acordo com as acções do utilizador. Um *servlet* é um objecto Java que implementa a API Java Servlet [12].
- Registo de recursos: Registo de recursos estáticos, tais como, ficheiros HTML, imagens, ficheiros Javascript.

A plataforma não define a versão da implementação do serviço Http, portanto podem ser utilizadas:

- HTTP 1.0 Specification RFC-1945 [13].
- HTTP 1.1 Specification RFC-2068 [14].

Este serviço foi desenvolvido para funcionamento conjunto com a Java Servlet API, o que requer que a implementação do HttpService tenha de suportar pelo menos a versão 2.1 da referida API. Para a implementação deste serviço, a plataforma define as seguintes interfaces:

- HttpContext: Permite aos *bundles* acederem a informação sobre o registo de *servlets* e recursos. São disponibilizados os seguintes métodos:
  - getMimeType(String name)
  - getResource(String name)
  - handleSecurity(HttpServletRequest req, HttpServletResponse res)
- HttpService: Permite que outros *bundles* activos na plataforma registem dinamicamente recursos e *servlets*, fornecendo os seguintes métodos:
  - createDefaultHttpContext()
  - registerResources(String alias, String name, HttpContext context)
  - registerServlet(String alias, Servlet servlet, Dictionary initparams, HttpContext context)
  - unregister(String alias)
- NamespaceException: Excepção que ocorre caso exista uma tentativa de registo de um ou mais recursos num URI específico.

2.3.4.1 - Registo de servlets

O registo de *servlets* pode ser concretizado com a interface HttpService através do método registerServlet(String alias, Servlet servlet, Dictionary initparams, HttpContext context). Cada tentativa de registo de um *servlet* é acompanhada pelos objectos HttpContext e Dictionary, sendo equivalentes aos definidos na API Servlet (ServletConfig e ServletContext). É também necessário criar o objecto *servlet* e definir o alias sob o qual irá ser disponibilizado. O funcionamento geral do serviço http está representado na figura 2.14.

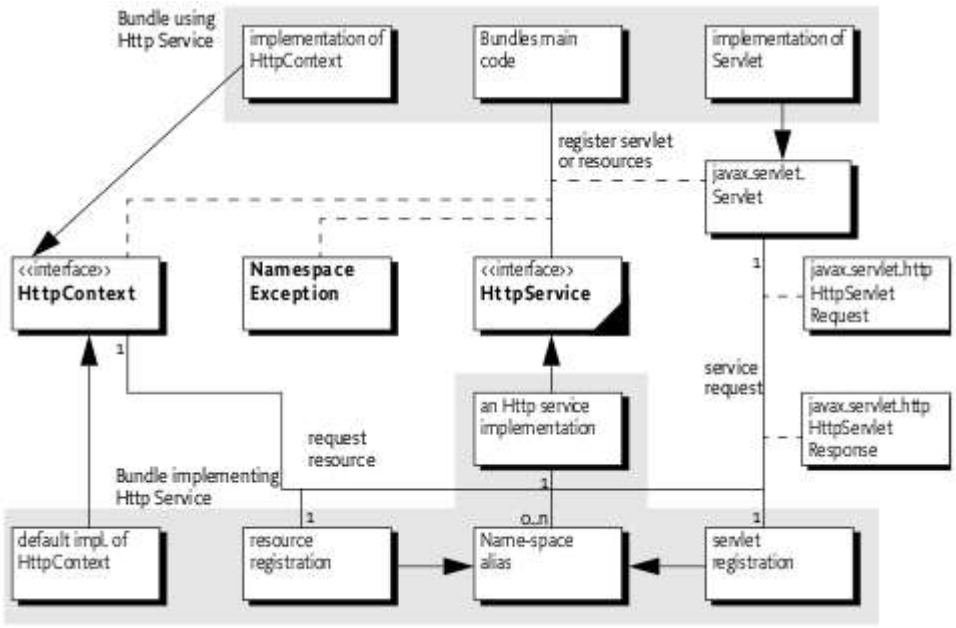

Figura 2.14 – Visão global do funcionamento do serviço HTTP [11].

2.3.4.2 - Registo de recursos

O processo é semelhante ao registo de *servlets*. Neste caso, o método a ser invocado é registerResources (String alias, String name, HttpContext context). O parâmetro que difere relativamente ao método de registo de *servlets* é o segundo. Este define a localização do ou dos ficheiros a serem disponibilizados sob o *alias* definido.

O método unregister(String alias) da interface HttpService permite libertar o *alias* especificado dos recursos alocados ao mesmo. No caso do *servlet*, o HttpService invoca o método destroy() do *servlet* originalmente registado.

2.3.4.3 - Autenticação

A especificação do HttpService separa a autenticação da execução, de um acesso. Para cada pedido de acesso, o serviço invoca o método handleSecurity(HttpServletRequest req, HttpServletResponse res), responsável por controlar se o acesso é considerado normal ou se retorna erro de autenticação.

Para efeitos de autenticação o HTTP providencia os seus mecanismos: *Basic* ou *Digest* [15]. O esquema de autenticação básica é considerado inseguro ao ser utilizado com o protocolo HTTP [16], no entanto pode ser usado em sistemas privados em que o nível de segurança seja baixo. O mecanismo limita-se a codificar o *username* e *password* em Base64 e a transmitir o resultado. No receptor esta *string* é posteriormente descodificada. Apesar das credenciais não serem transmitidas em *plaintext*, a descodificação da *string* é facilmente descodificável. Por outro lado, o mecanismo de *Digest Authentication* utiliza as funções de *hashing* MD5 [17] tornando o processo de autenticação mais seguro. A *string* transmitida é dificilmente descodificada, quando um atacante só tem acesso a essa informação. Apesar de continuar a ser utilizado, o MD5 em pontos fracos explorados em [18 - 20].

2.4 – TR-069 CPE WAN Management Protocol

Devido ao aumento exponencial de dispositivos com acesso à Internet presentes numa rede doméstica, a gestão de todos os equipamentos torna-se uma tarefa árdua e complexa para o utilizador final. Nesse sentido, e de forma a ilibar o utilizador da responsabilidade de configuração e manutenção do sistema residencial, um consórcio de empresas ligadas às tecnologias de informação optou por desenvolver um protocolo de gestão para os equipamentos de interface entre a rede doméstica e a rede do operador, apelidados de CPEs [21]. O protocolo tem como objectivos:

- Auto configuração e disponibilização de serviços dinâmicos: Oferece mecanismo de aprovisionamento automático fornecido ao CPE no momento de ligação à rede. Os mecanismos de identificação permitem um aprovisionamento diferenciado nas características do CPE.
- Gestão de *software/firmware* dos dispositivos: O protocolo oferece mecanismos de gestão de downloads de ficheiros de *software/firmware*. O standard prevê identificação de versões, requisição de transferência de ficheiros (por parte do ACS e opcionalmente pelo CPE) e notificação do ACS sobre o estado final da transferência. O CPE pode também efectuar transferência de pacotes de ficheiros com as respectivas instruções de instalação. Estes pacotes contêm uma assinatura digital de forma a verificar a integridade e a autenticidade dos ficheiros transferidos.
- Monitorização do estado e da performance dos equipamentos: Disponibiliza suporte para que o CPE seja capaz de fornecer informações sobre o seu estado, enviar estatísticas de performance e notificar o ACS em caso de alterações.
- Providenciar métodos de diagnóstico: Prevê que o ACS possua capacidade de execução de testes de diagnóstico e resolução de falhas de conectividade/serviços.

O modelo apresentado pelo BroadBand Forum, baseia-se numa arquitectura cliente-servidor ilustrada na figura 2.15.

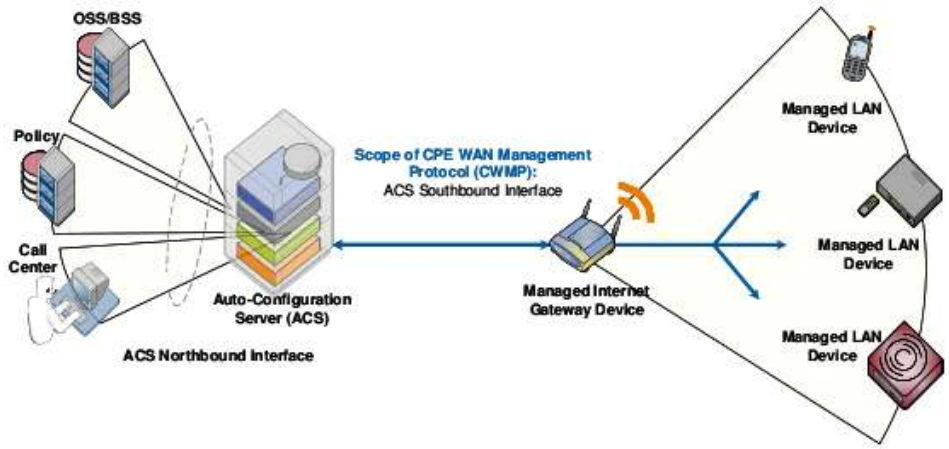

Figura 2.15 – Arquitectura do protocolo de gestão proposto pelo BroadBand Forum [22].

O modelo de gestão define 2 entidades principais: CPE e ACS (Auto Configuration Server). O dispositivo CPE, neste contexto, o *gateway* residencial comunica com o ACS, que executa todas as funções inerentes à gestão da rede doméstica. O meio de acesso ao CPE é indiferente para o protocolo, simplesmente á assumida a conectividade IP entre as duas entidades.

Tendo em conta os objectivos a que o protocolo CWMP se propõe, é utilizada a seguinte pilha protocolar:

Tabela 2.1 – Descrição da pilha protocolar utilizada pelo protocolo CWMP.

Camada	**Descrição**
Aplicação CPE/ACS	A aplicação utiliza o protocolo CWMP no CPE e ACS. A aplicação é definida localmente e não especificada como parte do standard.
Métodos RPC (Remote Procedure Call)	Métodos RPC específicos permitem a leitura e escrita de parâmetros do CPE.
SOAP	Protocolo baseado em sintaxe XML utilizado para encapsulamento dos RPCs.
HTTP	Versão 1.1 do standard.
SSL/TLS	Protocolos standard para segurança na camada de transporte. Versões SSL 3.0 [23] ou TLS 1.0 [24].
TCP/IP	Standard TCP/IP.

2.5 – Especificações OSGi

Desde do seu aparecimento em 1999, surgiram especificações que procuraram adaptar-se às novas aplicações da plataforma. As versões, por ordem cronológica, são:

- *Release* 1.0 Maio 2000
- *Release* 2.0 Outubro 2001
- *Release* 3.0 Março 2003
- *Release* 4.0 Outubro 2005
- Release 4.1 Maio 2007
- *Release* 4.2 Setembro 2009

2.6 – Implementações plataforma OSGi

Diversas implementações estão presentes no mercado, sendo que algumas são Open-Source. As mais importantes são:

- Open-Source
  - Apache Felix
  - Equinox
  - Knopflerfish
  - Concierge
- Comercial
  - Knopflerfish Pro
  - OpenRG
  - mBServer
  - Oscar

2.6.1 - Apache Felix

Apache Felix [25] é um projecto de implementação da especificação R4 da OSGi *Alliance* distribuída sob a licença Apache. A implementação da plataforma é organizada em sub

projectos, cada um centrada no desenvolvimento de um serviço ou especificação em concreto da especificação OSGi. A lista completa de projectos pode ser consultada em [26].

2.6.2 - Equinox

Equinox [27] é uma implementação da especificação R4, tal como Apache Felix. Os *bundles* disponibilizados por este projecto estão disponíveis em [28].

2.6.3 - Knopflerfish

O projecto Knoplerfish [29], mantido e desenvolvido pela Makewave [30] disponibiliza uma implementação das especificações OSGi R3 e R4, nas versões KF1 e KF2 respectivamente. A versão Knopflerfish Pro é uma versão comercial, certificada e com suporte. A versão KF1 define os seguintes componentes:

- Framework base: gestão de *bundles*. É composta pelos componentes internos:
  - Serviço PackageAdmin
  - Serviço PermissionAdmin
  - Serviço StartLevel
  - Serviço URL
- ServiceTracker
- Serviço Log
- Serviço HTTP
- Gestor de configurações
- Gestor de dispositivos
- Serviço UserAdmin
- Serviço de preferências
- API Posição
- API Medição
- API Metatype
- Ferramenta XML

A descrição em detalhe de cada um dos serviços listados acima pode ser consultada em nas especificações da plataforma anteriormente referidas. Para além destes, são incluídos vários componentes úteis:

- Desktop: ambiente gráfico da *framework*.
- Console: gestão da *framework* através de uma consola.
- OSGi Metatype XML: implementa formato de dados a utilizar por outros serviços.

Na versão KF2, foram efectuadas diversas alterações de forma a implementar a *release* 4. Para além das modificações na *framework* base, a nova versão inclui suporte a 2 novas funcionalidades: Declarative Services e EventAdmin.

2.6.4 - Concierge

Concierge [31] é uma implementação optimizada da *release* 3 da plataforma OSGi, desenvolvida no Institute for Pervasive Computing, ETH Zurich. De acordo com Rellermeyer e Alonso [32], esta distribuição tem uma melhor performance que as restantes alternativas *Open-Source*, o que a direcciona para implementação em dispositivos com recursos limitados.

2.7 – Aplicações OSGi

Dadas as características demonstradas ao longo deste capítulo, pode-se afirmar que a plataforma se apresenta como uma ferramenta extremamente útil para a execução de serviços em simultâneo e partilha de recursos entre os mesmos. Na secção seguinte são apresentadas diversas aplicações, que não se focam somente na aplicação num contexto residencial, mas expandem as vantagens da plataforma para utilização em veículos e dispositivos móveis. Como exemplo de aplicação, tem-se o serviço de monitorização médica remota. Este serviço possibilitaria a um hospital ou centro de saúde receber dados de um dado paciente, por exemplo, tensão arterial, através de uma ligação á Internet. O paciente estaria equipado com dispositivos próprios para as medições que por sua vez, estariam ligados ao *gateway* residencial. Os *bundles* OSGi seriam instalados no *gateway* e seriam responsáveis por publicar esses dados por exemplo, numa página WEB ao qual o médico responsável pelo paciente teria acesso.

Tendo em conta esta implementação, os *bundles* presentes no *gateway* seriam:

- *Bundle* para o serviço de comunicação entre os dispositivos e o *gateway*.
- *Bundle* para submissão dos dados obtidos.
- *Bundle* para localização dos *drivers* do dispositivo a conectar.
- *Bundle* para o serviço de *drivers* do dispositivo.

Outras aplicações interessantes seriam o controlo da iluminação, controlo de electrodomésticos (activados mediante a hora ou mediante acção remota do utilizador), sistemas de segurança desactivado mediante a aproximação do veículo do residente, ajuste automático do volume de som da TV quando é recebida uma chamada VOIP, etc.

Nas secções anteriores, foi analisada a plataforma OSGi com algum detalhe. O crescente interesse na aplicabilidade desta plataforma de serviços, fez com que diversas aplicações fossem desenvolvidas.

Em [33] foi desenvolvido um *middleware* sobre a plataforma OSGi, denominado de R-OSGi, permite que uma aplicação centralizada desenvolvida dobre OSGi seja expandida a outros pontos da rede através de proxies. Estes outros equipamentos não distinguem os serviços locais dos serviços remotos oferecidos pelo proxy R-OSGi. Uma visão geral da arquitectura é apresentada na figura 2.16:

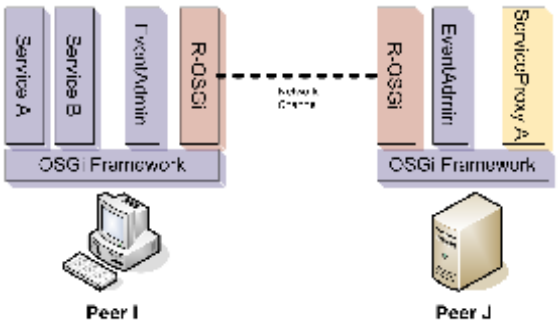

Figura 2.16 – Visão geral da interacção promovida pelo *middleware* R-OSGi [33].

O exemplo de aplicação exposto acima, apresenta um *Service Provider*(*Peer* I) e um *Service Consumer* (*Peer* J). Para o p*eer* J, o serviço A é tido como local e todo o processo de distribuição é transparente. O módulo R-OSGi é responsável pela invocação do serviço original

e os eventos gerados pelo serviço no sistema I são reencaminhados para o sistema consumidor do serviço. Os eventos são interpretados pelo *peer* J como sendo gerados pelo *bundle* local. O *middleware* proposto baseia-se em quatro componentes:

- Proxy dinâmico de invocação de serviços.
- Registo de serviços dinâmico.
- Monitorização da rede de distribuição.
- Resolução de dependências.

Para a demonstração do modelo proposto, foi desenvolvido o Robot Service que permitia controlar o robot Lego Mindstorms através de um PDA. O serviço é iniciado como um serviço local, fazendo a transferência da interface disponibilizada pelo robot de forma a ser possível controlá-lo. Por fim, uma análise da performance entre R-OSGi, RMI e UPNP, verificando-se um ligeiro melhor desempenho em testes de rede em relação ao RMI e uma maior rapidez na invocação do serviço, em comparação com o UPNP.

No modelo de um *home gateway*, apresentado em 2.2.4, o sistema é totalmente dependente do *home gateway* sendo esse o ponto de falha do sistema. Em [34], Thomsen optou por criar *sub-gateways* para cada tipo de rede de dispositivos presentes na rede doméstica. Assim sendo, em caso de falha do *gateway* principal, as redes separadas poderiam operar normalmente apesar de algumas limitações. O sistema desenvolvido baseia-se no conceito de replicação, um tipo de *backup* em que os *sub-gateways* são réplicas do *gateway* principal. Este processo implica a replicação dos *bundles*, que são compostos por código executável, meta data e estado interno. Para ser possível a réplica do *bundle*, será necessário copiar o conteúdo do JAR e replicar o conteúdo da memória e dos registos do CPU. É assumida a capacidade de replicação do JAR principal caso este sofra um *update*. Tendo em conta, o crescente número de dispositivos com capacidades multimédia num ambiente residencial, o interesse no desenvolvimento de CMMs tem vindo a aumentar gradualmente.

Com recurso às características únicas da plataforma OSGi, foi projectado um *Context-Aware Multimedia Middleware* num ambiente residencial [35], a ser implementado num *residential gateway*. A arquitectura do CMM desenvolvido é apresentada na figura 2.17, onde se definem os seguintes componentes:

- *Context Aggregator*: responsável por agregar informação do contexto com base em sensores ou programas de software.
- *Context Reasoner*: fornece informação de alto nível dos contextos.
- *Context Learner*: obtém as preferências do utilizador.
- *Content Filter*: Filtra o conteúdo de acordo com as preferências do utilizador.
- *Content Recommender*: estima o nível de preferência do conteúdo.
- *Content Adapter*: adapta o conteúdo para apresentação num determinado terminal ou sob determinadas condições da rede.

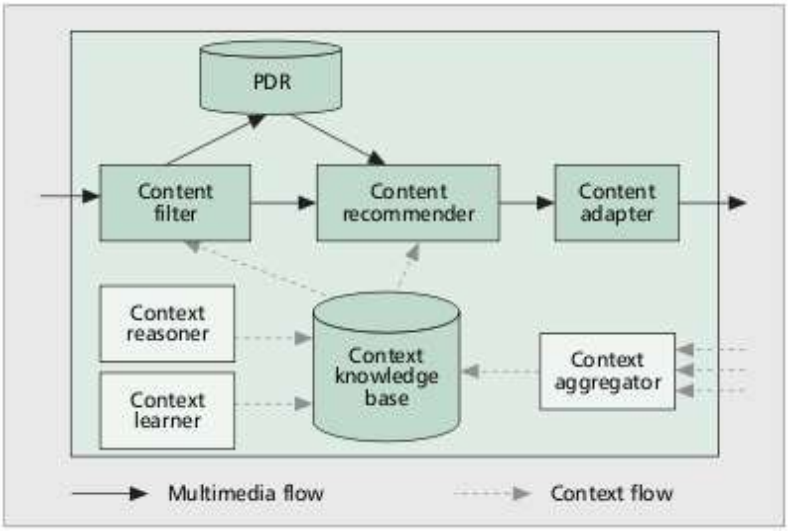

Figura 2.17 – Componentes do sistema de *Context-Aware Multimedia Middleware* [35].

O ambiente de simulação consistia num *home gateway*, que utilizava um processador Intel Celeron 600MHz, 256MB de RAM com sistema operativo Linux, um servidor multimédia, um PC, um telemóvel com capacidade multimédia e vários sensores. A plataforma OSGi escolhida foi a solução comercial da Prosyst, Prosyst mBedded Server. O serviço desenvolvido apelidado de ContAwareMovie, disponibilizava conteúdo aos utilizadores baseado nos vários contextos. Conteúdo esse que era obtido por *broadcast* do servidor multimédia para o *gateway*.

Nos últimos anos, tem-se verificado o crescimento das tecnologias móveis e das capacidades de processamento destes dispositivos. Nesse contexto, com a colaboração da OSGi *Alliance*, foi criado o Mobile Expert Group (MEG), responsável pela definição de standards para promover a implementação da plataforma OSGi em equipamentos móveis. Um grupo de investigadores [36], acredita que a forma correcta de interligação entre as redes PAN (*Personal Area Networks*) e as redes locais pode ser alcançada com a combinação de SIP e OSGi. SIP [37] é um protocolo de sinalização utilizado para gestão de sessões multimédia. Tendo em conta, o objectivo de alargar os dispositivos acessíveis pela PAN aos equipamentos residenciais, é necessário lidar com diferentes protocolos de comunicação. Para esse feito, é fulcral que a plataforma OSGi disponibilize serviços de descoberta de dispositivos e métodos para a sua comunicação, designados por *bridging bundles*. O SIP Service, representado na figura 2.18, é um desses *bundles*, sendo possível, módulos OSGi e dispositivos comunicarem utilizando SIP. Este serviço permite a mobilidade de serviços e dispositivos, podendo ser utilizado para comunicação entre 2 ou mais *gateways*.

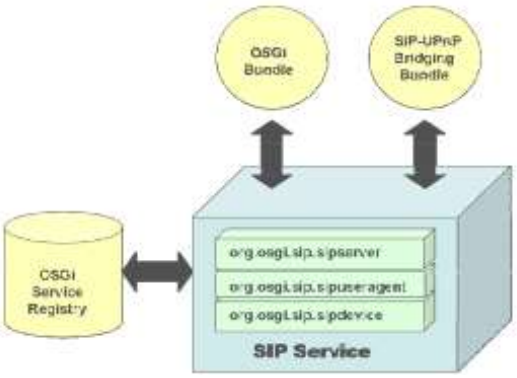

Figura 2.18 – Visão global do serviço SIP na interacção entre *bundles* e dispositivos [36].

O SIP Service permite que os *bundles* e dispositivos OSGi suportem SIP, fornecendo as funcionalidades de proxy, servidor, troca de mensagens, eventos e capacidade de registo. Para avaliação das funcionalidades do serviço, foi efectuada uma interligação entre um dispositivo móvel para controlo de iluminação SIP e um conjunto de lâmpadas UPNP. Neste caso, o *bridging bundle* comunica com os dispositivos UPNP e reencaminha a informação para o SIP Service. Outro caso de utilização descrito, é a interligação entre uma rede PAN e um *home gateway*, só possível se os serviços e dispositivos presentes num *gateway* forem exportados para um segundo gateway através da utilização do SIP Service e de *bridging bundles*. A informação é empacotada no *payload* da SIP MESSAGE e enviada para o *bridging bundle* da rede "exterior" e após a recepção dos dados, é criada um representação virtual do dispositivo sendo possível aceder a serviços do mesmo. A figura 2.19 apresenta a utilização de um gravador de DVD UPNP.

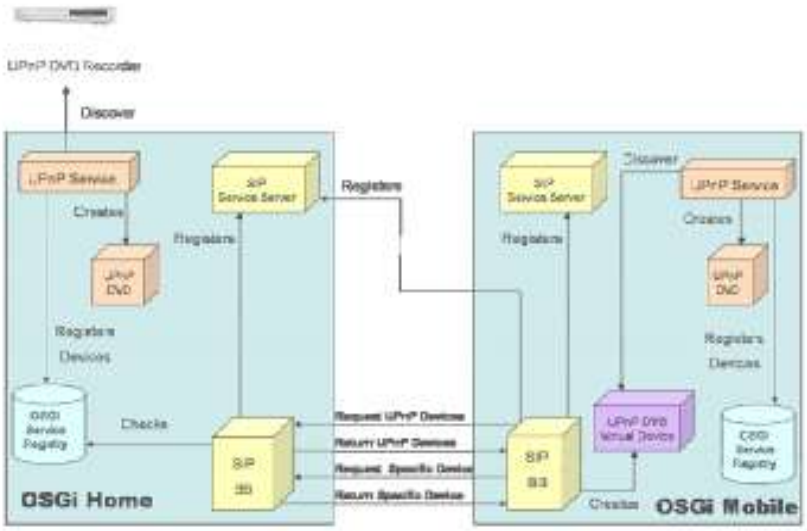

Figura 2.19 – Interacção entre *home gateway* e um dispositivo móvel através do serviço SIP [36].

O modelo de *gateway* residencial sugere diversos cenários de aplicação. Cenários esses que utilizam os mais diversos equipamentos disponíveis na residência, sendo que existem já propostas comerciais de serviços ao dispor dos utilizadores [38,39]. A BSH (Bosch e Siemens) implementou nos seus produtos '*serve@home*' (diversos electrodomésticos), a possibilidade de os gerir e controlar remotamente. A ShellHomeGenie oferece aos seus clientes um kit composto por um *Home Gateway*, uma *Webcam* sem fios, um sensor de contacto, etc. para

gestão remota com recurso à plataforma OSGi. A Philips, lançou a gama iPronto [40] para controlo remoto de dispositivos electrónicos e acesso aos mais variados serviços. A CentralCasa comercializa uma solução de controlo remoto de certos equipamentos, *Service Delivery Platform*. A solução para alguns problemas do produto, é apresentada em [41] e recorreu à implementação da plataforma OSGi nos *routers/switches* residentes. O software embebido inclui a implementação OSGi da ProSyst [42]. No entanto, muitas aplicações encontram-se ainda em fase de investigação e desenvolvimento, como as apresentadas de seguida.

Através da utilização de uma Webcam, foi desenvolvida uma aplicação de monitorização da residência [43]. A arquitectura do sistema, ilustrada na figura 2.20, consiste num *home gateway*, que executa uma plataforma OSGi com dois *bundles* para disponibilizar o serviço. Um *bundle* que implementa a comunicação com a Webcam e outro para permitir o acesso remoto que implementa um servidor Web.

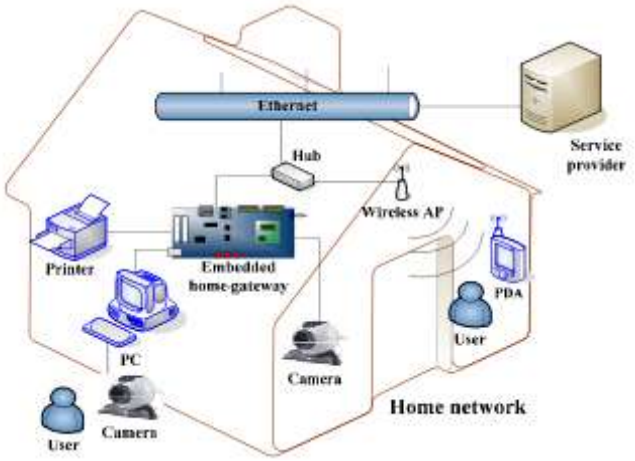

Figura 2.20 – Arquitectura de sistema de videovigilância baseado em OSGi [43].

Outra aplicação desenvolvida em [44], consiste na medição de sinais vitais de um paciente e apresentação dos resultados num portal para diagnóstico. A aplicação baseou-se na utilização de um dispositivo que monitoriza a actividade eléctrica do coração, designado de electrocardiograma. Este equipamento comunica com o *home gateway* através de um conversor RS-232 para USB. O *bundle* responsável pela gestão do dispositivo limita-se a receber dados, guardá-los e enviá-los para o ServiceCenter. Este ServiceCenter é responsável pelo empacotamento dos dados para envio para o centro de controlo, com recurso ao MOM Middleware. A função deste *middleware* é garantir a entrega das mensagens de diagnóstico de forma assíncrona apesar das possíveis falhas de comunicação. Dentro do mesmo contexto, em [45] é apresentado um sistema de receitas médicas para uma aplicação de telemedicina baseada em OSGi. Xie Lie e Wenjun Zhang [46] desenvolveram uma aplicação que permitia o controlo de equipamentos residenciais interligados através de uma rede de módulos Lonworks [47,48]. Os módulos referidos comunicam entre si através da rede eléctrica, não sendo necessária a instalação de novos cabos. No entanto, a largura de banda da rede é algo limitada. A aplicação permitia o controlo de iluminação e a activação de uma ventoinha.

A plataforma OSGi em conjunto com a API do JSR 272 [49], permitiu desenvolver um *middleware* para a transmissão de conteúdos digitais em dispositivos móveis [50]. A utilização do OSGi também se propagou aos transportes. A BMW apostou nesta tecnologia para desenvolver o sistema de navegação e centro de multimédia nos veículos pertencentes à série

5. No caso da Bombardier, o sistema de diagnóstico remoto (*RDS*) foi desenvolvido sobre OSGi e aplicado em locomotivas eléctricas ALP 46. Os autores em [51] descrevem uma série de serviços modulares implementados num veículo de teste, com recurso a um computador para interface com os vários sensores e dispositivos de localização. O veículo encontra-se equipado com interfaces Bluetooth, 802.11 e GPRS/UMTS para garantir total conectividade. Os serviços implementados são:

- Monitor de integridade: serviço que permite ao utilizador avaliar a integridade da posição emitida pelo sistema de localização.
- Visão traseira: permite ao condutor aceder às imagens da câmara instalada na traseira do veículo.
- *Media player*: leitor de conteúdo multimédia compatível com diversos formatos de vídeo e áudio.
- Navegador: sistema de navegação GPS com recurso ao módulo de síntese de voz.

Outros investigadores dedicaram o seu estudo às possíveis falhas do OSGi, tendo analisado a sua segurança [52], a tolerância a falhas [53] e formas de tornar a plataforma e os seus serviços mais fiáveis [54] nos cenários de aplicação descritos nos projectos acima.

2.8 – Resumo

Este capítulo focou-se na plataforma OSGi, onde foram detalhadas as características e funcionalidades de acordo com o modelo de camadas que a especifica. Foram também apresentadas as implementações disponíveis, com especial atenção nas alternativas *Open-Source*. Na secção final, apresentaram-se diversas aplicações e estudos da plataforma em questão.

Capítulo 3

Domótica

A palavra Domótica tem origem na junção da palavra latina Domus (casa) e do termo Robótica. O conceito surge com o intuito de melhorar a qualidade de vida através da utilização inteligente dos recursos disponíveis numa residência. Com a crescente disponibilização de ligações à Internet de banda larga por parte dos fornecedores de serviços, surgem novas formas de aproveitamento das mesmas. A capacidade de controlar diversos equipamentos numa residência de forma remota, surge como uma natural aplicação possível, tendo em conta que na maioria dos casos a ligação de banda larga se encontra activa durante as 24 horas. Neste contexto, emerge o conceito de casa inteligente, que se define como a integração dos diferentes sistemas residenciais, desde os equipamentos electrónicos aos sistemas de comunicação. Desta forma, é possível centralizar os serviços e a gestão dos sistemas, promover a cooperação entre estes e possibilitar o acesso exterior às funcionalidades da residência. A inteligência da habitação está implícita na correcta gestão dos sistemas tendo em conta os seus utilizadores e as suas necessidades, desde a qualidade de vida para pessoas com mobilidade reduzida até a eficiência energética de forma a reduzir os consumos desnecessários.

O mercado emergente da automação residencial, possibilitou o desenvolvimento de vários protocolos e produtos de forma a proporcionar várias aplicações, tais como: vigilância remota, controlo remoto de iluminação, estores, electrodomésticos, etc. Na secção seguinte, será apresentada uma descrição das tecnologias envolvidas na concepção de um sistema de automação residencial.

3.1 - Tecnologias Wired

3.1.1 - IEEE 802.3

Conjunto de standards [55], que definem a camada física e a camada de ligação do modelo OSI, destacando-se a tecnologia Ethernet, largamente utilizada nos dias de hoje, para a formação de LANs. Num ambiente residencial, a tecnologia é utilizada para a interligação de dispositivos para acesso à Internet, sendo que a velocidade de ligação é suficiente para a

maioria das aplicações, 10 ou 100Mbps. Para o efeito é utilizado cabo cruzado sem blindagem da categoria 5, com conectores 8P8C, usualmente designados por RJ45.

3.1.2 - HomePNA

Este standard [56], define a transmissão de dados através da utilização das linhas telefónicas comuns. HomePNA (Home Phoneline Networking Alliance) é o consórcio de várias empresas que desenvolvem e promovem o standard. Através da utilização de FDM, separa o tipo de dados em frequências diferentes possibilitando a transmissão de voz e dados no mesmo cabo. De acordo com a especificação 3.1, aprovado pelo ITU em Dezembro de 2006, o protocolo suporta uma taxa de transmissão até 320Mbps, o que possibilita a distribuição *Triple-Play*, IPTV, voz e Internet.

3.1.3 - PowerLine

Esta tecnologia [57], possibilita a transmissão de dados em sistemas de cabos de electricidade, sobrepondo frequências mais elevadas à frequência de normal do fornecimento da energia, cerca de 50Hz em Portugal. O desenvolvimento da tecnologia pode ser aproveitada para que os operadores de telecomunicações possam distribuir serviços utilizando a rede eléctrica normal. Esta tecnologia foi adoptada por diversos fabricantes, alguns deles na área de automação residencial, sendo as implementações mais influentes no mercado descritas.

3.1.3.1 - X10

A tecnologia X10 adopta a instalação eléctrica como meio físico para implementação do seu protocolo, com o propósito de controlar equipamentos electrónicos. Cada equipamento deverá ser ligado a um módulo X10. Este standard utiliza a modulação em amplitude para a transmissão binária de dados, codificando a informação numa portadora a 120KHz durante a passagem por zero da forma de onda que caracteriza a alimentação eléctrica (50 ou 60Hz). Um bit é transmitido por cada 2 transições, para reduzir os erros na transmissão.

Um sistema que utiliza esta tecnologia consiste normalmente num módulo controlador transmissor e um conjunto de módulos receptores, sendo estes distinguíveis pelo seu endereço único na rede. O endereço é composto por uma letra de A a P e um código numérico de 1 a 16, possibilitando a existência de 256 dispositivos na rede (16*16). Um pacote de dados X10 é composto pelo código do endereço e um código de comando, que define a função a executar pelo módulo receptor. As vantagens deste standard são:

- Baixo custo do sistema
- Facilidade de instalação
- Aproveitamento da instalação eléctrica residencial

No entanto, a tecnologia apresenta limitações e pontos fracos [58]. Numa implementação num ambiente real, os autores verificaram a existência de interferências com outros equipamentos que partilham a rede eléctrica. A maior fraqueza deste protocolo, é o facto de não existir um mecanismo de verificação da correcta recepção dos comandos enviados, sendo assim, o envio de um comando “OFF” pode efectivamente não ser recepcionado e o equipamento continuar “ON”, sem que o utilizador tenha conhecimento.

3.1.3.2 – INSTEON

Este protocolo desenvolvido pela SmartLabs, surgiu para colmatar as lacunas do protocolo X10. Para tal, para além da rede eléctrica é utilizada a comunicação RF. A vantagem da utilização de 2 meios de comunicação tem impacto importante na fiabilidade do sistema, sendo que um complementa o outro nos seus pontos fracos. Para além disso, um mecanismo de confirmação de comando foi implementado, possibilitando o reenvio do comando em caso de erro. Numa rede Insteon, todos os dispositivos são repetidores de sinal, contrariando assim a limitação em termos de alcance. As mensagens são enviadas para todos os dispositivos ao alcance ao mesmo tempo sendo repetidas até 3 vezes, evitando assim a existência de mecanismos de encaminhamento de pacotes. O protocolo define o número máximo de hops para a transmissão da mensagem e o número de *hops* que faltam para a retransmissão da mensagem. Um equipamento Insteon, utiliza a frequência de 131.65KHz com modulação BPSK e a frequência de 904MHz sobre RF com modulação FSK. Uma comparação entre as diferentes tecnologias PowerLine é apresentada em [59].

3.1.3.3 – UPB

O Universal Protocol Bus [60], é uma tecnologia desenvolvida pela Powerline Control Systems [61] e representa mais uma alternativa nas tecnologias Powerline. A tecnologia utiliza PPM, em que um conjunto de impulsos gerados é sincronizado com a frequência AC de 50 ou 60Hz. A posição relativa dos impulsos varia num determinado período de tempo ou depende da posição de impulsos anteriores. Em resumo, o dispositivo transmissor modula a informação através de impulsos precisos e o receptor detecta os impulsos, analisa as posições e descodifica a informação. Uma das principais vantagens relativamente ao X10 é a comunicação nos 2 sentidos, sendo possível aceder ao estado do dispositivo remoto. De acordo com [62], a velocidade de comunicação é bem mais rápida que o X10.

3.1.3.4 – LonWorks

LonWorks [63] define uma rede P2P de controlo, largamente utilizada na supervisão industrial e edifícios inteligentes, que suporta diversos meios de transmissão, cabo coaxial, fibra óptica, infra-vermelhos, etc. O protocolo sob LonWorks, foi em Janeiro de 2009 aprovado como standard para automação de edifícios, sob a designação de ISO/IEC14908-1. O componente fundamental deste sistema é o chip Neuron, que integra 3 processadores, em que 2 deles são dedicados à comunicação e o restante à aplicação. O protocolo de comunicação designado de LonTalk, implementa todas as camadas do modelo OSI.

3.1.3.5 – HomePlug

O HomePlug 1.0 [64,65], baseou-se na tecnologia Intellon PowerPacket, para transmissão de dados através da rede eléctrica a uma velocidade de 14Mbps. Numa versão mais recente, HomePlug AV, atinge a taxa de transmissão máxima de cerca de 200Mbps, possibilitando assim a transmissão de conteúdos multimédia e HDTV. O HomePlug C&C (Command and Control), adapta o conceito geral à aplicação de automação residencial, em que a velocidade não é um factor crítico, possibilitando um custo mais acessível dos equipamentos. Uma

análise da tecnologia é efectuada em [66], onde fica patente a rapidez e a robustez da tecnologia.

3.1.4 - UPNP

Universal Plug and Play [67] foi desenvolvido a partir do PnP – Plug and Play e foi concebido para permitir uma maior simplicidade na interligação de dispositivos inteligentes numa rede doméstica. O UPNP é compatível com uma diversidade de meios de comunicação que sejam compatíveis com IP, Ethernet, FireWire (IEEE 1394), RF (Bluetooth, Wi-Fi) e IR (IrDA). O protocolo define-se como independente do sistema operativo e da linguagem de programação, sendo possível desenvolver produtos UPNP em qualquer plataforma e através de qualquer linguagem. A arquitectura P2P é distribuída sendo baseada em standards padrão como TCP/IP, UDP, HTTP, XML e SOAP. Outras tecnologias, como as referidas anteriormente (X10, LonWorks), podem ser integradas com UPNP através de bridges ou proxies [68].

3.1.5 - BatiBUS

Este sistema, desenvolvido em 1988, foi inicialmente projectado para interligar unidades de controlo, actuadores e sensores. Utiliza como meio físico, o cabo UTP para interligar todos os dispositivos e alimentá-los caso não consumam mais de 3mA. O BatiBUS [69] pode utilizar qualquer topologia de rede: bus, estrela, token-ring, e emprega o mecanismo de CSMA/CA para controlo de acesso ao meio. Relativamente à velocidade de transmissão fica-se pelos 4800bps.

3.1.6 - EIB/KNX

European Installation Bus [70] é um sistema desenvolvido pela EIBA. Normalmente utiliza o BUS de comando, para transmissão de informações e ordens de comando. No entanto, podem ser utilizados outros meios de transmissão como rádio frequência, infravermelhos ou rede eléctrica. A utilização do sistema pressupõe uma rede de alimentação independente da rede de comando, para alimentação dos diferentes sistemas de saída (actuadores).

3.2 - Tecnologias Wireless

3.2.1 - Wi-Fi

As redes Wi-Fi ou WLAN definem um sistema de transmissão de dados utilizando ondas rádio em vez de uma estrutura física. Nos dias, de hoje a maioria das redes sem fio utilizam o standard 802.11 [71], que implementa a camada física e a camada de ligação do modelo OSI. Estas redes permitem a mobilidades dos utilizadores, sendo que o alcance num ambiente residencial ronda os 50m e cerca dos 100m em linha de vista. As elevadas taxas de transmissão não se adequam às velocidades requeridas para aplicações de automação residencial, sendo que seria um desperdício a sua utilização neste contexto.

3.2.2 - Bluetooth

O protocolo Bluetooth [72], foi desenvolvido com o objectivo de obter uma forma de comunicação sem fios e de baixo consumo para transmissão de voz e dados. É normalmente utilizado em dispositivos móveis e periféricos cuja utilização não implique uma distância elevada, uma vez que o alcance ronda os 10m. A comunicação é efectuada na frequência de 2.4.GHz com FHSS de forma a obter a velocidade de 3Mbps. A limitação do alcance é o revés desta tecnologia na sua aplicação na automação residencial.

3.2.3 – Infra-Vermelhos

A comunicação por infra-vermelhos [73] é largamente utilizada nos dias de hoje, para controlo de equipamentos electrónicos, televisores, leitores de DVD, etc. A comunicação é de apenas 1 sentido, não existindo portanto forma de confirmar a recepção dos dados. Uma vez que a transmissão infra-vermelhos pode sofrer interferências, devido aos sinais gerados pelos nossos corpos e outros equipamentos, é necessário "encriptar" a transmissão. O processo de proteger as transmissões dos falsos sinais é apelidado de modulação. O sinal a transmitir é modulado numa determinada frequência, utilizada pelo emissor e receptor. Todos os sinais IR com outras frequências são simplesmente ignorados. As frequências normalmente utilizadas variam entre os 30KHz e 60Khz, sendo a mais comum 36KHz, em que são gerados 36000 impulsos por segundo para a transmissão do nível lógico "1". Os módulos receptores identificam os impulsos e geram um sinal de saída, sendo composto por um conjunto de blocos internos, como presente na figura 3.1.

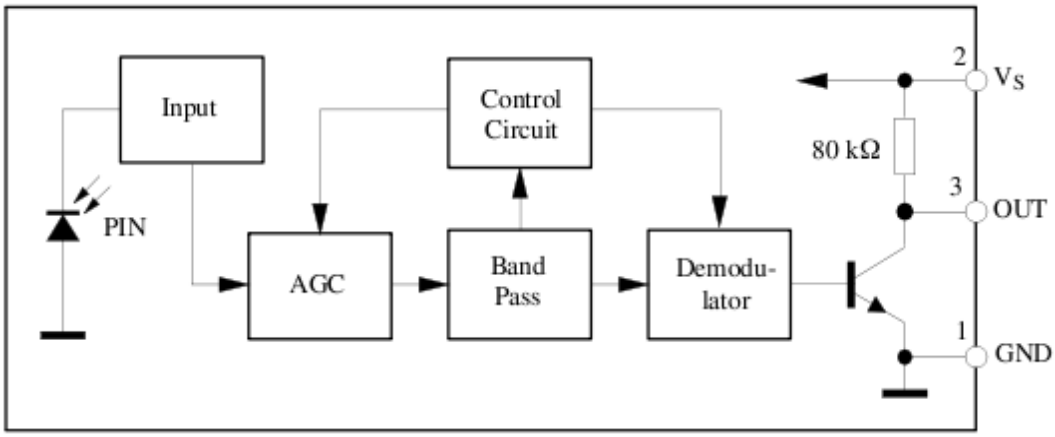

Figura 3.1 – Componentes de um receptor infra-vermelhos da série TSOP da Vishay [74].

Para além de um led IR receptor, o circuito é composto por um AGC, um circuito de controlo, um filtro passa-banda e um desmodulador. A saída é activa a "0", ou seja, quando é identificado o nível lógico "1" na entrada. Para evitar que um emissor actue em diversos equipamentos, foram definidos pelos fabricantes diferentes protocolos. Os mais importantes são o RC5 da Philips, RCA e Sony SIRC.

3.2.4 - Z-Wave

Z-Wave [75] é uma tecnologia sem fios desenvolvida pela Zensys para automação e controlo residencial, com os seguintes objectivos:

- Baixo custo
- Baixo consumo

- Fiabilidade
- Instalação simples

O protocolo define 4 camadas apresentadas na figura 3.2: camada MAC, camada de transporte, camada de *routing* e camada de aplicação. A camada MAC, utiliza o mecanismo de CSMA/CD para controlo de acesso ao meio RF. A camada de transporte, é responsável pela verificação da integridade das mensagens e controla a correcta recepção dos dados e retransmissões. A camada de *routing* é tem a função de encaminhar as mensagens de um nó para outro, com base na localização dos nós da rede. Em cada mensagem é definida uma lista dos nós repetidores, definindo o caminho a seguir. Uma tabela de rotas para todos os nós é definida com base na topologia da rede. Por fim, a camada de aplicação descodifica e executa os comandos recebidos.

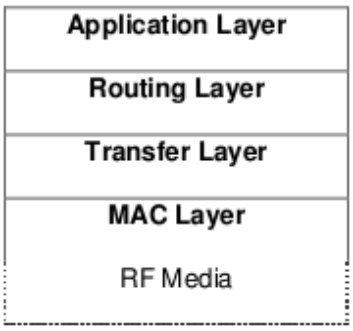

Figura 3.2 – Modelo de camadas do protocolo Z-Wave [75].

O protocolo Z-Wave, tal como o Zigbee, define tipos diferentes de nós na sua rede *mesh*: *controller device* e *slave device*. O controlador, é o gestor da rede, o dispositivo que define os parâmetros de ligação (canal de transmissão, identificação da rede, etc), atribui endereço de rede ao *slave device* e envia mensagens. Para cada rede, apenas um nó gestor pode ser designado. Este nó contém a tabela de encaminhamento da rede, tem portanto a possibilidade de comunicar com todos os nós. O *slave device* apenas recebe as mensagens e possui capacidade de reencaminhamento. Estes nós não têm a funcionalidade transmitir comandos. Em [76], os autores compararam Z-Wave com Zigbee, realçando que devido ao Z-Wave ser um protocolo proprietário impede uma análise comparativa mais profunda e que a configuração da rede é mais fácil no caso do Zigbee, dado que é um protocolo mais focado na monitorização.

3.2.5 - Zigbee

Zigbee [77] é uma especificação para a comunicação bidireccional sem fios para aplicações com requisitos de baixo consumo, baixa taxa de transmissão, alcance limitado, normalmente associados a monitorização e controlo remoto. A comunicação baseia-se no standard IEEE 802.15.4 [78], que define a camada MAC e a camada física das redes sem fio de baixo débito (WPANs). Na Europa, são utilizadas as frequências de 868MHz e 2.4GHz, com uma taxa de transmissão máxima de 250Kbps. O alcance varia de acordo com o módulo de comunicação utilizado, sendo o valor de 50m (*indoor*) o mais usual. O Zigbee utiliza a pilha protocolar representada na figura 3.3:

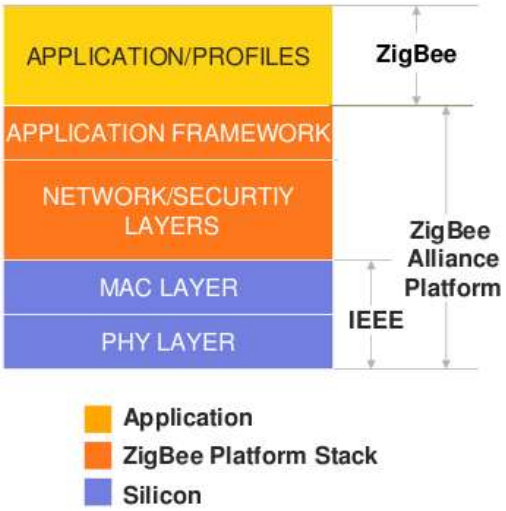

Figura 3.3 – Pilha protocolar Zigbee [79].

Na camada física, é utilizada DSSS com duas modulações PSK distintas para minimizar as interferências e na camada MAC adopta o CSMA/CD como mecanismo de acesso ao meio.

A tecnologia Zigbee define 2 tipos físicos de dispositivos: FFD e RFD. Os FFDs são geralmente alimentados pela rede eléctrica, comunicam com todos os nós da rede e possuem capacidade de descoberta de outros nós. Os RFDs, são os equipamentos mais simples e alimentados a bateria. A rede pode ser implementada segundo um destes três modelos presentes na figura 3.4: árvore, estrela e *mesh*.

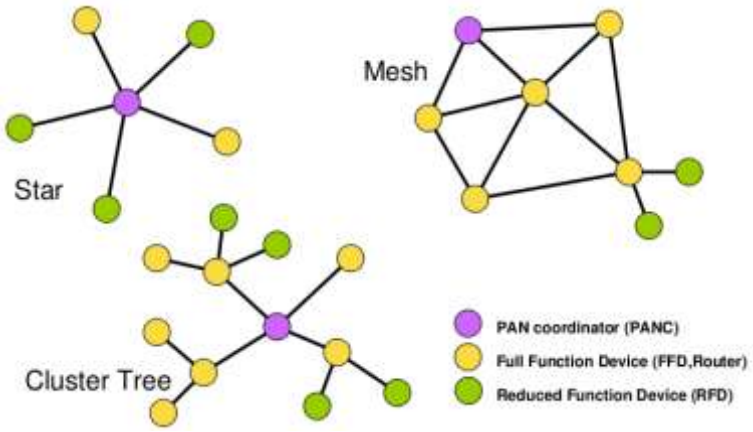

Figura 3.4 – Topologias de rede [79].

Em termos lógicos, uma rede Zigbee inclui um gestor, um ou mais routers e vários *end devices*. Numa topologia em estrela, o controlador da rede situa-se no centro com os routers e *end devices* presentes na sua área de cobertura. É normalmente utilizada na criação de sub-redes, onde um router actua como coordenador. Um exemplo de uma rede constituída por estes dispositivos está exemplificada na figura 3.5. Com a aplicação da rede *mesh* é possível ultrapassar a limitação do alcance, com os routers a comunicarem entre si, sendo esta a topologia mais popular.

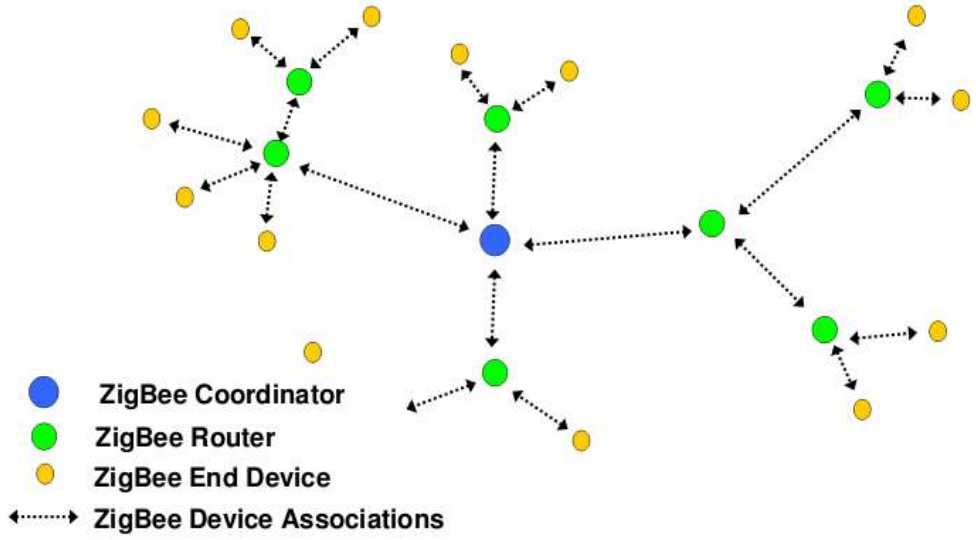

Figura 3.5 – Exemplo de uma rede de dispositivos Zigbee [79].

3.3 – Estado da Arte

Dada a quantidade de protocolos existentes, são apresentados nesta secção diversos estudos e aplicações das diferentes tecnologias no contexto da domótica. Com recurso a Zigbee, foi desenvolvido um sistema de monitorização do batimento cardíaco [80]. O sistema é composto por um módulo de comunicação integrado com sensores, um servidor residencial e um servidor central. Os dados recolhidos são enviados para o servidor residencial, responsável por coordenar a rede (gestor conectado por USB), guardar e processar a informação recebida. O servidor central obtém a informação do servidor residência, para posterior análise por parte de um profissional de saúde. Um projecto com uma arquitectura semelhante foi implementado por Ying-Wen Bai e Chi-Huang Hung [81], com o objectivo de ligar/desligar remotamente equipamentos e obter os seus consumos energéticos.

Os autores do projecto descrito em [82], investigaram formas de desenvolver interfaces com electrodomésticos para pessoas com dificuldade de mobilidade. Interfaces essas que deveriam ser seguras, inteligentes e extensíveis. Na perspectiva de desenvolver um MAS para o controlo dos equipamentos, foram analisados protocolos de comunicação entre eles, X10, Zigbee e Z-Wave. Tendo em conta o elevado preço de soluções como o X10 ou EIB/KNX, foi implementado um protocolo de comunicação Powerline com microprocessadores PIC [83]. Para tal utilizaram-se módulos PLM 24 integrados com PIC16F876, com recurso ao protocolo aberto SNAP. Este protocolo foi desenvolvido para a aplicação em automação residencial que representa um formato genérico para a transmissão de pacotes. O produto final consistia em duas unidades de comunicação, um *master* e um *slave*, desenvolvidos com a quantia de 90 euros. A tecnologia GSM, largamente utilizada nos dias de hoje, foi a base do sistema desenvolvido em [84]. O sistema é composto por um terminal móvel interligado a um microprocessador por RS-232 representado na figura 3.6.

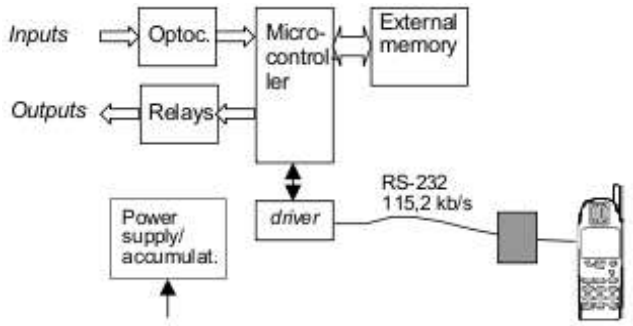

Figura 3.6 – Componentes de um sistema de automação residencial com recurso a GSM [84].

O microprocessador é responsável pelo processamento das mensagens escritas, que pretendem activar relés ou avisar o utilizador de alterações no estado das entradas. Soucek, Russ e Tamarit aplicaram a tecnologia EIB para o desenvolvimento de uma cozinha inteligente [85]. Instalaram-se diversos sensores para detecção de presença, fumo e fugas de água. O sistema pretendia implementar mecanismos de segurança, conforto e poupança de energia. O *gateway* residencial permitia o acesso remoto a rede de *bus*, com a utilização do protocolo SNMP [86] para acesso às MIBs dos pontos de acesso ao bus EIB.

Outros exemplos de soluções domóticas podem ser encontradas nos sites oficiais dos respectivos standards e fabricantes. De referir que as soluções mais acessíveis ao utilizador comum são produtos que utilizam a tecnologia Powerline, devido ao custos de implementação mais baixos, nomeadamente X10 e Insteon.

3.4 - Comunicação dispositivos electrónicos

Nesta secção são apresentados os protocolos mais utilizados para comunicação entre dispositivos electrónicos, como LCDs, memórias RAM e EEPROM.

3.4.1 - I2C

O protocolo I2C [87], desenvolvido pela Philips, permite a comunicação entre componentes no mesmo circuito electrónico. As comunicações são efectuadas utilizando duas linhas de barramento designadas por SDA e SCL. Para além destas são necessárias as duas linhas de alimentação VCC e GND, como presente no exemplo da figura 3.7.

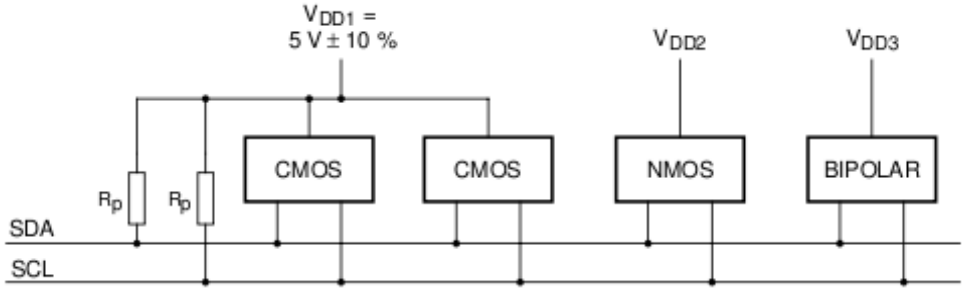

Figura 3.7 – Exemplo de aplicação do protocolo I2C [87].

As vantagens deste protocolo são:

- Apenas 2 linhas de barramento são necessárias.

- Cada dispositivo presente no barramento tem um endereço único.
- Endereços de 7 ou 10 bits que permite a existência de um numero elevado de dispositivos.
- Arquitectura *master/slave*.
- Permite a existência de até 8 *masters* com mecanismos de detecção de colisões.
- Suporte velocidades de transferência até 3.4Mbps.

3.4.2 - SPI

A interface *Serial Peripheral Interface Bus* [88] define-se como uma ligação de dados síncrona full-duplex. Os dispositivos comunicam segundo uma relação *master/slave* em que o *master* inicia a transmissão, como explícito na figura 3.8.

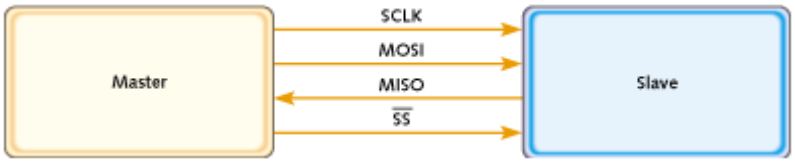

Figura 3.8 – Comunicação *master/slave* SPI [88].

Quando o *master* gera um *clock* e selecciona o dispositivo *slave*, os dados são transferidos num ou nos dois sentidos simultaneamente. O PSO especifica os seguintes sinais:

- SCLK: sinal de relógio utilizado para sincronizar *master/slave*.
- MOSI: envio de dados do mestre para o *slave*.
- MISO: envio de dados do *slave* para o mestre.
- CS: selecção do *slave*.

3.5 – Resumo

Este capítulo abordou os principais protocolos de automação residencial, quer os que requerem meios físicos para a transmissão quer sem fios. Dedicou-se uma secção à apresentação de alguns projectos que aplicaram essas tecnologias no âmbito da domótica. A secção final, apresentou dois dos protocolos mais utilizados na comunicação de periféricos.

Capítulo 4

Sistema

A aplicação desenvolvida focou-se no controlo e monitorização da temperatura de um ambiente residencial. Em resumo, as funcionalidades do sistema implementado são:

- Monitorização da temperatura ambiente através da leitura de sensores em tempo real.
- Ajuste da temperatura ambiente através da interacção com o sistema de ar condicionado.
- Interacção remota através de portal Web.
- Envio de mensagem de correio electrónico caso limite de temperatura definido pelo utilizador seja excedido.
- Descodificação de comandos utilizados pelo controlo remoto do sistema de ar condicionado LG MS09AH.

As funcionalidades pretendidas resultam num conjunto de requisitos. A restrição do sistema desenvolvido refere-se à utilização da plataforma OSGi.

4.1 - Requisitos

4.1.1 - Funcionais

Acesso a serviços

O sistema deve:

- F1.1 - Requisitar credenciais para acesso aos diversos serviços.

Apresentação de informações

O sistema deve:

- F2.1 – ser capaz de apresentar a temperatura actual através do acesso a sensores.
- F2.2 – apresentar informações relativas à configuração de notificações.
- F2.3 – apresentar resultados da descodificação de comandos infra-vermelhos.
- F2.4 – apresentar as informações numa página Web.

Controlo sobre dispositivos

O sistema deve:

- F3.1 – ser capaz de controlar o sistema de ar condicionado.
- F3.2 – possibilitar o controlo remoto através de uma interface Web.

Notificações

O sistema deve:

- F4.1 – ser capaz de notificar o utilizador por mensagem de correio electrónico.

Configurações

O sistema deve:

- F5.1 – possibilitar a configuração de endereço de correio electrónico para o serviço de notificações.
- F5.2 – possibilitar a definição de limite superior de temperatura.
- F5.3 – possibilitar a especificação do intervalo entre notificações.

4.1.2 – Não funcionais

- Segurança no acesso ao serviço.
- Facilidade de utilização.

4.1.3 - Restrições

- R1.1 – A aplicação deve ser desenvolvida com recurso à plataforma OSGi.

4.2 - Arquitectura

Na figura 4.1, é apresentada a arquitectura física do sistema. Os componentes principais que o constituem são:

- **Home Gateway:** equipamento que executa a plataforma OSGi com os serviços desenvolvidos e representa o dispositivo que possui ligação com a rede exterior.
- **Sensores:** grupo de dispositivos electrónicos que fornecem informação sobre diversos parâmetros como temperatura, humidade, etc.
- **Ar condicionado:** equipamento a controlar mediante pedido do utilizador.
- **Descodificador Infra-Vermelhos:** permite descodificar comandos IR.

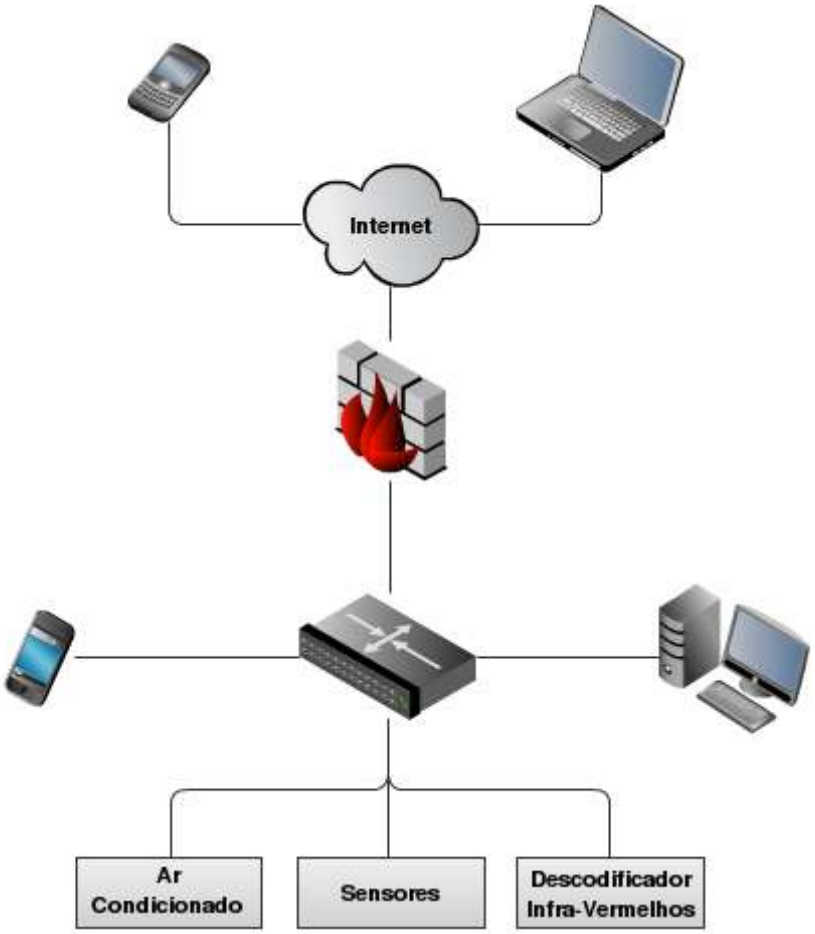

Figura 4.1 – Arquitectura física do sistema.

A ligação ao sistema de ar condicionado, representada na figura acima, não implica um meio de comunicação físico. A comunicação é efectuada por infra-vermelhos. Ao *home gateway* é adicionado um circuito emissor de IR, controlado via GPIO. Para a aquisição de informação relativa á temperatura ambiente, são utilizados 2 sensores distintos. Um conectado directamente a umas das interfaces GPIO e outro acedido via Zigbee. O circuito descodificador de IR comunica através de outra interface GPIO. Na arquitectura definida, o *gateway* residencial é responsável pela integração dos sistemas. O dispositivo comporta-se também como a interface de acesso à rede exterior e gestor da rede residencial.

4.2.1 - Tecnologias

4.2.1.1 - Plataforma de desenvolvimento

A implementação do sistema representado na figura 4.1, prevê a existência de um *gateway* residencial. Como já referido anteriormente, este equipamento será responsável por interligar a rede interna e a rede externa, possibilitando a um utilizador externo o acesso a sistemas presentes na rede residencial. Para a concretização do projecto é necessário que a plataforma residencial possua interfaces de forma a ser possível interagir com sensores e actuar sobre um sistema de ar condicionado. O resultado final da pesquisa por plataforma é apresentado em anexo.

A escolha final recaiu sobre a solução apresentada pela Omnima. Esta apresenta recursos suficientes para o sistema a implementar e uma excelente relação qualidade/preço. Diversa informação de desenvolvimento pode ser encontrada em [89]. Segue-se uma análise mais detalhada da plataforma apresentada nas figuras 4.2 e 4.3:

Figura 4.2 – Plataforma distribuída pela Omnima [90].

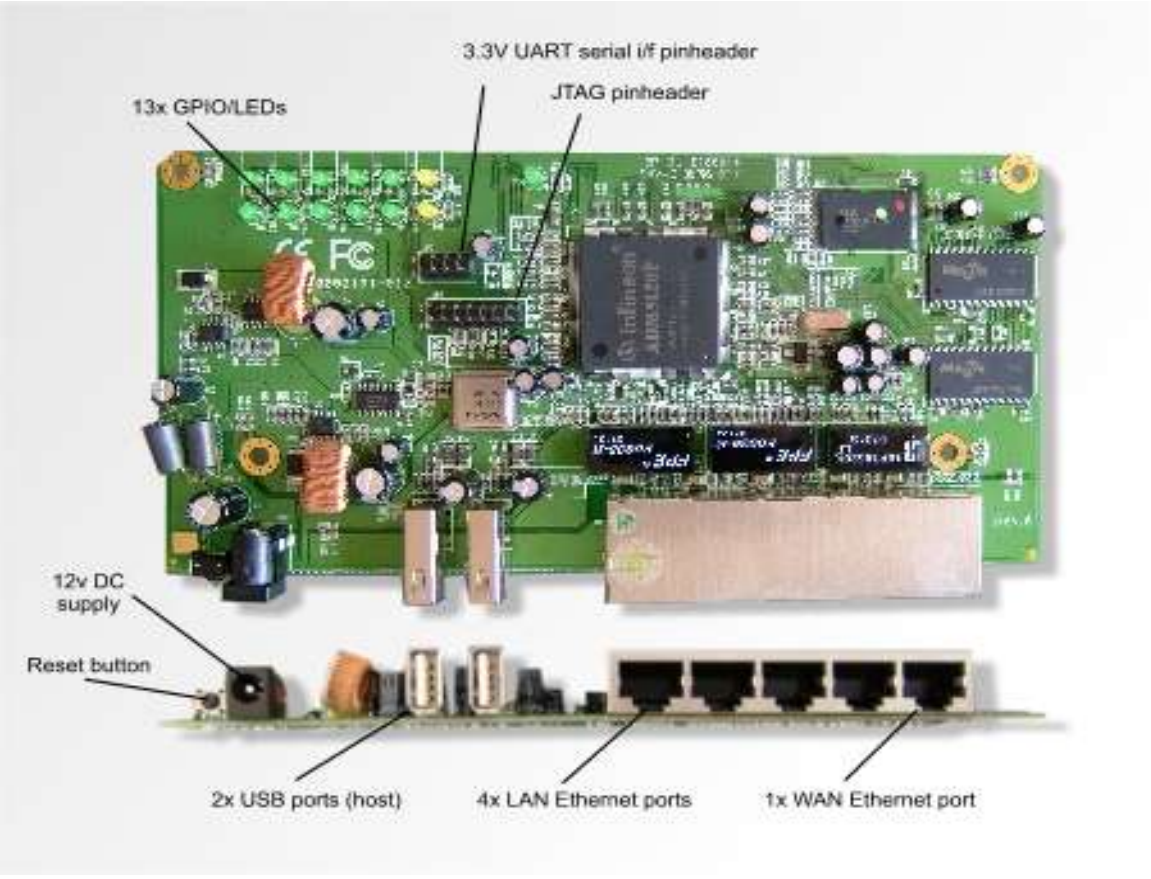

Figura 4.3 – Interfaces presentes na plataforma [90].

A plataforma escolhida tem as seguintes características:

- Processador Infineon ADM5120 (MIPS32)
- Memória: 16 MB RAM e 2 MB Flash
- 4 portas Ethernet LAN
- 1 porta Ethernet WAN
- 2 portas USB
- 1 interface serial UART
- 1 interface JTAG

- 12/13 GPIO com LEDs
- Alimentação 12V 500mA
- Sistema Operativo: Linux OpenWRT / NetBSD

4.2.1.1.1 – Sistema operativo

Segundo o fabricante, é possível executar a distribuição Squidge [91] do OpenWRT [92], a partir de uma *pen* USB de 1GB. Esta distribuição encontra-se desactualizada pelo que foi necessário actualizar o sistema operativo para uma versão mais recente. O OpenWRT é uma distribuição Linux para *embedded devices*, com especial incidência em routers comerciais. O seu desenvolvimento iniciou-se na série WRT54G da Linksys, tendo-se expandido progressivamente a outros fabricantes e arquitecturas. O seu principal atractivo é oferecer um sistema operativo aberto e modificável, com acesso a *packages* [93] compiladas para diversas aplicações. Rapidamente se tornou um sistema operativo popular e o número de pessoas envolvidas no seu desenvolvimento aumentou exponencialmente. Uma lista das plataformas compatíveis pode ser encontrada em [94]. Outras distribuições estão disponíveis, DD-WRT, FreiFunk, Midge, X-WRT, tendo como base o OpenWRT. Tendo em conta o suporte oferecido pela versão original, optou-se por instalar a versão Kamikaze 8.09.2, a versão estável no início do desenvolvimento do projecto. Entretanto, surgiu uma versão mais recente, designada de BackFire.

4.2.1.2 - Sensores

Os sensores a utilizar para a medição da temperatura podem ser de 2 tipos: analógicos ou digitais. Um sensor analógico traduz a temperatura medida em diferença de potencial no seu pino de saída, pelo que para uma interligação directa nas interfaces GPIO da plataforma Omnima, implicaria a utilização de um conversor analógico digital. Por outro lado, um sensor digital com interface I2C ou SPI, por exemplo, permite uma ligação directa à plataforma através dos pinos GPIO. É representada na figura 4.4, o diagrama da ligação entre os sensores e o *gateway* residencial.

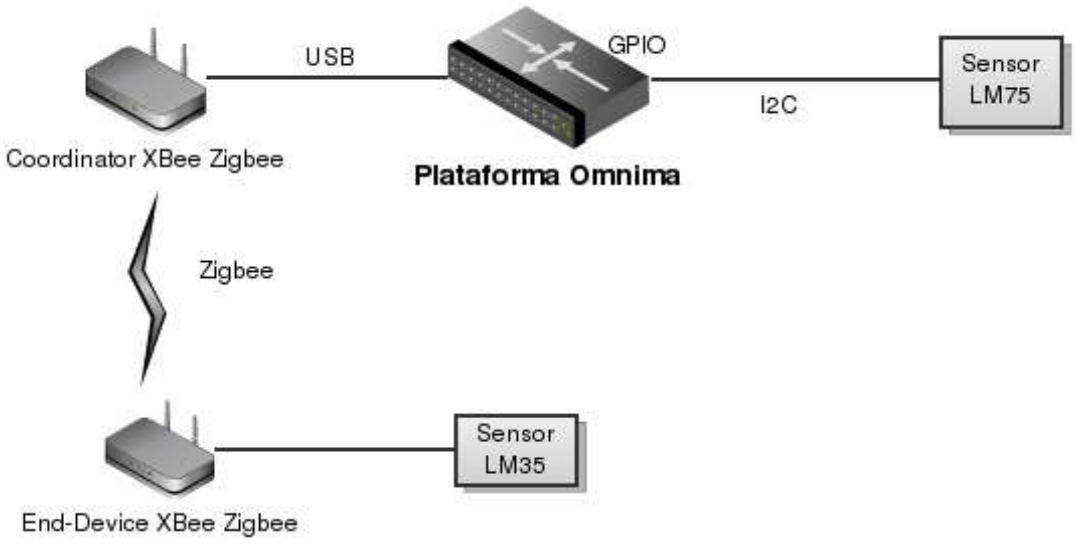

Figura 4.4 – Interligação entre sensores e a plataforma de desenvolvimento.

Para medições de temperatura em locais próximos da plataforma é utilizado um sensor digital ligado directamente. No entanto, para locais em que a distância é maior, serão utilizados módulos XBee que comunicam entre si recorrendo a Zigbee. O módulo XBee receptor dos dados será interligado à plataforma através de USB [95].

4.2.1.2.1 - Wired

Como referido anteriormente, existem 2 tipos de sensores: analógicos ou digitais. Para a ligação directa é utilizado um sensor digital com interface de saída I2C. Este será ligado aos pinos GPIO da plataforma residencial. Existem vários sensores que cumprem estes requisitos, sendo escolhido o seguinte:

Sensor Digital LM75 da National [96]
Características:

- Resolução de 9 bits
- Intervalo de temperatura: -55ºC a +125ºC
- Voltagem: 3 a 5.5V
- Permite comparação de valores
- Preço: $2.67

4.2.1.2.2 - Wireless

No caso da medição de temperatura sem fios, serão utilizados os módulos XBee, apresentado na figura 4.5, para a comunicação de dados. De seguida são apresentada as suas características:

Figura 4.5 – Módulo XBee [97].

- 3.3V @ 40mA
- Taxa de transmissão máxima de 250kbps
- 2mW (+3dBm) na saída
- Alcance de 120m (*Line-of-sight*)
- Antena integrada
- 6 Pinos ADC com resolução de 10-bit
- 8 Pinos digitais IO
- Encriptação 128-bit
- Configuração local ou remota
- Configuração via comandos AT ou API
- Certificação FCC

O sistema é composto por um módulo emissor e um receptor. O receptor necessita de uma *board* adicional para permitir a ligação directa por USB ao *gateway* residencial, representada na figura 4.6.

Figura 4.6 – Módulo XBee com interface USB [95].

Dado que os módulos possuem entradas analógicas e o menor custo de um sensor analógico, foi efectuada uma pesquisa de sensores analógicos. Uma boa solução é o seguinte sensor:

Sensor analógico LM35 da National [98]
Características:

- Calibrado em graus Celsius
- Factor de escala 10mV/˚C
- Precisão de 0.5˚C a 25˚C
- Range: -55˚C até +150˚C
- Voltagem: 4 a 30V
- Preço: 2€

4.2.1.3 - Ar Condicionado

O sistema de ar condicionado disponível para teste do serviço a implementar é o modelo MS-09AH da LG. A comunicação entre o sistema e o *gateway* residencial é efectuada por infra-vermelhos.

4.2.1.4 - Descodificador Infra-Vermelhos

Este dispositivo é responsável por descodificar a sequência de valores lógicos transmitida por um controlo remoto infra-vermelhos. O descodificador utilizado, TSOP1730 da Vishay, identifica sinais de frequência 30KHz. Esta é a frequência utilizada pela LG no modelo de ar condicionado referido acima.

4.2.1.5 – Plataforma OSGi

De entre as várias distribuições possíveis o*pen-source* da *framework* OSGi, apenas algumas implementam a especificação R4. As implementações mais utilizadas são:

- **Eclipse Equinox**: distribuição certificada que implementa as especificações *draft* da *release* 4 do OSGi. Última versão: 3.5.2.

- **Apache Felix**: implementação com suporte à versão 4.2 da plataforma OSGi. Última versão: 2.0.5.
- **Knopflerfish**: A versão KF2 implementa as especificações da *release* 4, sendo a versão 2.3.3 a mais recente. No entanto, a versão 3 ainda em desenvolvimento, já implementa o núcleo da versão 4.2.

As implementações da plataforma OSGi listadas acima, possuem uma grande quantidade de utilizadores e disponibilizam *bundles* com os serviços base da plataforma bem como outros serviços. Idealmente, os *bundles* gerados e testados numa implementação podem ser executados em qualquer uma das implementações. O ambiente de desenvolvimento utilizado foi o Eclipse. Esta ferramenta é um software *Open-Source*, foi desenvolvido em Java e é considerado um IDE muito completo que permite novas funcionalidades através da instalação de *plugins*. Um dos *plugins* disponíveis permite um desenvolvimento de *bundles* mais simples e intuitivo para a plataforma Knopflerfish [99]. Com o desenvolvimento facilitado nesta plataforma, os serviços implementados foram desenvolvidos neste ambiente.

4.2.2 – Casos de utilização

As funcionalidades pretendidas para o sistema resultam em vários casos de utilização representados na figura 4.7:

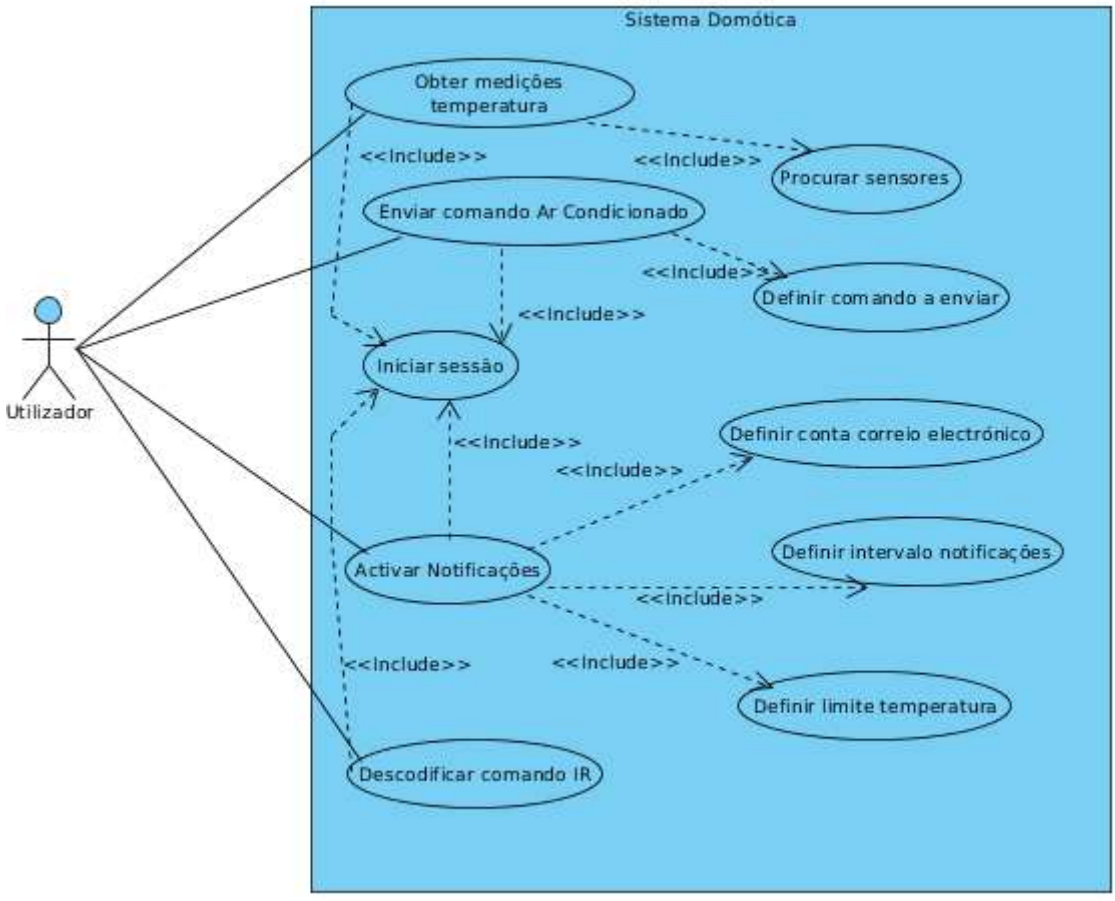

Figura 4.7 – Diagrama de casos de utilização.

4.2.3 – Descrição casos de utilização

Cada caso de utilização é descrito em detalhe nas tabelas seguintes, onde consta a sequência de funcionamento, as condições de ocorrência e os *bundles* que os implementam.

Tabela 4.1 – Descrição caso de utilização 'Iniciar sessão'.

Nome	Iniciar sessão
Descrição	Envio de informação para autenticação no sistema
Actores	Utilizador - Envia credenciais de acesso
Sequência de funcionamento	1 – Sistema solicita autenticação. 2 – Utilizador envia credenciais. 3 – Sistema compara os dados recebidos com os dados armazenados. 4 – Sistema permite acesso se credenciais correctas. Alternativa: 4 – Sistema nega acesso se credenciais incorrectas.
Pré-condição	As credenciais do utilizador devem estar armazenadas no sistema.
Pós-condição	Apresentação da página inicial da interface Web.
Bundles	WebService

Tabela 4.2 – Descrição caso de utilização 'Obter medições temperatura'.

Nome	Obter medições temperatura
Descrição	Fornece informação sobre a temperatura actual.
Actores	Utilizador – Efectua pedido de medição.
Sequência de funcionamento	1 – Utilizador solicita pedido de informação sobre medições. 2 - Sistema acede às medições dos sensores disponíveis. 3 – Sistema apresenta resultados na interface Web. Alternativa: 3 – Sistema apresenta página de erro em caso de anomalia do passo 2.
Pré-condição	Utilizador deve ter sessão iniciada. Sistema deve ter verificado o estado dos sensores.
Pós-condição	Apresentação das medições obtidas.
Bundles	WebService, TempService

Tabela 4.3 – Descrição caso de utilização 'Procurar sensores'.

Nome	Procurar sensores
Descrição	Fornece informação sobre o estado actual dos sensores (*online*/*offline*).
Actores	Utilizador – Efectua pedido de informação sobre estado dos sensores.
Sequência de funcionamento	1 – Utilizador solicita pedido de informação sobre estado dos sensores. 2 - Sistema tenta comunicação com sensores. 3 – Sistema apresenta resultados na interface Web. Alternativa: 3 – Sistema apresenta página de erro em caso de anomalia do passo 2.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação do estado dos sensores.
Bundles	WebService, TempService

Tabela 4.4 – Descrição caso de utilização ' Enviar comando Ar Condicionado'.

Nome	Enviar comando Ar Condicionado
Descrição	Envia ordem para sistema de ar condicionado.
Actores	Utilizador – Efectua pedido de envio de comando para o sistema de ar condicionado.
Sequência de funcionamento	1 – Utilizador solicita pedido de envio de comando. 2 - Sistema envia comando solicitado com confirmação. Alternativa: 2 – Apresentada página Web a indicar erro.
Pré-condição	Utilizador deve ter sessão iniciada. O comando a enviar deverá ter sido especificado pelo utilizador.
Pós-condição	Apresentação de informação se comando gerado com sucesso.
Bundles	WebService, AirService

Tabela 4.5 – Descrição caso de utilização 'Definir comando a enviar'.

Nome	Definir comando a enviar
Descrição	Definição da ordem a enviar para o sistema de ar condicionado.
Actores	Utilizador – Selecciona ordem a enviar.
Sequência de funcionamento	1 – Utilizador selecciona ordem a enviar. 2 - Sistema procede para o envio. Alternativa: 2 – Sistema apresenta informação de erro se comando configurado incorrectamente.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação de informação se comando gerado com sucesso.
Bundles	WebService

Tabela 4.6 – Descrição caso de utilização 'Activar notificações'.

Nome	Activar notificações
Descrição	Activação de mecanismo de notificações.
Actores	Utilizador – Activa envio de notificações.
Sequência de funcionamento	1 – Utilizador activa mecanismo de notificações. 2 - Sistema executa configurações e activa mecanismo. 3 – Sistema apresenta informação relativa à activação do mecanismo. Alternativa: 3 – Sistema apresenta informação de erro se mecanismo configurado incorrectamente.
Pré-condição	Utilizador deve ter sessão iniciada. Utilizador deverá ter inserido informação necessária para activação do serviço: conta de correio electrónico de destino, temperatura limite e intervalo entre notificações.
Pós-condição	Apresentação de informação com as configurações efectuadas.
Bundles	WebService, MailService

Tabela 4.7 – Descrição caso de utilização 'Definir conta correio electrónico'.

Nome	Definir conta correio electrónico.
Descrição	Definição do endereço da conta de correio electrónico destinatária das notificações
Actores	Utilizador – Insere endereço.
Sequência de funcionamento	1 – Utilizador insere endereço. 2 - Sistema procede para o envio. Alternativa: 2 – Sistema apresenta informação de erro caso parâmetros incorrectos.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação de informação com as configurações efectuadas.
Bundles	WebService

Tabela 4.8 – Descrição caso de utilização 'Definir intervalo notificações'.

Nome	Definir intervalo notificações.
Descrição	Definição do intervalo entre notificações.
Actores	Utilizador – Selecciona intervalo.
Sequência de funcionamento	1 – Utilizador selecciona intervalo.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação de informação se comando gerado com sucesso.
Bundles	WebService

Tabela 4.9 – Descrição caso de utilização 'Definir temperatura limite'.

Nome	Definir temperatura limite.
Descrição	Definição da temperatura limite para envio de notificações.
Actores	Utilizador – Selecciona limite.
Sequência de funcionamento	1 – Utilizador selecciona limite.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação de informação com as configurações efectuadas.
Bundles	WebService

Tabela 4.10 – Descrição caso de utilização 'Descodificar comando IR'.

Nome	Descodificar comando IR.
Descrição	Descodificação de dados enviados por controlo remoto.
Actores	Utilizador – Envia pedido de descodificação.
Sequência de funcionamento	1 – Utilizador envia pedido. 2 – Sistema descodifica sinal. Alternativa: 2 – Sistema apresenta página de erro se anomalia ocorrer no passo 2.
Pré-condição	Utilizador deve ter sessão iniciada.
Pós-condição	Apresentação da informação obtida.
Bundles	WebService, IrService

4.3 - Desenvolvimento

Esta secção apresenta o desenvolvimento do sistema nas suas diversas vertentes.

4.3.1 - Plataforma

As interfaces GPIO da plataforma, apresentas na figura 4.8, não se encontravam acessíveis pelo facto de estas conterem leds. Estes leds sinalizavam a utilização das entradas USB e Ethernet. Desta forma, foi necessário remover alguns leds para permitir a ligação com os dispositivos.

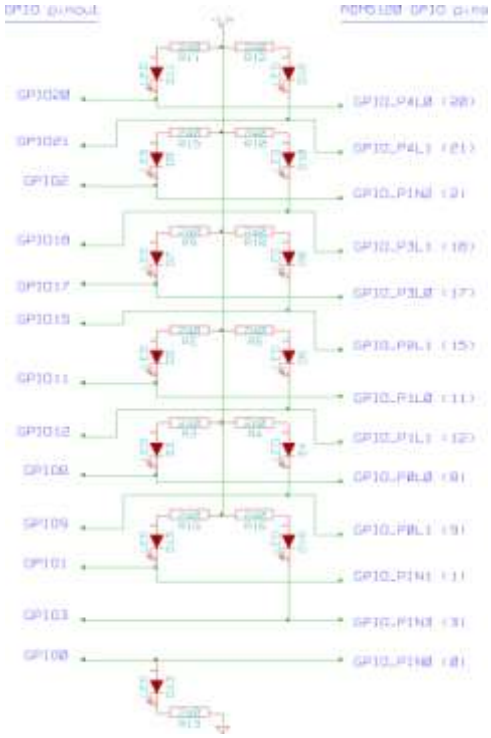

Figura 4.8 – Esquema das interfaces GPIO disponíveis na plataforma [100].

Existem diversas formas de acesso às interfaces GPIO. No entanto, a frequência de acesso é importante nas aplicações a desenvolver. A primeira forma de acesso é através do sistema de ficheiros /sys/class/gpio. A frequência de acesso ronda 400Hz. Outro método de acesso é através de *file descriptors*, com uma frequência de cerca de 150 KHz. A última forma de acesso é através de módulos kernel para acesso directo às interfaces GPIO, com frequências na ordem dos MHz. O acesso através de *file descriptors* revelou-se suficiente para as aplicações desenvolvidas.

Com o objectivo de dotar a plataforma com uma versão mais recente do sistema operativo e consequentemente ter acesso a *packages* opcionais actualizadas, foi necessário actualizar o *firmware* da plataforma. A distribuição utilizada é a Kamikaze 8.09.2. Dado que a plataforma possui uma memória *flash* de 2MB, seria impossível instalar uma versão normal devido à reduzida capacidade. Sendo assim, um suporte amovível USB é utilizado com o sistema de ficheiros. A configuração efectuada foi baseada em [101], com algumas alterações. A actualização implicou a instalação da nova imagem do *kernel* correspondente, através da interface UART. A imagem continha apenas as *packages* mínimas para ser possível aceder ao router sem ultrapassar o limite imposto pela capacidade da memória *flash*. As *packages* necessárias para além das básicas podem ser compiladas com a *toolchain* disponibilizada com a distribuição ou através do repositório. O acesso à plataforma é efectuado através de SSH, implementado com a aplicação Dropbear. A utilização do SDK e respectivos ambientes de compilação revelaram-se fulcrais no desenvolvimento do projecto, pois só com estas ferramentas seria possível executar compilações para a arquitectura do *gateway* residencial.

4.3.2 – Sensores

4.3.2.1 – Sensor digital

O sensor LM75 da National, utiliza o protocolo de comunicação I2C. De acordo, com a descrição em 4.4.1, são necessárias 2 linhas de comunicação: SDA e SCL. São necessárias portanto 2 interfaces GPIO, como ilustrado na figura 4.9. As duas linhas de alimentação são também obtidas a partir da plataforma. As *packages* disponíveis para acesso a dispositivos I2C foram compiladas e posteriormente instaladas.

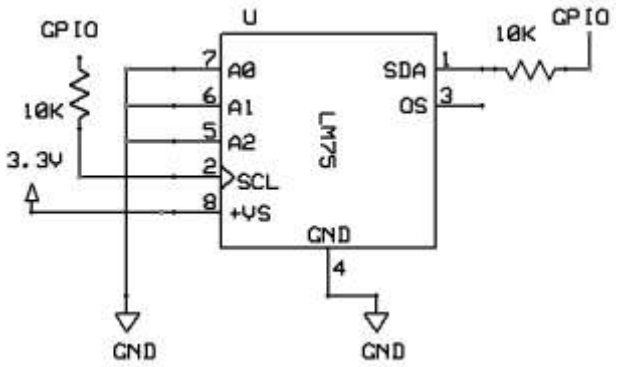

Figura 4.9 – Esquema electrónico da montagem do sensor digital LM75.

4.3.2.2 – Sensor Zigbee

Na interligação do coordenador da rede Zigbee com o *gateway* residencial foi utilizada uma *board* USB. Para a instalação dos *drivers* na plataforma foi necessário compilar e instalar as *packages* kmod-usb-serial e kmod-usb-serial-ftdi. É requerida a instalação anterior da *package* kmod-usb-core. A referida *board* também permite a programação dos módulos XBee com o *software* X-CTU, da Digi. O nó *End-Device* da rede foi integrado com um sensor de temperatura analógico, através de uma das entradas analógicas do módulo (Pino 20). No pino 15, é conectado um led que permite identificar o estado da associação do módulo, isto é, o led pisca com uma maior frequência no caso de o módulo estar associado a uma rede. O circuito resultante da integração é apresentado na figura 4.10.

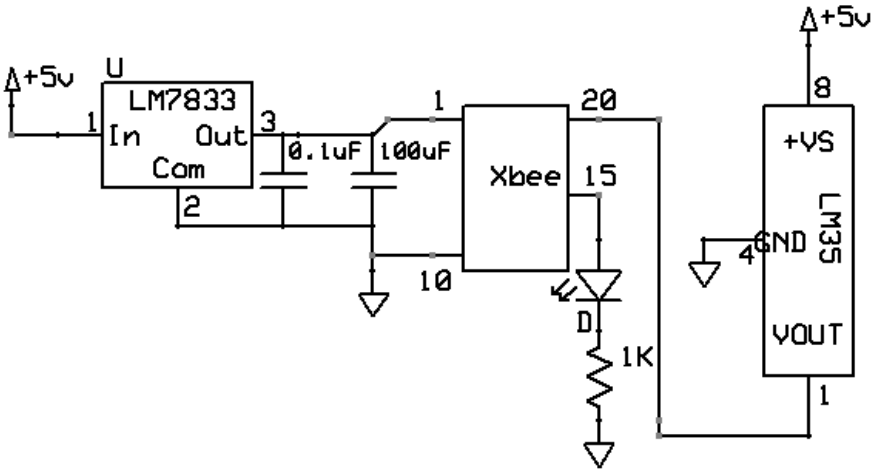

Figura 4.10 – Esquema electrónico da montagem do sensor Zigbee.

4.3.2.3 – Software X-CTU

X-CTU [102] é uma aplicação para sistemas operativos Windows, disponibilizada pela Digi [103], para a programação dos módulos XBee. O *firmware* encontrava-se desactualizado sendo necessário efectuar um *update* para uma versão mais recente. A versão recomendada

para a utilização da API Java dos módulos XBee [104] é a ZB Pro, pelo que foi necessário seguir o procedimento descrito em [105]. Os módulos permitem 2 modos de operação: AT ou API. O modo AT é uma forma de operação focada na ligação ponto a ponto, em que o módulo emissor simplesmente envia os dados para outro módulo através de um endereço especificado. Este modo poderia ter sido utilizado na aplicação em causa, mas a API utilizada solicitava o modo de operação API. Este modo permite no futuro a adição de novos módulos de comunicação sem necessidade de grandes alterações no sistema actual. O modo API permite o envio e a recepção de pacotes.

Modo API

- Permite a recepção de leituras de 1 ou mais módulos.
- Mecanismo de reenvio e confirmação. O módulo emissor recebe um ACK no caso de sucesso de envio. No caso de falha de recepção, o pacote é reenviado.
- Os pacotes recebidos diferenciam-se através do acesso ao endereço de origem.
- Configuração remota de módulos.
- Envio de pacotes para grupos de módulos ou para todos os nos da rede (*broadcast*).
- Pacotes contêm checksum para verificação de integridade.
- Verificação da qualidade do sinal (RSSI).

De seguida, foi necessário definir o ID da rede dos 2 módulos (PAN ID) e activar o modo AP (AP=2). No dispositivo *End-Device* definiu-se o parâmetro D0=2 (entrada analógica no pino 20), IR=5000 e D5=1. O parâmetro IR especifica o intervalo em milissegundos entre envio de leituras das entradas definidas e o parâmetro D5 activa o led indicador de estado. A interface das opções de configuração é apresentada na figura 4.11.

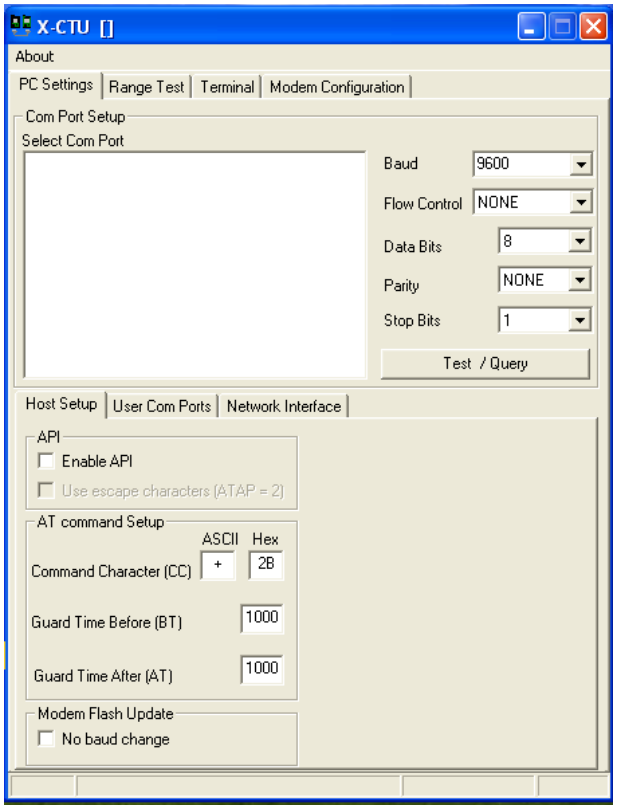

Figura 4.11 – Interface do *software* X-CTU.

4.3.3 - Emissor Infra-Vermelhos

Numa das interfaces GPIO acoplou-se um circuito emissor de infra-vermelhos para comunicação com o sistema de ar condicionado. O circuito é composto por um transístor NPN 2N2222 [106], um led IR emissor e uma resistência para limitar a corrente, ilustrado na figura 4.12.

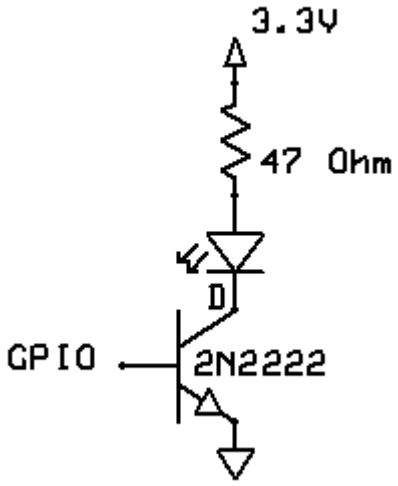

Figura 4.12 – Esquema electrónico da montagem do circuito emissor de IR.

4.3.4 - Descodificador Infra-Vermelhos

O seguinte circuito tem como função descodificar o sinal infra-vermelhos gerado pelo controlo remoto do sistema de ar condicionado. No entanto, pode ser aplicado na descodificação de qualquer protocolo que utilize a frequência de modulação de 30KHz. As transições na saída do receptor são interpretadas numas das interfaces GPIO da plataforma, como representado na figura 4.13.

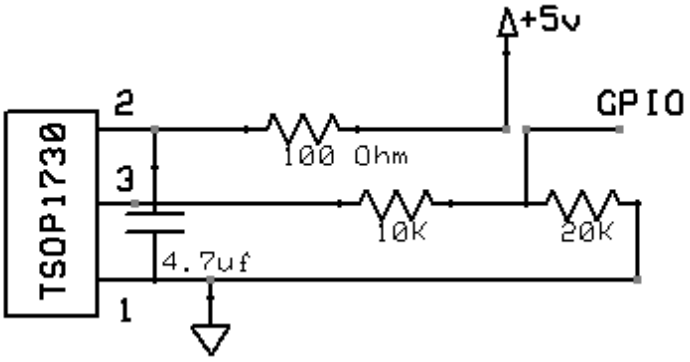

Figura 4.13 – Esquema electrónico da montagem do circuito descodificador de IR.

4.3.5 - Protocolo Infra-Vermelhos LG MS09AH

Com o intuito de gerar sinais IR que fossem interpretados pelo sistema de ar condicionado, foi necessário identificar o protocolo utilizado, uma vez que estão disponíveis inúmeros protocolos e os sinais transmitidos são modulados a frequências diferentes. O

primeiro passo consistiu em identificar a frequência do sinal modulado transmitido pelo controlo remoto. Para isso foi utilizado um led receptor de infra-vermelhos ligado a um osciloscópio, representado na figura 4.14.

O led receptor gera uma pequena voltagem quando atingido por um sinal infra-vermelhos. Esta pequena diferença de potencial permitiu identificar a frequência do sinal com recurso às capacidades de *trigger* do osciloscópio.

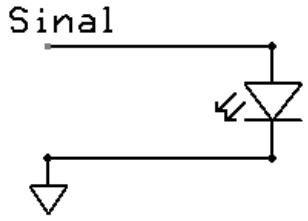

Figura 4.14 – Circuito utilizado para leitura de sinal IR no osciloscópio.

De seguida, com um receptor IR (TSOP1730), compatível com a frequência de 30KHz, analisou-se os bits transmitidos e os tempos de transição entre níveis lógicos. O pino de saída do receptor é activo a nível baixo, o que significa que quando recebe um sinal de frequência 30Khz apresenta 0V na saída. Com a recepção de vários comandos, foi possível verificar que todos continham um *header*, seguido de 28 bits.

O *header* transmitido permanece 8,8 milissegundos no nível alto e 4 milissegundos no nível baixo. Os bits 0 e 1, têm tempos totais, de 2 e 1 milissegundos respectivamente. O nível alto em ambos os bits mantém-se durante cerca de 600 micro segundos.

De acordo com os dados obtidos, observa-se que o protocolo utiliza modulação *pulse distance*, tal como no protocolo desenvolvido pela NEC [107]. Assumiu-se que o valor lógico "1" corresponde ao maior tempo de bit e o valor lógico "0" ao menor tempo. Dos 28 bits recebidos, os primeiros 8 bits são sempre iguais, 0x88. De referir, que os comandos que especificam a temperatura baseiam-se no valor final da temperatura, o que significa que o código de subida da temperatura para 19ºC é idêntico ao código de descida para 19ºC. O controlo remoto utilizado não envia sinais repetidos. Em anexo, apresentam-se os comandos obtidos.

4.3.6 - Java Native Interface

A interface Java *Native* [108-110], é uma funcionalidade da plataforma Java, que permite integrar código nativo C e C++ em aplicações Java. O código nativo não é independente do ambiente de execução, depende da arquitectura do CPU, ao contrário de código Java que é independente do sistema onde é executado. Apesar desta limitação, linguagens nativas continuam a ser necessárias, nomeadamente para programação de mais baixo nível e para efeitos de performance. Dessa forma, a Java *Native Interface* torna-se uma ferramenta útil para o desenvolvimento de uma aplicação que recorra às linguagens Java, C e C++. A interface permite a invocação de funções nativas, implementadas em bibliotecas nativas e vice-versa.

No contexto da aplicação desenvolvida, é necessário aceder às interfaces GPIO da plataforma, sendo que o acesso ao *hardware* é facilitado mediante a utilização de código nativo. Para conseguir a integração desejada, o código C necessário foi compilado em

diversas bibliotecas de acordo com o serviço a disponibilizar. O processo de integração é composto por:

- Implementar código Java com a declaração das funções nativas.
- Compilar ficheiro Java.
- Gerar ficheiro JNI-header a partir do ficheiro .class com javah –jni.
- Implementar funções nativas em C incluindo o *header* anteriormente criado.
- Compilar ficheiro C e gerar biblioteca nativa para a arquitectura do *host* (MIPS).
- Incluir biblioteca no *bundle* do serviço.

Para gerar as bibliotecas nativas foi utilizada a toolchain obtida em 4.2.1.1.1, com recurso a uma versão do gcc compatível.

4.3.7 – *Bundles* desenvolvidos

As diferentes funcionalidades de um sistema desenvolvido sobre OSGi, tipicamente são implementadas em vários serviços de forma a aproveitar a modularidade oferecida pela plataforma. Um serviço OSGi é a instanciação de um objecto Java registado na plataforma, que normalmente implementa uma interface acessível por outros serviços. O sistema implementado consistiu em 5 serviços disponibilizados sob a forma de *bundles*:

- WebService: serviço responsável pelo conteúdo do portal Web.
- AirService: serviço que permite o envio de comandos para o sistema de ar condicionado.
- MailService: serviço de notificações por correio electrónico.
- IrService: serviço que permite descodificar comandos infra-vermelhos.
- TempService: serviço que acede às medições dos sensores.

Os serviços foram executados num ambiente composto por uma máquina virtual Java, JamVM, e a implementação *open-source* das classes *standard* do Java, Gnu Classpath. Os serviços AirService, MailService, IrService, TempService são exportados para a plataforma através da declaração *Export-Package* nos ficheiros *manifest* do *bundle* respectivo. O diagrama de sequência da figura 4.15, mostra a ordem pela qual os eventos devem ocorrer para que os serviços executem de forma correcta.

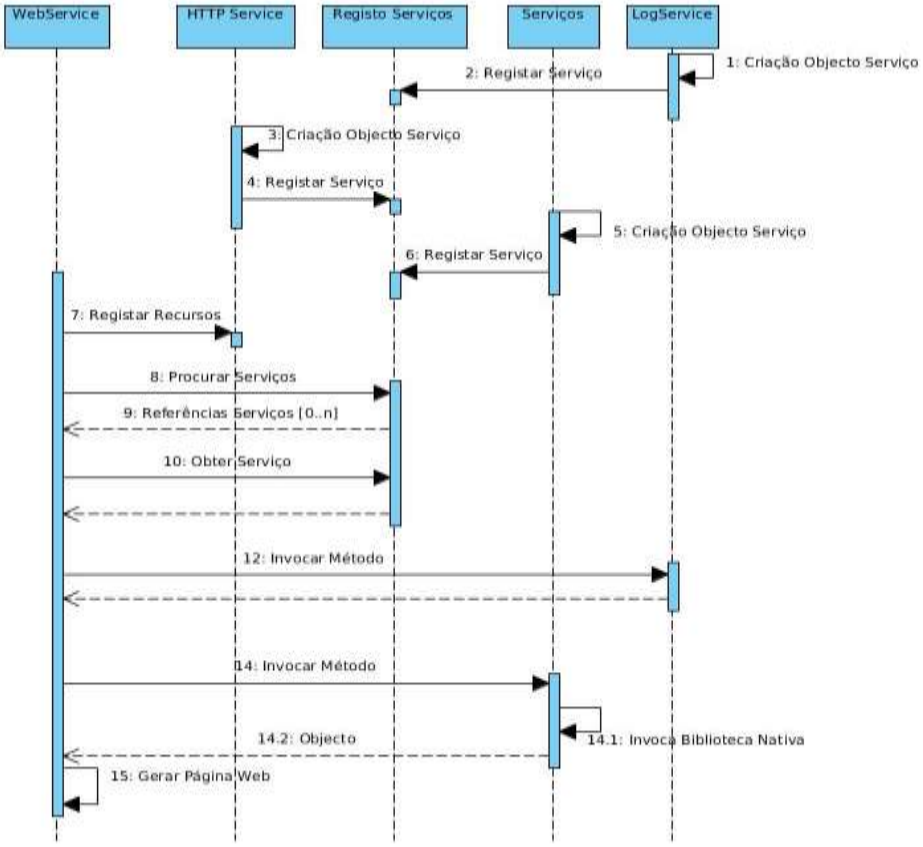

Figura 4.15 – Diagrama de sequência do funcionamento de serviços.

4.3.7.1 – WebService

O *bundle* WebService implementa o serviço do mesmo nome, que utiliza o serviço HTTP disponibilizado pelo servidor Jetty, na versão 2.0.4, proveniente da implementação Apache Felix para exportação de recursos acessíveis via HTTP. O *bundle* HTTPService base, na implementação Knopflerfish revelou-se extremamente lento na plataforma pelo que a opção recaiu sobre o servidor Jetty. O serviço HTTP permite a disponibilização de conteúdos estáticos ou dinâmicos. Os recursos dinâmicos são gerados utilizando a API Java Servlet. *Servlets* são objectos Java capazes de gerar conteúdo dinâmico de acordo com os pedidos recebidos. Um exemplo da interface proporcionada por este serviço é apresentado na figura 4.16.

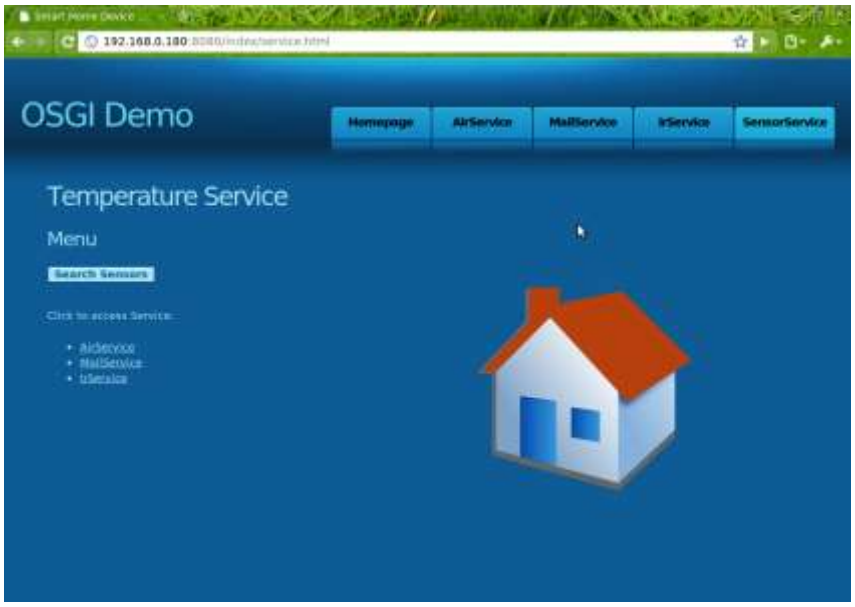

Figura 4.16 – Interface Web disponibilizada pelo *bundle* WebService.

A figura 4.17 apresenta o diagrama de pacotes, que explicitam as dependências do serviço:

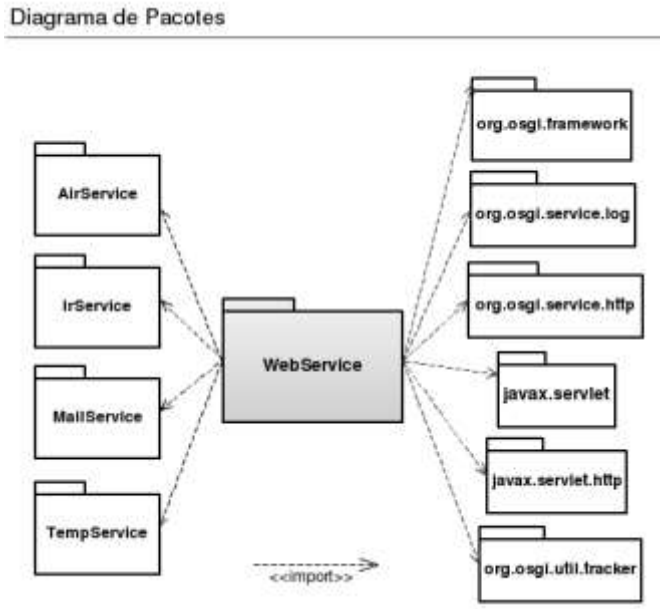

Figura 4.17 – Diagrama de pacotes do *bundle* WebService.

Este serviço depende dos restantes serviços implementados para o seu correcto funcionamento. As dependências, como já referido, estão definidas no ficheiro *manifest* do *bundle* na declaração *Import-Package*. A figura 4.18 apresenta o diagrama de classes correspondente a este serviço.

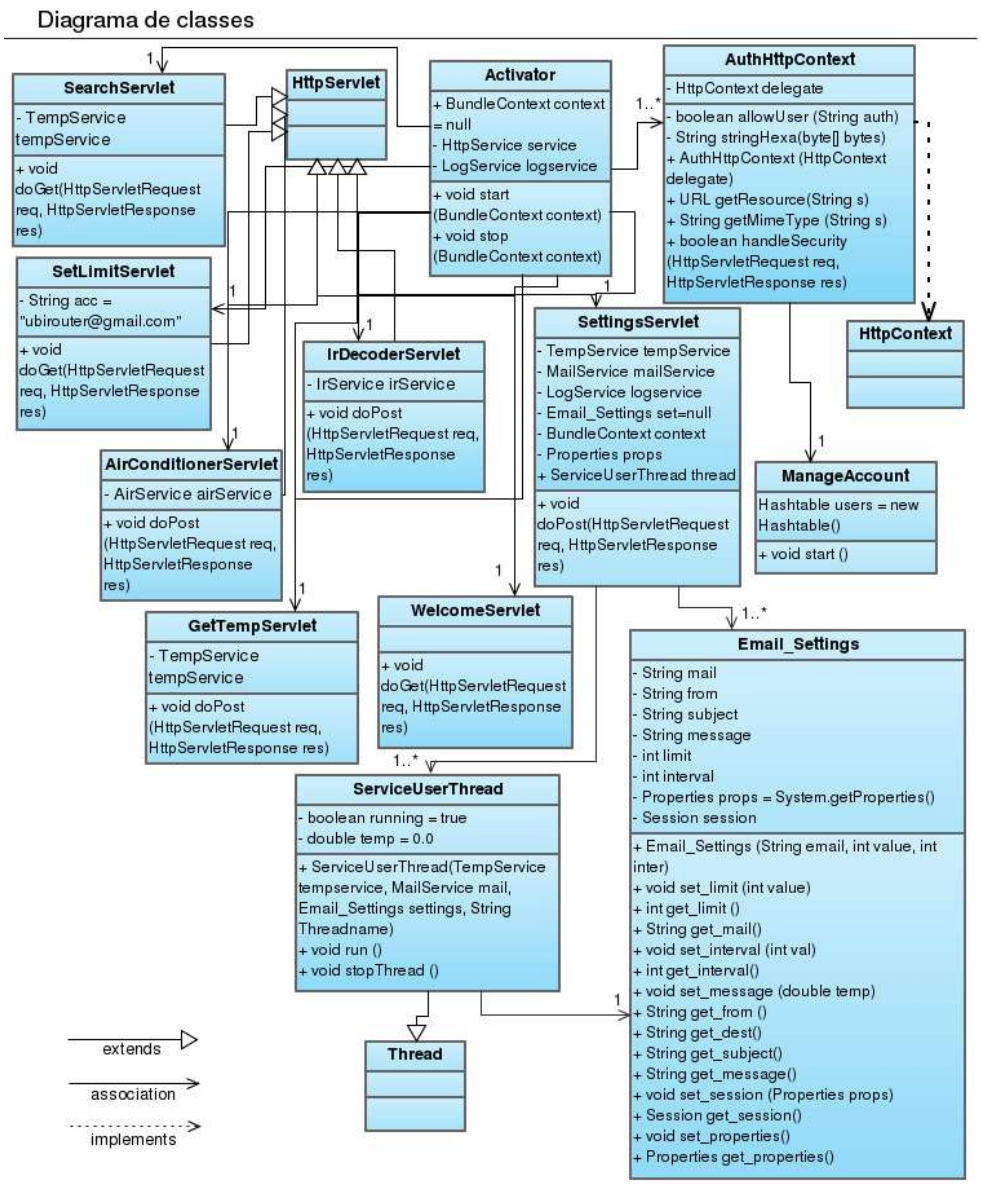

Figura 4.18 – Diagrama de classes de WebService.

Para acesso ao portal Web, implementou-se o mecanismo de *DIgest Authentication*. Em vez da transmissão das credenciais em *cleartext* como acontece em *Basic Authentication*, este mecanismo utiliza o algoritmo de *hashing* MD5 para envio das credenciais encriptadas. O cliente ao solicitar o acesso ao servidor, utiliza um pedido HTTP não autenticado. O servidor responde com o código HTTP 401 Unauthorized, um *nonce* (valor aleatório) e um *header* HTTP solicitando *Digest Authentication*. Os parâmetros utilizados para o processo de autenticação são:

- Username: nome do utilizador
- Password: código de acesso
- Nonce: valor aleatório associado a uma resposta com o código 401
- Realm: palavra que indica o serviço a ser acedido
- DigestURI: URI presente no Request-URI
- Qop: "Quality of protection" utilizada pelo servidor
- NonceCount: número de requests

- ClientNonce: valor gerado pelo cliente

O *browser* gera uma caixa de diálogo para inserção das credenciais e é gerada a resposta **MD5 (HA1:nonce:nonceCount:clientNonce:qop:HA2)** em que **HA1 = MD5 (username:realm:password)** e **HA2 = MD5 (method:digestURI).** O servidor com base na cópia das credenciais guardada em local seguro e nos restantes parâmetros gera uma resposta que posteriormente é comparada com a resposta recebida. Se os 2 *hashes* coincidirem o acesso é permitido, já no caso contrário o acesso é negado. O *bundle* WebService é composto por:

/META-INF/MANIFEST.MF
/WebService/* (classes)
/WebService/resources/* (páginas HTML)
/WebService/resources/css/
/WebService/resources/js/
/WebService/resources/images/

4.3.7.2 - AirService

O *bundle* AirService permite o envio de comandos infra-vermelhos para o sistema de ar condicionado. Para tal é necessário controlar a interface GPIO apenas acessível através da biblioteca libir.so. A biblioteca contém a sequência de valores a enviar de acordo com o comando definido pelo utilizador e modula o sinal IR à frequência de 30KHz. Uma vez que são conhecidos os tempos de bit, é fácil gerar o sinal infra-vermelhos pretendido. Para obter um sinal modulado a 30KHz, o led tem de "piscar" 30000 vezes por segundo, portanto um tempo de bit de 600µs equivale a "piscar" 18 vezes.

O *bundle* está estruturado da seguinte forma:

/META-INF/MANIFEST.MF
/airservice/
/airservice/impl/
/airservice/lib/libir.so

O diagrama de classes e pacotes é apresentado na figura 4.19:

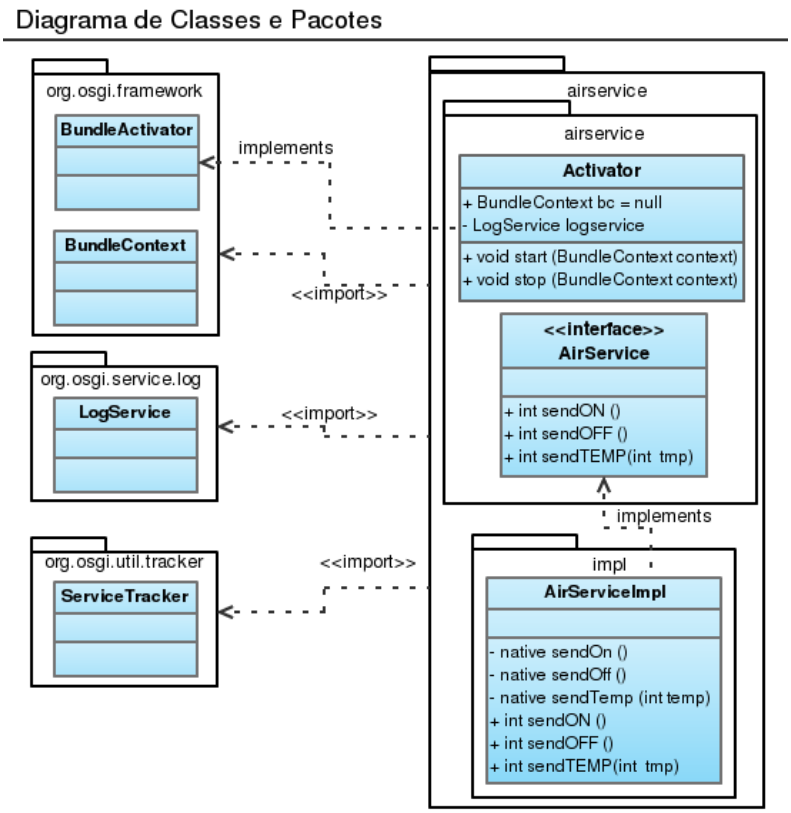

Figura 4.19 – Diagrama de classes e pacotes de AirService.

4.3.7.3 - MailService

O sistema de notificações por correio electrónico é implementado pelo MailService. Para efeitos de testes, foi criada uma conta Gmail. O utilizador para activação do serviço necessita definir o endereço de destino da notificação, o intervalo entre notificações e a temperatura limite a partir da qual é activo o sistema. Se activo, o utilizador tem a possibilidade de desactivar o envio de notificações.

Este serviço utiliza a API Gnu JavaMail [111]. No entanto, a API da Sun não pode ser utilizada devido à implementação das classes ser incompatível com as classes base do Gnu Classpath. No entanto, existe uma versão compatível disponibilizada pelo projecto Gnu Classpath Extensions. A API é apenas compatível com a especificação 1.3 da Sun, que neste momento está na versão 1.4.3. A compilação da API depende de Gnu JAF e de Gnu inetlib, que por sua vez contam com outras dependências. De referir que o servidor SMTP do Gmail solicita o estabelecimento de uma ligação segura, sendo portanto necessário utilizar um protocolo de transporte seguro SSL ou TLS. Este serviço utiliza o protocolo TLS na porta 587. O serviço é disponibilizado para a plataforma, através da interface MailService.MailService, na declaração Export-Package no *manifest*. O diagrama de classes e pacotes correspondente a este serviço é apresentado na figura 4.20. O ficheiro JAR correspondente a este serviço estrutura-se da seguinte forma:

/META-INF/MANIFEST.MF

/MailService/*

/MailService/impl/*

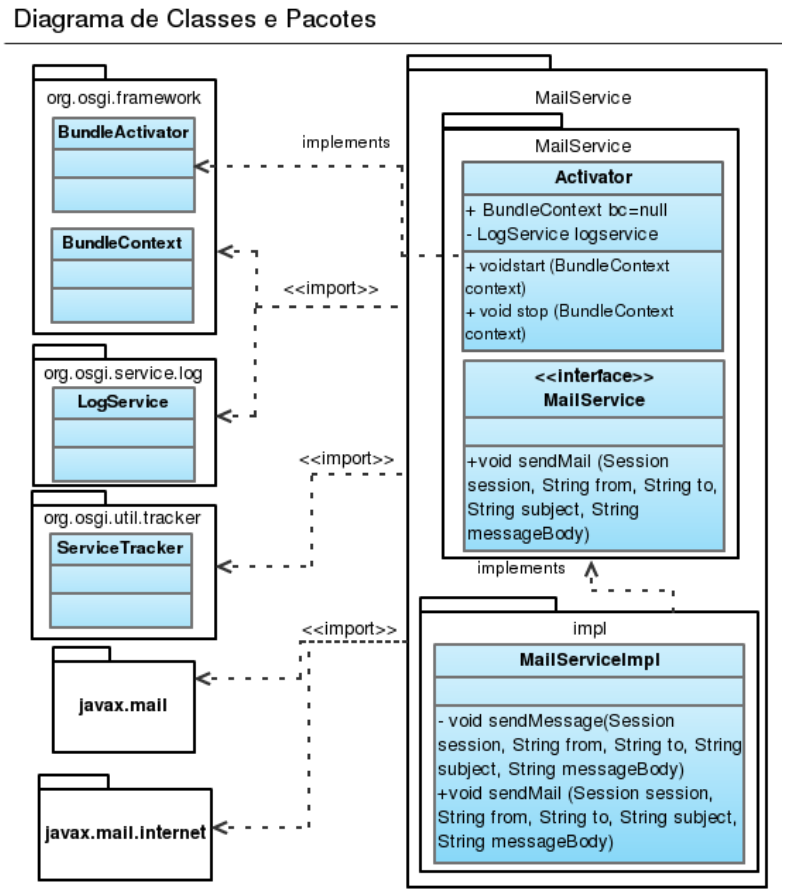

Figura 4.20 – Diagrama de classes e pacotes de MailService.

4.3.7.4 - IrService

O IrService permite descodificar dados enviados por infra-vermelhos a partir de um controlo remoto. Este serviço está no entanto limitado a sinais de frequência de 30 KHz dado que o receptor utilizado apenas identifica esta frequência. O serviço focou-se no protocolo IR utilizado pelo sistema de ar condicionado da LG, limitando o número de bits a serem identificados e tendo em conta os intervalos de transmissão de cada bit.

A técnica utilizada na descodificação consiste em cronometrar o tempo entre transições de nível. Com o conhecimento dos tempos implementados pelo protocolo, basta realizar uma comparação com o intervalo de tempo do nível baixo, para se atribuir o bit 1 ou 0. Outra técnica que pode ser utilizada é a verificação do valor lógico da interface GPIO, a partir da metade do tempo de duração do 1º nível alto e realizar repetidamente a verificação em intervalos com o dobro desse valor. O diagrama de classes e pacotes correspondente a este serviço é apresentado na figura 4.21. A interface de acesso ao serviço é declarada no *manifest* do *bundle* que apresenta a seguinte estrutura:

/META-INF/MANIFEST.MF
/IrService/*
/IrService/impl/*
/IrService/lib/libcode.so

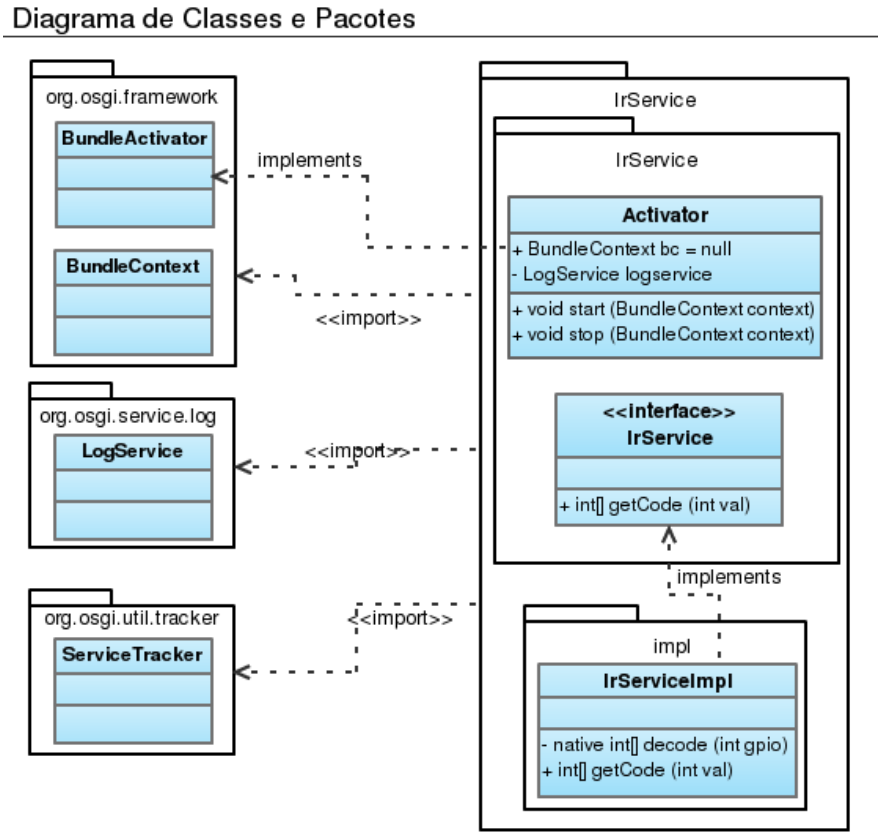

Figura 4.21 – Diagrama de classes e pacotes de IrService.

4.3.7.5 - TempService

O *bundle* TempService, implementa um serviço de acesso a sensores de temperatura. Um dos sensores está acessível através da interface GPIO, sendo que as funções de acesso estão incluídas numa biblioteca nativa liblm75.so, devidamente identificada na declaração *Bundle-NativeCode*. Para garantir a correcta configuração do sensor é necessário executar um pequeno script para especificação das interfaces GPIO utilizadas pela *package* instalada previamente. Este processo é efectuado na primeira execução do *bundle*. O sensor analógico é acedido através da interacção com o coordenador da rede Zigbee. Para acesso a este dispositivo, é utilizada a API RXTX [112] para comunicação com a porta série e a API de comunicação com os módulos XBee. As classes gnu.io são adicionadas ao directório de classes e importadas no *manifest* do *bundle*. A ligação por USB cria uma porta série acessível /dev/ttyUSB*, dependendo da interface USB utilizada. O *bundle* está disposto composto da seguinte forma:

/META-INF/MANIFEST.MF

/tempservice/*

/tempservice/impl/*

/lib/liblm75.so

O diagrama de classes e pacotes correspondente a este serviço é apresentado na figura 4.22.

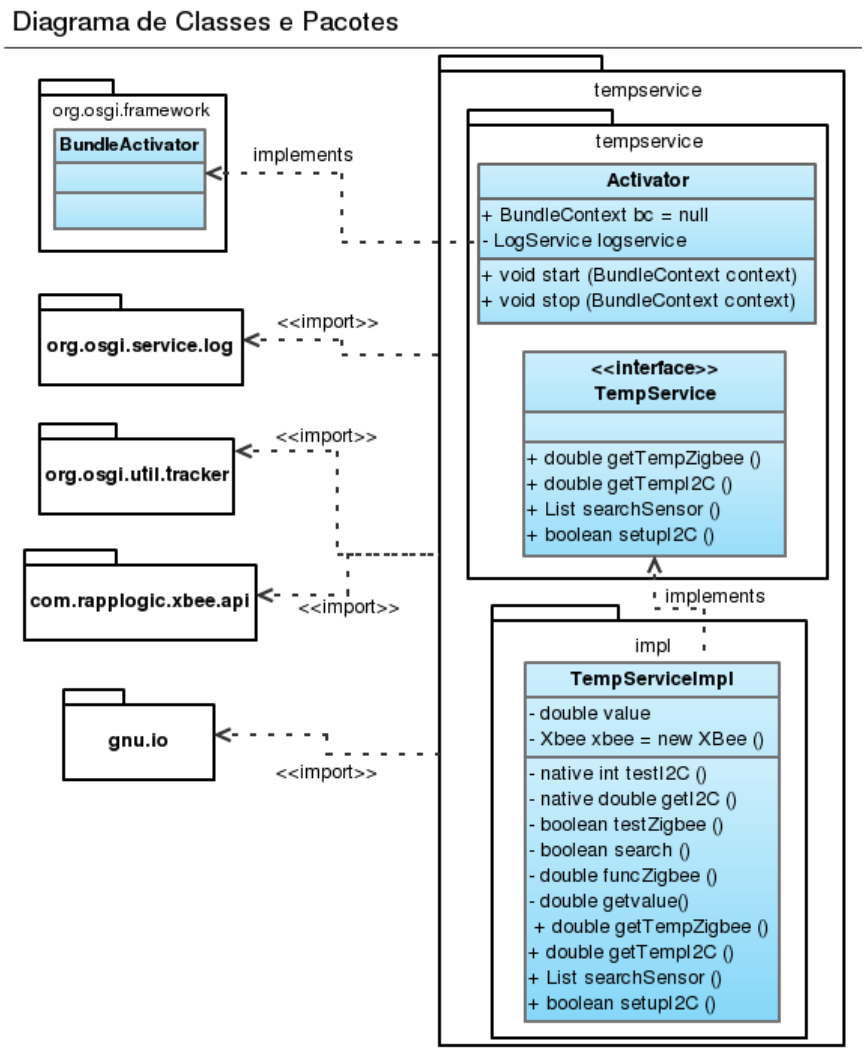

Figura 4.22 – Diagrama de classes e pacotes de TempService.

4.3.8 – Implementação OSGi

A plataforma Knopflerfish define um conjunto de *bundles* mínimos para o seu correcto funcionamento. De referir que a última versão disponível (2.3.3) não foi utilizada pois apresentou um funcionamento mais lento que as versões antecessoras, sendo assim utilizada a versão anterior, 2.3.2. Além destes, são adicionados os *bundles* LogReader e Apache Felix HTTP Jetty. O serviço LogReader permite o *output* de eventos gerados tanto pela Framework como por *bundles*. O serviço de HTTP é disponibilizado pelo *bundle* Jetty. O ambiente de execução mínimo que permite o correcto funcionamento dos serviços implementados, é apresentado na figura 4.23.

```
id  level/state name
-------------------
 0  0/active    System Bundle           1  1/resolved  LogService-API
 2  1/resolved  Console-API             3  1/active    cm-API
 4  1/active    LogService-IMPL         5  1/active    Console-IMPL
 6  1/active    TTY-Console-IMPL        7  1/active    FW-Commands-IMPL
 8  1/active    LogCommands-IMPL        9  1/active    UserAdmin-API
10  1/active    LogReader              11  1/active    Apache Felix Http Jetty
```

Figura 4.23 – Conjunto de *bundles* mínimos necessários para a execução dos serviços desenvolvidos.

Um exemplo das mensagens capturadas pelo serviço LogReader, apresentado na figura 4.24:

```
 INFO: org.apache.felix.http.jetty: BundleEvent RESOLVED
 INFO: org.apache.felix.http.jetty: ServiceEvent REGISTERED
 INFO: org.apache.felix.http.jetty: BundleEvent STARTED
 INFO: system.bundle: FrameworkEvent STARTED
Framework launched
 INFO: org.apache.felix.http.jetty: ServiceEvent REGISTERED
 DEBUG: org.apache.felix.http.jetty: ServiceEvent MODIFIED
 INFO: org.apache.felix.http.jetty: Started jetty 6.1.x at port 8080
```

Figura 4.24 – Registos gerados pelo serviço LogReader.

A implementação permite uma personalização do ambiente de execução através de um ficheiro de configurações, onde podem ser especificados os *bundles* a instalar, as suas configurações e propriedades da implementação. As propriedades importantes para o correcto funcionamento dos serviços implementados são:

- org.knopflerfish.verbosity: permite definir o nível de mensagens de *debug* geradas pela plataforma, em que o 0 é o valor com menor número de mensagens e 4 com o maior.
- org.knopflerfish.gosg.jars: especificação do directório de *bundles* a instalar.
- org.knopflerfish.framework.debug.packages: permite activar o *debug* na resolução de dependências de pacotes.
- org.knopflerfish.framework.debug.errors: permite activar as mensagens de erro da plataforma.
- org.knopflerfish.framework.debug.classloader: permite activar as mensagens de *debug* no carregamento de classes necessárias.
- org.OSGi.framework.system.packages: especifica os pacotes extra necessários para além dos obtidos por omissão pela *framework*.
- org.apache.felix.http.jettyEnabled: permite activar o serviço HTTP disponibilizado pelo *bundle* Jetty.
- org.OSGi.service.http.port: especifica a porta utilizada pelo serviço HTTP.
- org.knopflerfish.framework.bundlestorage.file.always_unpack: permite descompactar o conteúdo dos *bundles* do directório de cache da plataforma.

4.3.9 - Custo sistema

O custo total do sistema é resumido na tabela 4.11:

Tabela 4.11 – Custo dos componentes necessários ao desenvolvimento do sistema.

Componente	**Preço**
Plataforma Omnima	£23.08 ou 27.78 €
Módulos XBee	44 €
Interface XBee USB	9 €
TSOP1730	4 €
LM75	6 €
LM35	2 €
Componentes diversos	~5 €
Total	**97,78 €**

A taxa de conversão utilizada foi de 1£ = 1,20358 €. O custo dos componentes XBee foi obtido em [113].

4.4 – Resumo

Este capítulo iniciou-se com a apresentação das ferramentas utilizadas ao longo da dissertação, referindo-se às características do hardware e apresentando a integração dos diversos componentes. O sistema desenvolvido é apresentado em primeiro lugar a partir de uma perspectiva de alto nível, seguindo-se uma descrição detalhada de cada serviço.

Capítulo 5

Testes

Este capítulo pretende estudar a influência do ambiente de execução na performance dos serviços desenvolvidos. Os testes executados, tanto de *frameworks* como de máquinas Java, têm como objectivo encontrar o conjunto que apresenta a melhor performance e perceber qual a sua influência nos serviços disponibilizados ao utilizador. Dadas as limitações em termos de processamento da plataforma de desenvolvimento, podem existir grandes variações de performance de acordo com o ambiente de execução utilizado nos serviços implementados.

5.1 – Máquinas virtuais Java

A plataforma OSGi, segundo o modelo de camadas apresentado em, necessita ser executada sobre uma *Java Virtual Machine*. Nesse sentido foram analisadas as implementações *Open-Source* de JVMs e efectuados testes de performance em 2 sistemas com arquitecturas diferentes. Apesar de algumas das implementações existentes não permitirem a sua execução em todas as diferentes arquitecturas, considera-se que no futuro, as implementações que se destacarem em termos de performance e suporte, serão portáveis.

As mais importantes são:

- Apache Harmony [114]
- Cacao [115]
- Dalvik [116]
- Hotspot[117]
- JamVM [118]
- Jikes RVM [119]
- Kaffe [120]
- Mika VM [121]
- PhoneME [122]

5.1.1 - Benchmarking

Da lista de JVMs acima mencionada, apenas foram seleccionadas algumas: Cacao, Hotspot, JamVM, Jikes RVM, Kaffe, Mika VM e PhoneME. A Dalvik é uma máquina virtual desenvolvida para o sistema operativo Android que executa o seu próprio *bytecode*, em vez de Java *bytecode*, sendo por esse motivo excluída dos testes. Em relação à Apache Harmony, Kaffe e Mika VM, a compilação não foi conseguida com sucesso. Kaffe e Mika VM são ambas compatíveis com a arquitectura MIPS, pelo que a sua análise seria efectuada também na secção seguinte, impossibilitada pelo motivo referido.

Os testes efectuados são apresentados a seguir:

5.1.1.1 - SciMark 2.0

SciMark [123] são um conjunto de testes efectuados à JVM de forma a analisar a performance na execução de determinados algoritmos. Os testes são os seguintes:

- Fast Fourier Transform (FFT)
- Jacobi Sucessive Over-Relaxation (SOR)
- Monte Carlo Integration
- Sparse Matrix Multiply
- Dense LU Matrix Factorization

Composit Score apresenta o resultado médio dos 5 testes efectuados. A versão *large* foi utilizada em todas as JVMs, o que implica um maior número de operações.

5.1.1.2 - Linpack

O teste Linpack [124] resolve um sistema linear Ax=b, com a dimensão 500x500. A matriz é gerada aleatoriamente e o segundo membro é gerado para que a solução seja composta apenas por uns. O método de resolução é baseado no método de Gauss com pivotação parcial.

5.1.1.3 - Dhrystone

O Dhrystone consiste maioritariamente em operações aritméticas sobre inteiros e envolve a invocação de outros métodos que retornam um resultado após terem executado determinadas funções. A versão utilizada [125] contém algumas alterações em relação ao benchmark originalmente proposto por Dr. Reinhold Weicker [126]. O tempo obtido como resultado traduz a média do tempo de execução de 100000 iterações do ciclo principal.

5.1.2 – Resultados x86

Os testes efectuaram-se num sistema com as características especificadas na tabela 5.1 as versões das máquinas Java utilizadas estão presentes na tabela 5.2:

Tabela 5.1 – Características do sistema de teste.

Características	
Processador	Pentium M 1.8GHz
Memória	1 Gb RAM
Sistema Operativo	Ubuntu 9.10 32bits
Kernel	2.6.31-22

Tabela 5.2 – Versões das máquinas virtuais Java analisadas.

Java Virtual Machine	Versão
Cacao	0.99.4
Hotspot	16.3 -b01
JamVM	1.5.4
Jikes RVM	3.1.0
PhoneME	Advanced MR2-b97

A biblioteca de classes utilizada no caso da JamVM, Cacao e JikesRVM foi a GNU Classpath na versão 0.98. A PhoneMe utiliza a sua própria biblioteca que tem como objectivo a implementação da especificação J2ME. Para a execução destes testes nesta JVM, foi necessário compilar as classes para a versão 1.4 do JDK. A Hotspot, a JVM da Sun utiliza as classes de J2SE 1.6. Na figura 5.1, são apresentados os resultados da execução dos testes SciMark.

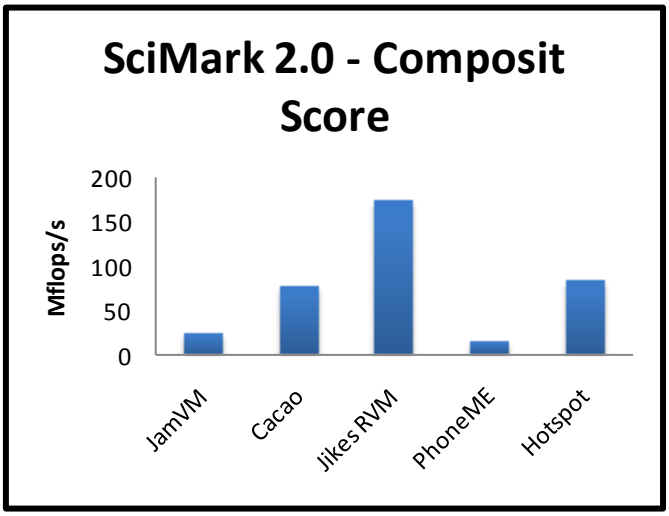

Figura 5.1 – Performance de várias máquinas virtuais Java segundo os testes SciMark.

Todos os testes efectuados estão disponíveis em anexo. No resultado geral, a PhoneME apresenta a pior performance seguida pela JamVM. Ao longo de todos os testes, Cacao e Hotspot apresentam performance semelhantes sendo que na média final Hotspot tem uma ligeira vantagem. A clara vencedora é a Jikes RVM, destacando-se em 4 dos testes efectuados. Apenas no teste de FFT, obtém uma performance semelhante à Cacao. O resultado do teste de Linpack é apresentado na figura 5.2.

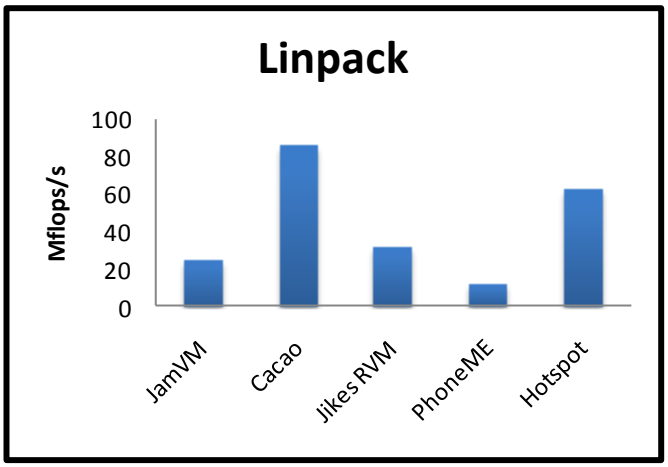

Figura 5.2 – Performance de várias máquinas virtuais Java segundo o teste Linpack.

Na avaliação de performance proporcionada pelo teste Linpack, Cacao é a mais rápida. Seguem-se Hotspot, Jikes RVM e JamVM. PhoneME apresenta novamente o pior resultado. O resultado do teste de DhryStone é apresentado na figura 5.3.

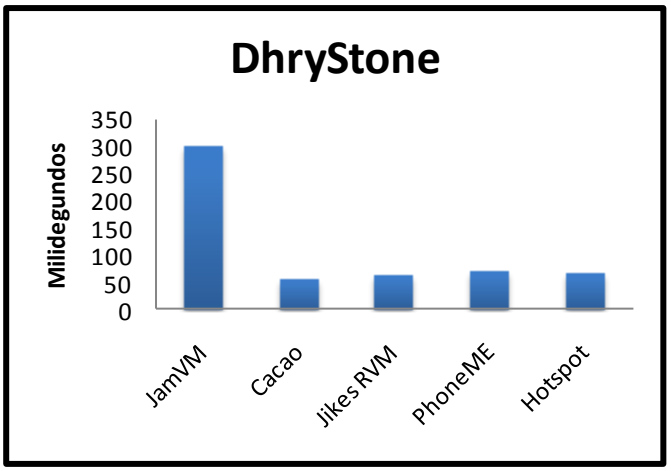

Figura 5.3 - Performance de várias máquinas virtuais Java segundo o teste DhryStone.

Neste teste, o menor tempo significa uma melhor performance. De notar, a disparidade da performance da JamVM em relação aos seus concorrentes. Cacao, Jikes RVM, PhoneME Hotspot obtêm resultados da mesma ordem, 54.3, 60.2, 65.2 e 68.5 respectivamente.

Em geral, a PhoneME apresenta os piores resultados, apenas com uma boa performance so teste DhryStone. Apesar de os testes terem sido efectuados, com a opção JIT activa, o aumento de performance face à versão por omissão não é notório. JikesRVM apresentou os resultados mais consistentes, com a melhor performance geral. Um aspecto a focar dos testes nesta arquitectura, é perceber o comportamento de JVMs alternativas à JVM oficial da Sun. Neste aspecto Jikes RVM e Cacao apresentam-se como boas alternativas segundo os testes efectuados. De entre o grupo de JVMs testadas, apenas 3 são compatíveis com a arquitectura MIPS: Cacao, JamVM e PhoneME. Neste grupo, destaca-se Cacao, que apresentou os melhores resultados neste ambiente de execução.

Estas análises apenas mostram a performance num sistema com determinadas características e podem indicar o comportamento previsto em sistemas semelhantes. No

entanto, no desenvolvimento de uma máquina virtual Java são definidas as plataformas alvo e realizadas optimizações de acordo com as suas características pelo que não deve ser assumido que o comportamento verificado acima se reflicta em sistemas diferentes.

5.1.3 – Resultados MIPS

Das 3 alternativas passíveis de teste, apenas 2 foram compiladas com sucesso para a arquitectura MIPS. Cacao apresenta um bug na compilação, reportado em [127], impossibilitando a comprovação dos resultados prometedores anteriormente obtidos. Sendo assim, apenas JamVM e PhoneME foram analisadas. Os resultados obtidos são apresentados nas figuras 5.4, 5.5 e 5.6.

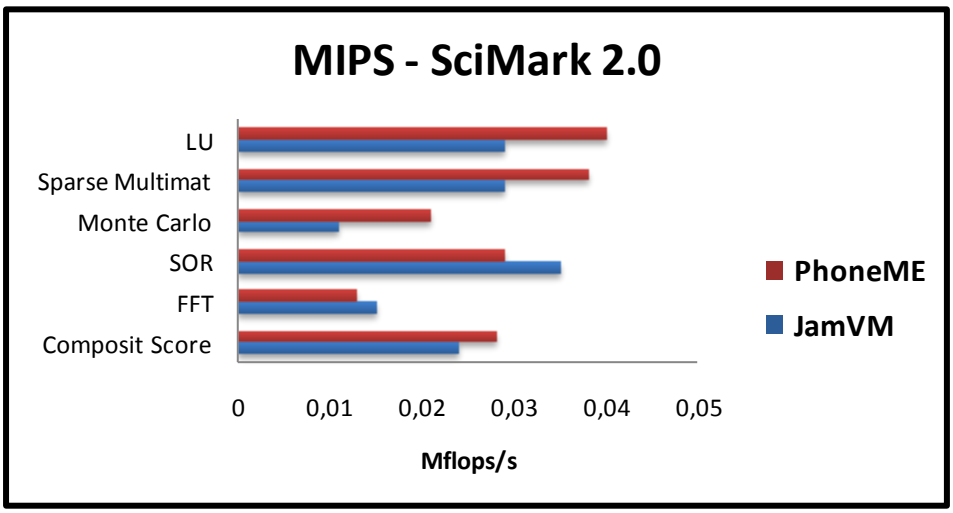

Figura 5.4 - Performance de JamVM e PhoneME segundo os testes SciMark na plataforma de desenvolvimento.

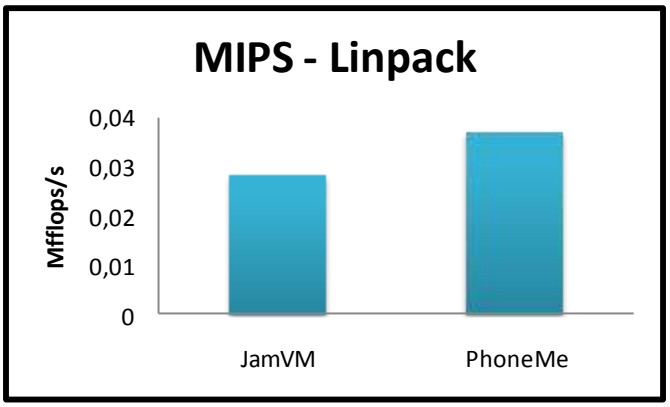

Figura 5.5 - Performance de JamVM e PhoneME segundo o teste Linpack na plataforma de desenvolvimento.

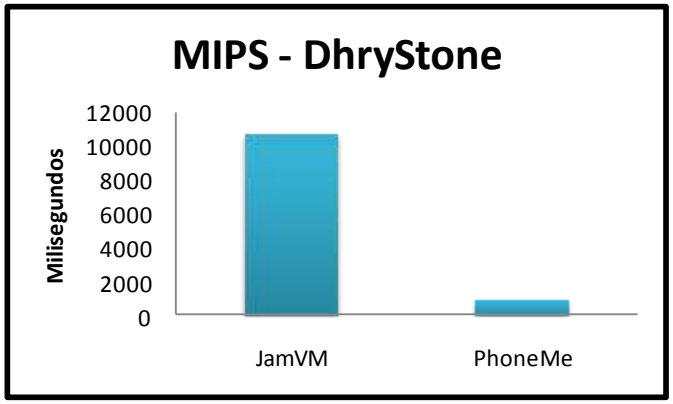

Figura 5.6 - Performance de JamVM e PhoneME segundo o teste DhryStone na plataforma de desenvolvimento.

PhoneME tem vantagem em três dos cinco testes efectuados pelo benchmark da SciMark e nos restantes dois testes. Na arquitectura MIPS, a PhoneME comporta-se de forma muito mais satisfatória em comparação com os resultados anteriormente obtidos. Por outro lado, confirmou-se a diferença de performance num sistema diferente.

5.2 – Implementações OSGi

Segundo a especificação OSGi, os *bundles* devem ser compatíveis com diferentes implementações da plataforma. Nesse sentido os serviços implementados foram executados em diferentes *frameworks*. As JVMs utilizadas foram a JamVM e PhoneME quando possível. A PhoneME utiliza a implementação CDC Java ME, sendo que os serviços implementados foram desenvolvidos tendo em conta outro ambiente de execução. Outra limitação reporta-se à especificação Java compatível com a PhoneME, a versão 1.4. Esse facto provoca excepções na execução dos *bundles*, que foram desenvolvidos com especificações superiores. Sendo assim, não foi possível testar a alternativa à JamVM. A *framework* Equinox na versão 3.5.2 não executa na máquina JamVM.

Tabela 5.3 – Versões das *frameworks* analisadas.

Framework	Versão
Knopflerfish	2.3.2
Apache Felix	2.0.5
Concierge	1.0.0
Equinox	3.5.2

Nesta secção é apresentada uma comparação de *frameworks* sobre diversos aspectos. A implementação Concierge apesar de não ser compatível com a especificação R4, foi integrada nos testes devido aos resultados prometedores anunciados. Os pontos de análise dividem-se em:

- Tempo de carregamento inicial
- *Bundles* mínimos necessários
- Memória

- Navegação Portal Web

Um aspecto importante a ter em conta, é o ambiente de execução mínimo de uma *framework*. Neste aspecto, as diferentes implementações solicitam *bundles* específicos para permitir a sua execução. Sendo assim, definiu-se comparar as *frameworks*, tendo em conta os *bundles* necessários para executar os serviços desenvolvidos nesta dissertação.

5.2.1 - Tempo de carregamento inicial

Os tempos de execução foram obtidos através da função Linux *time*, em que é extraído o valor de tempo real da execução de determinado comando. Neste caso, o comando *time* cronometrou o tempo de carregamento de uma *framework*, isto é, o tempo que a plataforma demora a instalar e executar os *bundles* mínimos respectivos. Os valores foram obtidos a partir da média de 3 tempos de carregamento e são apresentados nas figuras 5.7 e 5.8.

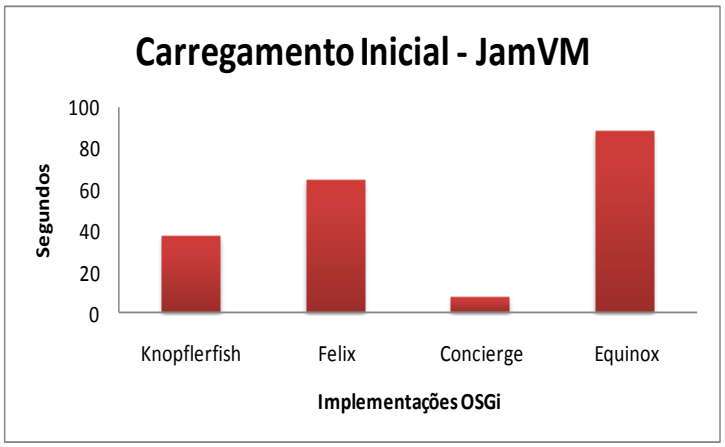

Figura 5.7 – Análise do tempo de carregamento inicial das *frameworks* em JamVM.

A *framework* Concierge é a mais rápida que as restantes implementações testadas. Seguem-se Knopflerfish e Felix. De referir, que na medição dos tempos de carregamento inicial de cada *framework*, os *bundles* mínimos necessários de cada uma, não são iguais em número. Na secção 5.2.2 é apresentado o ambiente de execução mínimo de cada *framework*.

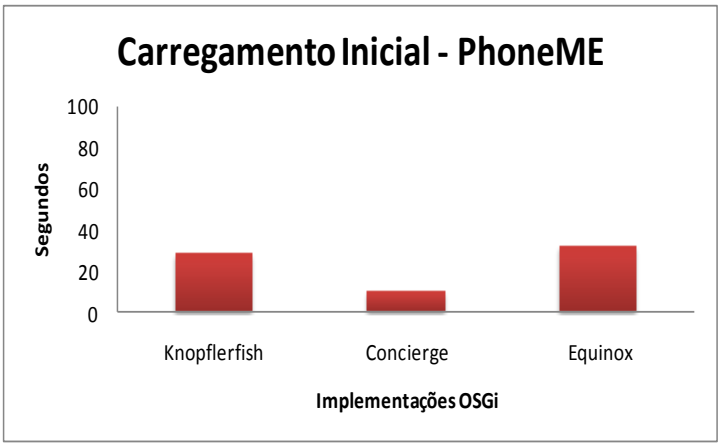

Figura 5.8 - Análise do tempo de carregamento inicial das *frameworks* em PhoneME.

Com a JVM alternativa, não foi possível analisar a implementação Felix devido a uma excepção de classes. No entanto nas restantes opções, os resultados revelaram-se consistentes com os anteriores, mantendo-se Concierge como a implementação mais rápida.

5.2.2 - Bundles mínimos

Para se retirarem conclusões acerca das diferentes implementações, é necessário analisar o ambiente mínimo de execução utilizado nas análises da secção anterior. A presente secção apresenta os *bundles* mínimos para a execução correcta dos serviços implementados, em cada alternativa.

5.2.2.1 - Knopflerfish

A implementação Knopflerfish requer um número elevado de *bundles* mínimos para que a *framework* seja carregada com sucesso. O ambiente de execução mínimo testado é composto pelos seguintes módulos:

- Bundle sistema
- API - Serviço de Log
- API - Consola
- API – Cm
- IMPL – Serviço de Log
- IMPL – Consola
- IMPL – Consola TTY
- IMPL – Comandos Framework
- IMPL – Comandos Log
- API – UserAdmin
- LogReader
- Apache Felix http Jetty

5.2.2.2 - Apache Felix

Os *bundles* necessários na implementação Felix são:

- Shell
- Shell TUI
- Compendium
- BundleRepository
- Serviço Log
- LogReader
- Apache Felix HTTP Jetty

5.2.2.3 - Concierge

A implementação Concierge é compatível com apenas a especificação R3 da plataforma OSGi. Devido a esse facto, o *manifest* dos *bundles* têm de ser alterados para uma correcta execução.

- Shell
- Serviço Log
- API – Cm

- Apache Felix HTTP Jetty
- Logger
- Service Tracker

5.2.2.4 - Equinox

A plataforma Equinox foi testada em 3 versões: 3.5.2, 3.5.1 e 3.5. Todas as versões analisadas são extremamente lentas na sua execução, sendo assim impossível testar os serviços desenvolvidos. Na instalação dos serviços, a execução da Equinox é interrompida abruptamente. Apesar do aumento de memória disponível, a situação manteve-se.

5.2.3 - Memória

Na análise da memória necessária na utilização da plataforma utilizou-se o parâmetro Virtual Memory Size para análise. Este parâmetro está disponível no sistema Linux no acesso aos processos a serem executados no momento. VSZ expressa a quantidade de memória a ser utilizada por determinado processo, que inclui a memória RAM e a memória SWAP. Para ser possível executar os serviços implementados sem que ocorressem excepções de falta de memória, gerou-se um ficheiro de SWAP com 16MB. Os valores são aproximados e foram obtidos no início da execução dos serviços. Obviamente, este valor varia de acordo com o processamento inerente à utilização dos mesmos, sendo por isso analisado o consumo de memória numa situação estacionária. O resultado da análise é apresentado na figura 5.9.

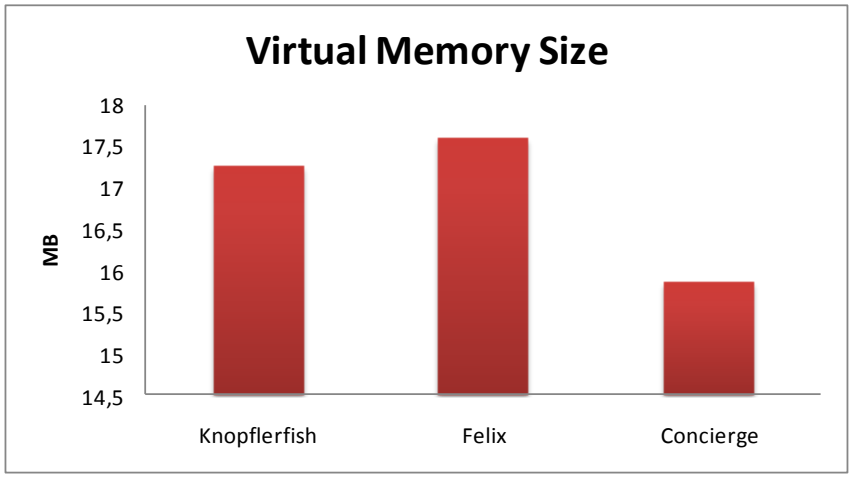

Figura 5.9 – Análise da utilização de memória de diferentes implementações OSGi.

O valor foi obtido após terem sido iniciados todos os serviços implementados. De acordo com os dados obtidos, Concierge é executada com a menor quantidade de memória. Knopflerfish e Felix obtêm resultados semelhantes. Em relação à implementação Equinox não foi possível efectuar a sua análise, uma vez que não foi possível executar os serviços, como referido anteriormente.

5.2.4 - Navegação WebService

Em relação ao serviço Web, analisou-se a rapidez de navegação no portal. O acesso ao portal foi efectuado através de acesso local. Knopflerfish e Concierge obtêm velocidades aceitáveis sendo que a *framework* Felix apresenta o pior comportamento, com um tempo de

carregamento bem mais lento. Apesar de Concierge ser mais rápida a responder a pedidos de acesso, o seu comportamento não é tão aceitável quando é necessário aceder às bibliotecas nativas de alguns serviços e posteriormente apresentar os resultados do acesso. Neste teste foram utilizados os browsers Chrome e Firefox, verificando-se igual comportamento. O tempo de resposta considerado razoável é de 3s.

5.2.5 - Conclusão

Os testes e análises efectuados permitiram aferir que a implementação da *framework* influencia efectivamente na performance oferecida ao utilizador. No que se refere ao tempo de carregamento inicial, apesar de Concierge apresentar o melhor resultado, esta necessita de metade do número de *bundles* necessários para carregar Knopflerfish. Atendendo a esse facto, não se pode afirmar que uma seja mais rápida que outra. Por outro lado, o resultado pretendido por essa análise é saber qual a *framework* que carrega mais rapidamente um ambiente adequado para os serviços desenvolvidos, independentemente dos *bundles* que cada implementação requer. Desta perspectiva, Concierge é claramente superior.

Em termos de rapidez de acesso ao portal Web, verificou-se uma lentidão incomportável na *framework* Felix, apesar de o servidor HTTP utilizado ser exactamente o mesmo. Os melhores resultados foram obtidos com as plataformas Knopflerfish e Concierge, sendo que esta última apresenta um comportamento lento ou instável no acesso a código nativo necessário em alguns dos serviços desenvolvidos.

No que se refere ao factor JVM, não foi possível retirar conclusões acerca da sua influência, uma vez que não foi possível executar os serviços na PhoneME. Apenas se conclui que a PhoneME, carrega mais rapidamente as implementações OSGi testadas, o que pode ser um indicador de melhor performance relativamente a JamVM. No entanto, não se pode inferir qualquer conclusão acerca da performance dos serviços nesta máquina Java.

De acordo com os testes efectuados, conclui-se que o ambiente de execução inicial constituído pela JVM, JamVM e a implementação OSGi, Knopflerfish, constitui a melhor opção em termos de eficácia, performance e estabilidade.

5.3 - Resumo

Este capítulo apresentou os resultados de testes de performance efectuados em diferentes máquinas virtuais Java em dois sistemas diferentes. As diferentes implementações OSGi foram analisadas e retiradas conclusões acerca da sua influência na performance dos serviços desenvolvidos ao longo da presente dissertação.

Capítulo 6

Conclusão

O sistema desenvolvido ao longo da dissertação, pretende disponibilizar um serviço de monitorização e controlo de temperatura, facilmente implementável e de baixo custo.

O sistema implementado é capaz de:

- Aceder a sensores de temperatura em tempo real.
- Enviar ordens para o sistema de ar condicionado via infra-vermelhos.
- Descodificar sequências de bits de um controlo remoto que opere na frequência de 30 KHz.
- Enviar notificações de alarme via correio electrónico.

Todas as funcionalidades acessíveis ao utilizador, são implementadas em componentes modulares definidos pela plataforma OSGi, denominados de *bundles*. A plataforma OSGi apresenta claras vantagens no desenvolvimento de aplicações, e neste caso em particular, permite a interacção entre subsistemas, como por exemplo, o sistema de ar condicionado e sensores, através da colaboração entre os diferentes módulos. A especificação OSGi define uma arquitectura orientada a serviços. Sendo assim, as funcionalidades de determinado módulo são acessíveis através do acesso ao serviço correspondente. Um serviço oferece uma interface bem definida e abstrai os outros componentes modulares de detalhes de implementação. Este facto tem como consequência a redução de complexidade no desenvolvimento sobre OSGi. Outro aspecto a ter em conta é o ambiente de execução dinâmico. A especificação OSGi fornece um registo de serviços dinâmico, onde os módulos podem registar e requisitar serviços. Um módulo pode adaptar as suas funcionalidades tendo em conta os serviços disponíveis na plataforma. Um exemplo disso mesmo é o serviço de portal Web, que importa diversos serviços. O serviço apesar das dependências não deixa de executar caso estas não sejam satisfeitas. Apenas não acede às funcionalidades dos serviços externos.

A plataforma em conjunto com um protocolo de gestão, como o TR-069, permite operações de manutenção efectuadas remotamente, dadas as interfaces disponibilizadas

nesse sentido. De referir, que as operações de manutenção podem ser efectuadas sem interferir na execução do sistema, isto é, sem necessidade de reiniciar a plataforma. A plataforma permite *updates* dinâmicos de serviços, isto é, um determinado módulo pode estar a ser alvo de um processo de *update* sem que os módulos que dele dependem sejam afectados. Este facto é também uma clara vantagem no caso de certos serviços conterem *bugs*. Todas estas vantagens foram perceptíveis ao longo do projecto.

Prevê-se que num futuro próximo, os *gateways* disponibilizados pelos operadores sejam capazes de executar serviços sobre uma implementação OSGi, permitindo ao utilizador dispor de inúmeros serviços sem ter de executar qualquer operação de instalação ou manutenção. No futuro, é previsível que cada fabricante disponibilize um *bundle* de acesso ao seu *hardware* específico e que pode ser importado por um serviço geral de manutenção a fornecer ao cliente.

Ao analisar o sistema desenvolvido na perspectiva do utilizador, os serviços disponibilizados podem ter as mais diversas aplicações mediante o contexto em que estão inseridos. Apesar de o serviço de sensores ter sido utilizado para a obtenção da temperatura ambiente, este pode ter utilizações mais específicas como a temperatura de um aquário ou de uma estufa. O sistema desenvolvido poderá ser também aplicado numa sala de servidores, sistemas que necessitam de um ambiente com determinadas características a nível de temperatura e humidade. Nas aplicações domóticas referenciadas não se verifica a integração de diferentes protocolos de transmissão de dados, como no caso do sistema desenvolvido. O facto de os módulos definirem APIs para acesso a dispositivos, permite a integração na plataforma de protocolos até então independentes e incompatíveis.

Todas as funcionalidades apresentadas foram devidamente testadas, executando como esperado. No entanto, há que referir um comportamento por vezes instável no acesso ao sensor Zigbee que culmina com a suspensão da execução da plataforma OSGi.

O sistema apresentado é encarado como um ponto de partida para um conjunto de outros serviços. O serviço de descodificação infra-vermelhos poderá ser melhorado e ampliada a sua actuação, mediante a compatibilidade com outros protocolos IR. O serviço de controlo do sistema de ar condicionado pode ser encarado como um exemplo e aplicado no controlo de outros dispositivos electrónicos como sistemas de som e televisores. Um serviço de controlo por SMS poderá ser também desenvolvido, aproveitando a disseminação do acesso à banda larga móvel e consequentemente dos modems USB. Um dos desenvolvimentos futuros importantes passará pela integração de serviços OSGi com os protocolos de automação residencial mais acessíveis como X10 ou Insteon. O melhor aproveitamento das características dos módulos de comunicação XBee é também um caminho a seguir, como por exemplo a inclusão de novos sensores no mesmo módulo.

Para além de todo o desenvolvimento do sistema e do estado da arte relacionado, apresenta-se uma avaliação de diferentes ambientes de execução para os serviços implementados, no que se refere à máquina virtual Java e à implementação da plataforma OSGi. A implementação da especificação OSGi influencia na performance geral do sistema desenvolvido, nomeadamente no que se refere ao serviço Web. Este capítulo, é considerado uma introdução para futuras análises da influência dos factores referidos.

Os desenvolvimentos efectuados permitiram a aquisição de conhecimento no âmbito da plataforma OSGi e protocolos de automação residencial, contribuindo esta dissertação para uma apresentação geral da especificação OSGi e das tecnologias mais relevantes do mercado

domótico. De referir também, a experiência adquirida em ambientes Linux e em processos de compilação para diferentes arquitecturas.

Anexo A

Tabela A.1 – Plataformas de desenvolvimento.

Fabricante	Cpu	Ethernet	RS 232	Usb	OS	Memória	Flash	GPIO	UART	JTAG	Custo
Cirrus Logic	200 MHz ARM920T	Sim		Sim	Linux/Windows	Expansível		Sim	Sim		£233.75
Avr	AT32AP7000	Sim	Sim	Sim	Linux	32 MB SDRAM	8MB			Sim	US$69
Mbest	266+ MHzSamsung S3C2410A	Sim	Sim	Sim	Linux/WinCE	64MB SDRAM	64MB	Sim		Sim	US$159
	AT91SAM9261S	Sim	Sim	Sim	Linux/WinCE	64MB SDRAM	128MB	Sim		Sim	US$129
JK Microsystems	200MHz ARM Processor	Sim	Sim	Sim	Linux	32MB RAM	16 MB	Sim			US$169
Mycable	500MHz AU1500	Sim	Sim	Sim	Linux/WinCE	64/128MB SDRAM	32MB	Sim		Sim	
Esd	133 or 266MHz IBM PPC405GP	Sim	Sim	Sim	Linux	32 até 128 MB	32MB				
Omnima	Infineon ADM5120	Sim		Sim	Linux/NetBSD	16 MB	2MB	Sim	Sim	Sim	£23.08
Biiferboard	Intel 486SX 150MHz	Sim		Sim	Linux	32 MB SDRAM	8MB	Sim		Sim	£35
NSLU2	Intel IXP420	Sim		Sim	Linux	32 MB SDRAM	8MB	Sim			84.95€

Anexo B

Comandos LG MS09AH

Cabeçalho=0x88

Tabela B.1 – Sequências de bits utilizada pelo sistema LG MS09AH.

Comando	Dados (20 bits)
ON	00549
OFF	C0051
Down 18	0830B
Up 19	0840C
Up 20	0850D
Up 21	0860E
Up 22	0870F
Up 23	08800
Up 24	08901
Up 25	08A02
Up 26	08B03
Up 27	08C04
Up 28	08D05
Up 29	08E06
Up 30	08F07

Anexo C

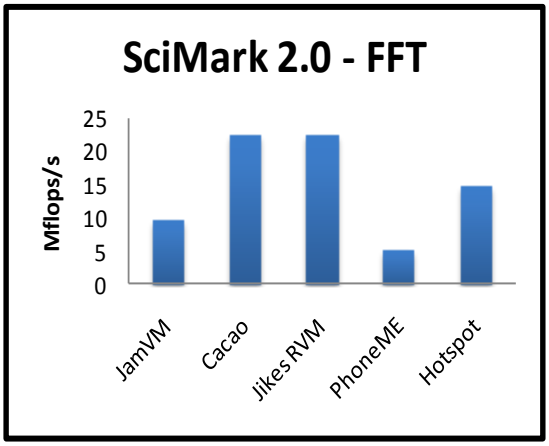

Figura C.1 – Resultado FFT.

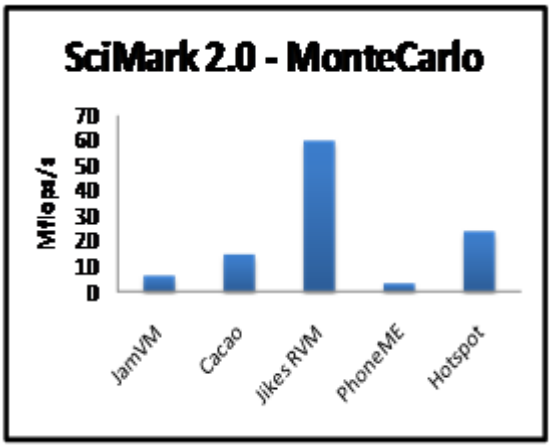

Figura C.2 - Resultado MonteCarlo.

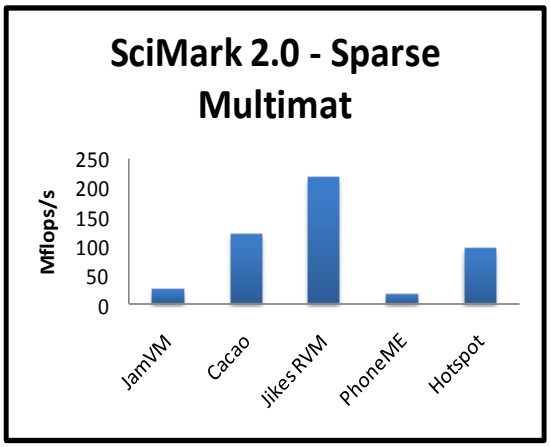

Figura C.3 – Resultado Sparse Multimat.

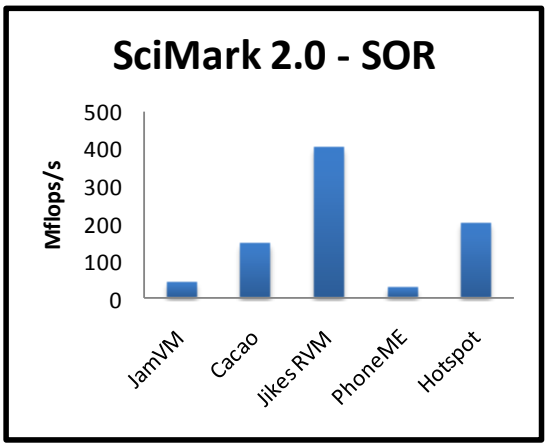

Figura C.4 – Resultado SOR.

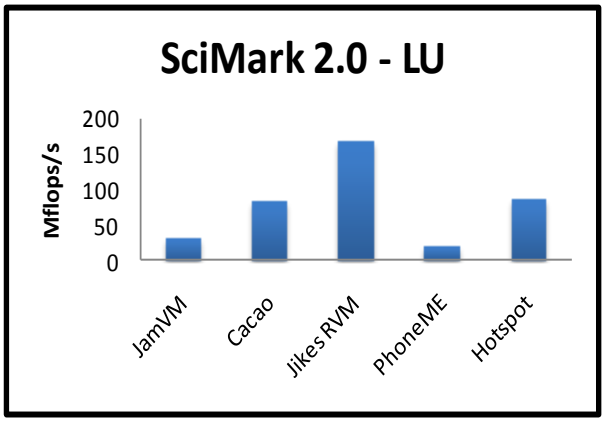

Figura C.5 – Resultado LU.

Referências

[1] The OSGi Alliance, “OSGi Service Platform service Compendium”, Release 4, Version 4.2, Agosto 2009.

[2] Paolo Bertoldi, Bogdan Atanasiu, “Electricity Consumption and Efficiency Trends In European Union”, European Comission, pp. 7-14, 2009.

[3] Anacom, “Serviço de acesso à internet”, pp. 148-150, Junho 2009.

[4] Business Wire, “Gain Insight into the Global Commercial Air Conditioner Market”, Junho 2008.

[5] Sun Microsystems, “Java 2 Standard Edition JAR File Specification”. Disponível em http://java.sun.com/j2se/1.5.0/docs/guide/jar/index.html. Acesso em 23/Junho/2010.

[6] James Gosling, Bill Joy, Guy Steele Jr., “Java Language Specification”, 3rd ed., capítulo 7, Sun Microsystems.

[7] Tim Lindholm, Frank Yellin, “The Java Virtual Machine Specification”, 2nd ed., capítulo 5, Sun Microsystems.

[8] M. Warres, “Class Loading Issues in Java RMI and Jini Network Technology”. Sun Microsystems, Mountain View: CA Sun Labs, pp. 2-7, 2006.

[9] The OSGi Alliance, “OSGi Service Platform Release 4, Core Specification”, Julho 2006.

[10] R. Hall. “OSGi R4 Service Platform: Java Modularity and Beyond”. Berlin, Março 2007.

[11] The OSGi Alliance. “OSGi Service Platform Release 3, Core Specification”, Março 2003.

[12] Java Servlet Technology. Disponível em http://java.sun.com/products/servlet/. Acesso em 23/Junho/2010.

[13] HTTP 1.0 Specification RFC-1945. Disponível em http://www.ietf.org/rfc/rfc1945.txt. Acesso em 23/Junho/2010.

[14] HTTP 1.1 Specification RFC-2068. Disponível em http://tools.ietf.org/html/rfc2068.html. Acesso em 23/Junho/2010.

[15] HTTP Authentication: Basic and Digest Authentication. Disponível em http://tools.ietf.org/html/rfc2617. Acesso em 23/Junho/2010.

[16] Hypertext Transfer Protocol – HTTP/1.1. Disponível em http://tools.ietf.org/html/rfc2616. Acesso em 23/Junho/2010.

[17] The MD5 Message-Digest Algorithm. Disponível em http://tools.ietf.org/html/rfc1321. Acesso em 23/Junho/2010.

[18] M.M.J. Stevens. “On Collisions for MD5”, capítulo 7. Eindhoven, Junho 2007.

[19]J. Black, M. Cochran, T. Highland. “A Study of the MD5 Attacks: Insights and Improvements”, pp. 9-16. Colorado USA, Março 2006.
[20]X. Wang, H. Yu. “How to Break MD5 and Other Hash Functions”, pp. 1-12. China 2006.
[21]Customer Premises Equipment. Disponível em http://www.cybertelecom.org/ci/cpe.htm. Acesso em 23/Junho/2010.
[22]BroadBand Forum,“TR-069 CPE WAN Management Protocol”, v1.1, Dezembro 2007.
[23]The SSL Protocol, Version 3.0. Disponível em http://wp.netscape.com/eng/ssl3. Acesso em 23/Junho/2010.
[24]RFC 2246, The TLS Protocol, Version 1.0. Disponível em http://www.ietf.org/rfc/rfc2246.txt. Acesso em 23/Junho/2010.
[25]Apache Felix. Disponível em http://felix.apache.org/site/index.html. Acesso em 23/Junho/2010.
[26]Apache Felix Subproject Documentation. Disponível em http://felix.apache.org/site/subprojects.html. Acesso em 23/Junho/2010.
[27]Eclipse Equinox. Disponível em http://download.eclipse.org/equinox/. Acesso em 23/Junho/2010.
[28]Equinox Bundles. Disponível em http://eclipse.org/equinox/bundles/. Acesso em 23/Junho/2010.
[29]Knopflerfish OSGi. Disponível em http://www.knopflerfish.org/index.html. Acesso em 23/Junho/2010.
[30]Makewave. Disponível em http://www.makewave.com/. Acesso em 23/Junho/2010.
[31]Concierge. Disponível em http://concierge.sourceforge.net/. Acesso em 23/Junho/2010.
[32]Jan S. Rellermeyer, Gustavo Alonso, “Concierge: A Service Platform for Resource – Constrained Devices”, in *The Proceedings of the Eurosys 2007 Conference and ACM SIGOPS Operating Systems Review*, Junho 2007, pp. 1-14.
[33]Rellermeyer, J., G. Alonso, T. Roscoe, “R-OSGi: Distributed applications through software modularization”, Zurich 2007.
[34]Thomsen, J. “OSGi-based gateway replication” in *Proceedings of the IADIS Applied Computing Conference 2006*, 2006, pp, 123-126.
[35]Y. Zhiwen, Z. Xingshe, Y. Zhiyong, Z. Daqing, C. Chung-Yau. “An OSGi-Based Infrastructure for Context-Aware Multimedia Services” in *IEEE Communications Magazine*, Vol. 44, Outubro 2006, pp. 136-142.
[36]B. Alan, K. Mario, B.Dennis, L. Gennady, M. Mathew, “A SIP-based OSGi Device Communication Service for Mobile Personal Area Networks” in *Consumer Communications and Networking Conference*, Vol.1, Jan. 2006, pp. 502-506.
[37]RFC 3261, SIP: Session Initiation Protocol. Disponível em http://tools.ietf.org/html/rfc3261. Acesso em 23/Junho/2010.
[38]Eclipse-OSGi. Disponível em http://www.aqute.biz/OSGi/Eclipse. Acesso em 23/Junho/2010.
[39]OSGi Markets and Solutions. Disponível em http://www.OSGi.org/Markets/HomePage. Acesso em 23/Junho/2010.
[40]Philips iPronto digital home controller. Disponível em http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Device-profile-Philips-iPronto-digital-home-controller/. Acesso em 23/Junho/2010.
[41]Ferreira, Paulo, “Componentes OSGi para aplicações de segurança domótica e vigilância”, 2006.

[42]Smart Home Solutions. Disponível em http://www.prosyst.com/index.php/de/html/content/37/Smart-Home-Solutions/. Acesso em 23 de Junho de 2007.

[43] Ying-Wen, H. Jui-Po. "Design and Implementation of an Embedded Home Gateway for Remote Monitoring Based on OSGi Techonology" in *IASTED European Conference: internet and multimedia systems and applications*, Anaheim, CA, USA, 2007, pp. 63-68.

[44]C. Ing-Yi, H. Chao-Chi. "A Remotely Manageable Electrocardiogram Measurement System for Home Healthcare using OSGi Framework" in *Journal of Medical and Biological Engineering*, Set. 2004, pp. 133-139.

[45]Bobbie Patrick O., Ramisetty Sailaja H., Yussiff Abdul-Lateef, Pujari Sagar. "Designing an Embedded Electronic-Prescription Application for Home-Based Telemedicine using OSGi Framework" in *Proceedings of the International Conference on Embedded Systems and Applications*, Junho 2003, pp. 16-21.

[46]L.Xie, Z.Wenjun. "The Design and Implementation of Home Network System Using OSGi Compliant Middleware" in IEEE Transactions on Consumer Electronics, Vol.50, No.2, Maio 2004, pp. 528-534.

[47]LonWorks Control Networking. Disponível em http://www.echelon.com/products/lonworks_control_networking.htm. Acesso em 23/Junho/2010.

[48]LonMark International. Disponível em http://www.lonmark.org/connection/what_is_lon. Acesso em 23/Junho/2010.

[49]JSR 272: Mobile Broadcast Service API for Handheld Terminals. Disponível em http://jcp.org/en/jsr/detail?id=272. Acesso em 23/Junho/2010.

[50]S.G. Kiev. "An OSGi Middleware for Mobile Digital TV Applications" in *International Conference On Mobile Technology, Applications, And Systems: Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, pp. 690-695, NY, USA, 2007.

[51]S. José, U. Benito, S. Antonio. "A Multiplatform OSGi Based Architecture for Developing Road Vehicle Services" in *Consumer Communications & Networking Conference 2007*, Las Vegas, 2007.

[52]P.Parrend e S. Frenot. "Security Benchmarks of OSGi platforms: toward hardened OSGi" in *Software, Practice and Experience*, Vol. 39, Abril 2009, pp 471-499.

[53]A. Heejune, Oh Hyukjun, O.S. Chang. "Towards Reliable OSGi Framework and Applications" in *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1456-1461, Dijon, France, 2006.

[54]Torrão, C. "Fault Tolerance in the OSGi Service Platform" in *OTM Conferences*, 2009, pp. 653-670.

[55]IEEE 802.3. Disponível em http://standards.ieee.org/getieee802/802.3.html. Acesso em 23/Junho/2010.

[56]HomePNA. Disponível em http://www.homepna.org/. Acesso em 23/Junho/2010.

[57]Yousuf, M.S., El-Shafei, M., "Power Line Communications: An Overview - Part I" in *4th International Conference on Innovations in Information Technology*, Novembro 2007.

[58]David Liu, Dao Xian, "Home environmental control system for the disabled" in *Proceedings of the 1st international convention on Rehabilitation engineering & assistive technology*, Abril 2007, pp. 164-168.

[59]SmartLabs, “Insteon Compared”, Jan. 2006.
[60]Powerline Control Systems, “UPB System Description Document”, Versão 1.4, Abril 2007.
[61]Powerline Control Systems. Disponível em http://www.pcslighting.com/. Acesso em 23/Junho/2010.
[62]How UPB Compares to X-10. Disponível em http://www.pulseworx.com/upb/x10_compare_upb_.htm. Acesso em 23/Junho/2010.
[63]LonWorks Technology Overview. Disponível em http://www.echelon.com/communities/energycontrol/developers//lonworks/default.htm. Acesso em 23/Junho/2010.
[64]HomePlug Powerline Alliance. Disponível em http://www.homeplug.org/home/. Acesso em 23/Junho/2010.
[65]Steve Gardner, Brian Markwalter, Larry Yonge, “HomePlug Standard Brings Networking to the Home”. Disponível em http://www.commsdesign.com/main/2000/12/0012feat5.htm. Acesso em 23/Junho/2010.
[66]HomePlug, “How HomePlug Technologies Enhance the Consumer Experience”, 2007.
[67]UPNP Forum, “UPNP Device Architecture 1.1”, Outubro 2008.
[68]Jordi Palet, Francisco Ortiz, “6Power, IPv6, and PLC for Home Automation”, Terena, 2004.
[69]Allen, Bob e Brian Dillon, "Environmental Controland Field Bus Systems ", Dublin, 1997.
[70]Protocolo EIB. Disponível em http://www.knxportugal.com/imagens/EIBA.pdf. Acesso em 20/Junho/2010.
[71]IEEE 802.11-2007. Disponível em http://standards.ieee.org/getieee802/802.11.html. Acesso 23/Junho/2010.
[72]Chatschik, B., “An overview of the Bluetooth wireless technology”, in *IEEE Communications Magazine*, Dezembro 2001.
[73]InfraRed Remote Control. Disponível em http://www.ustr.net/infrared/infrared1.shtml. Acesso em 20/Junho/2010.
[74]Vishay, “TSOP17.. Datasheet - Photo Modules for PCM Remote Control Systems”.
[75]Zensys, “Z-Wave Protocol Overview”, Versão 4, Maio 2009.
[76]G. Ferrari, P. Medagliani, S. Di Piazza, M. Martalò, “Wireless Sensor Networks: Performance Analysis in Indoor Scenarios” in *EURASIP Journal on Wireless Communications and Networking*, Vol. 2007, pp. 14, 2007.
[77]Zigbee Alliance, “Zigbee Specification”, Dezembro 2006.
[78]IEEE 805.15.4 2006. Disponível em http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf. Acesso em 23/Junho/2010.
[79]Matt Maupin, “Zigbee: Wireless Control Made Simple” in Wireless and Mobile WorldExpo, Toronto, Canada, 2005.
[80]S. Dagtas, G. Pekhteryev, Z. Sahinoglu, “Multi-stage Real Time Health Monitoring via Zigbee in Smart Homes” *in Proceedings of the 21st International Conference on Advanced Information Networking and ApplicationsWorkshops*, Ontario, Canada, Maio 2007, pp. 782-786.
[81]Ying-Wen Bai, Chi-Huang Hung, “Remote Power On/Off Control and Current Measurement For Home Electric Outlets Based On a Low-Power Embedded Board and Zigbee Communication” in *Proceedings of the 2008 International Symposium on Consumer Electronics*, Algarve, Portugal, Abril 2008, pp. 14-16.

[82]A. R. Delgado, A. Robinet, John McGinn, Vic Grout, R. Picking, "Assistive Human-Machine Interfaces for Smart Homes". Disponível em http://www.newi.ac.uk/groutv/papers/nrg2007/royrobinetmcginngroutpicking.pdf. Acesso em 23/Junho/2010.

[83]E. Mainardi, S. Banzi, M. Bonfe, S. Beghelli, "A low-cost Home Automation System based on Power-Line Communication Links" in *22nd International Symposium on Automation and Robotics in Construction*, Setembro 2005.

[84]Mafalda Seixas, João Palma, "Remote Alarm and Command System for Residential Domotics trough GSM-SMS" in *Proceedings of the Ninth Spanish-Portuguese Congress of Electrical Engineering*, Marbella, Spain, pp.1-4.

[85]Stefan Soucek, Gerhard Russ, Clara Tamarit, "The Smart Kitchen Project – An Application of Fieldbus Technology to Domotics" in *Third International Workshop on Networked Appliances*, 2001.

[86]SNMP RFC1157. Disponível em http://tools.ietf.org/html/rfc1157. Acesso em 23/Junho/2010.

[87]"I2C-bus specification and user manual", Rev. 03, Junho 2007.

[88]David Kalinsky, Roee Kalinsky, "Introduction to Serial Peripheral Interface". Disponível em http://www.embedded.com/story/OEG20020124S0116. Acesso em 20/Junho/2010.

[89]Omnima Forum Embedded Controller. Disponível em http://www.omnima.co.uk/forums/index.php?s=41d0219289d5eeffcd15847134b5a006&showforum=6. Acesso em 23/Junho/2010.

[90]Omnima Embedded Controller. Disponível em http://www.omnima.co.uk/store/catalog/Embedded-controller-p-16140.html. Acesso em 23/Junho/2010.

[91]Squidge Packages. Disponível em http://www.omnima.co.uk/forums/index.php?act=attach&type=post&id=25. Acesso em 24/Junho/2010.

[92]OpenWRT. Disponível em http://openwrt.org/. Acesso em 20/Junho/2010.

[93]Packages ADM5120. Disponível em http://downloads.openwrt.org/kamikaze/8.09.2/adm5120/packages/. Acesso em 20/Junho/2010.

[94]OpenWRT Supported Devices. Disponível em http://oldwiki.openwrt.org/TableOfHardware.html. Acesso em 23/Junho/2010.

[95]XBee Explorer USB. Disponível em http://www.sparkfun.com/commerce/product_info.php?products_id=8687. Acesso em 20/Junho/2010.

[96]National Semiconductor, LM75 Datasheet.

[97]XBee Series 2.5 Wire Antenna. Disponível em http://www.sparkfun.com/commerce/product_info.php?products_id=8695. Acesso em 20/Junho/2010.

[98]National Semiconductor, LM35 Datasheet.

[99]Knopflerfish Eclipse Plugin. Disponível em http://www.knopflerfish.org/eclipse_plugin.html. Acesso em 22/Junho/2010.

[100] GPIO Info. Disponível em http://www.omnima.co.uk/forums/index.php?showtopic=110. Acesso em 20/Junho/2010.

[101] USB-rootfs on boot on Edimax BR-6104KP (ADM5120). Disponível em https://forum.openwrt.org/viewtopic.php?id=18570. Acesso em 20/Junho/2010.
[102] Digi, "X-CTU Configuration and Test-Utility Software", 2008.
[103] Digi. Disponível em http://www.digi.com. Acesso em 23/Junho/2010.
[104] A Java API for Digi XBee/XBee-Pro OEM RF Modules. Disponível em http://code.google.com/p/xbee-api/. Acesso em 20/Junho/2010.
[105] Digi, "Upgrading from ZNet 2.5 to ZB".
[106] Discrete Semiconductors, 2N2222 Datasheet.
[107] Nec Protocol. Disponível em http://www.sbprojects.com/knowledge/ir/nec.htm. Acesso em 23/Junho/2010.
[108] Calvin Austin, Monica Pawlan, "Advanced Programming for the Java 2 Platform", capítulo 5.
[109] Sheng Liang, "The Java Native Interface: Programmer´s Guide and Specification".
[110] Bruce Eckel, "Thinking in Java", 4th Edition.
[111] Gnu JavaMail. Disponível em http://www.gnu.org/software/classpathx/javamail/javamail.html. Acesso em 24/Junho/2010.
[112] RXTX API. Disponível em http://www.rxtx.org/. Acesso em 23/Junho/2010.
[113] CoolComponents. Disponível em http://www.coolcomponents.co.uk/catalog/. Acesso em 20/Junho/2010.
[114] Apache Harmony. Disponível em http://harmony.apache.org/. Acesso em 23/Junho/2010.
[115] CacaoVM. Disponível em http://www.cacaovm.org/. Acesso em 23/Junho/2010.
[116] Dalvik. Disponível em http://www.dalvikvm.com/. Acesso em 23/Junho/2010.
[117] Hotspot. Disponível em http://openjdk.java.net/groups/hotspot/. Acesso em 23/Junho/2010.
[118] JamVM – Java Virtual Machine. Disponível em http://jamvm.sourceforge.net/. Acesso em 20/Junho/2010.
[119] JikesRVM. Disponível em http://jikesrvm.org/. Acesso em 23/Junho/2010.
[120] Kaffe. Disponível em http://www.kaffe.org/. Acesso em 23/Junho/2010.
[121] Mika VM. Disponível em http://www.k-embedded-java.com/index.php?option=com_content&view=article&id=25:openmika&catid=2:products&Itemid=14. Acesso em 23/Junho/2010.
[122] PhoneME. Disponível em https://phoneme.dev.java.net/. Acesso em 23/Junho/2010.
[123] SciMark 2.0. Disponível em http://math.nist.gov/scimark2/. Acesso em 23/Junho/2010.
[124] Linpack – Java Version. Disponível em http://www.netlib.org/benchmark/linpackjava/. Acesso em 23/Junho/2010.
[125] Marcus Weiskirchner, "Comparison of Execution Times of ADA, C and Java", Setembro 2005
[126] Dr. Reinhold P. Weicker, "DhryStone Benchmark: Rationale for version 2 and measurement rules" in *SIGPLAN Notices*, Agosto 1988, pp. 49-62.
[127] Bugzilla Bug 127. Disponível em http://server.complang.tuwien.ac.at/cgi-bin/bugzilla/show_bug.cgi?id=127. Acesso em 20/Junho/2010.