



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área departamental de Engenharia Electrónica e Telecomunicações e
de Computadores**



Sistema Integrado de Localização de Contentores de Correio

JOAQUIM EDUARDO MARTINS LUZ CARAÇA
(Bacharelato em Engenharia Informática)

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Informática e de Computadores

Orientador (es):

Doutor Manuel Martins Barata

Júri:

Presidente: Mestre Pedro Alexandre Seia Cunha Ribeiro Pereira

Vogais:

Doutor Pedro Miguel Florindo Miguéns Matutino

Doutor Manuel Martins Barata

Março de 2016

Declaro que esta dissertação/trabalho de projeto/relatório de Mestrado é o resultado da minha investigação pessoal e independente. O seu conteúdo é original e todas as fontes consultadas estão devidamente mencionadas no texto, nas notas e na bibliografia.

Sistema Integrado de Localização de Contentores de Correio

Copyright © Joaquim Eduardo Martins Luz Caraça, Instituto Superior de Engenharia de Lisboa.

O Instituto Superior de Engenharia de Lisboa tem o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

DEDICATÓRIA

Dedico este trabalho ao meu querido Pai

À minha querida Esposa

Ao meu filhinho

À minha família

Aos meus amigos

AGRADECIMENTOS

Ao meu orientador, Professor Doutor Manuel Martins Barata, pela imensa disponibilidade, incansável motivação e apoio que demonstrou ao longo de todo o projeto, exprimo a minha viva e cordial gratidão.

Aos CTT – Correios, pela oportunidade de me proporcionarem realizar um trabalho sobre um tema de estudo relacionado com a própria empresa.

Aos meus superiores hierárquicos, Dra. Julieta Cainço, Dra. Dulce Pires, Dr. Albino Fortunato Costa, pela forma como me apoiaram, incentivaram e acreditaram no meu projeto.

Aos colegas dos CTT que de diferentes formas se envolveram comigo neste projeto, e me prestaram todo o apoio e ajuda necessária à realização do mesmo, nomeadamente ao Alberto Bigotte de Almeida, Nuno Galão, Hernâni Silva, Rui São Pedro, Pedro Pinheiro e Rui Santos da *PayShop*, ao Paulo Gonçalves pela revisão do texto e tantos outros que comigo partilharam esta ideia.

Aos meus professores, colegas e amigos do ISEL, que me acompanharam ao longo do meu percurso académico e que sempre me deram alento para continuar.

Como não podia deixar de ser, a família, que é o principal suporte de vida de qualquer indivíduo, e como tal, tudo o que tenho conseguido, devo-o principalmente a eles, em especial a minha esposa pela sua compreensão, amizade e incentivo constantes, compreendendo os dias em que estou mais ausente.

Por fim, quero agradecer ao meu falecido pai, pela forma como me educou, mostrando-me os valores da vida, e pelo seu imenso amor, que me ajudou a ser a pessoa que sou hoje e que recordarei eternamente. Ao meu filho Filipe, que surgiu já no decorrer deste trabalho, e trouxe uma nova luz à minha vida, dedico, este trabalho, e peço desculpa pelo tempo que precisei de estar ausente.

A todos, os meus mais sinceros agradecimentos.

RESUMO

A eficiência dos processos de logística e transporte relativamente aos contentores de correio é um fator decisivo para a otimização dos custos e melhoria de qualidade de toda a operação empresarial. Neste sentido, é determinante conhecer em tempo útil onde os mesmos se encontram, quais os percursos que efetuaram, locais de origem, locais de destino ou seja, poder realizar a sua rastreabilidade.

Este conhecimento permite gerir de forma mais eficaz toda a frota de contentores utilizados no transporte de correio. Identificar a localização dos mesmos, possibilita antecipar necessidades operacionais que possam ocorrer em determinados locais, mediante maior ou menor fluxo de trabalho. Visa ainda o redesenho e automatização de processos para que os vários departamentos da empresa possam, otimizar o ciclo operativo e consequentemente obter uma melhor qualidade de serviço assim como enorme redução de custos. Por outro lado, não menos importante, e como consequência da melhoria dos processos envolvidos, pode ainda ter um impacto ambiental positivo, com a otimização dos trajetos de transporte dos referidos contentores.

No presente trabalho, apresenta-se um protótipo de demonstração do conceito de implementação, para um sistema de localização e rastreio de contentores. Esta área aplicacional suscita um grande interesse na medida em que se possam encontrar soluções que permitam efetuar a rastreabilidade dos contentores com custos reduzidos. O mesmo é baseado em tecnologia RFID e com suporte da linguagem de programação Java.

Palavras-chave

Contentores, RFID, Java

ABSTRACT

The efficiency of transport and logistics processes related with containers carrying mail is a key factor to optimize costs and improve the quality of the operating companies. It thus becomes crucial to know on time where containers are, what routes they made, place of origin, destination, among other information necessary to their track.

This knowledge enables a more effective management of the containers fleet used in mail transport. Knowing the location of containers, allows us to anticipate and optimize operational needs that may occur in certain places, according to the more or less workflow forecasted. It can create conditions for automating processes so that the various departments of the company can optimize the entire operation and the costs associated therewith. Moreover, and no less important, and as a result of improvement of the processes involved, can have a positive environmental impact by reducing or optimizing transport paths.

In this work is presented a demonstration prototype for a system of tracking and tracing of containers within the mail processing centers, distribution centers, mail and other buildings as well as outside them. This application area raises a lot of interest in that it can find solutions that enable the trace of this activity with a reduced cost. The project is based on RFID technology supported by programs developed in the programming language Java.

Keywords

Containers, RFID, Java

ÍNDICE

RESUMO	I
ABSTRACT	II
ÍNDICE	III
ÍNDICE DE FIGURAS	VII
ÍNDICE DE TABELAS	VIII
ACRÓNIMOS	IX
CAPÍTULO 1	1
1. Introdução	1
1.1. Descrição do Problema	1
1.2. Motivação	2
1.3. Objetivos	2
1.4. Organização do Trabalho	3
CAPÍTULO 2	5
2. Estado de Arte	5
2.1. RFID no Sector Postal	6
2.1.1. Caso dos Correios Finlandeses	6
2.1.2. Caso dos Correios Suíços	7
2.2. Sistemas RFID	8
2.2.1. Contexto Histórico RFID	9
2.2.2. Modo de Funcionamento	10
2.2.3. Normas e Padrões RFID	11
2.2.4. Componentes do Sistema	13
2.2.5. Segurança em RFID	15

2.2.6.	Cenários de Utilização	15
2.2.7.	Conclusão	17
CAPÍTULO 3		19
3.	Análise de Requisitos	19
3.1.	Requisitos Funcionais	19
3.2.	Requisitos Não Funcionais	20
3.3.	Visão Geral do Sistema.....	20
3.4.	Casos de Utilização.....	20
3.4.1.	Registar Comunicação da Etiqueta	21
3.4.2.	Consultar Entrada Contentor	21
3.4.3.	Consultar Permanência Contentor	22
3.4.4.	Consultar Saída Contentor	22
3.5.	Definição das Mensagens	23
3.5.1.	Diagrama de Troca de Mensagens.....	24
3.6.	Arquitetura do Sistema	25
3.7.	Diagrama de Instalação do Protótipo	26
CAPÍTULO 4		27
4.	Desenho e Modelação do Sistema.....	27
4.1.	Sistema ALOHA.....	27
4.2.	Caracterização das Colisões das Mensagens.....	30
4.3.	Desenho das Etiquetas	31
4.3.1.	Montagem da etiqueta.....	32
4.3.2.	Programação do controlador da etiqueta.....	34
4.4.	Desenho dos Leitores	34
4.4.1.	Modulo recetor de mensagens rádio	35

4.4.2.	Módulo de processamento	36
CAPÍTULO 5		39
5.	Implementação do Sistema	39
5.1.	Camada de Acesso a Dados	39
5.2.	Módulos de Software	40
5.2.1.	Pacote de Software <i>ContentoresClientKeepAlive</i>	41
5.2.2.	Pacote de Software <i>ContentoresRFIDMiddleware</i>	42
5.2.3.	Pacote de Software <i>ContentoresRFIDReader</i>	43
5.2.4.	Pacote de Software <i>ContentoresRFIDShared</i>	44
5.2.5.	Aplicação Web - SIGCC.....	45
5.3.	Configuração do Modem GSM.....	48
CAPÍTULO 6		51
6.	Testes e Análise de Resultados	51
6.1.	Plano de Testes	51
6.2.	Análise de Resultados	52
CAPÍTULO 7		55
7.	Conclusões e Trabalho Futuro	55
7.1.	Conclusões	55
7.2.	Trabalho Futuro	56
BIBLIOGRAFIA.....		57
ANEXOS.....		58
Anexo A – Código Fonte do Emissor e do Recetor.....		58
Anexo B – Modelo de Dados		73
Anexo C – Programa Java para testar os leitores.....		74

Anexo D – Shell Script Autoconnectnet.sh	77
Anexo E – Leituras Registadas pelas Etiquetas.....	78
Anexo F – Pacotes e Classes Java do Leitor.....	80
Anexo G – Pacotes e Classes <i>Middleware</i>	91

ÍNDICE DE FIGURAS

Figura 1 - Espectro Eletromagnético das bandas de frequência de um sistema RFID [5]	11
Figura 2 - Fornecimento de energia para etiqueta passiva.....	13
Figura 3 - Fornecimento de energia para etiqueta ativa.....	14
Figura 4 - Pacote de dados enviados pela etiqueta	23
Figura 5 - Mensagem XML de evento de Entrada.....	24
Figura 6 - Diagrama de alto nível de troca de mensagens	25
Figura 7 - Arquitetura física do protótipo.....	25
Figura 8 - Diagrama de instalação da solução	26
Figura 9 - Multiplexagem de comunicação ALOHA	28
Figura 10 - Probabilidade de colisão das mensagens	31
Figura 11 - Etiquetas e fonte de alimentação.....	31
Figura 12 - Etiqueta com caixa protetora	32
Figura 13 - NRF24L01 2.4GHz <i>Wireless Transceiver</i>	32
Figura 14 - Esquema eletrónico da etiqueta	33
Figura 15 - Adaptador de comunicação série para USB.....	36
Figura 16 - Ligação USB do módulo de rádio ao <i>Raspberry</i>	37
Figura 17 - Caixa com o leitor de Etiquetas	37
Figura 18 - Ficheiro LogKeepAlive.hbm.xml	39
Figura 19 - Ficheiro hibernate.cfg.xml	40
Figura 20 - Organização das peças de <i>software</i> desenvolvidas	40
Figura 21 - Diagrama de classes do projeto <i>ContentoresClientKeepAlive</i>	41
Figura 22 - Diagrama de classes que tratam as mensagens com informação das etiquetas	42
Figura 23 - Diagrama de classes que tratam as mensagens com informação dos leitores	43
Figura 24 - Diagrama de classes do pacote de <i>software</i> do leitor.....	44
Figura 25 - Diagrama de classes que compõem o pacote comum	45
Figura 26 - Página de administração do <i>apache-tomcat</i>	47
Figura 27 - Aplicação SIGCC	47
Figura 28 - Output do comando <i>lsusb</i> no <i>Raspberry Pi</i>	49
Figura 29 - Ficheiro configuração 19d2:0117	49
Figura 30 - Relatório de informação da conectividade GSM.....	50
Figura 31 - Tensão da bateria em função do nº de dias em funcionamento.....	54

ÍNDICE DE TABELAS

Tabela 1 - Evolução Histórica RFID	9
Tabela 2 - Regiões ITU-R	10
Tabela 3 - Características Frequência RFID	11
Tabela 4 - Características das etiquetas ativas e passivas	15
Tabela 5 - Dispersão de contentores por local	52
Tabela 6 - Número de leituras RFID por local	53
Tabela 7 - Leituras obtidas com a etiqueta do identificador 51	53
Tabela 8 - Leituras obtidas com a etiqueta do identificador 53	78
Tabela 9 - Leituras obtidas com a etiqueta do identificador 54	78
Tabela 10 - Leituras obtidas com a etiqueta do identificador 55	79
Tabela 11 - Leituras obtidas com a etiqueta do identificador 56	79

ACRÓNIMOS

API	-	<i>Application Programming Interface</i>
APN	-	<i>Access Point Name</i>
CDP	-	<i>Centro de Distribuição Postal</i>
CPLS	-	<i>Centro de Produção e Logística do Sul</i>
EEPROM	-	<i>Erasable Programmable Read-Only Memory</i>
EPC	-	<i>Electronic Product Code</i>
GPS	-	<i>Global Positioning System</i>
GSM	-	<i>Global System for Mobile Communications</i>
IEC	-	<i>International Electrotechnical Commission</i>
ISM	-	<i>Industrial, Scientific and Medical</i>
ISSO	-	<i>International Organization for Standardization</i>
LAN	-	<i>Local Area Network</i>
LF	-	<i>Low Frequency</i>
MIT	-	<i>Massachusetts Institute of Technology</i>
MVC	-	<i>Model View Controller</i>
OCR	-	<i>Optical character recognition</i>
PDF	-	<i>Portable Document Format</i>
RFID	-	<i>Radio Frequency Identification</i>
RTLS	-	<i>Real-time locating system</i>
SMS	-	<i>Short Message Service</i>
SSH	-	<i>Secure Shell</i>
TCP/IP	-	<i>Transmission Control Protocol</i>
UHF	-	<i>Ultra-High Frequency</i>
UML	-	<i>Unified Modeling Language</i>
USB	-	<i>Universal Serial Bus</i>
WAN	-	<i>Wide Area Network</i>
XML	-	<i>eXtensible Markup Language</i>

CAPÍTULO 1

1. Introdução

O presente trabalho está inserido no âmbito do projeto de mestrado, do curso de Engenharia Informática e Computadores do Instituto Superior de Engenharia de Lisboa. O tema escolhido resulta da minha experiência enquanto auditor interno de uma grande empresa nacional e da preocupação com o controlo sobre ativos empresariais de elevado custo. Embora o tema tratado aborde a temática relativa a contentores utilizados no contexto do sector postal ele é transversal a todo o tipo de contentores usados em operações de distribuição e logística.

Neste primeiro capítulo apresentam-se as principais razões que determinam a importância deste trabalho e que motivaram a realização do mesmo. Em primeiro lugar, expõe-se o que motivou este estudo, e a seguir a importância e o impacto do conhecimento obtido para as empresas, relativamente à localização dos contentores que transportam o correio. São também apresentados os principais objetivos do estudo, terminando o capítulo com a descrição da organização do documento.

1.1. Descrição do Problema

Os contentores de correio são transportados por todo o país numa vasta rede de itinerários. Para este efeito são utilizadas viaturas da empresa e de terceiros, através de subcontratação.

A rede de distribuição é caracterizada por uma enorme dispersão geográfica que vai desde os centros de produção e logística aos centros de distribuição postal situados em cada localidade. Nos centros de produção e logística, o correio vindo dos diversos pontos do país é colocado em máquinas que efetuam o reconhecimento dos endereços das cartas por leitura OCR. Neste processo, é colocado um código de barras nas cartas, para posterior tratamento por leitura ótica. Com base nesta leitura encaminham o correio para o seu destino, agrupando-o e colocando-o em cassetes (caixas de plástico) que por sua vez são colocadas dentro de contentores. Estes contentores são rotulados com o local de destino, e posteriormente encaminhados para as portas de saída no cais, para serem introduzidos dentro das viaturas de transporte. Cada porta de saída onde está a viatura corresponde a um conjunto de locais de destino conhecido.

No percurso inverso, e diariamente, são entregues nos locais de distribuição e reenviados ao final do dia, aquando do processo de expedição de correio. No entanto, existem outros fluxos de transporte como por exemplo os que são utilizados na receção de correio de grandes clientes, com entrega e recolha direta nas instalações dos mesmos.

Atualmente, a falta de informação sobre o fluxo e a localização de cada contentor, levanta questões preocupantes relativamente à área de negócio que efetua a gestão dos mesmos. Para se saber ao certo

Sistema Integrado de Localização de Contentores de Correio

quantos existem – estima-se que existem atualmente cerca de 10.000 contentores, onde se encontram, o seu estado de conservação, quantos devem ser substituídos no final de cada período, se houve perda de alguns e onde, datas previsíveis de abate, são tarefas hoje em dia impossíveis de realizar de forma sistemática.

1.2. Motivação

Empresas de logística e distribuição operam num mercado cada vez mais vasto. No caso concreto de empresas de distribuição de correio existe uma dispersão geográfica enorme o que leva a que o correio seja transportado por todo o país em viaturas das próprias empresas de distribuição de correio ou por empresas contratadas para realizar essas funções. Estas empresas têm necessidades específicas de informação relativamente à rede de transporte e daquilo que transportam independentemente do tipo de produtos. Ora neste contexto, e observando como exemplo os Correios, estes operam numa rede de distribuição conhecida e sobre a qual transportam diverso tipo de correio. Desta forma a principal motivação deste trabalho é encontrar uma solução que permita responder de forma positiva ao controlo de fluxo de contentores associados às atividades empresariais de distribuição e logística. A falta de controlo sobre estes ativos resulta em perdas financeiras avultadas para as empresas. A maior parte das vezes nem se consegue quantificar exatamente o valor dessas perdas. Essa perceção evidencia-se quando surge a necessidade de aquisição de novos equipamentos e muitas das vezes a área de negócio envolvida não foi alvo de um processo de expansão.

O conhecimento da localização destes ativos permite uma gestão com maior rigor e controlo com impacto ao nível de todos os custos produtivos. Desta forma consegue-se controlar e reduzir o risco da perda destes ativos e ao mesmo tempo melhorar os demais processos de logística. Permite uma gestão eficiente destes recursos, e assim maximizar o seu uso e até da própria rede de transporte tornando esta mais eficiente e mais amiga do ambiente. Ao nível da relação de negócio com os clientes possibilita desencadear processos de maior transparência possibilitando a disponibilização de informação sobre os produtos colocados na rede de distribuição através de uma relação de compromisso e cooperação com estes.

1.3. Objetivos

Para além de estudar as tecnologias e plataformas existentes e emergentes nesta área, são objetivos gerais deste trabalho, a formulação e apresentação de uma solução que responda ao problema descrito na secção anterior. Pretende-se demonstrar a possibilidade de utilização da tecnologia RFID, no contexto da rastreabilidade e localização de contentores de correio que permita o controlo interno destes ativos. Pretende-se apresentar uma proposta de uma solução baseada em sensores e desenvolvimento aplicacional que permita implementar um sistema capaz de endereçar uma solução que responda ao

problema descrito. Para demonstração da validade do trabalho efetuado deve ser implementado um protótipo de referência e testado num conjunto de percursos reais, que permita tirar conclusões relativamente a:

- ✚ Adequação do sistema proposto como solução do problema que é proposto resolver.
- ✚ Adequação da definição dos processos de distribuição e logística de contentores de correio internamente à organização e interações com clientes.
- ✚ Verificação de controlos implementados no sentido de uma minimização de riscos de perda destes ativos.
- ✚ Verificação do comportamento e desempenho do sistema através da realização de casos de testes.

1.4. Organização do Trabalho

Neste primeiro capítulo expõem-se as principais motivações para a realização deste estudo nomeadamente a importância e impacto consequente do conhecimento obtido relativamente à localização e rastreabilidade de contentores. Apresentam-se também quais os principais objetivos deste estudo e o mesmo termina com a descrição da organização de todo o documento.

No segundo capítulo apresenta-se o estudo do “Estado de Arte” efetuado, sobre a tecnologia RFID, vantagens e desvantagens, quando comparada com outros sistemas de rastreabilidade. Na parte final apresentam-se as conclusões e que são o ponto de partida para a formulação de uma solução para o problema objeto deste trabalho.

No terceiro capítulo apresenta-se a formulação teórica sobre a conceção e desenvolvimento das etiquetas e leitores RFID, bem como, a descrição sobre as mensagens rádio transmitidas.

O quarto capítulo descreve, através da metodologia UML¹, como o sistema pode ser implementado. Fornece uma visão geral do mesmo, os requisitos funcionais, e não funcionais apresentados sob a forma de casos de utilização, diagramas de classes e de atividade. Apresentam-se os vários componentes que constituem o sistema e a forma como estes interagem como um todo.

No quinto capítulo apresentam-se os vários módulos em termos de programação e a descrição do processo de implementação dos mesmos. A linguagem de programação utilizada e alguns aspetos técnicos utilizados e associados ao projeto.

No sexto capítulo é descrito o plano de testes e seus objetivos. É apresentada e analisada a recolha de dados durante a fase destes, de forma a permitir formular a conclusão apresentada, no capítulo seguinte.

¹ *Unified Modeling Language*

Sistema Integrado de Localização de Contentores de Correio

Por fim no sétimo capítulo apresenta-se a conclusão retirada de todo o trabalho efetuado, assim como os próximos trabalhos passíveis de serem desenvolvidos sobre o tema estudado.

CAPÍTULO 2

2. Estado de Arte

Os sistemas de identificação por radiofrequência ou RFID estão hoje em dia amplamente estudados e são utilizados nos mais diversos fins com uma longa história de existência [1]. Caracterizam-se por utilizar comunicações sem fios através dos campos eletromagnéticos das frequências de rádio para transmissão de dados [1]. O propósito da sua utilização é a identificação e rastreabilidade automática de objetos, pessoas ou animais, através da comunicação de dados entre as componentes que constituem estes sistemas. São exemplos da sua utilização os sistemas de controlo de acessos nos transportes públicos, cartões multibanco para realização de transações entre muitos outros. Um passo importante para a consolidação desta tecnologia foi o aparecimento das etiquetas inteligentes, uma versão de custo reduzido, altamente integrável e que permitem a construção de soluções de identificação por caminhos inacessíveis a qualquer outra tecnologia desta área [1].

Também no contexto da área de logística e nomeadamente no sector postal, a tecnologia RFID conta já com uma larga experiência, existindo alguns operadores postais a nível mundial, que a utilizam por forma a melhorar os seus processos de negócio e operacionalidade. De seguida apresentam-se alguns exemplos desta utilização no setor postal e posteriormente dá-se a conhecer os aspetos principais relacionados com a tecnologia e o seu modo de funcionamento.

Um estudo realizado em 2015 sobre a tecnologia RFID, pela *IDTechEx* [2] projetou o valor deste mercado para esse mesmo ano em 10,1 bilhões de dólares. Efetua ainda uma previsão para uma subida na ordem dos 13.2 milhões dólares americanos em 2020, o que realça a importância desta tecnologia na economia mundial. É ainda divulgado neste mesmo estudo que em termos de tendências, a tecnologia RFID vai continuar a ser adotada para etiquetar peças de vestuário - prevendo atingir 4,6 mil milhões de etiquetas em 2016. Tem no entanto ainda algum caminho a percorrer já que a penetração no mercado, representa apenas 15% do mercado total de vestuário em 2016. Na marcação de animais (como porcos, ovelhas e animais de estimação) espera um aumento substancial, uma vez que continua a ser um requisito legal em muitos países, com 420 milhões de etiquetas a ser utilizadas para este sector em 2016. No total, a *IDTechEx* projeta a venda de 8,9 bilhões de etiquetas em 2015 e 10,4 bilhões em 2016, sendo que a maior parte desse crescimento é de etiquetas RFID UHF passivas. De notar ainda que o crescimento maior esperado é nas etiquetas RFID ativas e sistemas RTLS.

De seguida dá-se a conhecer os principais aspetos relacionados com esta tecnologia e o seu modo de funcionamento.

2.1. RFID no Sector Postal

Por consulta à página da internet do *RFIDjournal*² e pesquisando pelo tema “RFID Post” encontram-se algumas publicações de interesse nesta área. No sentido de perceber quais as preocupações que outros operadores postais sentem relativamente à logística dos contentores efetuou-se pesquisa para tentar obter respostas a esta temática. De entre os vários exemplos de aplicação da tecnologia RFID e sem quaisquer critérios de escolha apresentam-se aqui dois exemplos de casos de sucesso. Pretende-se que ao analisar estes casos de sucesso se retirem ilações no sentido de nos ajudar a perceber quais as motivações que os levaram a introduzir esta tecnologia nos seus processos de negócio e também de que forma os implementaram.

2.1.1. Caso dos Correios Finlandeses

Na página de internet do *RFIDJournal*, pode encontrar-se uma publicação (Collins, 2006)³, que descreve como os correios finlandeses testaram a utilização de um sistema RFID para controlar os contentores utilizados no transporte de correio. Após a duração do projeto, que levou dois meses de execução experimental, os correios finlandeses observaram que era claramente vantajoso utilizar etiquetas RFID - UHF 856 MHz passiva encapsuladas em plástico para evitar danos físicos às antenas das etiquetas, para rastreio dos seus contentores.

Em cada ano perdia cerca de 17.000 desses contentores, custando cerca de 1,3 milhões de euros (\$ 1,6 milhões) para os substituir. Além do mais, por causa da alta taxa de evasão, nem sempre tinham um número suficiente de contentores disponíveis para as suas atividades. Este facto fazia aumentar os custos operacionais e reduzia a qualidade dos serviços que prestava, uma vez que não conseguia disponibilizar aos clientes, os contentores que estes necessitavam.

Os correios Finlandeses tinham uma cláusula nos seus contratos com os clientes que especificam que dentro de duas semanas após a entrega dos contentores vazios nas instalações destes, deveriam ser recolhidos pelos correios com a carga do cliente. No entanto, não havia uma forma de saber onde tinham entregado contentores ou há quanto tempo estavam os mesmos nos clientes.

Ao mesmo tempo, era também intenção dos correios Finlandeses fazer uso da informação recolhida por RFID nas suas operações para fornecer aos clientes e operadoras parceiras informações cada vez mais detalhada e pró-ativa sobre o estado das suas remessas.

Nos contentores, em 200 de cerca de 200.000, para além de lhes ter sido colocada uma etiqueta RFID fixa com braçadeiras de plástico, também lhes foi colocada uma fita adesiva à volta, a indicar que tinham sido marcados para o projeto piloto e que deveriam ser devolvidos ao terminal de correio em Helsínquia, um

² <http://www.rfidjournal.com/>

³ <http://www.rfidjournal.com/articles/view?2207/2>

Sistema Integrado de Localização de Contentores de Correio

centros de distribuição em todo o país. Neste terminal foram colocados três leitores fixos UHF. Dois dos leitores foram designados para verificar os contentores dentro e fora da instalação, enquanto o terceiro foi localizado num ponto de entrada separada no edifício, onde os contentores vindos de uma cidade vizinha entravam. Por seu lado, a equipa de trabalho dentro das instalações, também tinha leitores portáteis para verificação das leituras, sem ter que movê-los através de um dos cais.

Na relação com os clientes, os Correios Finlandeses optaram por equipar o trabalhador dos correios com um leitor portátil para interrogar as etiquetas dos contentores, bem como uma etiqueta RFID montada nas portas de entrega de área de entrega do cliente. O dispositivo portátil também gravava o dia e a hora de cada leitura. O motorista introduzia um evento específico no dispositivo, registando-o como uma entrega ou uma recolha, e se o contentor estava vazio ou carregado. Quando o motorista voltava para o terminal principal, os dados recolhidos durante as suas rondas poderiam, então, ser enviados para a aplicação de teste através de uma conexão LAN sem fio.

Em termos de relatórios, a informação registada podia ser analisada em termos de eventos, como “entregas” ou “recolhas” por cliente ou rota, por contentores entregues e devolvidos por um determinado cliente de um dia específico, ou pelo número de contentores solicitados por um cliente para cada dia ou semana.

Ao etiquetar os contentores, os Correios finlandeses descobriram que alguns dos seus clientes mantinham em sua posse contentores por mais de duas semanas, e que alguns estavam usando os contentores para o seu próprio uso de distribuição. Esta verificação decorreu de estarem a receber contentores de clientes para os quais não tinham entregado qualquer contentor marcado.

O facto de estarem a controlar o fluxo dos contentores, não só fez os clientes mais responsáveis a voltar a enviar os contentores, mas também deu aos funcionários dos Correios Finlandeses um incentivo para recolhê-los.

Apesar do valor da informação, os correios Finlandeses dizem-se (à altura do projeto) não estar ainda preparados para etiquetar todos os seus contentores pelo facto de não existirem normas acordadas entre clientes e os correios. Será necessário que os clientes adotem a tecnologia e que utilizem a mesma frequência e o mesmo tipo de etiqueta usada, permitindo uma troca de informação de negócio que beneficie ambas as partes.

2.1.2. Caso dos Correios Suíços

Uma outra publicação na mesma página de internet (Wessel, 2008)⁴ aborda como os correios suíços pensaram este problema e como o tentam resolver. O referido artigo diz que antes de introduzir a etiquetagem nos contentores era impossível realizar um inventário confiável destes ativos. A empresa

⁴ <http://www.rfidjournal.com/articles/view?4270>

Sistema Integrado de Localização de Contentores de Correio

contava-los manualmente a cada dois anos, na altura cerca de 45 000, um processo trabalhoso que exigia em média 200 dias de trabalho envolvendo duas pessoas em cada local.

Não obstante, observaram, que não havia uma explicação plausível para os contentores em falta e que não conseguiam criar qualquer tipo de estatística para ajudar a uma utilização eficiente destes recursos. Foram também, estimados custos muito elevados, no transporte entre locais da própria empresa e no empréstimo a clientes.

Na implementação, cada contentor foi equipado com uma etiqueta *EPC Gen 2* [3] colocada numa caixa de plástico e fixa ao bordo superior do contentor. Para a escolha deste tipo de etiqueta, pesou o facto da mesma ser considerada como um padrão de mercado e também o seu baixo custo. Para além disso, os correios suíços esperam que os seus clientes - incluindo as empresas para as quais presta serviços de logística, comecem também a utilizar este tipo de etiqueta com mais frequência, perspetivando desta forma uma relação de maior satisfação com os seus clientes nos seus processos de negócio.

Sempre que um contentor passa por um portal RFID é lida a etiqueta e inserida a informação numa base de dados com a indicação da localização do contentor. A partir daqui, podem gerar-se relatórios com a indicação do número de contentores disponíveis num determinado local. Ao saber quantos contentores estão à disposição e onde se encontram fisicamente pode certificar-se de que tem o número necessário de contentores em cada local, a fim de lidar com os volumes de trabalho esperado.

Por fim, a informação obtida permite à empresa gerir de forma mais adequada e amiga do ambiente estes recursos pelo facto de realizar menos transportes entre os diversos centros e evitar atrasos devido à não disponibilidade de contentores.

2.2. Sistemas RFID

Como já referido, os sistemas RFID continuam a ser hoje em dia amplamente estudados e são utilizados nos mais diversos fins. Caracterizam-se por utilizar comunicações sem fios através dos campos eletromagnéticos das frequências de rádio para transmissão e receção de dados [4]. O propósito da sua utilização é a identificação e rastreabilidade automática de objetos, pessoas ou animais, conseguida através da comunicação de dados entre as componentes que constituem estes sistemas. Um passo importante para a consolidação desta tecnologia foi o aparecimento das etiquetas inteligentes (e.g. colocadas sob a pele dos animais para efetuar a sua rastreabilidade), com um custo reduzido e altamente integrável que permitem a construção de soluções de identificação por caminhos inacessíveis a qualquer outra tecnologia desta área [4]. A seguir apresentam-se as principais características desta tecnologia e a forma como a mesma pode ser utilizada.

2.2.1. Contexto Histórico RFID

A utilização de tecnologia RFID conta já com uma longa história de existência, que remonta ao tempo da segunda guerra mundial. Os Alemães, Japoneses, Americanos e Ingleses usavam sistemas de radar descoberto em 1935 pelo físico escocês *Sir Robert Alexander Watson-Watt*, para alertar sobre a aproximação de aviões enquanto estes ainda estavam a quilómetros de distância. Contudo não havia maneira de identificar quais os aviões que pertenciam ao inimigo e os dos próprios pilotos do país que voltavam de uma missão. Os alemães então descobriram que se os seus pilotos girassem os seus aviões quando estivessem de regresso à base o sinal de rádio seria modificado e refletido de volta ao radar. Esse método simples alertava os técnicos responsáveis pelo radar que se tratava de aviões alemães (este foi, considerado o primeiro sistema passivo de RFID) (Roberti, 2005)⁵. Sob o comando de Watson-Watt, que liderou um projeto secreto, os ingleses desenvolveram o primeiro identificador ativo de amigo ou inimigo (IFF – *Identify Friend or Foe*). Foi colocado um transmissor em cada avião britânico e quando esses transmissores recebiam sinais das estações de radar no solo, começavam a transmitir um sinal de resposta, que identificava o avião como amigo. Os sistemas RFID funcionam no mesmo princípio básico. Um sinal é enviado a um transmissor, o qual é ativado e reflete de volta o sinal (sistema passivo) ou transmite seu próprio sinal (sistema ativo). Os avanços na área de radares e de comunicação RF (Radio Frequência) continuaram através das décadas de 50 e 60. Cientistas e académicos dos Estados Unidos, Europa e Japão realizaram pesquisas e apresentaram estudos explicando como a energia RF poderia ser utilizada para identificar objetos remotamente.

Esta tecnologia não tem parado de evoluir, sendo atualmente o *ibeacon* a mais recente aplicação nesta área. Esta nova tecnologia desenvolvida pela Apple, anunciada no Verão de 2013 (a nova API do iOS 7), permite às empresas, lojas e qualquer tipo de comércio, configurar transmissores que enviam para os *smartphones* ou *tablets*, próximos à sua presença, notificações com qualquer tipo de informação. A tabela 1 seguinte apresenta um quadro resumo da evolução da tecnologia RFID.

Tabela 1 - Evolução Histórica RFID

Evolução Histórica RFID	
1940-1950	Radar refinado e usado, grande esforço de desenvolvimento da segunda guerra mundial. RFID inventado em 1948.
1950-1960	Primeiras explorações da tecnologia RFID, experimentações de laboratório.
1960-1970	Desenvolvimento da teoria da RFID. Início de ensaios no campo de aplicações.
1970-1980	Explosão do desenvolvimento com RFID. Aceleração dos testes com RFID. Implementações muito precoces de RFID.
1980-1990	Aplicações comerciais RFID começam a surgir.
1990-2000	Surgimento de padrões e normas RFID. Sistemas RFID largamente implementados. RFID começa a fazer parte da vida de todos os dias.
2000-	Explosão RFID continua.

⁵ <http://www.rfidjournal.com/articles/view?1338>

2.2.2. Modo de Funcionamento

Existem diferentes bandas de frequência que podem ser utilizadas por um sistema RFID e estas podem variar de país para país. Estas bandas estão diretamente relacionadas com as bandas ISM (*Industrial, Scientific and Medical*) que não carecem de licenciamento obrigatório levando a que cada pedaço do espectro seja alvo de enorme disputa. Por este facto torna-se necessário um controlo muito rigoroso por parte das entidades reguladoras de cada país que ao mesmo tempo leva a que seja difícil atuar em termos de regulamentação e normativos a nível mundial.

Apesar de não haver uma banda de funcionamento comum para todos os países, existem no entanto, três grandes áreas regulamentares definidas pelo ITU-R⁶ como se pode ver pela tabela 2.

Tabela 2 - Regiões ITU-R

Região ITU-R		
1	2	3
EU e África	Américas	Ásia
EU	USA	Austrália
CEPT/ETSI	Canadá	Nova Zelândia
África do Sul	América Central	Japão
Israel	América do Sul	Coreia do Sul
		Singapura
		Hong Kong
		China
		Tailândia
		Índia

Desta forma, o grupo de países inserido numa determinada região tem de cumprir as normas da entidade reguladora da sua região. Nos últimos tempos temos assistido, a uma maior intenção dos governos e centros de decisão, em aumentar o esforço no sentido de encontrar soluções, mais abrangentes para esta área como é o caso da união europeia⁷. As bandas de funcionamento dos sistemas RFID podem subdividir-se em subgrupos mais ou menos comuns:

- ❖ LF (Low Frequency), 125 – 134,2 kHz
- ❖ HF (High Frequency), 13,56 MHz
- ❖ UHF (Ultra High Frequency), 860 MHz – 960 MHz
- ❖ SFH (Super High Frequency), 2,4 GHz – 2,48 GHz

A figura 1 mostra o espectro eletromagnético com as faixas de frequência que os sistemas de RFID podem utilizar.

⁶ *International Telecommunication Union – Radiocommunication Sector*

⁷ <https://ec.europa.eu/digital-single-market/en/news/commission-calls-industry-and-european-standardisation-organisations-speed-development-common>

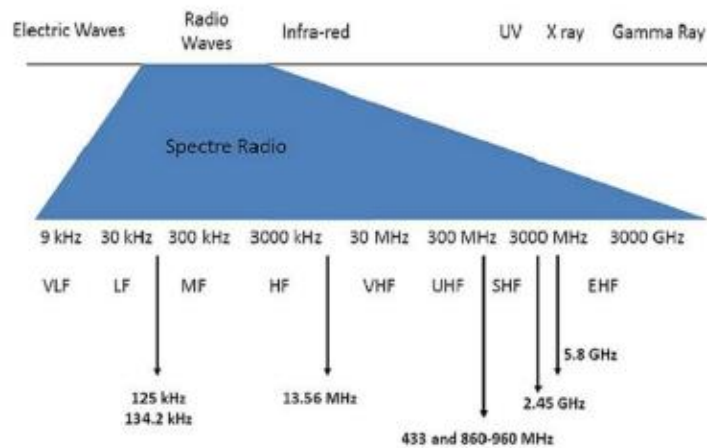


Figura 1 - Espectro Eletromagnético das bandas de frequência de um sistema RFID [5]

A tabela 3 apresenta um resumo com base num conjunto de características na banda utilizada de um sistema RFID, tais como a frequência, o alcance de leitura, vantagens, desvantagens e cenários de utilização.

Tabela 3 - Características Frequência RFID

RFID – Frequências de Utilização				
Banda	LF (Baixa Frequência)	HF (Alta Frequência)	UHF (Ultra Alta Frequência)	Micro-ondas
Frequência RFID	125 – 135 KHz	13,56 MHz	433 MHz ou 860 a 960 MHz	2,45 MHz ou 5,8 GHz
Alcance	Inferior a 0,5 metros	Até 1,5 metros	Até 100 metros	Até 30 metros
Velocidade de Transferência	Inferior a 1 Kbit/s	Aproximadamente 25 Kbit/s	Aproximadamente 30 Kbit/s	Até 1 Mbit/s
Exemplos de Aplicação	Controlo de Acessos, Trace de Animais	Bibliotecas, Livrarias, Cartões Pessoais de Identificação	Indústria Automóvel, Controlo de Armazéns, Logística	Portagens, Veículos em Movimento, Logística



2.2.3. Normas e Padrões RFID

Os padrões e as normas de RFID são orientações ou especificações para o desenvolvimento e utilização desta tecnologia para diferentes produtos. Estes padrões fornecem diretrizes sobre como os sistemas RFID devem funcionar, em que frequências devem operar, a forma como devem ser transmitidos os dados e como a comunicação deve funcionar entre o leitor e a etiqueta.

Sistema Integrado de Localização de Contentores de Correio

São essenciais para a maioria das aplicações que utilizam esta tecnologia pois ajudam a garantir que os produtos de RFID são interoperáveis, independentemente do fornecedor ou utilizador. Também fornecem diretrizes pelas quais as empresas podem desenvolver produtos complementares, tais como diferentes tipos de etiquetas, leitores, programas informáticos e acessórios. Além disso, os padrões ajudam a ampliar os mercados e aumentar a concorrência dentro da indústria, o que faz com que os preços dos produtos com padrão RFID possam ser competitivos e aumentar a confiança generalizada na tecnologia.

Existem dois organismos principais de normalização RFID a nível internacional:

-  ISO - Organização Internacional de Normalização
-  EPCglobal – Código Eletrónico de Produto Global Incorporado

Embora estas duas organizações forneçam as principais normas sobre esta tecnologia, há também uma infinidade de outras normas que se aplicam a áreas de nicho de RFID.









Em termos de organizações de normalização a ISO é a mais antiga. Em 1996, foi constituída uma comissão conjunta com o IEC⁸ para se debruçarem sobre a normalização da tecnologia RFID.

As normas ISO para RFID estão divididas por uma série de categorias, de acordo com a aplicação dos sistemas RFID. Estas incluem: interface aérea e protocolos associados; conteúdo de dados e respetiva formatação; testes de conformidade; aplicações; e várias outras áreas menores.

Para além das normas ISO RFID, existem também os padrões da EPC global. Em 1999, um número de empresas industriais em conjunto com o MIT definiram um consórcio conhecido como o consórcio *Auto-ID*, com o objetivo de pesquisar e padronizar a tecnologia RFID.

Em 2003, esta organização foi dividida ficando a maioria das atividades de normalização na *EPCglobal*, nome pelo qual se passou a designar. O centro *Auto-ID* manteve as suas atividades relacionadas com a investigação nesta tecnologia.

Dentro das normas existentes, enumeram-se as principais com uma breve descrição de cada uma delas:

-  ISO 10536 – define padrões para acoplamento de cartões.
-  ISO 11784 - norma que define a forma pela qual os dados são estruturados numa etiqueta RFID.
-  ISO 11785 – norma que define o padrão do protocolo para a identificação por radiofrequência (RFID) dos animais.
-  ISO 14443 - padrão que fornece as definições para o protocolo para etiquetas RFID utilizadas em sistemas de proximidade - destinado ao uso em sistemas de pagamento.
-  ISO 15459 - identificadores únicos para unidades de transporte (utilizados na gestão da cadeia de abastecimentos).
-  ISO 15693 - padrão que define o uso de cartões de proximidade.
-  ISO 18000 - padrão que define a interface aérea sobre as frequências RFID em todo o globo.
-  ISO 18046 – define a implementação dos testes de desempenho das etiquetas e leitores.

⁸ *International Electrotechnical Commission*

🚦 ISO 24730 – padrão para os sistemas de localização em tempo real.

2.2.4. Componentes do Sistema

Tipicamente, um sistema de identificação por radiofrequência é constituído pelos seguintes componentes:

🚦 Leitor

Os leitores ou interrogadores como referidos em muita literatura, são dispositivos que transmitem e recebem ondas de rádio para comunicação sem fios com as etiquetas RFID. Existe uma grande variedade de tipos de leitores com formatos muito diferenciados uns dos outros.

As unidades portáteis são uma combinação leitor / antena, enquanto os sistemas maiores costumam separar as antenas do leitor. O leitor recupera a informação a partir da etiqueta e pode ser autossuficiente e registrar as informações internamente. No entanto, também pode ser parte de um sistema localizado, como uma caixa registadora (POS⁹), uma grande rede de área local (LAN) ou uma rede de área ampla (WAN¹⁰).

🚦 Etiqueta

As etiquetas são constituídas por uma antena para transmitir e receber sinais de rádio ligada a um circuito integrado, onde é colocado o identificador único da etiqueta e outro tipo de informação.

Podem assumir diferentes formatos e por vezes são de dimensões muito reduzidas. Têm capacidade de produzir e modelar um sinal de rádio para ligação a uma antena ou leitor. Estão normalmente ligadas fisicamente aos objetos ou itens que se quer rastrear. Com base nas suas características podem classificar-se da seguinte forma:

- **Etiquetas Passivas**

Internamente não possuem bateria nem transmissor. Através da antena interna, o campo eletromagnético do leitor fornece toda a energia necessária à mesma (Figura 2).

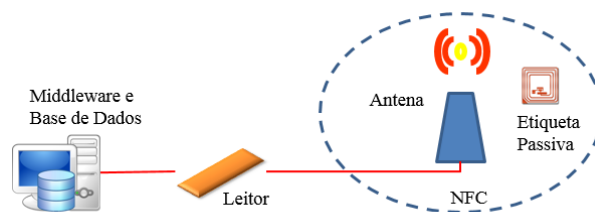


Figura 2 - Fornecimento de energia para etiqueta passiva

⁹ *Point of Sale or Point of Service*

¹⁰ *Wide Area Network*

As comunicações são realizadas através da energia emitida pelo leitor e esta é utilizada para a transmissão de dados tanto a partir do leitor para a etiqueta como de volta para o leitor.

Caso a etiqueta esteja localizada fora do intervalo do sinal emitido pelo leitor, e dado não possuir fonte de alimentação, não será capaz de comunicar. Contêm um identificador único e pode possuir uma memória interna (EEPROM¹¹) para armazenar dados. São mais baratas, mecanicamente mais flexíveis e possuem um tempo de vida útil maior mas apenas permitem um alcance reduzido.

○ Etiquetas Ativas

Possuem internamente a sua própria fonte de energia que pode ser por exemplo, na forma de uma pilha ou de uma célula solar. Neste caso a fonte de alimentação é utilizada para providenciar energia ao circuito eletrónico o que lhes permite enviar por *broadcast*¹² sinais rádio para o leitor. O campo magnético ou eletromagnético do leitor deixa de ser utilizado para fornecer energia ao circuito eletrónico (Figura 2). Pelo facto de possuir energia armazenada significa que comunica a uma distância maior do que uma etiqueta passiva. Essa distância está dependente da capacidade de detetar o sinal de interrogação proveniente do leitor.

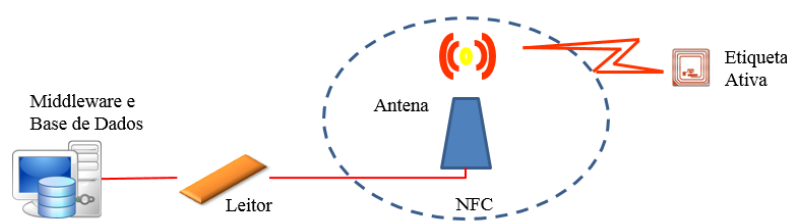


Figura 3 - Fornecimento de energia para etiqueta ativa

No caso de utilização de uma pilha internamente, estas etiquetas podem ter um tempo de vida útil até cerca de dez anos, dependendo do número de operações a que estejam sujeitas. Possuem um nível de potência mais elevado do que as passivas e semi-passivas/ativas, possibilitando-lhes um maior alcance de comunicação. Normalmente permitem operações de leitura e escrita possibilitando alterar a informação nelas contida. Têm um custo mais elevado que as restantes e são normalmente de dimensões maiores [6].

○ Etiquetas Semi-Passivas/Ativas

Tal como as etiquetas ativas, possuem alimentação própria através de uma bateria ou pilha. No entanto esta fonte de energia não contribui para a transmissão dos dados da etiqueta para o leitor e precisam do campo eletromagnético deste para efetuar a comunicação. Esta limitação tem impacto no alcance da comunicação e varia consoante a frequência em que operam [6].

Para melhor compreender as características das etiquetas e respetivas diferenças entre elas apresenta-se a tabela 4 em que são analisadas as mesmas segundo vários parâmetros.

¹¹ *Erasable Programmable Read-Only Memory*

¹² Processo de difusão da informação para múltiplos recetores ao mesmo tempo

Tabela 4 - Características das etiquetas ativas e passivas

Parâmetro	Etiqueta RFID Ativa	Etiqueta RFID Passiva
Energia	Interna	Transferida do leitor por RF
Força de sinal requerida do leitor para a etiqueta	Muito baixo	Muito alto
Força de sinal disponível da etiqueta para o leitor	Alta	Muito baixo
Alcance	Longo alcance (100m ou mais)	Curto ou muito curto (3m ou menos)
Capacidade	Larga capacidade de leitura e escrita (128 KB) com capacidade de pesquisa e acesso disponíveis	Pequena capacidade de leitura e escrita (128 bytes)

2.2.5. Segurança em RFID

Uma das grandes preocupações na implementação de sistemas de radiofrequência é a segurança da informação tornando possível o não roubo ou alteração da mesma. Neste sentido as etiquetas mais baratas tais como as etiquetas passivas EPC¹³, não têm qualquer poder de computação para executar operações básicas de criptografia.

Estas etiquetas são na melhor das hipóteses capazes de utilizar chaves estáticas, ou seja, códigos de segurança e palavras passe como mecanismos de segurança. Estas limitações são hoje vistas como um desafio especial que este tipo de etiqueta contempla.

Por outro lado, as etiquetas com maior custo, dispõem de funcionalidades avançadas, e muitas vezes incluem a capacidade de executar algoritmos criptográficos básicos, tais como criptografia de chave simétrica e protocolos de identificação de desafio-resposta.

Pelo fato da criptografia de chave pública ter custos bastante elevados, apenas é utilizada em sistemas que sintam a necessidade de sigilo e confidencialidade da informação colocada nas mesmas.

2.2.6. Cenários de Utilização

Esta tecnologia pode ser utilizada nos mais diversos cenários de diferentes organizações ou indústrias. Enumeram-se aqui alguns exemplos meramente informativos, sobre a possibilidade da sua utilização:

¹³ *Electronic Product Code*

Sistema Integrado de Localização de Contentores de Correio

- 🚦 Controlo de acessos a quase todo o tipo de instalações;
- 🚦 Controlo de acesso aos mais variados meios de transporte;
- 🚦 Localização e identificação de pessoas ou animais;
- 🚦 Nas bibliotecas para controlo dos livros;
- 🚦 Nos meios hospitalares;
- 🚦 Para rastreabilidade de paletes, contentores, entre outros;

Na escolha de qualquer tipo de tecnologia para ajudar a resolver ou dar a melhor resposta a determinado tipo de problema, não é tão relevante a tecnologia em si mesma, mas sim aquela que apresenta maiores possibilidades de endereçar para uma solução mais adequada. Neste sentido, a tecnologia RFID, pode em muitos cenários ser aquela que mais possibilidade oferece, quando comparativamente, com outras similares, apresentando um maior número de vantagens. Para o paradigma deste trabalho, permite uma versatilidade de opções de resolução que não se encontrou em nenhuma outra. Contudo e como qualquer outro tipo de tecnologia, apresenta vantagens e desvantagens que devem ser consideradas. Enumeram-se aqui de forma não exaustiva aquelas que podem ser consideradas as suas principais vantagens e também as suas desvantagens.

Vantagens

- 🚦 Eliminação de erros de escrita e leitura dos dados;
- 🚦 O armazenamento de dados é realizado de forma automática e com enorme rapidez;
- 🚦 Sem necessidade de aproximação nem campo visual para o reconhecimento dos dados;
- 🚦 Capacidade de correto funcionamento em ambientes hostis;
- 🚦 Possibilidade de escrita de dados nas etiquetas;
- 🚦 Grande tempo de duração das etiquetas com possibilidades de reutilização;
- 🚦 Prevenção de roubos e falsificação de mercadorias;
- 🚦 Segurança e confidencialidade através da possibilidade de encriptação da informação trocada;

Desvantagens

- 🚦 O custo ainda elevado desta tecnologia quando comparado com os sistemas de código de barras;
- 🚦 O uso de materiais metálicos e condutivos pode afetar o alcance de transmissão das antenas. Como a operação é baseada em campos eletromagnéticos, o metal pode interferir negativamente no desempenho.
- 🚦 A ausência de padrões e normas a nível mundial das frequências utilizadas para que os produtos possam ser lidos por toda a indústria uniformemente. Algumas das etiquetas produzidas por determinados fabricantes só podem ser lidas por equipamentos destes;
- 🚦 Invasão da privacidade dos consumidores por causa da monitorização das etiquetas coladas nos produtos.

2.2.7. Conclusão

Verifica-se que existe ainda algum trabalho a realizar por parte das entidades reguladoras para que este tipo de tecnologia possa evoluir num sentido positivo e que possa ser utilizado em grande escala permitindo competir a preços de mercado de igual com outras tecnologias. Por outro lado a tecnologia RFID, pela sua multiplicidade de suportes, fiabilidade, facilidade de integração, potencial de crescimento tecnológico e preço acessível perfila-se como uma muito interessante solução de recolha automática de dados.

No caso concreto da atividade de logística postal esta tecnologia apresenta inúmeras possibilidades para desenvolver um sistema de rastreabilidade e controlo de contentores de correio que deve ser explorado. O trabalho desenvolvido neste projeto procura pensar e encontrar uma solução boa para cada um dos vários cenários que são a realidade do processo de logística nos correios.

Para os contentores que circulam apenas entre edifícios da empresa existe a alternativa de colocar leitores nos locais de passagem (interiores e nos locais de saída/entrada) dos contentores. Permite assim, efetuar as leituras sempre que estes circulam e entram ou saem dos edifícios e desta forma possibilitar conhecer a sua permanência, ausência, entrada ou saída dos respetivos edifícios. Em casos de edifícios de dimensões maiores ou em que se pretenda efetuar uma rastreabilidade mais detalhada dever-se-á colocar leitores em zonas de passagem que não sejam zonas de entrada ou saída. Pode também ser equacionada a colocação de um leitor nas viaturas da empresa como os que foram utilizados no projeto. Esta abordagem pode reduzir a quantidade de equipamentos a adquirir. Para o caso das entregas e recolhas nos grandes clientes e instalando um programa de recolha de dados GPS consegue-se saber a localização das entregas e recolhas. Deve entender-se aqui como pressuposto o carregamento prévio numa base de dados das coordenadas GPS de cada cliente.

Podem ainda os correios rever e sugerir alterações aos contratos com os clientes, prevendo uma evolução das suas relações com estes numa base de satisfação e qualidade de serviço, sugerindo a colocação de etiquetas RFID nas suas instalações. Estas poderiam ser colocadas nas zonas de entrega e recolha possibilitando a leitura pelos leitores colocados nas viaturas. Esta relação pode ser complementada com a prestação de informação mais detalhada aos clientes do processo de circulação das remessas destes dentro da rede dos correios. Permite assim, desencadear processos de respeito mútuo e maior confiança pela qualidade de serviço prestado.

Nos centros de produção e logística, poderá ser atribuído um destino a cada contentor/etiqueta através de uma aplicação informática antes da sua dispersão. Em caso de colocação numa determinada viatura que não a correta seria enviado um alerta na forma de som ou mostrado num ecrã junto ao cais a informação do encaminhamento errado. Esta funcionalidade permitiria melhorar a qualidade de serviço e otimizar o trabalho com ganhos de produtividade e redução de custos.

Sistema Integrado de Localização de Contentores de Correio

Em suma, a solução para o problema da rastreabilidade dos contentores de correio pode encontrar através da tecnologia RFID uma ou mais soluções que endereçam para uma otimização de custos e melhoria dos processos organizacionais assim como nos processos da relação da empresa com os seus clientes.

CAPÍTULO 3

3. Análise de Requisitos

Tendo por base o adequado processo de desenvolvimento de programas informáticos, uma das fases mais importantes em termos de projeto, é a fase de análise, que inclui a identificação detalhada das funcionalidades do sistema (levantamento de requisitos) e a respetiva descrição (especificação do sistema) de modo a que os mesmos requisitos possam ser validados pelos utilizadores finais do sistema [7]. Nesse sentido e sem um processo formal de análise detalhada junto do cliente interno, apresenta-se neste ponto a especificação funcional e não funcional dos requisitos que se subentendeu que o sistema deve incorporar. De referir que alguns destes requisitos, embora tivessem sido identificados, não chegaram a ser implementados.

3.1. Requisitos Funcionais

RF01: O sistema deve permitir obter e registar informação relativamente ao rastreio dos contentores que transportam o correio dentro de toda a rede de distribuição e nomeadamente nos processos de entrega e recolha junto dos grandes clientes.

RF02: O sistema deve registar a informação sobre os diversos locais de permanência, saída e entrada de contentores.

RF03: Os leitores devem disponibilizar um conjunto de serviços que permitam receber comandos para efetuar operações sobre os mesmos por um utilizador com perfil de administrador.

RF04: Os leitores devem filtrar a informação recebida das etiquetas por forma a excluir a que é redundante e comunicar apenas aquela que é importante.

RF05: O sistema deve permitir visualizar a informação recolhida aplicando filtros (e.g. por local ou data) e possibilidade de impressão de relatórios ou com extração para o formato Excel.

RF06: O sistema deve disponibilizar um módulo de alertas para comunicação de situações anómalas que possam ocorrer e passíveis de ser visualizados na aplicação.

RF07: O sistema deve permitir a impressão de guias de remessa em formato relatório com a informação dos contentores entregues por cliente, data / hora e local de entrega.

RF08: O sistema deve publicar um conjunto de serviços que permita outras aplicações subscrever para receção e envio de informação.

RF09: O sistema deve conter áreas de acesso diferenciadas por perfil de utilizador.

3.2. Requisitos Não Funcionais

RNF01: O sistema deve manter-se operacional 24x7, com garantia de 99% de disponibilidade.

RNF03: Apesar de não se tratar de um sistema crítico para o negócio é fundamental a definição de uma política de cópias de segurança da base de dados e sistema de ficheiros.

RNF04: O sistema deve ser intuitivo e simples de utilizar pelos utilizadores.

3.3. Visão Geral do Sistema

O sistema deve permitir obter e registar informação relativamente aos movimentos dos contentores que transportam o correio dentro de toda a rede de distribuição.

A informação deve ser registada sobre os diversos locais de permanência, saída e entrada de contentores e permitir perceber qual o sentido de trânsito dos contentores. Deve ainda permitir saber quais os destinos e origem nos seus movimentos logísticos.

Os leitores devem disponibilizar um conjunto de serviços que permitam receber comandos para efetuar configurações e ou operações nos mesmos por um utilizador com perfil de administrador. Devem filtrar a informação recebida das etiquetas por forma a comunicar apenas aquela que é relevante.

Deve permitir aos utilizadores a visualização da informação recolhida aplicando filtros (e.g. por local ou data, sentido de trânsito dos contentores) e possibilidade de impressão de relatórios ou extração para Excel ou PDF.

O sistema deve disponibilizar um módulo de alertas para registo e visualização de situações anómalas que possam ocorrer.

Deve disponibilizar um interface com o utilizador através de um portal na intranet da empresa.

Deve disponibilizar um conjunto de serviços que permita outras aplicações subscrever para receção e envio de informação.

O sistema deve registar sob a forma de log em ficheiro ou base de dados os eventos ocorridos no mesmo, sejam eles de informação, erro ou chamada de atenção.

3.4. Casos de Utilização

Os casos de utilização permitem a especificação dos requisitos funcionais segundo uma aproximação focada primordialmente nos utilizadores do sistema. Um caso de utilização é uma sequência de ações que um ou mais atores realizam num sistema de modo a obterem um resultado particular [7].

Sistema Integrado de Localização de Contentores de Correio

Dentro dos casos de utilização, um cenário, é uma determinada sequência de ações que ilustra o comportamento de um sistema [7].

No contexto deste projeto, e seguindo a modelação UML, apresentam-se de seguida alguns casos de utilização do sistema, numa perspetiva de possibilitar uma melhor compreensão do mesmo. Contudo, os mesmos, não refletem uma captura de requisitos exaustiva, focando apenas alguns aspetos do sistema e que se descrevem de seguida.

3.4.1. Registar Comunicação da Etiqueta

Nome: Registar Comunicação da Etiqueta

Cenário Principal

O caso de utilização inicia quando um contentor se aproxima da zona de leitura do leitor. O leitor recebe a comunicação enviada pela etiqueta colocada no contentor e coloca essa informação numa estrutura em memória. A informação obtida na comunicação é enviada para o servidor aplicacional para ser registada na base de dados como um evento de primeira comunicação com a informação da data e hora da comunicação, identificador da etiqueta, estado da pilha, identificador do leitor que envia a mensagem.

Cenário Alternativo 1 (Comunicações repetidas)

Nas comunicações seguintes (2 em 2 segundos), dado que a informação já existe na estrutura em memória apenas é atualizada a data e hora da comunicação nessa estrutura em memória sem efetuar qualquer registo na base de dados.

Cenário Alternativo 2 (Sem comunicação após 10 minutos)

O caso de utilização inicia quando um leitor deixa de receber comunicação da etiqueta colocada no contentor. Após um período parametrizável de 10 minutos sem comunicação é registado um evento de última comunicação. Nesta ação é enviada a informação para o servidor aplicacional para ser registada na base de dados como um evento de última comunicação com a última informação constante da estrutura em memória relativa à etiqueta, data e hora da comunicação e identificador do leitor que envia a mensagem.

3.4.2. Consultar Entrada Contentor

Nome: Consultar Entrada Contentor

Cenário Principal

O caso de utilização inicia quando o utilizador pretende consultar a entrada de contentores num determinado local. Por consulta à base de dados são verificadas as primeiras comunicações por cada leitor

e por contentor desse mesmo local para um determinado dia. É devolvida a lista de registos das primeiras comunicações dos contentores nesse determinado lugar uma vez que podem entrar e sair mais do que uma vez por dia.

3.4.3. Consultar Permanência Contentor

Nome: Consultar Permanência Contentor

Cenário Principal

O caso de utilização inicia quando o utilizador pretende consultar a permanência de contentores num determinado local. Por consulta à base de dados são verificadas as últimas comunicações por cada leitor desse mesmo local. Caso a última comunicação tenha sido registada por um leitor no espaço interior é assumido a permanência do contentor nas instalações.

Cenário Alternativo 1 (Espaços físicos com 1 leitor)

Para os espaços físicos mais reduzidos em que existe apenas 1 leitor é verificado se existe uma última comunicação para aquele local. Caso não exista esse registo é assumida a permanência do contentor nas instalações.

Cenário Alternativo 2 (Espaços de Clientes)

Para consulta aos contentores que permanecem nos clientes, é efetuada pesquisa à base de dados para as comunicações realizadas pelos leitores colocados nas viaturas em que para a data e hora da última comunicação de determinado contentor e determinada coordenada geográfica (associada a determinado cliente) não exista registo de comunicação posterior por leitores colocados em instalações da empresa ou em viaturas. É devolvida uma listagem.

3.4.4. Consultar Saída Contentor

Nome: Consultar Saída Contentor

Cenário Principal

O caso de utilização inicia quando o utilizador pretende consultar as saídas de um determinado local. Por consulta à base de dados são verificadas as últimas comunicações por cada leitor desse mesmo local.

Cenário Alternativo 1 (Espaços físicos com n leitores)

É efetuada consulta à base de dados para obter as últimas comunicações efetuadas e registadas pelos leitores colocados nas portas de saída/entrada e devolvida a listagem ao utilizador.

Cenário Alternativo 2 (Espaços de Clientes)

É efetuada consulta à base de dados para obtenção dos registos em que se verifique; última comunicação de registo de leitura por um leitor colocado na viatura para um determinado cliente/área geográfica e data mais o par de primeira e última comunicação efetuada por um leitor colocada na viatura para uma data posterior.

3.5. Definição das Mensagens

As mensagens enviadas das etiquetas para o leitor, foram definidas com 4 bytes de dimensão, tendo em conta que se pretendia um tempo de comunicação curto, para que a probabilidade de ocorrência de colisões, fosse de encontro ao método que se pretendia demonstrar. Com estas opções apenas será possível distinguir 256 etiquetas, situação que é suficiente para o protótipo que se quer implementar.

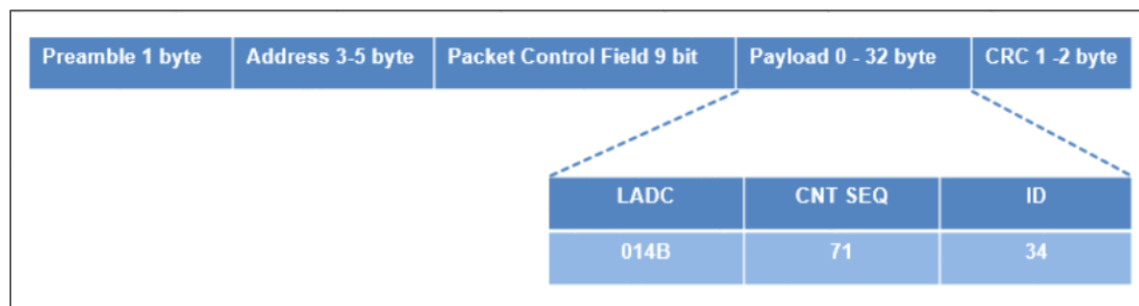


Figura 4 - Pacote de dados enviados pela etiqueta

O formato do pacote de dados transmitido pelo rádio *modem* implementado a partir do chip nRF24L01+ é descrito no capítulo 4. O *payload* do pacote é onde vai a mensagem que foi definida e a qual é composta da seguinte forma:

- ✚ Os dois primeiros bytes contêm a informação da tensão da pilha;
- ✚ O byte seguinte contém a informação do contador de sequência;
- ✚ O último byte contém a informação com o identificador único da etiqueta.

No entanto a esta mensagem e após ser efetuada a sua leitura, seriam ainda adicionados pelo leitor outros atributos, como o tipo de etiqueta, o código de barras da etiqueta que se assumiu com igual valor do identificador e a hora e data da mensagem. De referir que esta mensagem está representada pela classe *TagObject* que se encontra na biblioteca comum aos projetos – *ContentoresRFIDShared.jar* com os respetivos métodos de obtenção e atribuição de valores.

Por seu lado, a mensagem final a enviar para o servidor, é representada pela classe *MessageStagingObject* também incluída na biblioteca comum, já referida, para além dos próprios atributos, define também um atributo do tipo *TagObject*, com a informação enviada pela etiqueta e outro do tipo *ReaderObject*. A classe *ReaderObject* também pertencente à biblioteca comum representa o leitor adicionando assim mais alguns atributos deste e incorporando-os à mensagem final que se julgou ser relevante. A mensagem final ficou estruturada com a representação em XML como mostra a figura 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageStagingObject>
  <MessageDate>Jan 28, 2016 3:15:52 PM</MessageDate>
  <MessageStatus>Entrada</MessageStatus>
  <TagObject>
    <TagType>Transponder ISO CTT</TagType>
    <TagID>51</TagID>
    <TagBaterlyLive>3.1352098765432097</TagBaterlyLive>
    <TagBarCode>51</TagBarCode>
    <TagNumSequential>15</TagNumSequential>
    <TimeStamp>Jan 28, 2016 3:15:52 PM</TimeStamp>
    <TagNumSequential></TagNumSequential>
  </TagObject>
  <ReaderObject>
    <ReaderIdentifier>RaspBerryPiCPLS</ReaderIdentifier>
    <ReaderIPAdress>10.0.67.5</ReaderIPAdress>
    <ReaderHostName>RaspBerryPiCPLS</ReaderHostName>
  </ReaderObject>
</MessageStagingObject>
```

Figura 5 - Mensagem XML de evento de Entrada

3.5.1. Diagrama de Troca de Mensagens

A troca de mensagens entre o leitor, o servidor aplicacional e a base de dados é realizada sempre que se verificam as seguintes condições:

Trata-se da primeira mensagem – gera um evento de entrada, de uma determinada etiqueta, rececionada pelo leitor antes de qualquer comunicação dessa mesma etiqueta ou na primeira comunicação, após um período de 10 minutos assumido, sem comunicação da etiqueta;

Trata-se da última mensagem – gera um evento de saída, o *timestamp* da mensagem é atualizado, a cada comunicação na tabela de *hash*, para todas as comunicações dessa etiqueta, e após um período de 10 minutos sem comunicar é enviada a mensagem em memória com o último *timestamp* registado.

Na figura 6, apresenta-se a troca de mensagens de alto nível efetuada entre a etiqueta, leitor, servidor aplicacional e o envio destas para a Base de Dados e respetivo retorno do identificador da mensagem – gerado por uma sequência na base de dados. De referir, aqui, que caso ocorra algum erro de comunicação com o servidor aplicacional ou com a base de dados, é gerada uma exceção e o identificador da transação que foi inicializado a zero, permanece com esse valor e determina a ação seguinte que será manter na tabela de *hash*, a informação do registo da etiqueta e voltar a enviar na próxima iteração. Neste processo, o ideal seria registar a ocorrência e proceder ao envio de uma mensagem de correio eletrónico ou SMS ao administrador da aplicação a informar para que fossem tomadas as devidas diligências, no entanto esta funcionalidade não foi contemplada no âmbito deste trabalho.

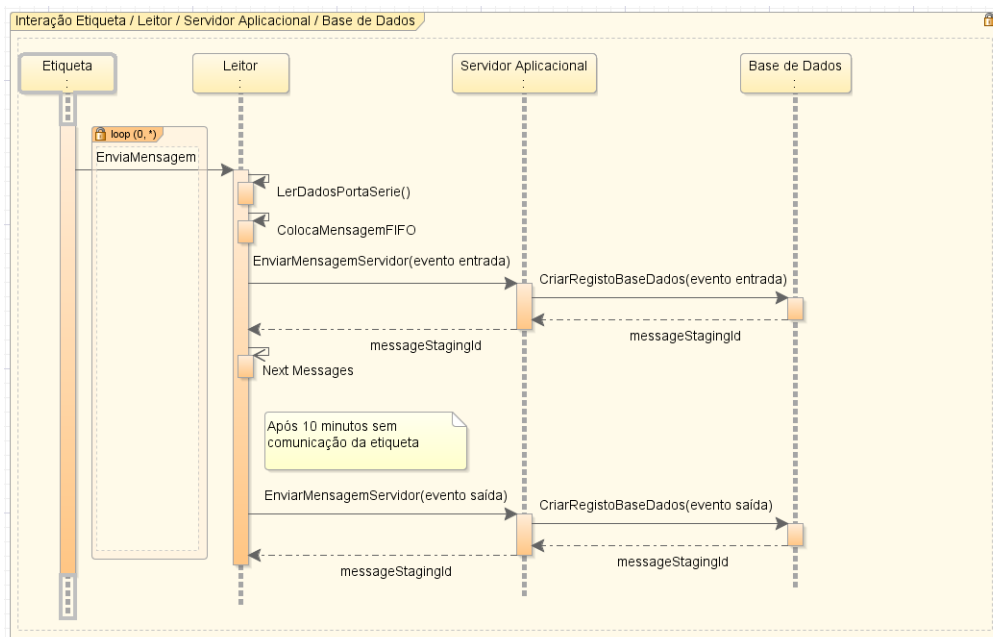


Figura 6 - Diagrama de alto nível de troca de mensagens

3.6. Arquitetura do Sistema

Em termos de arquitetura física de um sistema RFID, a mesma pode variar bastante em função do propósito da sua aplicação, no caso de estudo realizado a mesma assumiu a tipologia representada na figura 7.

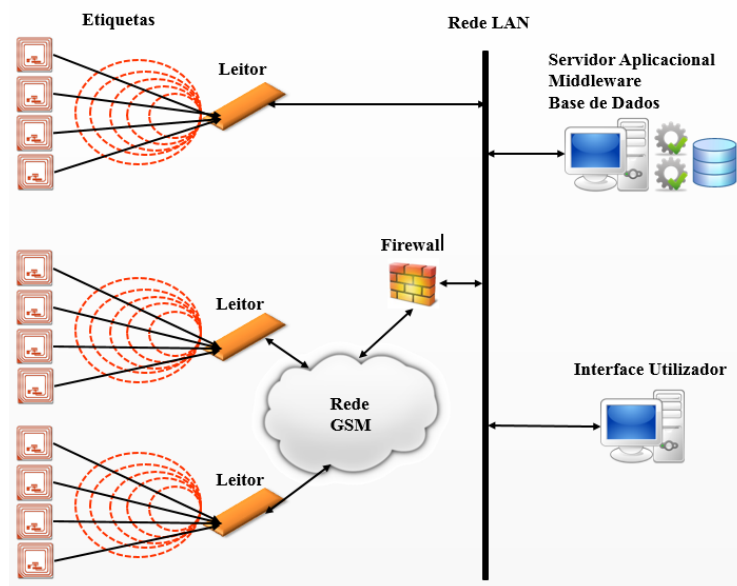


Figura 7 - Arquitetura física do protótipo

A arquitetura física necessária à implementação do protótipo foi simplificada nos aspetos da componente servidores, leitores e etiquetas por motivos custo e duração da prova de conceito. Neste trabalho, não

foram avaliados alguns aspetos associados à tecnologia e que importa referir, tais como a existência de água, metal, iluminação fluorescente, máquinas de grande porte, e frequências concorrentes (outras ondas de rádio) pois podem afetar o desempenho do sistema. A melhor maneira de maximizar o alcance de leitura é observar as várias formas possíveis de interferência e tentar que as mesmas sejam mitigadas. Neste sentido, para obter o melhor desempenho possível, será necessário a realização de um projeto de âmbito mais alargado com a realização de testes pertinentes ao sistema de forma que se compreenda quais as melhores opções a tomar.

3.7. Diagrama de Instalação do Protótipo

Apresenta-se neste ponto o diagrama de instalação da totalidade do sistema (figura 8).

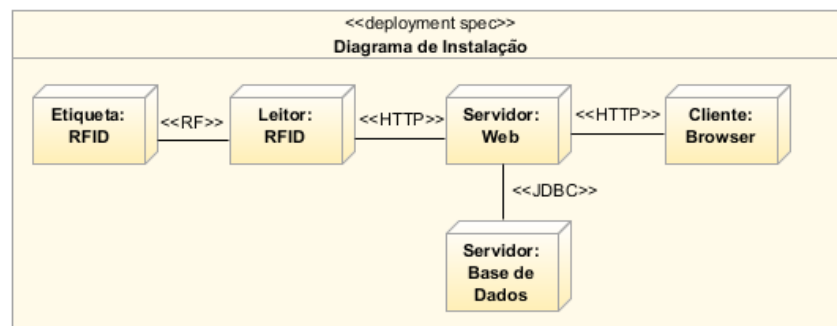


Figura 8 - Diagrama de instalação da solução

CAPÍTULO 4

4. Desenho e Modelação do Sistema

Na fase de escolha da tecnologia a utilizar era objetivo que a mesma não fosse proprietária pelos custos associados ao licenciamento e também permitisse alguma liberdade de programar os componentes que iriam ser utilizados. Efetuada consulta na internet sobre fabricantes de *hardware* e *software* na área do RFID, encontrou-se um projeto - “*OpenBeacon Active RFID Project*¹⁴” de tecnologia de código aberto e com custos bastante acessíveis quer para as etiquetas, quer para os leitores. No entanto no momento em que se pretendeu adquirir os equipamentos não existia disponibilidade na loja. Dado ser necessário avançar com o projeto, optou-se por desenvolver internamente no ISEL tanto as etiquetas como os leitores adquirindo apenas os módulos de rádio.

Quanto ao *hardware* a utilizar optou-se por adquirir três *Raspberry Pi* - um mini computador com um sistema operativo baseado em *Linux*. Neste *hardware* foram instalados os programas desenvolvidos em linguagem de programação java que rececionam as mensagens transmitidas pelas etiquetas e outros programas necessários ao projeto.

Ao nível das comunicações de dados entre os *Raspberry Pi* e o servidor, optou-se por definir dois cenários, um em que teríamos uma comunicação por cabo (Ethernet – cabo RJ45) ligado diretamente à rede da empresa e uma outra ligação, sem fios, através de um dispositivo GSM USB ligado a uma APN de uma das empresas do grupo.

Para simplificação dos leitores e permitir leituras a distâncias de mais de 10 m entre etiquetas e leitores, optou-se por etiquetas ativas que transmitem a intervalos regulares (2 segundos no nosso caso) uma mensagem com a sua identificação (*beacon*). Por se tratar de um *hardware* muito simples e também para se poder economizar a energia das pilhas da etiqueta, estas não implementam nenhum protocolo de comunicação no sentido de evitar ou minimizar colisões entre as mensagens.

Recorrendo-se à modelação estatística introduzida pelo sistema [8], concluiu-se que dadas as características das mensagens transmitidas, existe apenas uma pequena probabilidade de colisão, sendo a solução adotada viável.

4.1. Sistema ALOHA

O sistema ALOHA [8] foi proposto pela primeira vez nos EUA na década de 70 pela universidade do Hawaii sobre o uso das comunicações rádio entre computadores.

¹⁴ Página da internet <http://www.openbeacon.org/>

Com o projeto do sistema O ALOHA pretendeu-se averiguar sobre a possibilidade de utilizar as comunicações rádio entre computadores como alternativa às redes convencionais de ligação por fio para utilizadores interativos num sistema em rede. É analisado o método de acesso aleatório ao meio de comunicação e demonstrado que o número máximo de utilizadores interativos que o sistema suporta é cerca de 160 [8]. Tem como princípio a transmissão da informação por uma estação sempre que tem algo para enviar. Sempre que é detetada uma colisão, a estação retransmite a informação após esperar durante um período de tempo aleatório.

Uma mensagem transmitida pode ser recebida de forma incorreta por causa de dois tipos diferentes de erros; (1) erros de ruído aleatório e (2) erros causados por interferência com uma mensagem transmitida por um outro utilizador. O primeiro tipo de erro não foi considerado que fosse um problema sério. O segundo tipo de erro, que é causado por interferência, tem apenas importância quando um grande número de utilizadores está a tentar utilizar o canal rádio ao mesmo tempo. Os erros de interferência vão limitar o número de utilizadores e a quantidade de dados que pode ser transmitida neste canal de acesso aleatório. A figura 9 indica a sequência das mensagens transmitidas por κ utilizadores ativos no acesso aleatório ao canal de comunicação no sistema ALOHA.

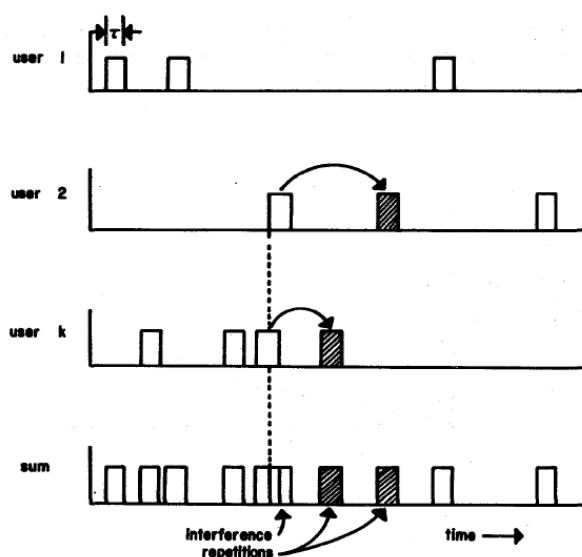


Figura 9 - Multiplexagem de comunicação ALOHA

Para fins de análise foi feita a suposição pessimista de que quando uma sobreposição ocorre ambos as mensagens são recebidas com erro e, por conseguinte, ambas são retransmitidas. Assim com o aumento do número de utilizadores também aumenta o número de interferências e, portanto, o número de retransmissões aumenta até que o canal satura. O estudo visa calcular o número médio de utilizadores ativos que podem ser suportados pelo esquema de transmissão acima descrito. De notar que o esquema de comunicação de acesso aleatório a um canal rádio, apresenta a dificuldade adicional em detetar antecipadamente as colisões de comunicação. Usando o canal de rádio, como se descreve, cada utilizador pode aceder ao mesmo canal mesmo que se encontrem geograficamente dispersos. O método de

comunicação de acesso aleatório utilizado no sistema ALOHA pode, assim, ser pensado como uma forma de concentração de dados por utilizadores geograficamente dispersos.

Foi definido um processo de acesso aleatório para cada um dos utilizadores ativos k , concentrando a atenção no início da transmissão das mensagens enviadas por cada utilizador. Refere o autor que é útil fazer uma distinção entre a transmissão de uma dada mensagem, por um dado utilizador, para a primeira vez que transmite e as mensagens enviadas repetidas. Como tal designa, as mensagens do primeiro tipo, como mensagens e as do segundo tipo como repetições. Designando por λ a taxa média por segundo de ocorrência de mensagens de um único utilizador ativo, e assumindo que esta taxa é idêntica de utilizador para utilizador, então a taxa média de ocorrência de mensagens enviadas por todos os utilizadores ativos num segundo é dada pela equação (1).

$$r = \kappa \lambda \quad (1)$$

Onde r dá a média do número de mensagens por unidade de tempo de κ utilizadores ativos. Designando por τ a duração de cada mensagem, então para uma unidade de tempo, obtém-se a relação dada pela equação (2) designada por utilização do canal:

$$r\tau = 1 \quad (2)$$

e que corresponde a preencher completamente todo o espaço livre numa unidade de tempo do canal com mensagens sem colisões. Sendo τ fixo, resta saber qual o valor de r que leva a esta utilização máxima do canal.

Definindo R como o número médio de mensagens, incluindo as retransmissões, por unidade de tempo de κ utilizadores ativos, então, caso existam algumas retransmissões deve ter-se $R > r$. Foi definido $R\tau$ como o tráfego do canal e esta quantidade representa a média do número de mensagens e retransmissões por unidade de tempo multiplicado pela duração de cada mensagem ou retransmissão.

A hipótese da probabilidade exponencial de que não haverá eventos (início de mensagem ou retransmissões) em um intervalo de tempo T é dada pela equação (3):

$$\exp(-RT) \quad (3)$$

Usando esta assunção, pode-se calcular a probabilidade para que uma dada mensagem tenha de ser retransmitida devido a interferência com uma outra mensagem ou retransmissão. A primeira mensagem enviada vai sobrepor-se com uma outra, se existir pelo menos uma outra mensagem que tenha sido transmitida no intervalo $[(t - \tau), (t + \tau)]$, relativamente ao ponto de partida, sendo t o início da mensagem enviada. Portanto $T = 2\tau$ e por isso, a probabilidade de colisão P_{col} é dada pela equação (4):

$$P_{col} = [1 - \exp(-2R\tau)] \quad (4)$$

Após algum trabalho algébrico apresentado em [9] pode-se relacionar a utilização do canal $r\tau$ com o tráfego do canal $R\tau$ através da equação (5):

$$r\tau = R\tau \exp(-2R\tau) \quad (5)$$

Analisando-se a equação (5) constata-se que para um $r\tau = 0,186$ obtém-se $R\tau = 0,5$ e que também corresponde ao seu valor máximo. Conclui-se que o acesso ALOHA não é muito eficiente para comunicação de dados, pois o melhor que se pode obter será menos de 18,6% de eficiência de utilização do canal. Por essa razão, atualmente este método é usado apenas como estudo académico fundador das raízes dos que se sucederam. Em 4.2 usaremos esta base para estabelecer o modelo de caracterização das colisões das mensagens enviadas pelas etiquetas.

4.2. Caracterização das Colisões das Mensagens

O modelo estatístico do ALOHA fornece as bases suficientes para podermos estabelecer o modelo estatístico das etiquetas RFID emissoras de mensagens periódicas. Uma vez que apenas existe transmissão de mensagens sem necessidade de confirmação de receção (esta consegue-se através da repetição periódica), apenas nos interessa saber se para uma dada concentração de etiquetas, existem colisões em número tal que impeçam os leitores de identificarem as diversas etiquetas em tempo útil.

Como não existem repetições, tem-se que $R = r$ e ainda substituindo r pela equação (1), a equação (4) toma a forma da equação (6):

$$P_{col} = [1 - \exp(-2\kappa \lambda \tau)] \quad (6)$$

que nos dá a probabilidade de pelo menos uma colisão durante uma transmissão. Assume-se que as etiquetas enviam as mensagens a intervalos regulares mas todas desfasadas umas das outras aleatoriamente.

Para o caso deste trabalho, considera-se que:

- Na vizinhança de um leitor, poderão encontrar-se no máximo 100 etiquetas ($k = 100$),
- Cada etiqueta transmite a sua mensagem de 2 em 2 segundos ($\lambda = 0,5 \text{ msgs/s}$),
- A duração de uma transmissão é de $105 \mu\text{s}$ ($\tau = 0,000105 \text{ s}$).

Substituindo os valores acima na equação (6) obtém-se $P_{col} = 0,0104$, que corresponde a uma percentagem de 1,04% em que uma etiqueta transmite em colisão com outra. Tendo também em conta a independência estatística, quando ao fim de 2 segundos retransmitir a mensagem a probabilidade de colisão mantêm-se. Tem-se assim que duas colisões seguidas têm uma percentagem de ocorrência de 0,011% e para três tem-se 0,0001%. Estes resultados permitem-nos esperar que ao fim de 6 segundos (3 transmissões) todas as 100 etiquetas devem ter sido detetadas pelo leitor. Estes resultados devem-se ao fato de se transmitirem mensagens muito pequenas, minimizando-se assim o tempo de exposição à colisão. Claro que na prática tal valor será difícil de se verificar, pois existem muitos outros fenómenos, tais como o ruído e também as colisões com outros equipamentos que podem poluir o canal de comunicação e que não entraram neste modelo. Contudo este resultado é encorajador, pois fornece uma ordem de grandeza da percentagem de colisões que se poderá obter num ensaio prático que utilize estes parâmetros.

Por forma a verificar a probabilidade das colisões para um número de etiquetas variável, assumindo o tempo de transmissão igual para cada uma e o intervalo de tempo a que cada etiqueta transmite,

substituíram-se os valores de k na equação (6) do qual resulta gráfico da figura 10. Iniciou-se o cálculo de probabilidade colisão com 10 etiquetas a transmitir simultaneamente na presença do leitor e aumentou-se de 10 em 10 até à presença de 350 etiquetas.

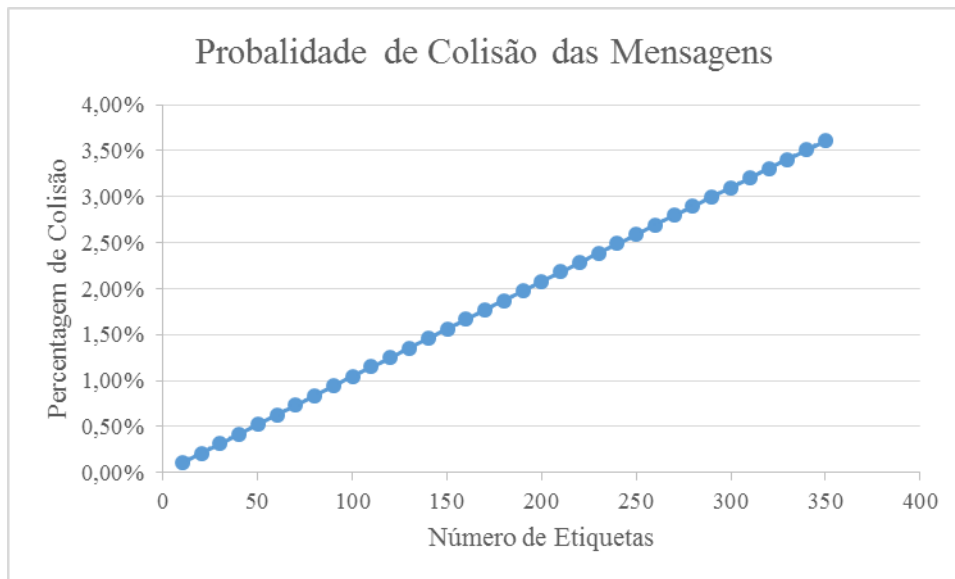


Figura 10 - Probabilidade de colisão das mensagens

Verifica-se que mesmo para um número elevado de etiquetas a probabilidade de ocorrência de colisões é bastante baixa, sendo que no caso dos contentores, a concentração em torno de um leitor no mesmo momento não deve ser superior a 100.

4.3. Desenho das Etiquetas

As etiquetas, e como já referenciado anteriormente, foram desenvolvidas internamente no ISEL. São constituídas por um circuito eletrónico que por sua vez se encontra ligado a uma fonte de alimentação com duas pilhas de tamanho AAA (figura 11).

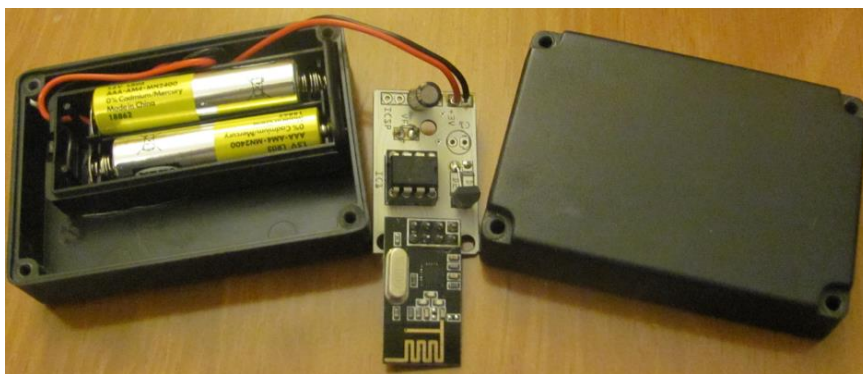


Figura 11 - Etiquetas e fonte de alimentação

Sistema Integrado de Localização de Contentores de Correio

A opção de instalar as pilhas na etiqueta partiu do princípio que um contentor pode ter um tempo de vida útil de mais ou menos dez anos e dado que iríamos utilizar etiquetas ativas. Pretendeu-se encontrar uma solução que por um lado permitisse demonstrar o algoritmo ALOHA e por outro perceber o gasto de energia das pilhas colocadas em cada contentor. Desta forma, permitir-nos-ia saber em que momento se devia trocar as pilhas de cada contentor para manter o sistema funcional. Para a prova de conceito, o desenho das etiquetas foi pensado para um cenário de presença de 100 contentores/etiquetas a comunicar simultaneamente. Por fim, foram colocadas numa caixa de plástico duro para proteção (figura 12).



Figura 12 - Etiqueta com caixa protetora

4.3.1. Montagem da etiqueta

A etiqueta é conseguida através da ligação de um módulo OEM de rádio frequência que opera na gama dos 2.4-2.5GHz e que utiliza o circuito integrado nRF24L01+ como rádio *modem*. Este módulo integra o circuito integrado, a eletrónica de acoplamento e antena em circuito impresso. Por se tratar de um produto OEM, foi adquirido no mercado com facilidade (figura 13).

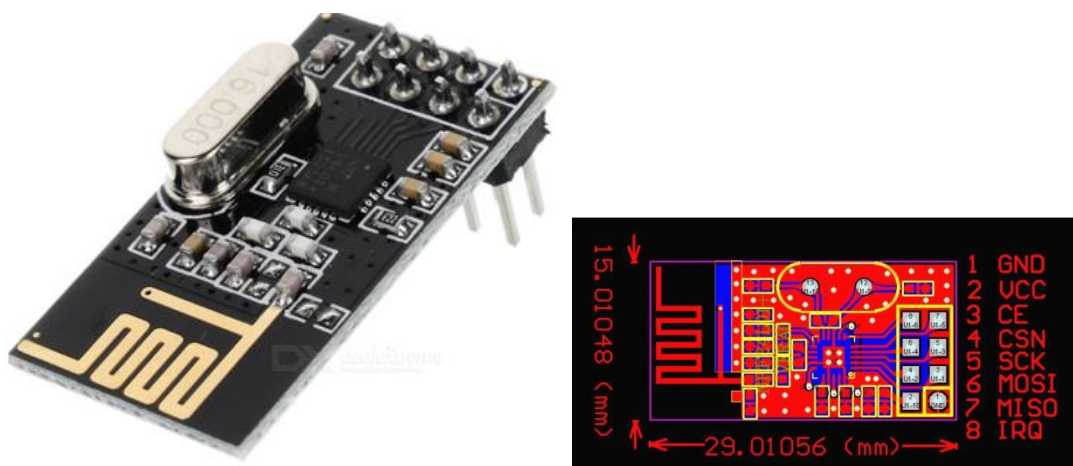


Figura 13 - NRF24L01 2.4GHz Wireless Transceiver

O NRF24L01 é um módulo transceptor 2.4GHz de baixo consumo de energia e baixo custo. É ideal para muitas aplicações que requerem comunicações sem fio. Pode ser configurado para taxas de transmissão de 2Mbps, 1Mbps e 250Kbps e ao ar livre, a distância de comunicação pode chegar até 100 m.

Especificações Técnicas.

- ✚ Tensão: 1,9 a 3,6V (recomendado 3,3V)
- ✚ Potência máxima de saída: 0 dBm
- ✚ Consumo (transmitindo): 11,3 mA (pico)
- ✚ Consumo (recebendo): 13,5 mA (pico)
- ✚ Consumo (*power-down*): 0,9 uA
- ✚ Sensibilidade em modo de receção -64 dBm
- ✚ Taxa de 1MB (área aberta): 100m
- ✚ Velocidade SPI: 0 a 10Mbps
- ✚ Canais: 125
- ✚ Número de pinos: 8
- ✚ Temperatura de trabalho: -40 a 85°C

Para se controlar o módulo NRF24L01 recorreu-se a um microcontrolador PIC12F675. Este microprocessador de 8 bit é adequado para aplicações de muito baixo consumo de energia e disponibiliza todas as funções necessárias para desenvolver o programa de controlo do rádio *modem*.

Para efetuar a ligação do microcontrolador foi desenvolvida uma pequena placa de circuito impresso, cujo circuito lógico é apresentado na figura 14.

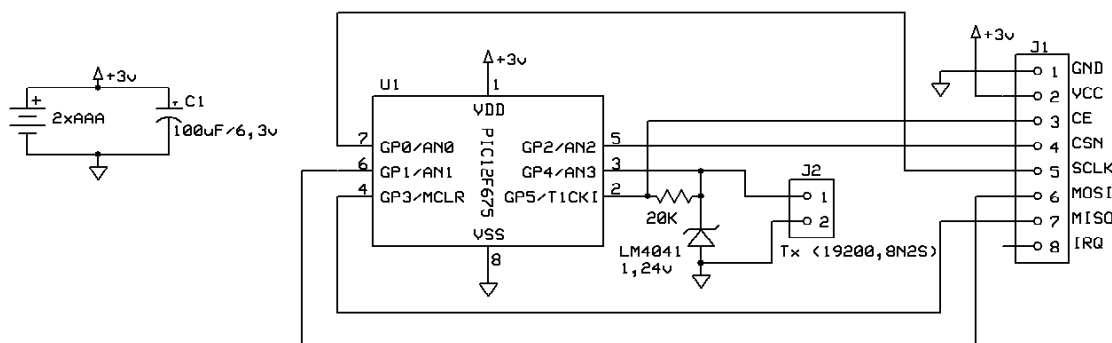


Figura 14 - Esquema eletrónico da etiqueta

Nesse circuito o conector *J1* corresponde à interface de ligação com o módulo NRF24L01, sendo a interface SPI implementada por *software* no PIC12F675. A alimentação é feita por duas baterias tipo AAA ligadas em série e foi também instalada uma referência de tensão LM4041 de 1,24 V ligada a uma das entradas analógicas do PIC12F675. Esta referência de tensão vai permitir, por *software*, medir a tensão de alimentação fornecida ao circuito através da seguinte equação:

$$V_{cc} = 1,24 * 1024 / V_{adc} \quad (7)$$

Onde *V_{adc}* corresponde ao valor lido na ADC a 10 bit do PIC12F675. Como a tensão de entrada da ADC não varia, qualquer variação na tensão da bateria (*V_{cc}*) repercute-se no valor *V_{adc}* lido podendo-se assim calcular o valor de *V_{cc}* através desta fórmula. Uma alternativa a esta solução seria optar por um microcontrolador que tenha esta função implementada.

O conector *J2* vai permitir que o mesmo *hardware* possa funcionar como recetor das mensagens rádio, como iremos ver adiante em 4.4.1.

4.3.2. Programação do controlador da etiqueta

O modo de funcionamento da etiqueta é definido pelo programa gravado no PIC12F675. Este foi desenvolvido usando o ambiente MPLAB IDE V8.46 e usou-se o programador *Picstart Plus* da *Microchip*. As fontes de todo o *software* encontram-se disponíveis no anexo A. O software foi feito de forma a incluir os dois modos de funcionamento: transmissor e recetor de mensagens. No CD de apoio a este relatório, encontra-se o projeto completo. Para se gerar o *software* da transmissão basta no ficheiro “*config.h*” marcar como ativa a macro “*#define _transmitter*” e colocar em comentário a macro “*//#define _receiver*”. Compilando o projeto obtém-se o *software* para a etiqueta.

A mensagem transmitida encontra-se na variável *msg* declarada na função *main*:

```
static char msg[4]={0x34,0x33,0x32,0x30}; //Msg to transmit
```

com as seguintes atribuições:

msg[0] – 2 bit de maior peso do valor da leitura da ADC ;

msg[1] – 8 bit de menor peso do valor da leitura da ADC;

msg[2] – 8 de contador de sequência; por cada vez que envia uma msg, incrementa uma unidade;

msg[3] – 8 bit para identificação da etiqueta; Este valor nunca muda.

O *software* está feito de forma que após a transmissão tanto o microcontrolador como o módulo NRF24L01 ficam em modo de baixo consumo (*low power*) até que expirem os 2 segundos de intervalo das mensagens. Aqui os 2 segundos são aproximados, não havendo necessidade de os garantir sempre iguais e são contados pelo *timer* do *Watchdog* que é comandado por um oscilador RC. Este modo de funcionamento permite uma operação em que o consumo de corrente médio solicitado às baterias é da ordem dos 12μA.

4.4. Desenho dos Leitores

Um leitor de etiquetas é constituído por duas partes: i) Módulo recetor das mensagens rádio e ii) Módulo de processamento local e comunicação com o servidor central. Os requisitos que se colocam a cada um destes módulos são os seguintes:

- *Módulo recetor das mensagens rádio*: Deve ser capaz de receber as mensagens rádio enviadas pelas etiquetas rejeitando todas as que estejam corrompidas; as mensagens recebidas devem ser

disponibilizadas ao módulo de processamento através de uma interface USB que simula uma porta de comunicação série assíncrona. A informação transmitida é constituída pelos 4 bytes da mensagem recebida codificados em hexadecimal e terminado pelos caracteres de controlo CR (13) e LF (10). Assim o *software* de processamento das mensagens recebidas, encara as etiquetas como produtores de linhas de texto com 10 caracteres.

- *Módulo de processamento*: Tendo em conta o ambiente industrial em que vai ser utilizado, o requisito principal coincide com o que se pede a um PC industrial i.e., não necessitar de refrigeração forçada, baixo consumo e robustez mecânica (nº mínimo de fichas). Adicionalmente deve poder executar um sistema operativo que suporte comunicação TCP/IP com ligação por cabo ou sem fios, poder executar uma máquina virtual *Java* e ter capacidade de gestão remota.

Os requisitos postulados permitem que se possa utilizar um PC normal (com *Windows*, *Linux* ou *Macintosh*) como ambiente de desenvolvimento, fazendo-se posteriormente a colocação em exploração do *software* do leitor no hardware selecionado para implementar o leitor.

4.4.1. Modulo recetor de mensagens rádio

A parte de receção de mensagens e garantia de integridade é assegurada pelo mesmo *hardware* utilizado para a implementação das etiquetas, já apresentado em 4.3.1, havendo apenas necessidade de alterar o programa que se instala no PIC12F675 para que este programe o módulo NRF24L01 em modo de receção e faça o encaminhamento das mensagens recebidas para o exterior através de uma porta série. Nesta solução optou-se por utilizar o mesmo microprocessador que se usou na etiqueta apenas por razões de não aumentar o orçamento do projeto e também por se ter entendido que com um pequeno esforço de programação, o PIC12F675 pode suportar a transmissão série a uma velocidade razoável (19.200,8,1,N). O módulo NRF24L01 é colocado em modo receção, usando o endereço 0xE7E7E7E7, que é usado como endereço destino de todas as mensagens enviadas pelas etiquetas. A restante configuração é a seguinte: 4 bytes de dados, 1 Mbps de velocidade de transmissão de dados, 2 byte de CRC e canal 2 de comunicação. No envio de uma mensagem por uma etiqueta não é feita a confirmação de receção; apenas se certifica se o endereço destino é o seu e se o CRC da mensagem não indica erro. O integrado nRF24L01+ que é utilizado pelo módulo NRF24L01, possui internamente um FIFO capaz de armazenar até 3 mensagens assim como todo o processamento de verificação automática de integridade do CRC, descartando as que apresentam erro. Este FIFO é monitorizado pelo microprocessador que executa a leitura dos dados das mensagens entretanto recebidas, codificando-as em hexadecimal junta-lhe os terminadores CR e LF, transmitindo-as pelo canal série (pino 3-GP4).

Para se gerar o programa para o microprocessador desempenhar as funções descritas basta no ficheiro “*config.h*” marcar como ativa a macro “*#define _receiver*” e colocar em comentário a macro “*//#define _transmitter*”. Compilando o projeto obtém-se o *software* para a receção de mensagens.

Finalmente utilizou-se o adaptador de comunicação série para USB apresentado na figura (15) devidamente ligado ao módulo recetor por um cabo com 3 condutores (3,3V, GND e RXD).



Figura 15 - Adaptador de comunicação série para USB

4.4.2. Módulo de processamento

De entre as várias opções de seleção de *hardware* capaz de responder aos requisitos para este módulo, constatou-se que atualmente existem no mercado muitas ofertas de módulos baseados na arquitetura ARM e que também oferecem boas condições de processamento tais como: mais de 500Mbyte de memória Flash, até 1 Gbyte de memória RAM e suporte de cartão SD superiores ou igual a 4Gbytes. Estes sistemas de *hardware* dispõem também de versões do sistema operativo *Linux*, devidamente configurados para aplicações em Sistemas Embebidos.

Considerando o balanceamento entre preço, funcionalidades oferecidas e facilidade de aquisição, decidiu-se por adotar o cartão *Raspberry Pi 2 Modelo B 1GB* para o qual existem várias distribuições de sistemas operativos como por exemplo *Debian Linux* que foi usado.

As principais características deste cartão são:

- CPU: Broadcom BCM2836 ARMv7 Quad-Core a 900 MHz;
- RAM: 1GB SDRAM;
- Armazenamento: MicroSD;
- Portas USB: 4 x USB 2.0;
- Porta LAN: 1 x 10/100Mbps;
- Porta HDMI: 1 x HDMI 1.4 (full size);
- Slot microSD: Sim;
- Áudio: 1 x Audio In/Out;
- GPIO: 40;

Sistema Integrado de Localização de Contentores de Correio

- Máx. Power Draw/voltage: 1.8A @ 5V.

Os módulos de mensagens rádio foram ligados através da entrada USB ao cartão *Raspberry PI* como se pode ver na figura 16.

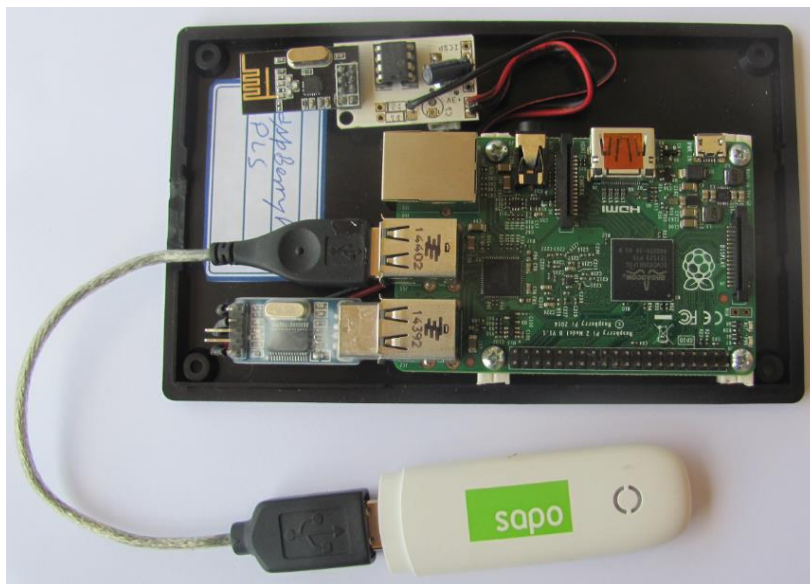


Figura 16 - Ligação USB do módulo de rádio ao *Raspberry*

Estes por sua vez foram montados em caixas de plástico duro às quais foi colada uma estrutura de apoio para os fixar (figura 17).



Figura 17 - Caixa com o leitor de Etiquetas

Adotou-se um cartão micro SD de 8Gbyte com uma distribuição de *Debian Linux* para sistemas embebidos, disponibilizada pelo sitio internet de código aberto www.raspberrypi.org. Esta versão de Linux disponibiliza uma versão de máquina virtual Java Versão 8 e também todas as ferramentas para acesso remoto nomeadamente servidor e cliente SSH e ainda se pode instalar um servidor remoto de acesso remoto ao *desktop*.

Como se sabe, a parte de comunicação série não tem um suporte adequado pelos pacotes *standard* do ambiente Java por um lado e por outro, necessita-se de uma forma de resolver o acesso à porta série para o caso do Java ser executado em ambientes de multiplataforma. Após alguma pesquisa optou-se pela implementação *Java Simple Serial Connector* (jSSC) disponível em <https://github.com/scream3r/java-simple-serial-connector> e que dá suporte aos seguintes sistemas operativos: Windows(x86, x86-64), Linux(x86, x86-64, ARM soft & hard float), Solaris(x86, x86-64), Mac OS X(x86, x86-64, PPC, PPC64).

O hardware do leitor foi testado utilizando o programa java apresentado no anexo C, especialmente desenvolvido para o efeito (função `public static void main(String[] args)` da classe `ReadCom_evt`). Este programa permite testar não só a receção de mensagens das etiquetas como também procede ao seguimento do valor do contador sequencial de mensagem. Sempre que o contador entre duas mensagens consecutivas provenientes da mesma etiqueta difere em mais de uma unidade, assume-se que houve colisão com as suas mensagens. Assim o programa de teste mostra em contínuo numa janela tipo texto uma linha por cada mensagem recebida indicando os valores que esta transporta (tensão da bateria, contador de sequência e nº Identificador) bem como o nº de colisões que já ocorreram com mensagens vindas dessa etiqueta.

A classe `ReadCom_evt` apresentada no anexo, tanto pode ser usada para testar os módulos recetores através da execução da sua função `main()` como pode ser usada como classe utilitária de interface com uma porta série. A sua execução necessita que o módulo de biblioteca `jssc.jar` que se encontra disponível nas diretórias de ambiente de execução da máquina virtual java. Este ficheiro contém no seu interior não só o código java de acesso nativo aos dispositivos de comunicação disponível no sistema operativo hospedeira da máquina virtual, como também as bibliotecas dinâmicas necessárias para dar suporte a esse acesso. O *software* tem capacidade para, no início da instanciação da classe `SerialPort`, identificar o sistema operativo e carregar a biblioteca dinâmica nativa correspondente.

Outro aspeto a mencionar para além do recurso a um FIFO implementado com a classe `ArrayBlockingQueue<String>(100)` diz respeito ao método recetor de eventos gerados pela porta série: `public void serialEvent(SerialPortEvent spe)`. Este método implementa um filtro no sentido de organizar a receção dos caracteres recebidos na porta série de forma que estes sejam armazenados numa *string* por mensagem recebida. A máquina de estados que executa garante esse comportamento e por cada mensagem recebida é depositada numa *string* no *Fifo* de receção. Esta máquina de estado é necessária para garantir que inicialmente existe sincronização com o início de uma mensagem.

CAPÍTULO 5

5. Implementação do Sistema

Neste capítulo descreve-se a implementação prática dos componentes *software* do projeto – classes Java dos diferentes módulos de *software*, criação da base de dados e respetivos objetos, configurações da rede GSM e processo de instalação do *software* e *hardware*.

5.1. Camada de Acesso a Dados

A camada de acesso a dados foi gerida pela *framework* – *Hibernate*¹⁵, por permitir ao programador uma maior abstração ao nível do acesso aos dados e das transações com a base de dados. Esta *framework* é escrita em linguagem java e facilita imenso o mapeamento dos atributos entre uma base de dados relacional e o modelo objeto de uma aplicação, através do uso de ficheiros XML ou anotações Java. *Hibernate* é uma das *frameworks ORM* mais amplamente utilizada na indústria e implementa a persistência em Java (JPA) definida nos *Enterprise JavaBeans (EJB)* na especificação 3.0 [9]. É um *framework* poderoso para o desenvolvimento de aplicações Java, sendo necessário importar as bibliotecas para o *classpath* do projeto. De seguida deve-se criar os ficheiros XML de mapeamento de uma entidade Java para as colunas da tabela na correspondente da base de dados. A figura 18 ilustra o mapeamento no ficheiro XML da entidade *logkeepalive* na base de dados do projeto.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="server.rfid.dao.LogKeepAlive" table="sigcc.Logkeepalive">
<id column="LOGKEEPALIVE_ID" name="id" type="java.lang.Integer">
<generator class="sequence">
<param name="sequence">sigcc.logkeepalive_logkeepalive_id_seq</param>
</generator>
</id>
<property generated="never" lazy="false" name="client_host" type="string">
<column length="100" name="CLIENT_HOST" not-null="true"/>
</property>
<property generated="never" lazy="false" name="client_ip" type="string">
<column length="30" name="CLIENT_IP" not-null="true"/>
</property>
<property generated="never" lazy="false" name="client_time" type="timestamp">
<column length="16" name="CLIENT_TIME" not-null="true"/>
</property>
</class>
</hibernate-mapping>
```

Figura 18 - Ficheiro LogKeepAlive.hbm.xml

A partir da aplicação Java que se está a utilizar, podem ser realizadas as operações CRUD sobre as entidades do modelo de objetos e a *framework* encarrega-se de traduzir o estado do objeto do modelo de

¹⁵ <http://hibernate.org/>

objetos para o modelo relacional. O mapeamento objeto / relacional é automatizado, de forma transparente, para a persistência de objetos numa aplicação Java para as tabelas da base de dados do modelo relacional, utilizando meta dados que descrevem o mapeamento entre os objetos e a base de dados [10][11]. Ao nível dos projetos que implementam a camada de acesso a dados – SIGCC e *ContentoresRFIDMiddleware*, foi necessário criar o ficheiro de configuração para utilização da *framework* *Hibernate*, no qual foi indicada qual a base de dados a utilizar assim como as credenciais de acesso à mesma (figura 19).

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory name="">
    <!-- hibernate dialect -->
    <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
    <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
    <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/SIGCC</property>
    <property name="hibernate.connection.username">postgres</property>
    <property name="hibernate.connection.password">#####</property>
    <property name="transaction.factory_class">
      org.hibernate.transaction.JDBCTransactionFactory
    </property>
    <!-- Simple memory-only cache -->
    <property name="hibernate.cache.provider_class">
      org.hibernate.cache.HashtableCacheProvider
    </property>
    <property name="hibernate.show_sql">true</property>
    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>
    <!-- #####
# mapping files with external dependencies #
##### -->
  </session-factory>
</hibernate-configuration>
```

Figura 19 - Ficheiro hibernate.cfg.xml

5.2. Módulos de Software

Os módulos de *software* produzidos, agregam as classes em pacotes de acordo com as funcionalidades que implementam. Basicamente, foram definidos quatro projetos e que podem ser vistos no Eclipse IDE como pastas: *ContentoresRFIDShared*, *ContentoresClientKeepAlive*, *ContentoresRFIDMiddleware* e *ContentoresRFIDReader*. Nas secções seguintes, efetua-se a descrição de cada um dos projetos e o detalhe da sua implementação referindo os aspetos considerados mais relevantes.

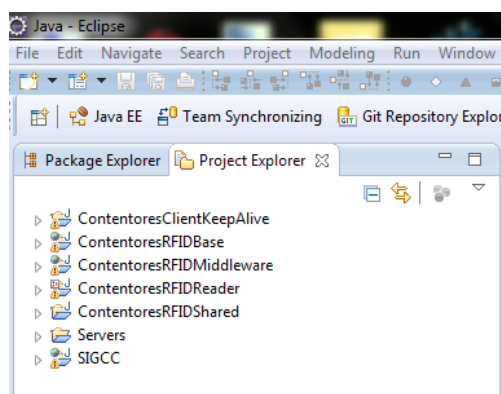


Figura 20 - Organização das peças de *software* desenvolvidas

5.2.1. Pacote de Software *ContentoresClientKeepAlive*

Projeto agregador de um conjunto reduzido de classes java que têm as seguintes responsabilidades:

- Leitura de um ficheiro de configuração para obtenção de parâmetros;
- Leitura da configuração da interface PPP para obtenção do IP dinâmico atribuído ao *Raspberry* dentro da rede sem fios;
- Envio da informação de dez em dez minutos do nome da máquina, IP e data e hora de envio para o servidor aplicacional através do protocolo *http* por invocação a um java *servlet*.

Este projeto resulta da necessidade de dois leitores precisarem efetuar as comunicações por rede sem fios. Uma vez que o *IP* dos equipamentos é obtido por *DHCP* houve a necessidade de registar essa informação para se poder aceder remotamente quer para instalar *software*, como para consultar ficheiros de log. Ao nível das classes Java utilizadas, foram criados três *packages* para as agrupar (figura 21).

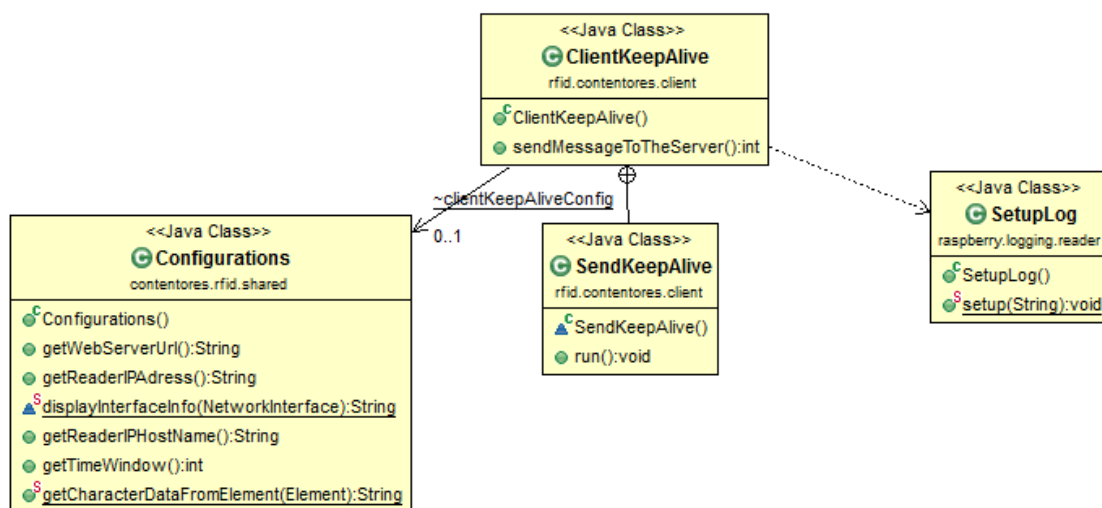


Figura 21 - Diagrama de classes do projeto *ContentoresClientKeepAlive*

A classe *main* encontra-se no pacote *rfid.contentores.client* e executa uma tarefa de 10 em 10 minutos que após obtenção dos parâmetros do ficheiro *Configurations.xml*: endereço do protocolo de internet e nome da máquina através dos métodos *public String getReaderIPAdress()* e *public static String getReaderIPHostName()*, serializa uma mensagem, à qual é adicionada o *timestamp*, através da API JAXB (*Java Architecture for XML Binding*). Esta mensagem representa o objeto *LogKeepAliveObject* da biblioteca comum aos projetos. A API JAXB permite mapear classes Java para representações XML e oferece duas características principais: a capacidade de organizar objetos Java em XML e o inverso, ou seja, de XML *unmarshal* novamente em objetos Java [12]. Após serializada a mensagem é enviada para o servidor aplicacional através do método *public int sendMessageToTheServer() throws SocketException*, onde é deserializada para um objeto java e o mesmo é entregue à *framework Hibernate* para inserir na base de dados a informação associada ao mesmo e discutida na secção 5.1.

5.2.2. Pacote de Software *ContentoresRFIDMiddleware*

Projeto que contém as classes que fornecem as funcionalidades de receber as mensagens, enviadas pelos leitores e após efetuar a desserialização das mesmas, as entrega à *framework Hibernate* para gerir as transações com a base de dados. As comunicações são realizadas de forma síncrona, pois após a inserção da informação na base de dados é devolvido para o leitor o identificador da chave primária da tabela onde é registada a informação. O *middleware* disponibiliza dois conjuntos de classes sendo que um trata as mensagens referentes às comunicações das etiquetas (figura 22), enquanto o outro conjunto de classes trata as mensagens enviadas pelos leitores com a informação associada à máquina que está a efetuar a comunicação (figura 23).

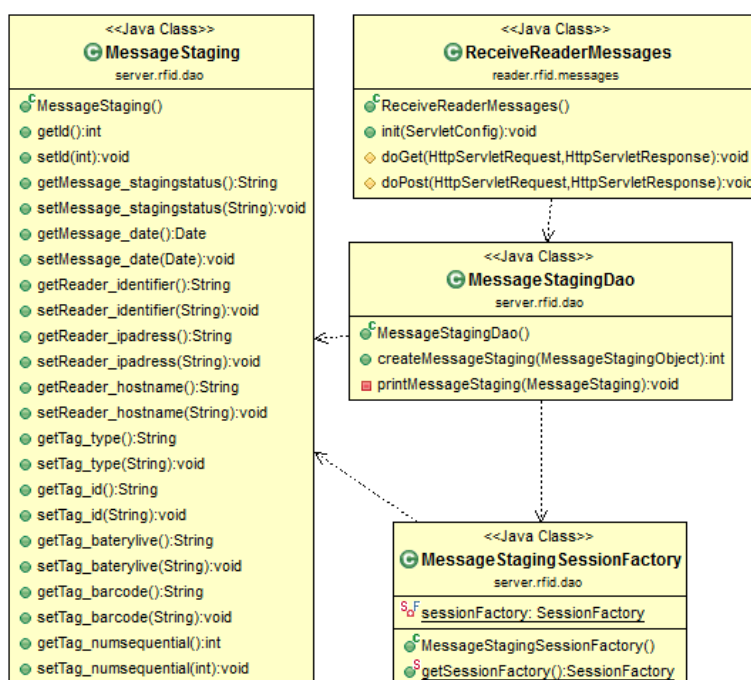


Figura 22 - Diagrama de classes que tratam as mensagens com informação das etiquetas

As classes que tratam as mensagens com a informação referente às etiquetas têm como ponto de entrada uma *Java Servlet* – *ReceiveReaderMessages*. Esta *Servlet* após leitura do *Stream* de dados (mensagem XML) enviados na chamada, desserializa o seu conteúdo para o objeto Java *MessageStagingObject* pertencente à biblioteca comum e incluída no projeto como uma biblioteca. Esta desserialização é realizada recorrendo à já referida API JAXB. Importa referir a vantagem de se utilizar esta API do lado do servidor para desserialização das mensagens, pois independentemente da implementação do lado do cliente, desde que este envie as mensagens no formato XML acordado são garantidas que estas são processadas.

O segundo conjunto de classes (figura 23) que efetua o tratamento das mensagens enviadas pelos leitores com a informação do seu endereço de IP e restantes atributos, foi utilizada a mesma abordagem diferindo apenas na *Servlet* que recebe os pedidos do cliente – *ReceiveKeepAlive*.

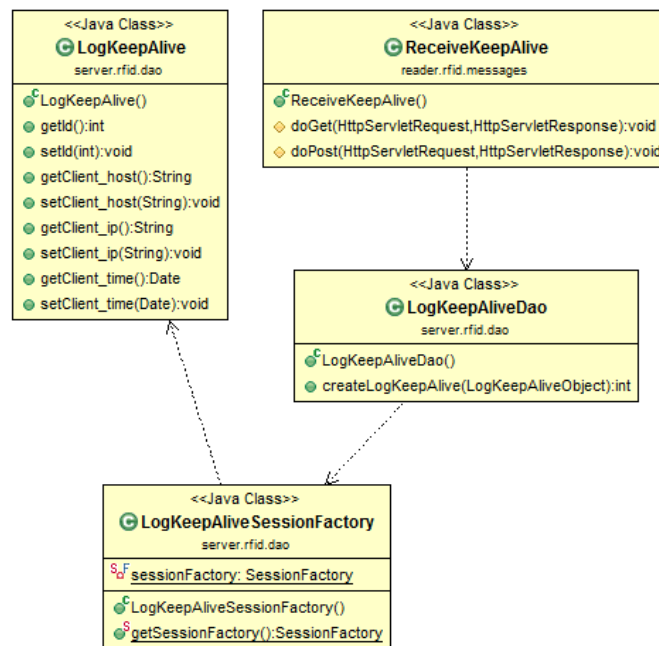


Figura 23 - Diagrama de classes que tratam as mensagens com informação dos leitores

5.2.3. Pacote de Software *ContentoresRFIDReader*

Projeto que agrega as classes (figura 24) com o *software* que foi instalado nos leitores. A classe *RunReader* representa o programa principal através do método main: `public static void main(String[] args)`. Na sua execução, são criadas três tarefas distribuindo as responsabilidades da seguinte forma:

- Tarefa de filtrar as mensagens recebidas – instancia a classe *ReadCom_evt* passando o nome da porta série ao método `public boolean comOpen(String com_name) throws SerialPortException`. É adicionado um *event listener* através do método `addEventListener()` à porta onde está ligado o dispositivo para ler os bytes que vão ser rececionados. Para garantir a sincronização dos caracteres recebidos implementou-se uma máquina de estados que verifica se é o início de uma linha e nesse caso passa ao próximo estado que verifica se é uma mudança de linha. Neste estado, caso seja uma mudança de linha passa ao estado seguinte para sincronização onde se verifica se o número de caracteres é maior do que 9, visto que esperamos uma *String* de tamanho 10 e caso se verifique efetua-se a leitura dos bytes para um objeto do tipo *String* e colocada no FIFO.
- Tarefa que tem como responsabilidades obter do FIFO das mensagens a enviar para o servidor as mensagens entregues lidas do FIFO cliente e recebidas na porta série. Após obtenção destas, envia-as ao método `public int sendMessageToTheServer(MessageStagingObject _messageStagingObject) throws SendTagInformationException`. Este método pertence à classe *ServerMessage* do *package raspberry.server.communications* e tem como trabalho, efetuar a serialização do objeto para uma mensagem XML e envio desta para o servidor, ficando a aguardar o identificador da transação ou um inteiro negativo, definido para o caso de ocorrência de alguma exceção. Caso ocorra uma exceção, o objeto é mantido na FIFO servidor e é tentado o seu envio mais tarde, caso contrário é removido.

- Tarefa que tem como trabalho, verificar de 5 em 5 minutos, se existem mensagens na tabela de *hash* em memória associadas a uma etiqueta para a qual não existem comunicações numa janela temporária ao fim de 10 minutos. Esta verificação é realizada pelo método `public synchronized int existsTagExpired(ConcurrentHashMap<String, TagObject> _hashTable)` da classe `MessageFilter` do package `raspberry.parse.filter` que devolve um inteiro com o número de registos encontrados. Caso existam objetos nestas condições, os mesmos são adicionados ao FIFO servidor e removidos da tabela de *hash*.

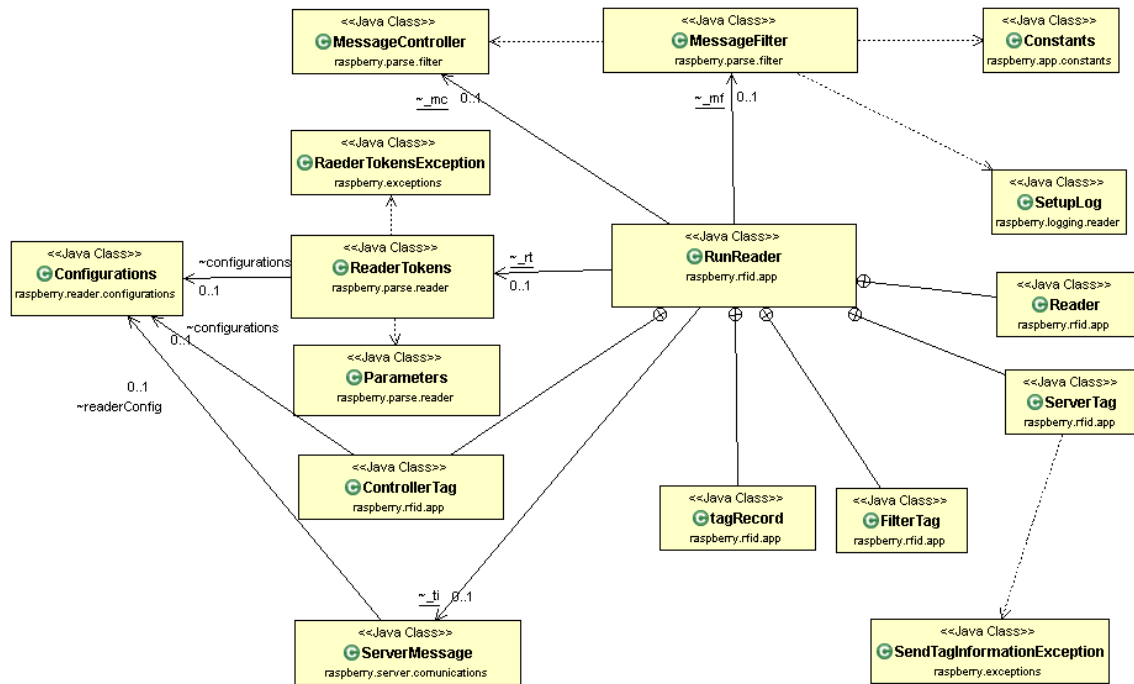


Figura 24 - Diagrama de classes do pacote de *software* do leitor

5.2.4. Pacote de Software *ContentoresRFIDShared*

Neste projeto foram agrupadas as classes que se pretendia que fossem comuns ou utilizadas pelos restantes projetos. Basicamente são as classes que mapeiam a estrutura dos objetos enviados nas mensagens e as entidades da base de dados. Algumas destas classes dado que vão ser utilizadas no contexto da *framework Hibernate* foi necessário criar as devidas anotações ao nível do nome da classe para mapeamento da sua representação para *XML*. De seguida ilustra-se este mapeamento num excerto do código da classe `TagObject`, em que a primeira linha representa a diretiva para um tipo de esquema *XML*. A segunda linha é também utilizada no topo da classe e fornece controlo sobre a serialização padrão de propriedades e campos de uma dada classe. A terceira linha indica qual o elemento raiz da estrutura *XML*.

```
@XmlType
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "TagObject")
public class TagObject implements Serializable {

}
```

Sistema Integrado de Localização de Contentores de Correio

Estas classes contêm apenas os métodos de atribuição de valores aos seus atributos e de obtenção dos valores desses mesmos atributos. Para além das classes anotadas, existe uma outra que efetua a leitura do ficheiro de configuração do projeto – `Configurations.XML` para obtenção dos parâmetros do URL do servidor aplicacional, tempo de expiração das mensagens em memória na tabela de *hash*.

Para criação da biblioteca a incluir nos projetos que dela necessitaram, efetuou-se a sua exportação ao nível do IDE para um ficheiro do com a extensão “*.jar*” - `ContentoresRFIDShared.jar`. As classes agrupadas neste pacote são as representadas na figura 25.

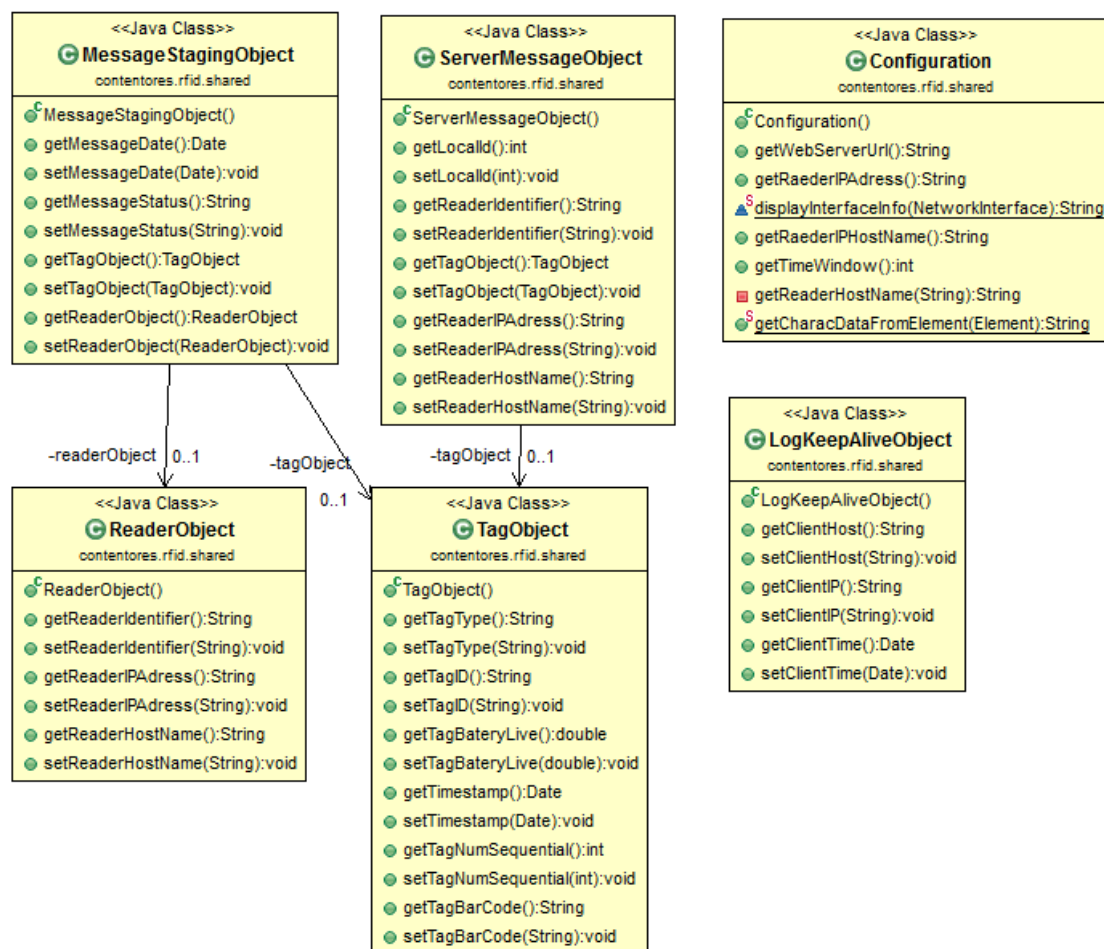


Figura 25 - Diagrama de classes que compõem o pacote comum

5.2.5. Aplicação Web - SIGCC

O projeto SIGCC disponibiliza uma pequena aplicação web para visualização das leituras efetuadas nos locais de ensaio do protótipo. Esta aplicação foi desenvolvida através do padrão MVC¹⁶ sobre a *framework Stripes*. Esta *framework* torna o desenvolvimento de aplicações java para a web de forma

¹⁶ *Model View Controller Architecture*

Sistema Integrado de Localização de Contentores de Correio

rápida e simples. Não necessita dos tradicionais ficheiros de configuração em XML, sendo o ficheiro *web.xml* o único necessário à sua utilização e que ficou definido da seguinte forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <filter>
    <filter-name>StripesFilter</filter-name>
    <filter-class>
      net.sourceforge.stripes.controller.StripesFilter
    </filter-class>
    <init-param>
      <param-name>ActionResolver.Packages</param-name>
      <param-value>sigcc.action</param-value>
    </init-param>
  </filter>
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>
      net.sourceforge.stripes.controller.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <filter-mapping>
    <filter-name>StripesFilter</filter-name>
    <servlet-name>DispatcherServlet</servlet-name>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
  </filter-mapping>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

A *framework Stripes* é mais visível nas partes que interagem com o modelo (*Model*) e na da camada de apresentação (*View*). Interage com o modelo mas não atua neste mantendo-o independente [13]. O *Stripes* é baseado em eventos de ação do utilizador no browser e o seu comportamento é o mesmo do *http*. Após uma ação do utilizador é enviado um pedido ao servidor que efetua o seu trabalho e devolve uma resposta. Um pedido é traduzido para um *action bean* que executa um método java para executar o trabalho para que foi concebido e devolve um resultado. Este resultado é interpretado pela *framework* e apresentado no *browser*.

A aplicação foi instalada a partir da página de administração do *apache-tomcat* versão 7.0.37, ficando a mesma disponível no endereço configurado no ficheiro *server.xml* da diretoria *conf* de instalação do *apache-tomcat* (figura 26).

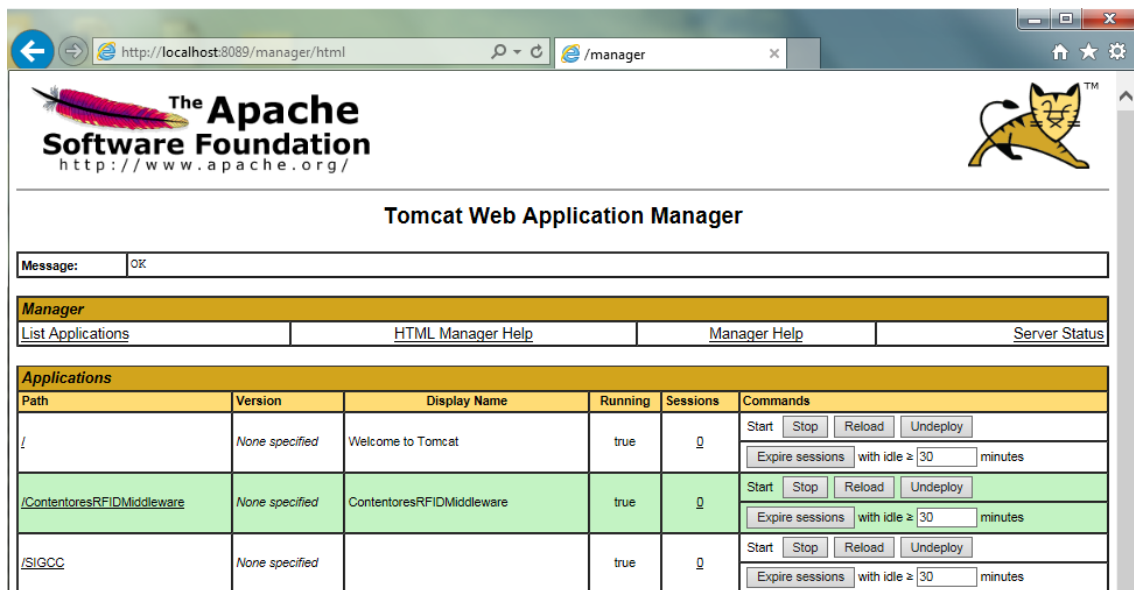


Figura 26 - Página de administração do *apache-tomcat*

Em termos da camada de apresentação dos dados ao utilizador, e procurando um formato simples optou-se por criar vários separadores (figura 27) de acordo com os locais: um separador que permite visualizar as leituras efetuadas em todos os locais, um outro com apenas as leituras no CPLS, CDP 1500 e CDP 1600.



Figura 27 - Aplicação SIGCC

5.3. Configuração do Modem GSM

Pelo facto de não dispor de pontos de rede disponíveis em dois dos locais escolhidos para a realização do protótipo e dado que a sua colocação iria trazer custos para a empresa, optou-se por encontrar uma solução para as comunicações que passou por utilizar uma APN de uma empresa do grupo. Esta limitação inicial revelou-se mais tarde de grande utilidade, pois permitiu testar o conceito da utilização de leitores nas viaturas com idêntico cenário de comunicações.

Realça-se aqui que esta parte do trabalho trouxe mais alguma morosidade ao projeto, pois foi necessário adquirir dois cartões GSM, gentilmente cedidos por uma empresa do grupo, e duas canetas USB onde foram colocados os cartões.

O passo seguinte consistiu em proceder às devidas configurações em cada um dos *Raspberry Pi* para utilização da ligação de rede sem fios através dos cartões. Para este efeito, foi necessário instalar o *script Sakis3g*¹⁷ - que permite criar uma ligação à internet através da utilização de *modem 3G*. Este *script* pode ser utilizado em modo gráfico ou em linha de comandos. Para instalar o *script*, foi necessário ligar os *Raspberry Pi* à internet através de cabo RJ45 e executar os seguintes comandos:

- `sudo wget "http://www.sakis3g.com/downloads/sakis3g.tar.gz" -O sakis3g.tar.gz`
- `sudo tar -xvzf sakis3g.tar.gz`
- `sudo chmod +x sakis3g`

Este utilitário pode ser executado através de interface gráfico ou em linha de comandos. O primeiro teste foi executado em modo gráfico. Neste passo, foi solicitada a APN que queríamos ligar e de seguida as credenciais de acesso associadas ao cartão. Para executar o utilitário em modo gráfico, utiliza-se o seguinte comando:

- `./sakis3g -interactive`

No entanto, houve ainda algum trabalho adicional que se teve de realizar. Dado que a maioria dos *modems USB* poder operar dois dispositivos, um de armazenamento e outro como *modem*, houve a necessidade de garantir que ao ligar o dispositivo USB, este seria assumido como *modem*. Por omissão, quando ligamos o dispositivo ao *Raspberry Pi*, é geralmente assumido o modo de armazenamento USB. Para contornar esta situação, utilizou-se um outro programa que permite, configurar o sistema para efetuar essa comutação, sendo necessário introduzir no ficheiro `/etc/usb_modeswitch.conf` a configuração correta para o *firmware* do dispositivo. A criação deste ficheiro é necessária porque aquando do arranque do sistema operativo, nem sempre o dispositivo está ativo quando o programa é executado e desta forma o mesmo permanece em modo de armazenamento. Para tal, foram realizados os seguintes passos:

Obter os códigos do dispositivo USB – sem ter uma ligação LAN ou Wifi, executar o seguinte comando:

- `lsusb`

¹⁷ <http://www.sakis3g.com/>

Que mostra a informação dos dispositivos USB, como se pode ver na figura 28.

```
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.  
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.  
Bus 001 Device 006: ID 192f:0916 Avago Technologies, Pte.  
Bus 001 Device 005: ID 1c4f:0002 SiGma Micro Keyboard TRACER Gamma Ivory  
Bus 001 Device 014: ID 19d2:0031 ZTE WCDMA Technologies MSM MF110/MF627/MF636  
Bus 001 Device 004: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
```

Figura 28 - Output do comando *lsusb* no Raspberry Pi

Os números sublinhados representam o fabricante e o código do produto. Estes valores vão ser utilizados no ficheiro customizado que se tem de criar: */etc/usb_modeswitch.conf*. De seguida é feita a extração dos códigos do fornecedor e produto existentes no ficheiro *configPack.tar.gz* utilizando o seguinte comando:

- `cd /tmp`
- `tar -xvzf /usr/share/usb_modeswitch/configPack.tar.gz 19d2:0117`

Após esta operação, e pelo comando: `sudo leafpad 19d2:0117` edita-se o ficheiro com a configuração respetiva. O conteúdo do mesmo deve ser idêntico ao apresentado na figura 29.

```
# ZTE devices  
DefaultVendor = 0x19d2  
DefaultProduct = not set  
  
# ZTE devices  
TargetVendor= 0x19d2  
TargetProductList="0001,0002,0015,0016,0017,0031,0037,0052,0055,0061,0063,0064,  
0066,0091,0108,0117,0128,0157,1402,2002,2003"  
  
MessageContent="55534243123456780000000000000061e00000000000000000000000000000"  
MessageContent2="55534243123456790000000000000061b00000002000000000000000000000"  
MessageContent3="555342431234567020000000800000c85010101180101010101000000000000"  
  
NeedResponse=1
```

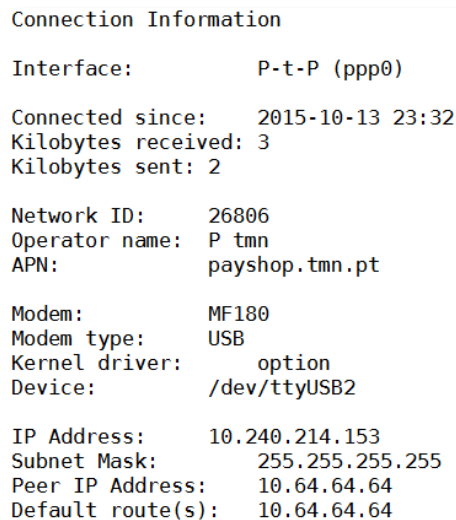
Figura 29 - Ficheiro configuração 19d2:0117

Esta informação vai ser utilizada no ficheiro de configuração que queremos criar e para tal executamos o comando: `sudo leafpad /etc/usb_modeswitch.conf`, substituindo os valores do fabricante e código do produto, assim como os valores do *MessageContent* referentes ao dispositivo que estamos a utilizar.

Com base na informação obtida neste relatório (figura 30), introduziu-se a informação referente aos diversos parâmetros do ficheiro de configuração *Sakis3g.conf* para que o mesmo seja executado de forma automática:

- 🚩 `USBDRIVER="option"`
- 🚩 `OTHER=CUSTOM_TTY`
- 🚩 `CUSTOM_TTY="/dev/ttyUSB2"`
- 🚩 `APN=CUSTOM_APN`
- 🚩 `CUSTOM_APN="PAYSHOP.TMN:PT"`
- 🚩 `APN_USER="payshop"`

```
APN_PASS="*****"  
MODEM="19d2:0117"
```



The screenshot shows a terminal window with a title bar. The text inside the terminal is as follows:

```
Connection Information  
  
Interface:          P-t-P (ppp0)  
  
Connected since:    2015-10-13 23:32  
Kilobytes received: 3  
Kilobytes sent: 2  
  
Network ID:         26806  
Operator name:      P tmn  
APN:                payshop.tmn.pt  
  
Modem:              MF180  
Modem type:         USB  
Kernel driver:      option  
Device:             /dev/ttyUSB2  
  
IP Address:         10.240.214.153  
Subnet Mask:        255.255.255.255  
Peer IP Address:    10.64.64.64  
Default route(s):  10.64.64.64
```

Figura 30 - Relatório de informação da conectividade GSM

Após realização das configurações, e para iniciar o *script* de ligação *gsm* de forma automática aquando do arranque do sistema operativo, criou-se um *Shell script*: *autoconnectnet.sh* (Anexo D), e que foi colocado na diretoria: */etc/init.d/autoconnectnet.sh* onde se colocou a instrução para arrancar o *script* de ligação após um *delay* de 10 segundos.

CAPÍTULO 6

6. Testes e Análise de Resultados

O plano de testes elaborado tem alguns objetivos comuns e outros diferenciados para cada uma dos tipos de ensaio descritos anteriormente e assim possibilitar verificar o comportamento da solução no seu todo.

6.1. Plano de Testes

Ensaio em Ambiente Simulado

- I. Acesso ao servidor aplicacional por um número crescente de leitores – simulados através de uma aplicação cliente em Java instalada em alguns computadores de colegas da empresa.
- II. Verificar o comportamento e acesso concorrenciais à base de dados.
- III. Verificar a quantidade de mensagens comunicadas pelas etiquetas e após aplicação dos filtros confrontar com as rececionadas na base de dados.
- IV. Registrar em ficheiros o *log* das mensagens produzidas, comunicadas e confrontar com as que foram registadas.
- V. Visualizar na aplicação web a informação recolhida pelo sistema podendo sobre a mesma ser aplicados alguns tipos de filtros (e.g. identificador da etiqueta).

Ensaio em Ambiente Real

- I. Recolha de informação sobre o número de possíveis colisões no envio das mensagens pelas etiquetas.
- II. Testar o alcance do leitor para as comunicações com as etiquetas.
- III. Verificar comportamento do *software* em cenário de ambiente real.
- IV. Constatar o registo de saída e entrada de contentores no centro de tratamento e logística de cabo ruivo e em dois centros de distribuição postal: 1500 e 1600.
- V. Analisar desempenho geral do sistema.

Com objetivos de posterior análise de resultados optou-se por definir dois cenários para a realização dos ensaios:

- a) Realizar um número significativo de testes em ambiente controlado de laboratório com um processo simples de instalação da solução,
- b) Realizar um número possível de testes com implementação do protótipo nas instalações da empresa e recolha de dados durante um período expectável de mais ou menos quinze a vinte dias.

6.2. Análise de Resultados

Com base no plano de testes, e após instalação da solução em ambiente real, definiu-se o envio e receção dos contentores, nos três locais escolhidos. De referir aqui, que foi necessário um trabalho adicional, com o envolvimento das equipas CTT para garantir que os contentores seguiam estes mesmos percursos.

Dos dez contentores etiquetados, cinco deles deveriam fazer o percurso entre o CPLS e o CDP 1500 e os outros cinco entre o CPLS e o CDP 1600, ficando organizada a sua dispersão como indicado na tabela abaixo.

Tabela 5 - Dispersão de contentores por local

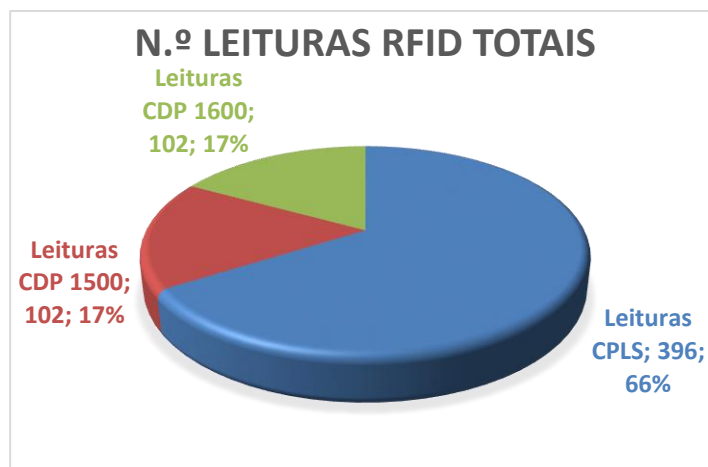
Contentores / Locais	CPLS	CDP 1500	CDP 1600
Contentor 51	Envio + Receção	Receção + Envio	
Contentor 53	Envio + Receção	Receção + Envio	
Contentor 54	Envio + Receção		Receção + Envio
Contentor 55	Envio + Receção	Receção + Envio	
Contentor 56	Envio + Receção		Receção + Envio
Contentor 57	Envio + Receção	Receção + Envio	
Contentor 58	Envio + Receção		Receção + Envio
Contentor 59	Envio + Receção		Receção + Envio
Contentor 60	Envio + Receção		Receção + Envio
Contentor 61	Envio + Receção	Receção + Envio	

O envio e receção de contentores a partir do CPLS ocorreu pelo período de duas semanas, com início a 29-02-2016 e término a 11-03-2016. Durante este período, diariamente os contentores foram enviados, do CPLS para os CDP e rececionados no mesmo dia, no processo de expedição. Os testes revelaram um comportamento expetável, sendo que um dos contentores entrou na rede de distribuição e perdeu-se o seu rastreio (devido a erro de encaminhamento dos operadores).

Foram registadas as leituras por local, sendo que com base nos pressupostos, sempre que um contentor entrava na zona de captação de mensagem pelo leitor, e rececionada a primeira comunicação, o sistema registava a informação na base de dados associando-a a um evento de “**Entrada**”. Por seu lado e tal como assumido, após dez minutos sem comunicação da etiqueta, foi gerado um evento de “**Saída**” com introdução dessa informação na base de dados. Estas assunções tiveram origem, por um lado, devido ao facto de possuímos apenas um leitor por local e por outro, para gerir todas as mensagens enviadas pelas etiquetas enquanto na presença de um leitor.

Durante o período de testes, foram rececionadas e registadas na base de dados um total de 600 mensagens, ficando as mesmas distribuídas por local como mostra o gráfico na tabela 6.

Tabela 6 - Número de leituras RFID por local



O elevado número de leituras no CPLS, relativamente aos CDP, é justificado pelo facto ser o local onde se concentravam todos os contentores, mas, também por existirem muitos eventos de “Entrada” e “Saída” fictícios. Uma vez que o movimento dos contentores, dentro das instalações, fazia com que entrassem e saíssem da zona de alcance do leitor, levava a que o leitor gerasse eventos adicionais de “Entrada” e “Saída”. No entanto, a análise à informação recolhida, permitiu isolar estes eventos por forma a avaliarmos se sempre que um contentor saía e entrava do CPLS com destino ao CDP e respetiva entrada e saída deste tinha o respetivo evento registado na base de dados. Constata-se que todos os eventos de dispersão foram recolhidos na base de dados, por local, data/hora e contentor. A tabela 7 abaixo mostra os movimentos do contentor nº 51 nos respetivos locais com a hora da chegada e partida em cada dia.

Tabela 7 - Leituras obtidas com a etiqueta do identificador 51

Datas	CPLS		CDP 1500	
	Partida	Chegada	Partida	Chegada
29-02-2016	06:20:00			06:40:00
		09:05:50	07:50:00	
01-03-2016				07:20:00
		08:47:00	08:25:00	
02-03-2016	06:41:00			07:24:00
		08:51:00	08:26:00	
03-03-2016	06:47:00			07:05:00
		08:59:00	08:31:00	
04-03-2016	06:57:00			07:13:00
		10:18:00	08:17:00	
07-03-2016	07:24:00			07:45:00
		09:13:00	08:43:00	
08-03-2016	07:04:00			08:29:00
		09:28:00	08:39:00	
09-03-2016	07:00:00			07:41:00
		09:40:00	08:34:00	
10-03-2016	06:45:00			07:04:00
		08:55:00	08:25:00	
11-03-2016	06:46:19			06:58:00
		08:57:33	08:16:34	

Pela análise, podemos verificar que excetuando o dia 01-03-2016, em todos os dias foram gerados os eventos de saída dos contentores do CPLS, no horário da manhã entre as 06h00 e as 08h00 com destino

ao CDP. No dia 01-03-2016, o não registo da saída do CPLS, deveu-se a uma atualização que foi necessário efetuar ao *software* em execução no leitor. Verifica-se também que diariamente ficou registado o evento de entrada e respetiva saída dos contentores em cada CDP. No anexo E mostram-se as leituras obtidas para os restantes contentores.

Avaliação da autonomia da bateria

Após a colocação das baterias nas etiquetas, ficaram em funcionamento sem interrupção enviando mensagens de 2 em 2 segundos aproximadamente. Para se poder construir um modelo de previsão de duração da bateria, foram feitos registos dos valores medidos e enviados nas mensagens, durante todos os dias em que o projeto esteve em funcionamento. Tendo em consideração que as medidas dos valores das tensões das baterias estão sujeitas a pequenas interferências devidas ao ruído induzido nos valores da tensão e ainda as diferenças de temperatura a que estas são feitas, para cada dia, calculou-se a média dos valores medidos. Fazendo uma regressão linear com os valores medidos função do número de dias de funcionamento, obteve-se a equação (8) como modelo de previsão do valor da tensão da bateria função do número de dias em que esta está em funcionamento:

$$V_{bat} = -0,0005 \text{ Ndias} + 3,15161 \quad (8)$$

A Figura (31) mostra o gráfico dos dados usados e também a reta de previsão obtida pela regressão linear. Da física dos materiais de fabricação e funcionamento interno de uma bateria, sabe-se que inicialmente (primeiros dias) a descarga não tem a forma de reta, sendo mais parecida com uma exponencial decrescente. De igual forma o processo de descarga apresenta uma forma aproximadamente linear durante o restante processo de descarga até que quando chega a cerca de 50% do valor nominal cai abruptamente. A equação da reta determinada pela regressão modela a descarga da bateria durante essa zona linear. Considerando que as etiquetas podem funcionar com um valor mínimo de tensão de 1,9 V e usando a equação da reta da equação (8), determina-se que são precisos 2512 dias (6 anos, 10 meses e 17 dias) para que as pilhas se considerem esgotadas.

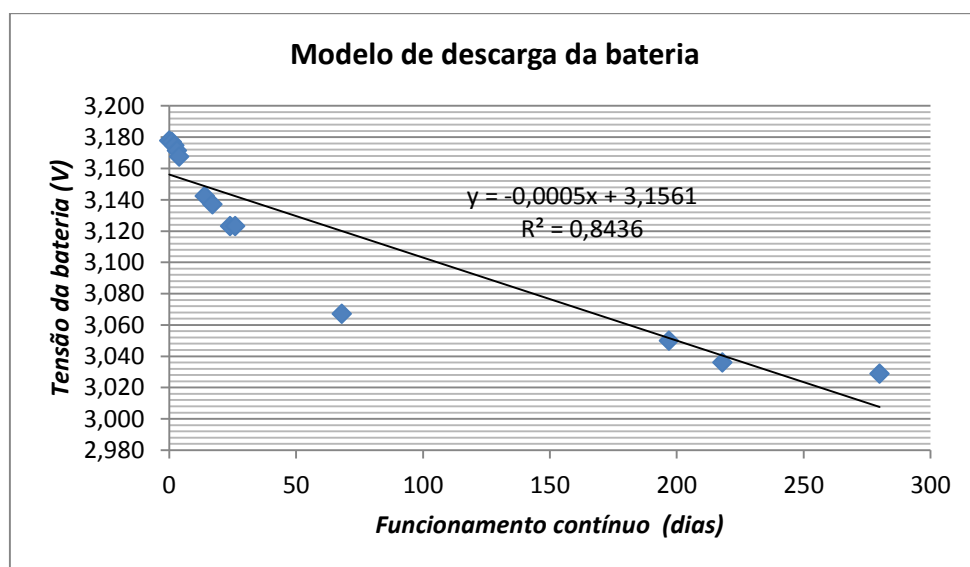


Figura 31 - Tensão da bateria em função do n° de dias em funcionamento

CAPÍTULO 7

7. Conclusões e Trabalho Futuro

Apresentam-se neste ponto as conclusões retiradas de todo o trabalho realizado relativamente aos aspetos positivos e também negativos o que poderá ser uma perspetiva de trabalho futuro sobre o tema estudado.

7.1. Conclusões

Como conclusões a retirar deste trabalho, podemos enumerar desde logo a possibilidade de implementar um sistema RFID com etiquetas ativas de baixo custo económico mas com uma durabilidade da pilha bastante grande. Este foi, alias um dos requisitos a ter em conta desde o início do projeto de forma que se demonstra a viabilidade do mesmo.

Foram produzidas 10 etiquetas RFID ativas com aquisição dos módulos rádio e programadas internamente no ISEL. Uma das preocupações era o grau de resistência que as etiquetas poderiam oferecer quando instaladas nos contentores e verificou-se que as mesmas permaneceram intatas durante as duas semanas do piloto.

Foram também adquiridos 3 mini computadores *Raspberry Pi* para ligação dos módulos de leitor e respetivo *software* de leitura e filtro das mensagens rádio recebidas pelos mesmos. Estes equipamentos para além de terem um custo muito acessível respondem como uma boa solução de arquitetura, com um ótimo desempenho de *hardware* e facilidade de integração em diferentes tipologias de rede.

Relativamente ao processo de instalação quer das etiquetas nos contentores, quer dos leitores o mesmo foi efetuado de forma simples e sem qualquer complicação.

Quanto aos resultados práticos obtidos, durante a implementação do protótipo, permitem considerar esta tecnologia como uma boa possibilidade para a rastreabilidade de contentores de correio. Aferiu-se assim, que sem qualquer trabalho manual adicional no manuseamento dos contentores se obteve informação relevante sobre os seus movimentos diários durante as duas semanas do projeto.

Conclui-se também a dificuldade e morosidade de implementar no terreno um protótipo deste género numa empresa com uma grande dimensão como é o caso dos CTT. Esta dificuldade advém do facto de ser necessário dialogar e envolver pessoas de diversos departamentos, estas, com agendas muito preenchidas, tornando a tomada de decisões mais morosa e com consequente impacto ao nível das datas no projeto final. Apesar das grandes dificuldades encontradas, com persistência e também boa vontade dos colegas envolvidos, foi possível a sua realização.

7.2. Trabalho Futuro

Como trabalho futuro, a solução apresentada, abre as portas para um sem número de possibilidades de utilização da tecnologia RFID no contexto da rastreabilidade de contentores de correio permitindo o controlo destes ativos empresariais.

Desde logo, seria de todo interessante poder efetuar-se um protótipo idêntico, mas com um âmbito mais alargado, quer em número de locais, quer quanto ao número de contentores. Neste âmbito mais alargado, dever-se-á também equacionar a colocação de um número maior de leitores, alguns em zonas mais interiores dos edifícios, e também no interior das viaturas para permitir tirar conclusões mais sustentadas.

Ao nível do modelo de dados, e dado que neste trabalho apenas foi criada uma entidade considerada como uma tabela de recolha de dados, fica para realizar em trabalho futuro a implementação de um modelos de dados relacional idêntico ao que se encontra no anexo B que foi pensado para este problema mas que não chegou a ser implementado. O desenho do mesmo pode ser reanalisado e implementar processos de transformação da informação da área de recolha de dados para o modelo relacional com vista a fornecer aos utilizadores a informação que necessitam.

Quanto aos módulos de *software* desenvolvido na componente de *Middleware*, como trabalho futuro podem ser implementadas as funcionalidades já descritas nalguns dos requisitos, que permitam através de uma interface web a comunicação com os leitores, para que se possa enviar comandos diversos para serem executados a partir desse interface. Podem também ser desenvolvidos um conjunto de serviços com vista a disponibilizar informação a outros sistemas que dela necessitem.

Poderão ainda ser equacionadas tecnologias alternativas como é o caso do *Bluetooth Low Energy*, que poderá permitir a implementação de sistemas normalizados, dada a sua grande adoção em equipamentos e dispositivos, nomeadamente dispositivos móveis.

BIBLIOGRAFIA

- [1] Hunt, V. D., Puglia, A. and Puglia, M. (2007) History and Evolution of RFID Technology, in RFID-A Guide to Radio Frequency Identification, John Wiley & Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/9780470112250.ch3
- [2] R. Das and P. Harrop, "RFID Forecasts, Players and Opportunities 2016-2026," IDTechEx Market Study, October 2015, acedido em 15-07-2015.
<http://www.idtechex.com/research/reports/rfid-forecasts-players-and-opportunities-2016-2026-000451.asp>
- [3] "Regulatory status for using RFID in the EPC Gen 2 band (860 to 960 MHz) of the UHF spectrum" (PDF). GS1.org. 2014-10-31, acedido em 20-07-2015.
http://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_2_0_0_standard_20131101.pdf
- [4] Miodrag Bolic (Editor), David Simplot-Ryl (Co-Editor), Ivan Stojmenovic (Co-Editor), RFID Systems: Research Trends and Challenges, July 2010
- [5] <http://www.centrenational-rfid.com/les-gammes-de-frequences-rfid-article-16-fr-ruid-17.html>, acedido em 12-09-2015.
- [6] Juraj Vaculík, Peter Kolarovszki and Jiří Tengler (2013). Possibility of RFID in Conditions of Postal Operators, Radio Frequency Identification from System to Applications, Dr. M. I. B. Reaz (Ed.), ISBN: 978-953-51-1143-6, InTech, DOI: 10.5772/53285.
- [7] Alberto Silva, Carlos Videira, UML - Metodologias e Ferramentas CASE - 2ª Edição - Volume 1, Maio 2005
- [8] N. Abramson, "The Aloha system-another alternative for computer communications," in AFIPS Conf. Proc., 1970, vol. 36, pp. 295–298.
- [9] <http://download.oracle.com/otndocs/jcp/ejb-3.1-pfd-oth-JSpec/>, acedido em 15-02-2016.
- [10] Ottinger, J., Guruzu, S., Mak, G., & Safari Books Online (Firm). (2015). Hibernate recipes: A problem-solution approach (Second edition.). [New York, NY]: Apress.
- [11] Christian Bauer, Gary Gregory, Gavin King (2015). Java Persistence with Hibernate, 2nd Edition
- [12] https://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding, acedido em 25-03-2016.
- [13] Frederic Daoud (2008). Stripes ...and Java Web Development Is Fun Again.

ANEXOS

Anexo A – Código Fonte do Emissor e do Recetor

Ficheiro: *config.h*

```

/*****
 * This module contains the configuration macros
 * Used by all modules
 *
 *****/

#ifndef CONFIG_H
#define CONFIG_H

#include "gpio.h"

//uncomment only one: _transmitter or _receiver!!!

#define _transmitter
//#define _receiver

/*****
 *
 * CPU peripherals settings
 *
 * configuration:
 *      ???      nRF_IRQ      input      //Not used by the software
 *      GPIO0     SCK          output
 *      GPIO1     MOSI         output
 *      GPIO3     MISO         input
 *      GPIO2     CSN          output
 *      GPIO5     nRF_CE       output
 *      GPIO4     LED          output //Activity Led or ADC input
 *
 *****/

#define CONFIG_CPU_TRIS_tx      0x18      //GPIO3 as input, GPIO4 analog input

#define CONFIG_CPU_TRIS        0x08      //GPIO3 as input

#define PIN_SCK_R                GPIO0
#define PIN_SCK_0()              clrGP_0()
#define PIN_SCK_1()              setGP_0()

#define PIN_MOSI_R                GPIO1
#define PIN_MOSI_0()              clrGP_1()
#define PIN_MOSI_1()              setGP_1()

#define PIN_MISO_R                GPIO3      //For read MSIO

#define PIN_CSN_R                GPIO2
#define PIN_CSN_0()              clrGP_2()
#define PIN_CSN_1()              setGP_2()

#define PIN_CE_R                GPIO5
#define PIN_CE_0()              clrGP_5()
#define PIN_CE_1()              setGP_5()

#define PIN_LED_R                GPIO4

```

Sistema Integrado de Localização de Contentores de Correio

```
#define PIN_LED_0()                clrGP_4()
#define PIN_LED_1()                setGP_4()

#define _XTAL_FREQ 4000000 //4Mhz clock

typedef unsigned char byte;

/*****
 *
 * RF chip settings
 *
 *****/

#define CONFIG_DEFAULT_CHANNEL 81

#undef CONFIG_HIRES_LOCATION

#ifdef CONFIG_HIRES_LOCATION
#define CONFIG_MAX_POWER_LEVELS 8
#else /*CONFIG_HIRES_LOCATION */
#define CONFIG_MAX_POWER_LEVELS 4
#endif /*CONFIG_HIRES_LOCATION */

#endif
```

Ficheiro: *timer.h*

```
/*****
 * This module contains the header software functions related
 * with the timing functions
 *****/

#ifndef _TIMER_H
#define _TIMER_H

/*
#define TIMER1_HZ      32768
#define JIFFIES_PER_MS(x) ((x*TIMER1_HZ)/1000)
*/

void timer_init(void);
void sleep_2ms(void);
void sleep_jiffies(unsigned short jiffies);

#endif /* _TIMER_H */
```

Ficheiro: *timer.c*

```
/*****
 * This module contains the software functions related
 * with the timing functions
 *****/

#include <htc.h>
#include "timer.h"
#include "config.h"

void timer_init (void) //No initializations necessary
{
}

void sleep_jiffies (unsigned short jiffies)
{
    _delay(jiffies); //Pic CC internal delay;
```

```

}
void sleep_2ms (void)
{
    __delay_ms(2); //Pic CC internal delay;
}

```

Ficheiro: *gpio.h*

```

/*****
 * This module contains the header definitions for PIC12F
 * IO pins manipulation functions.
 *
 *****/

#ifndef GPIO_H
#define GPIO_H

/*GPIO pins numbers to be used in function setGP_*( ) */

void setGP_0();
void setGP_1();
void setGP_2();
void setGP_4();
void setGP_5();
/*****/
void clrGP_0();
void clrGP_1();
void clrGP_2();
void clrGP_4();
void clrGP_5();

#endif //GPIO_H

```

Ficheiro: *gpio.c*

```

/*****
 * This module contains the software necessary for PIC12F
 * IO pins manipulation functions.
 *
 *****/

#include <htc.h>
#include "gpio.h"

/*****IO/BITS manipulation *****/

volatile union {
    unsigned char    port; // shadow copy of GPIO
    struct {
        unsigned GP0 : 1;
        unsigned GP1 : 1;
        unsigned GP2 : 1;
        unsigned GP3 : 1;
        unsigned GP4 : 1;
        unsigned GP5 : 1;
    };
} sGPIO;

void setGP_0(){
    sGPIO.GP0 = 1;
    GPIO = sGPIO.port;
}
void setGP_1(){
    sGPIO.GP1 = 1;

```

Sistema Integrado de Localização de Contentores de Correio

```
        GPIO = sGPIO.port;
    }
    void setGP_2(){
        sGPIO.GP2 = 1;
        GPIO = sGPIO.port;
    }
    void setGP_4(){
        sGPIO.GP4 = 1;
        GPIO = sGPIO.port;
    }
    void setGP_5(){
        sGPIO.GP5 = 1;
        GPIO = sGPIO.port;
    }
    /*****/
    void clrGP_0(){
        sGPIO.GP0 = 0;
        GPIO = sGPIO.port;
    }
    void clrGP_1(){
        sGPIO.GP1 = 0;
        GPIO = sGPIO.port;
    }
    void clrGP_2(){
        sGPIO.GP2 = 0;
        GPIO = sGPIO.port;
    }
    void clrGP_4(){
        sGPIO.GP4 = 0;
        GPIO = sGPIO.port;
    }
    void clrGP_5(){
        sGPIO.GP5 = 0;
        GPIO = sGPIO.port;
    }
}
```

Ficheiro: *soft_uart.h*

```
/*****/
* This module contains the software header functions related      *
* with UART software emulation in the PIC 12F                      *
*****/
```

```
#ifndef _SOFT_ART_H
#define _SOFT_ART_H

void InitSoftUart();
//Return Null if bad stop bit
unsigned char UART_Receive(void);
void UART_Transmit(unsigned char DataValue);
void echoTxRx();
void txByte(byte ch);

#endif
```

Ficheiro: *soft_uart.c*

```
/*****/
* This module contains the software functions related with UART *
* software emulation in the PIC 12F                             *
*****/
```

```
#include <htc.h>

#include "config.h"
```

Sistema Integrado de Localização de Contentores de Correio

```
#include "soft_uart.h"

//_XTAL_FREQ --> 4Mhz

//#define Baudrate      1245    //1200
//#define Baudrate      2500    //2400
//#define Baudrate      13000   //9600
#define Baudrate        38817   //19200

#define OneBitDelay      (1000000/Baudrate)
#define DataBitCount  8

#define UART_RX          GPIO3

#define UART_TX          GPIO4
#define UART_TX_0()      clrGP_4()
#define UART_TX_1()      setGP_4()

#define UART_RX_DIR      TRIS3
#define UART_TX_DIR      TRIS4

void echoTxRx(){
    while(1){
        if(UART_RX) UART_TX_1();
        else UART_TX_0();
    }
}
//////////
void InitSoftUart(){
    UART_RX_DIR = 1;
    UART_TX_DIR = 0;
    UART_TX_1();
}

//Return Null if bad stop bit
unsigned char UART_Receive(void){
    unsigned char DataValue=0;

    while (UART_RX == 1);
    __delay_us(OneBitDelay);
    __delay_us(OneBitDelay/2);

    for( unsigned char i=0; i < DataBitCount; i++)
    {
        DataValue >>= 1;
        if(UART_RX == 1) DataValue |= 0x80;
        __delay_us(OneBitDelay);
    }
    DataValue = (UART_RX == 1)? DataValue : 0;
    __delay_us(OneBitDelay/2);
    return DataValue;
}

void UART_Transmit(unsigned char DataValue){

    UART_TX_0();
    __delay_us(OneBitDelay);

    for( unsigned char i=0; i < DataBitCount; i++)
    {
        if( DataValue & 1 ) UART_TX_1();
    }
}
```

```

        else UART_TX_0();

        DataValue >>= 1;
        __delay_us(OneBitDelay);
    }
    UART_TX_1();
    __delay_us(OneBitDelay);
    __delay_us(OneBitDelay);
}

char toHex(byte n){
    if(n < 10) return n+ '0';
    else return n+'A'-10;
}

void txByte(byte ch)
{
    UART_Transmit(toHex(ch >> 4));
    UART_Transmit(toHex(ch & 0x0F));
}

```

Ficheiro: *nRF_drv.h*

```

/*****
 * This module contains the header software functions related
 * with the operation of the NRF24L00+ rf device
 *****/

#ifndef _nRF_drv_H
#define _nRF_drv_H

unsigned char nRFCMD_XcieveByte (unsigned char byte);
void configure_transmitter(void);
void configure_receiver(void);

void transmit_data(char *data);
void receive_data (char *data);

void rdTX_adress(byte* data);
void rdRX_adress(byte* data);
void wrTX_adress(byte* data);
void wrRX_adress(byte* data);

byte pool_RX_DR();
byte pool_RX_FIFO();
byte pool_TX_FIFO();

#endif // _nRF_drv_H

```

Ficheiro: *nRF_drv.c*

```

/*****
 * This module contains the software functions related with the
 * operation of the NRF24L00+ rf device
 *****/

#include <htc.h>

#include "config.h"
#include "gpio.h"
#include "timer.h"
#include "nRF_drv.h"

unsigned char
nRFCMD_XcieveByte (unsigned char byte)

```

```

{
    unsigned char idx;

    for (idx = 0; idx < 8; idx++)
    {
        //CONFIG_PIN_MOSI((byte & 0x80) ? 1 : 0);
        if((byte & 0x80)) PIN_MOSI_1(); else PIN_MOSI_0();
        byte <<= 1;
        PIN_SCK_1();
        _delay(4);
        byte |= PIN_MISO_R;
        PIN_SCK_0();
    }
    PIN_MOSI_0();

    return byte;
}

void inputDataBlock(byte * p, char sz){
    byte j;
    for (j = 0; j < sz; j++)
    {
        p[j] = nRFCMD_XcieveByte(0x00);
    }
}

void outputDataBlock(byte * p, char sz){
    byte j;
    for (j = 0; j < sz; j++)
    {
        nRFCMD_XcieveByte(p[j]);
    }
}

#ifdef _transmitter

void configure_transmitter(void)
{
    unsigned char i, j, data, cmd;

    PIN_CE_0();
    PIN_CSN_0();

    // PTX, CRC enabled, mask a couple of ints
    nRFCMD_XcieveByte(0x20);
    nRFCMD_XcieveByte(0x38); //0x78?
    PIN_CSN_1();
    PIN_CSN_0();

    //auto retransmit off
    nRFCMD_XcieveByte(0x24);
    nRFCMD_XcieveByte(0x00);
    PIN_CSN_1();
    PIN_CSN_0();

    //address width = 5
    nRFCMD_XcieveByte(0x23);
    nRFCMD_XcieveByte(0x03);
    PIN_CSN_1();
    PIN_CSN_0();

    //data rate = 1MB
    nRFCMD_XcieveByte(0x26);
    nRFCMD_XcieveByte(0x07); //0x26?
    PIN_CSN_1();
    PIN_CSN_0();
}

```

```

    //set channel 2, this is default but we did it anyway...
    nRFCMD_XcieveByte(0x25);
    nRFCMD_XcieveByte(0x02);
    PIN_CSN_1();
    PIN_CSN_0();

    //set address E7E7E7E7E7, also default...
    nRFCMD_XcieveByte(0x30);
    for (j = 0; j < 5; j++)
    {
        nRFCMD_XcieveByte(0xE7);
    }
    PIN_CSN_1();
    PIN_CSN_0();

    //disable auto-ack, RX mode
    //shouldn't have to do this, but it won't TX if you don't
    nRFCMD_XcieveByte(0x21);
    nRFCMD_XcieveByte(0x00);
    PIN_CSN_1();
}
#endif // _transmitter

#ifdef _receiver

//configure nRF24L01 for receive
void configure_receiver(void)
{
    unsigned char i, j;

    PIN_CSN_0();
    PIN_CE_0();

    //PRX, CRC enabled
    nRFCMD_XcieveByte(0x20);
    nRFCMD_XcieveByte(0x39);
    PIN_CSN_1();
    PIN_CSN_0();

    //disable auto-ack for all channels
    nRFCMD_XcieveByte(0x21);
    nRFCMD_XcieveByte(0x00);
    PIN_CSN_1();
    PIN_CSN_0();

    //address width = 5 bytes
    nRFCMD_XcieveByte(0x23);
    nRFCMD_XcieveByte(0x03);
    PIN_CSN_1();
    PIN_CSN_0();

    //data rate = 1MB
    nRFCMD_XcieveByte(0x26);
    nRFCMD_XcieveByte(0x07);
    PIN_CSN_1();
    PIN_CSN_0();

    //4 byte payload
    nRFCMD_XcieveByte(0x31);
    nRFCMD_XcieveByte(0x04);
    PIN_CSN_1();
    PIN_CSN_0();

    //set channel 2

```

```

        nRFCMD_XcieveByte(0x25);
        nRFCMD_XcieveByte(0x02);
        PIN_CSN_1();
        PIN_CSN_0();

        //set address E7E7E7E7
        nRFCMD_XcieveByte(0x2A);          //0x30
        for (j = 0; j < 5; j++)
            nRFCMD_XcieveByte(0xE7);
        PIN_CSN_1();
        PIN_CSN_0();

        //Flush RX FIFO
        nRFCMD_XcieveByte(0xE2);
        PIN_CSN_1();
        PIN_CSN_0();

        //PWR_UP = 1
        nRFCMD_XcieveByte(0x20);
        nRFCMD_XcieveByte(0x3B);
        PIN_CSN_1();
        PIN_CE_1();
    }
#endif // _receiver

#ifdef _transmitter

void transmit_data(char *data)
{
    unsigned char i;

    PIN_CSN_0();

    //clear previous ints
    nRFCMD_XcieveByte(0x27);
    nRFCMD_XcieveByte(0x7E);

    PIN_CSN_1();
    PIN_CSN_0();

    //clear TX fifo
    //the data sheet says that this is supposed to come up 0 after POR,
    //but that doesn't seem to be the case

    nRFCMD_XcieveByte(0xE1);
    PIN_CSN_1();
    PIN_CSN_0();

    //4 byte payload
    nRFCMD_XcieveByte(0xA0);

    outputDataBlock(data,4);

    PIN_CSN_1();
    PIN_CSN_0();

    //PWR_UP = 1
    nRFCMD_XcieveByte(0x20);
    nRFCMD_XcieveByte(0x3A);

    PIN_CSN_1();

    __delay_ms(6);

    //Pulse CE to start transmission
    PIN_CE_1();
}

```

```

        __delay_us(20);
        PIN_CE_0();

        CLRWD();
        do { } while(pool_TX_FIFO());

        PIN_CSN_1();
        PIN_CSN_0();
        //PWR_UP = 0
        nRFCMD_XcieveByte(0x20);
        nRFCMD_XcieveByte(0x38);

        PIN_CSN_1();
    }
#endif // _transmitter

#ifdef _receiver

void receive_data(char *data)
{
    unsigned char j;

    //Read RX payload
    PIN_CSN_0();
    nRFCMD_XcieveByte(0x61);
    inputDataBlock(data,4);
    PIN_CSN_1();

    //Flush RX FIFO

    PIN_CSN_0();

    //reset int
    nRFCMD_XcieveByte(0x27);
    nRFCMD_XcieveByte(0x40);
    PIN_CSN_1();
}

#endif // _receiver

#ifdef _transmitter

void wrTX_adress(byte* data){

    //write TX_ADDRESS register
    PIN_CSN_0();           //CSN low
    nRFCMD_XcieveByte(0x30);
    outputDataBlock(data,5);
    PIN_CSN_1();           //CSN high
}

#endif // _transmitter

#ifdef _receiver

void wrRX_adress(byte* data){

    //write TX_ADDRESS register
    PIN_CSN_0();           //CSN low
    nRFCMD_XcieveByte(0x2A);
    outputDataBlock(data,5);
    PIN_CSN_1();           //CSN high
}

#endif // _receiver

```

Sistema Integrado de Localização de Contentores de Correio

```
#ifdef _transmitter

void rdTX_adress(byte* data){

    //read TX_ADDRESS register
    //Check that values are correct using the MPLAB debugger

    PIN_CSN_0();          //CSN low
    nRFCMD_XcieveByte(0x10);
    inputDataBlock(data, 5);
    PIN_CSN_1();          //CSN high
}

byte pool_TX_FIFO(){
    byte st;

    PIN_CSN_0();
    st = nRFCMD_XcieveByte(0x17); //Read FIFO_STATUS
    st = nRFCMD_XcieveByte(0xff);
    PIN_CSN_1();

    return !(st & 0x10);
}

#endif // _transmitter

#ifdef _receiver

void rdRX_adress(byte* data){

    //read TX_ADDRESS register
    //Check that values are correct using the MPLAB debugger

    PIN_CSN_0();          //CSN low
    nRFCMD_XcieveByte(0x0A);
    inputDataBlock(data, 5);
    PIN_CSN_1();          //CSN high
}

byte pool_RX_DR(){
    byte st;

    PIN_CSN_0();
    st = nRFCMD_XcieveByte(0xff);
    PIN_CSN_1();
    return st & 0x40;
}

byte pool_RX_FIFO(){
    byte st;

    PIN_CSN_0();
    st = nRFCMD_XcieveByte(0x17); //Read FIFO_STATUS
    st = nRFCMD_XcieveByte(0xff);
    PIN_CSN_1();

    return !(st & 0x01);
}

#endif // _receiver

Ficheiro: main.c
```

Sistema Integrado de Localização de Contentores de Correio

```

/*****
* This module contains the main function for PIC12F
* According to macros' configuration a different main program
* is generated (tests, Tx and Rx)
*
*****/

#include <htc.h>

#include "config.h"
#include "gpio.h"
#include "timer.h"
#include "nRF_drv.h"
#include "soft_uart.h"

//WDTEN
//WDTDIS

#ifdef _transmitter
__CONFIG( WDTEN & INTIO & PWRTDIS & MCLRDIS & BORDIS & UNPROTECT); //pic12f6x.h
#endif

#ifdef _receiver
__CONFIG( WDTDIS & INTIO & PWRTDIS & MCLRDIS & BORDIS & UNPROTECT); //pic12f6x.h
#endif

//for pic 12f675

#define _XTAL_FREQ 4000000 //4Mhz clock

byte cmpareNotEqual(byte *p1, byte *p2, byte sz){
    for(sz=sz; sz && (*p1++ == *p2++); sz--);
    return sz;
}

void myDelay100ms(unsigned char n){
    char i;
    for( i = 0; i < n; i++) { __delay_ms(100); /*CLRWDT();*/ }
}

void mySleep576ms(byte n){
    while(n--){
        SLEEP();
    }
}

void showStatus(char stat){
    unsigned char delay;

    if( stat ) delay = 10;
    else delay = 2;
    while(1){
        PIN_LED_0();
        myDelay100ms(delay);
        PIN_LED_1();
        myDelay100ms(delay);
    }
}

//Soft for UART test
// macro _receiver must be defined to shutoff watch dog

//#define yyyy
```

Sistema Integrado de Localização de Contentores de Correio

```
#ifdef yyyy

////////////////////
void main(){
    unsigned char ch = 0;

    ANSEL = 0;
    ADCON0 = 0;
    CMCON = 7;
    VRCON = 0;
    //TRISIO = 0x0E;

    InitSoftUart(); //Also initializes I/O pins

    //echoTxRx();

    while(1){
        ch = UART_Receive();
        //__delay_ms(50);
        UART_Transmit(ch);
    }
}

#endif //yyyy

//Soft for I/O and GPIO functions test

//#define xxxx
#ifdef xxxx
void main(void) //main function to test IO
{
    TRIS5 = 0; //GPIO0 as Output
    TRIS1 = 0; //GPIO1 as Output
    PSA = 1; //Assign prescaler to WatchDog
    PS0 = 1; PS1 = 0; PS2 = 1; // Set prescaler to 32 (576 ms)

    while(1){
        clrGP_5(); //GPIO0 = 0;
        mySleep576ms(2);
        //myDelay100ms(5);
        setGP_5(); //GPIO0 =1;
        mySleep576ms(2);
        //myDelay100ms(5);

        if( GPIO2 == 1 )
            setGP_1();
        else clrGP_1();
    }
}

#endif //xxxx

//Soft for TX or RX according _transmitter and _receiver macros' definition
//If _transmitter macro in config.h is defined, configures the device as a
//beacon tag
//If _receiver macro in config.h is defined, configures the device as a
//receive node

//Macro zzz must be defined to generate the transmitter or receiver software

#define zzz
#ifdef zzz

byte data1[5]={0x11,0x22,0x33,0x44,0x55}; //Variable for node address storage
```

Sistema Integrado de Localização de Contentores de Correio

```
byte data [5]={0,0,0,0,0};           //Variable for Rx/Tx data packet
unsigned int cnt;                     //aux generic variable

byte adCH = 0;    // AD bits 9-8
byte adCL = 0;    // AD bits 7-0
byte seqCnt = 0;  //Sent Tx packets sequence Counter
byte adcCnt = 0;  //Wakeup counter; When zero the Batery voltage is sampled

void main(){
    char i;        // for index variable

    TRISIO = CONFIG_CPU_TRIS; //Configure I/O pins properly

    timer_init();
    PIN_CSN_1();    // CSN high
    PIN_SCK_0();    // SCK low
    PIN_CE_0();     // CE low

    __delay_ms(150); //Wait for system stabilization

    PSA = 1;        //Assign prescaller to WatchDog
    PS0 = 1; PS1 = 0; PS2 = 1; // Set prescaler to 32 (576 ms) sleep time

    //Check for device proper operation
#ifdef _transmitter
    wrTX_adress(data1);
    rdTX_adress(data);
#endif
#ifdef _receiver
    wrRX_adress(data1);
    rdRX_adress(data);
#endif

    if(cmpareNotEqual(data,data1,5)) showStatus(0);

    // for(i=0; i < 5 ; i++)
    //     if(data[i] != data1[i]) showStatus(0);
    //     showStatus(1);

    // Ok device is operating!

    static char msg[4] = { 0x34,0x33,0x32,0x30}; //Msg to transmit ultimo ID=3d

#ifdef _transmitter
    //Module works as a transmitter

    ANSEL = 0x38; // RcOsc + AN3 on GPIO4
    ADCON0 = 0x8C; // Vref=Vdd, AN3
    CMCON = 0x07; //Turn off analog comparator

    TRISIO = CONFIG_CPU_TRIS_tx; //Configure I/O pins
    configure_transmitter();

    __delay_ms(1);

    //Enter beacon loop

    while(1){

        msg[0] = adCH; msg[1] = adCL; msg[2] = seqCnt++;

        transmit_data(msg);
```

Sistema Integrado de Localização de Contentores de Correio

```
if(adcCnt-- == 0){
    adcCnt=10;

    //Sample Vbat
    PIN_CE_1();
    ADCON0 = 0x8D;    // Vref=Vdd, AN3, ADON
    _delay(30);       // ADC setup time

    GODONE = 1; //Start conversion
    while(GODONE == 1); //Wait until conversion done
    PIN_CE_0();

    adcH = ADRESH;    // AD bits 9-8
    adcL = ADRESL;    // AD bits 7-0

    ADCON0 = 0x8C; //Vref=Vdd, AN3, ADOff
    /*
    Vbat = ((1,24 x 1024)/adcHL) V
    */
}

mySleep576ms(3); //Sleep and waikup 3 times (total = 3*576 = 1,728 segundos)
}

#endif // _transmitter

#ifdef _receiver
//Module works as a recceivar

configure_receiver();
__delay_ms(1);

InitSoftUart(); //Also inializes I/O pins

//Enter Rx Loop
while(1){

    if( pool_RX_FIFO() ){           //Msg received
        receive_data(data);        //Ge the message from nrf's FIFO

        //Transmit message to USB module
        for(i=0; i < 4; i++) { txByte(data[i]); data[i]=0;}
        UART_Transmit(13), UART_Transmit(10);

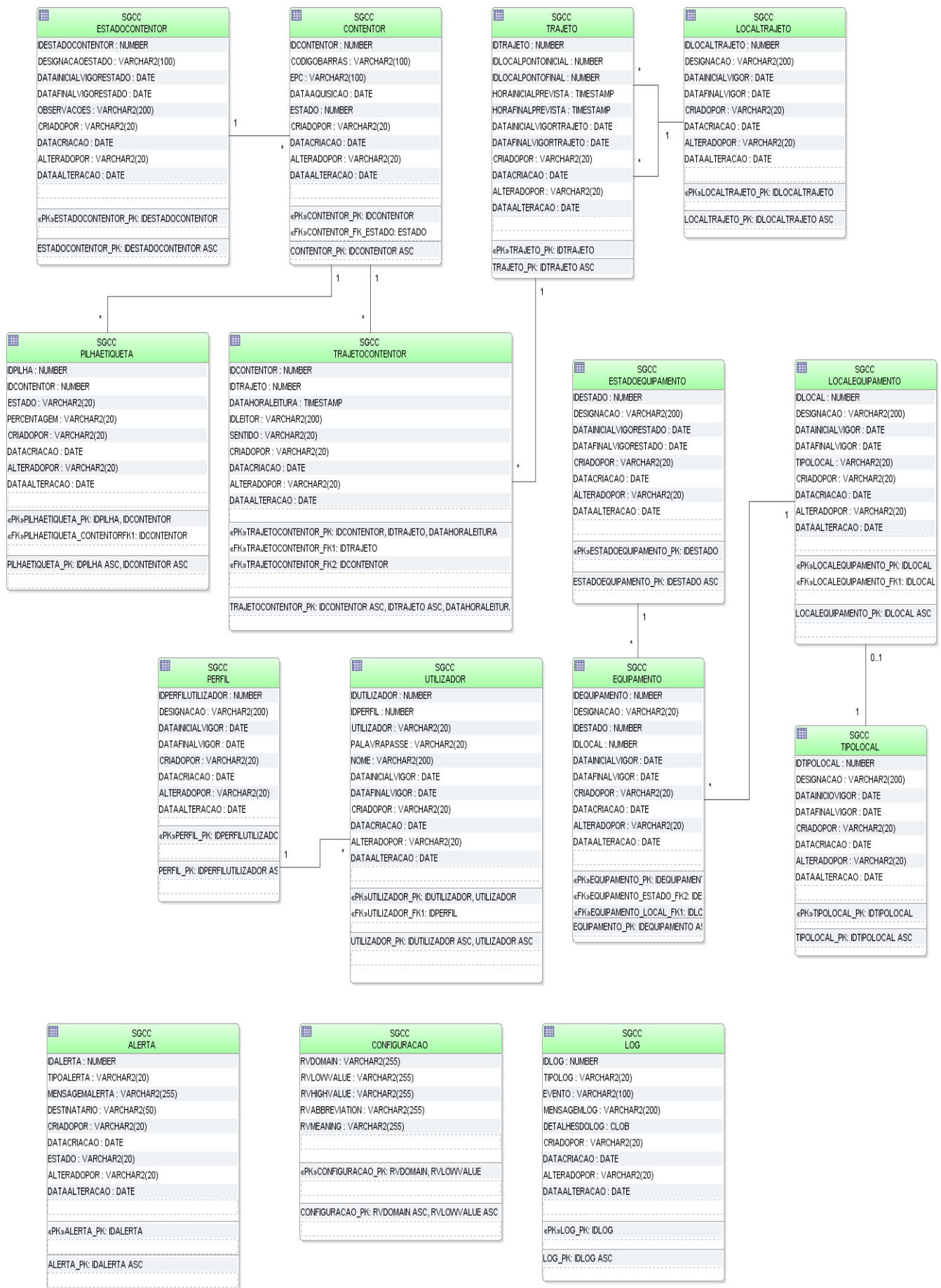
    } //if
} //while

#endif // _receiver
}

#endif //zzz
```

Sistema Integrado de Localização de Contentores de Correio

Anexo B – Modelo de Dados



Anexo C – Programa Java para testar os leitores

```

/*****
* Neste anexo apresenta-se a classe java que implementa a leitura *
* das mensagens na porta série.                                     *
*****/

Ficheiro: ReadCom_evt.java
import jssc.*;

import java.util.Date;
import java.util.Hashtable;
import java.util.concurrent.*;

public class ReadCom_evt {

    int estado = 0;           //Estado do automato
    SerialPort com = null;    //Porta serie
    BlockingQueue<String> Fifo = null;

    private class Reader implements SerialPortEventListener {

        public void serialEvent(SerialPortEvent spe) {
            if(spe.isRXCHAR()) {
                if( spe.getEventValue() > 0) // when RXCHAR - getEventValue()
                                                // returns bytes count in input
                                                // buffer
                {
                    switch(estado){
                        case 0: //Sincronizar com inicio de uma linha
                            try {
                                byte C [] = com.readBytes(1);
                                if(C[0] == 0x0D ) estado = 1; //Prx Est.
                            } catch (/*SerialPort*/Exception e)
                                {System.out.println(e);}
                            break;
                        case 1:
                            try { byte C [] = com.readBytes(1);
                                if(C[0] == 0x0A ) estado = 2; // Prx Est.
                            } catch (SerialPortException e) {}
                            break;
                        case 2: //Estado de sincronização
                            if(spe.getEventValue() > 9){
                                try { String linha = com.readString(10);

                                    Fifo.put(linha);

                                } catch (SerialPortException e) {}
                                catch (InterruptedException e){}
                            }
                            break;
                        default: estado = 0; //forçar sincronização
                    }
                }
            }
        }

        public ReadCom_evt() { //Construtor
            Fifo = new ArrayBlockingQueue<String>(100);
        }

        public boolean comOpen(String com_name) throws SerialPortException{

            com = new SerialPort(com_name);
            if( com.openPort())
            {
                com.setParams(SerialPort.BAUDRATE_19200, SerialPort.DATABITS_8,
                               SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
                com.addEventListener(new Reader());
            }
            return true;
        }
    }
}

```

```

    }

    public boolean comClose() throws SerialPortException{
        boolean res = com.closePort();
        com = null;
        return res;
    }

    public String comReadLine(){
        String res = null;
        try {
            res = Fifo.poll(3, TimeUnit.SECONDS);
        } catch (InterruptedException e) {}
        return res; //String vazio significa timeout
    }

    public static void main(String[] args) {

        //String com_name = "/dev/ttyUSB0";
        String com_name = "COM4"; //Nome da COM por omissão

        if(args.length == 1) com_name = args[0]; //Usar nome da linha de comando

        ReadCom_evt comObj = new ReadCom_evt(); //Criar objecto de interface com a COM

        Hashtable<Integer,tagRecord> tagsIds = new Hashtable<Integer,tagRecord>();

        try {
            comObj.comOpen(com_name);

            for(;;) { //i=0; i <100; i++){

                String L = comObj.comReadLine();

                if (L == null ) L="timeout on com_name\n";
                else
                {
                    //System.out.print(L);

                    int id = getByte(L, 6);
                    int cnt = getByte(L, 4);
                    int adcH = getByte(L,0);
                    int adcL = getByte(L,2);

                    float Vpilha = (1.24f*1024.0f/ (adcH << 8 | adcL));

                    System.out.println( //new Date()+" "+
                        String.format ("id=%2d",id)+
                        String.format (" V=%.2f", Vpilha) +
                        String.format (" Nsq=%3d",cnt )+
                        String.format (" Ncol=%3d",
                            comObj.getNumColisoes(tagsIds,id,cnt)) +
                        " "+
                        String.format("%1$tT,%1$tL ",new Date())
                    );

                }

            }

        } catch (SerialPortException e) {
            e.printStackTrace();
        } finally {
            try { comObj.comClose(); } catch (SerialPortException e)
            { e.printStackTrace(); }
        }

    }

    static int getByte(String S, int i){
        int n = 0;
    }

```

```

        n = getHex(S.charAt(i));
        n = n*16 + getHex(S.charAt(i+1));

        return n;
    }

    static int getHex(char C){
        if( C >='0' && C <= '9') return (int)(C - '0');
        else return (int)(C - 'A')+10;
    }

    private class tagRecord {
        int id;                //Id da tag
        int cnt;                //Contador cíclico de beacons
        int cntColisoese;       //Contador de colisões

        public tagRecord(int id, int cnt){
            this.id = id; this.cnt = cnt; this.cntColisoese = 0;
        }
    }

    int getNumColisoese(Hashtable<Integer,tagRecord> tab, int id, int cnt)
    {
        tagRecord tg = tab.get(new Integer(id));

        if( tg == null) { tab.put(new Integer(id), new tagRecord(id,cnt)); return 0;}

        if( ((tg.cnt + 1) & 0xff) != cnt){
            tg.cntColisoese++;
        }
        tg.cnt = cnt;

        return tg.cntColisoese;
    }
}

```

Anexo D – Shell Script Autoconnectnet.sh

```
#####
#!/bin/sh
#!/etc/init.d/autoconnectnet
### BEGIN INIT INFO
# Provides:          noip
# Required-Start:    $remote_fs $syslog
# Required-Stop:$remote_fs $syslog
# Default-Start:2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Simple script to start a program at boot
# Description:       A simple script ...
### END INIT INFO
case "$1" in
    start)
        sleep 10
        echo "connecting via sakis3g"
        /home/pi/sakis3g --sudo "connect"
        ;;
    stop)
        echo "disconnecting via sakis3g"
        /home/pi/sakis3g --sudo "disconnect"
        ;;
    *)
        echo "Usage: /etc/init.d/autoconnectnet {start|stop}"
        exit 1
        ;;
esac
exit 0

#####
```

Anexo E – Leituras Registadas pelas Etiquetas

Tabela 8 - Leituras obtidas com a etiqueta do identificador 53

Datas	CPLS		CDP 1500	
	Partida	Chegada	Partida	Chegada
29-02-2016	06:30:00			06:40:00
		09:05:50	07:50:00	
01-03-2016		08:47:00	08:25:00	07:20:00
02-03-2016	06:51:00			07:17:00
		08:51:00	08:16:00	
03-03-2016	06:57:00			08:23:00
		08:59:00	08:31:00	
04-03-2016	07:07:00			08:10:00
		10:18:00	08:17:00	
07-03-2016	07:34:00			07:44:00
		09:13:00	08:43:00	
08-03-2016	07:14:00			08:36:00
		09:28:00	08:39:00	
09-03-2016	07:10:00			08:16:00
		09:40:00	08:29:00	
10-03-2016	06:55:00			
		08:55:00		
11-03-2016	06:56:19			08:13:15
		08:59:05	08:16:34	

Tabela 9 - Leituras obtidas com a etiqueta do identificador 54

Datas	CPLS		CDP 1600	
	Partida	Chegada	Partida	Chegada
29-02-2016	06:30:00			06:56:00
		09:05:50	08:00:00	
01-03-2016	07:56:00			08:13:00
		08:42:00	08:35:00	
02-03-2016	06:51:00			06:53:00
		08:42:00	08:36:00	
03-03-2016	07:02:00			07:09:00
		08:54:00	08:46:00	
04-03-2016	06:40:00			06:56:00
		08:45:00	08:32:00	
07-03-2016	07:34:00			07:37:00
		08:52:00	08:38:00	
08-03-2016	07:04:00			07:06:00
		09:10:00	08:59:00	
09-03-2016	07:05:00			07:05:00
		08:51:00	08:30:00	
10-03-2016	06:55:00			06:55:00
		08:45:00	08:30:00	
11-03-2016	06:56:19			06:57:54
		08:51:28	08:35:39	

Tabela 10 - Leituras obtidas com a etiqueta do identificador 55

Datas	CPLS		CDP 1500	
	Partida	Chegada	Partida	Chegada
29-02-2016	06:30:00			06:40:00
		09:05:50	08:00:00	
01-03-2016				07:38:00
		08:46:00	08:35:00	
02-03-2016	06:51:00			07:17:00
		08:51:00	08:31:00	
03-03-2016	07:02:00			08:25:00
		08:59:00	08:41:00	
04-03-2016	07:07:00			07:12:00
		10:18:00	08:27:00	
07-03-2016	07:34:00			07:45:00
		09:14:00	08:53:00	
08-03-2016	07:14:00			08:29:00
		09:28:00	08:49:00	
09-03-2016	07:05:00			08:31:00
		09:40:00	08:44:00	
10-03-2016	06:55:00			08:06:00
		08:56:00	08:35:00	
11-03-2016	06:56:20			07:38:13
		08:59:05	08:26:35	

Tabela 11 - Leituras obtidas com a etiqueta do identificador 56

Datas	CPLS		CDP 1600	
	Partida	Chegada	Partida	Chegada
29-02-2016	06:30:00			07:00:00
		09:05:50	08:00:00	
01-03-2016				07:19:00
		08:42:00	08:35:00	
02-03-2016	06:46:00			06:53:00
		08:44:00	08:36:00	
03-03-2016	06:57:00			07:06:00
		08:54:00	08:46:00	
04-03-2016	04:32:00			04:45:00
		08:45:00	08:32:00	
07-03-2016	07:29:00			07:39:00
		08:52:00	08:38:00	
08-03-2016	06:54:00			07:06:00
		09:10:00	08:59:00	
09-03-2016	06:55:00			07:05:00
		08:51:00	08:30:00	
10-03-2016	06:45:00			06:57:00
		08:45:00	08:30:00	
11-03-2016	06:46:20			06:58:43
		08:53:50	08:35:39	

Anexo F – Pacotes e Classes Java do Leitor

```

/*****
* Módulo principal do software que corre no leitor. A classe RunReader *
* tem o método main que se encarrega de instanciar os objetos das *
* restantes classes e iniciar a execução das tarefas. *
*****/

Ficheiro: RunReader.java
package raspberry.rfid.app;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.logging.Level;
import java.util.logging.Logger;

import jssc.SerialPort;
import raspberry.app.constants.Constants;
import raspberry.exceptions.RaederTokensException;
import raspberry.exceptions.SendTagInformationException;
import raspberry.parse.filter.MessageController;
import raspberry.parse.filter.MessageFilter;
import raspberry.parse.reader.ReaderTokens;
import raspberry.reader.configurations.Configurations;
import raspberry.server.communications.ServerMessage;
import app.testes.GenerateRandomTags;
import contentores.rfid.shared.MessageStagingObject;
import contentores.rfid.shared.ReaderObject;
import contentores.rfid.shared.TagObject;

public class RunReader {

    public RunReader() {
        Fifo = new ArrayBlockingQueue<String>(100);
        FifoSrv = new ArrayBlockingQueue<MessageStagingObject>(100);
    }

    Thread filtro = null; /* Thread à espera de uma linha */
    Thread srv = null;
    Thread controller = null;
    BlockingQueue<String> Fifo = null;
    int estado = 0; /* Estado do automato */
    SerialPort com = null; /* Porta serie */
    BlockingQueue<MessageStagingObject> FifoSrv = null;
    ConcurrentHashMap<String, TagObject> _hashTable = new ConcurrentHashMap<String,
TagObject>();
    MessageStagingObject _messageStagingObject;
    ReaderTokens _rt;
    private final static Logger LOGGER = Logger
.getLogger(Logger.GLOBAL_LOGGER_NAME);
    MessageFilter _mf;
    MessageController _mc;
    Configurations _cf;
    int transactionId = 0;
    ServerMessage _ti;
    GenerateRandomTags grt = new GenerateRandomTags();

    public static void main(String[] args) {

        RunReader rr = new RunReader();
        rr._messageStagingObject = new MessageStagingObject();
        rr._cf = new Configurations();
        rr._rt = new ReaderTokens();
        rr._mf = new MessageFilter();
        rr._mc = new MessageController();
        rr._ti = new ServerMessage();

        /* Threads */

```

Sistema Integrado de Localização de Contentores de Correio

```
FilterTag ft = rr.new FilterTag();
ft.start();

ServerTag st = rr.new ServerTag();
st.start();

ControllerTag ct = rr.new ControllerTag();
ct.start();

}

private class FilterTag extends Thread {

    /* interface com a COM */
    FilterTag() {
    }

    @Override
    public void run() {

        ReadCom_evt comObj = new ReadCom_evt(); /*

        * Criar objecto para
        * leitura na porta série
        */

        try {
            /* comObj.comOpen("/dev/ttyUSB0"); *//*

        * Porta série no RaspBerry
        * PI
        */

            comObj.comOpen("COM8"); /* Porta série no Windows */

            for (;;) {
                String L = comObj.comReadLine(); /*

        * Utilizado para testes
        * iniciais: String L =
        * grt.getRandomTagTokens();
        */

                if (L == null)
                    L = "timeout on com_name\n";
                else {

                    /*
                    * Efetua o parse da string com os tokens e
                    coloca no
                    * objeto
                    */
                    try {
                        _messageStagingObject =

                        _rt.parseTokens(L);

                    } catch (RaederTokensException e) {
                        LOGGER.setLevel(Level.SEVERE);
                        LOGGER.severe(e.getMessage());
                    }

                    _mf.checkTagExistsHashTable(_hashTable,

                    _messageStagingObject.getTagObject(), _mc);

                    if (_mc.getMsgType().equalsIgnoreCase(
                        Constants.MSGTYPEDBFIRST)
                        &&
                    _mc.getMsgBody().equalsIgnoreCase(
```

Sistema Integrado de Localização de Contentores de Correio

```
Constants.MSGBODYDBINSERT)) {

    /* Adicionar A FIFO */
    _messageStagingObject

    .setMessageStatus(Constants.FIRSTMESSAGE);
    FifoSrv.add(_messageStagingObject);

    } else if (_mc.getMsgType().equalsIgnoreCase(
        Constants.MSGTYPEAPP)
        &&
        _mc.getMsgBody().equalsIgnoreCase(
            Constants.MSGBODYAPP)) {

        System.out
            .println(" Tag já
processada na tabela de HASH (repetida)");
    }
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private class ServerTag extends Thread {
    ServerTag() {
    }

    @Override
    public void run() {
        transactionId = 0;
        for (;;) {
            MessageStagingObject _messageStagingObject = new
MessageStagingObject();
            if (!FifoSrv.isEmpty()) {
                _messageStagingObject = FifoSrv.peek();
                if (_messageStagingObject != null) {
                    try {

                        transactionId = _ti

                        .sendMessageToTheServer(_messageStagingObject);

                        if (transactionId > 0) {
                            FifoSrv.poll();
                        }
                    } catch (SendTagInformationException e) {
                        LOGGER.setLevel(Level.SEVERE);
                        LOGGER.severe(e.getMessage());
                    }
                }
            }
        }
    }

    private class ControllerTag extends Thread {
        public ControllerTag() {
        }

        @Override
        public void run() {
            for (;;) {
                try {
```



```

MessageStagingObject _messageStagingObject = new MessageStagingObject();
TagObject _to = new TagObject();
ReaderObject _ro = new ReaderObject();
int adcs[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
try {

    if (tokens != null) {

        int id = getByte(tokens, 6);
        int cnt = getByte(tokens, 4);
        int adcH = getByte(tokens, 0);
        int adcl = getByte(tokens, 2);

        double Vpilha = (1.24f * 1024.0f / (adcH << 8 | adcl));
        adcs[0] = adcH << 8 | adcl;
        String vPilha = null;
        vPilha = String.format("%.2f", Vpilha);
        vPilha = vPilha.replace(",", ".");

        _to.setTagBaterlyLive(Float.valueOf(vPilha));
        _to.setTagID(String.format("%2d", id));
        _to.setTagBarCode("Contentor CTT " + String.format("%2d", id));
        _to.setTimestamp(new java.util.Date());
        _to.setTagNumSequential(cnt);
        _to.setTagType("Transponder ISO CTT");

        _ro.setReaderIdentifier(_cf.getRaederIPHostName().toString());
        _ro.setReaderHostName(_cf.getRaederIPHostName().toString());
        _ro.setReaderIPAdress(_cf.getReaderIPAdress());

        _messageStagingObject.setMessageDate(new java.util.Date());
        _messageStagingObject.setTagObject(_to);
        _messageStagingObject.setReaderObject(_ro);

    }

} catch (Exception e) {
    throw new RaederTokensException(
        "Exception in class: ReaderTokens -> parseTokens: " +
e);
}

return _messageStagingObject;
}

static int getByte(String S, int i) {
    int n = 0;

    n = getHex(S.charAt(i));
    n = n * 16 + getHex(S.charAt(i + 1));

    return n;
}

static int getHex(char C) {
    if (C >= '0' && C <= '9')
        return (int) (C - '0');
    else
        return (int) (C - 'A') + 10;
}
}

/*****
* Classe que tem a responsabilidade de filtrar as mensagens gerindo a
* estrutura em memória: insere novas mensagens, remove mensagens após 10
* minutos sem comunicação.
*****/
Ficheiro: MessageFilter.java
package raspberry.parse.filter;

import java.io.IOException;
import java.util.Enumuration;

```

Sistema Integrado de Localização de Contentores de Correio

```
import java.util.concurrent.ConcurrentHashMap;
import java.util.logging.Logger;

import raspberry.app.constants.Constants;
import raspberry.logging.reader.SetupLog;
import raspberry.reader.configurations.Configurations;
import contentores.rfid.shared.TagObject;

public class MessageFilter {

    boolean keyExists = false;
    int janelaTemporal = 0;

    private final static Logger LOGGER = Logger
        .getLogger(Logger.GLOBAL_LOGGER_NAME);
    Configurations configurations;

    public MessageFilter() {
        configurations = new Configurations();
        try {
            SetupLog.setup("GarbageLogging.txt");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public synchronized void checkTagExistsHashTable(
        ConcurrentHashMap<String, TagObject> _hashTable, TagObject _tag,
        MessageController _mc) {

        try {

            /* A tabela de hash não está vazia */
            if (_hashTable.isEmpty() == false) {

                /* A informação da etiqueta não existe na tabela de hash */
                if (_hashTable.containsKey(_tag.getTagID()) == false) {
                    System.out.println("checkTagExistsHashTable -> "
                        +
                        _hashTable.containsKey(_tag.getTagID()));
                    _tag.setTimestamp(new java.util.Date());
                    _hashTable.put(_tag.getTagID(), _tag);

                    /*
                     * Devolve informação para a chamada com indicação de
                     * para a base de dados
                     */
                    _mc.setMsgType(Constants.MSGTYPEDBFIRST);
                    _mc.setMsgBody(Constants.MSGBODYDBINSERT);

                } else {
                    /*
                     * A informação da etiqueta já existe na tabela de hash,
                     * atualiza informação do objeto e ainda não passou a
                     * temporal
                     */
                    _tag.setTimestamp(new java.util.Date());
                    _tag.setTagNumSequential(_tag.getTagNumSequential());
                    _hashTable.put(_tag.getTagID(), _tag);

                    /*
                     * Devolve informação para a chamada sem necessitar
                     * interação com a base de dados
                     */
                    _mc.setMsgType(Constants.MSGTYPEAPP);
                    _mc.setMsgBody(Constants.MSGBODYAPP);
                }
            }
        }
    }
}
```

```

    }

    } else { /*
        * A tabela de hash está vazia - Inserir informação da
        * etiqueta na tabela de hash
        */

        _tag.setTimestamp(new java.util.Date());

        _hashTable.put(_tag.getTagID(), _tag);

        /*
        * Devolve informação para a chamada que etiqueta com indicação
        * de enviar para a base de dados
        */
        _mc.setMsgType(Constants.MSGTYPEDBFIRST);
        _mc.setMsgBody(Constants.MSGBODYDBINSERT);
    }
} catch (Exception ex) {
    /*
    * Devolve informação para a chamada que etiqueta com indicação de
    * enviar para a base de dados
    */
    _mc.setMsgType(Constants.ERRORMSGPARSEHASH);
    _mc.setMsgBody(Constants.ERRORBODYPARSEHASH);
}

}

public synchronized void garbageTagExpired(
    ConcurrentHashMap<String, TagObject> _hashTable) {
    // configurations = new Configurations();
    /*
    * Obter janela temporal do ficheiro de configuração (em minutos) e
    * converter em milisegundos
    */
    janelaTemporal = 10; // configurations.getTimeWindow();
    int janelaTemporalMilSegundos = ((janelaTemporal * 60) * 1000);

    /* Obter a data e hora atual em milisegundos */
    long tempoAtual = new java.util.Date().getTime();

    /* Variável para guardar o tempo de cada objeto em milisegundos */
    long tempoObjeto = 0;

    /* Caso a tabela de hash não se encontre vazia */
    if (_hashTable.isEmpty() == false) {
        Enumeration<String> keys = _hashTable.keys(); // key do objeto
        while (keys.hasMoreElements()) {
            String key = keys.nextElement();
            tempoObjeto = _hashTable.get(key).getTimestamp().getTime();
            if (tempoAtual - tempoObjeto > janelaTemporalMilSegundos) {
                /* Remover objeto */
                _hashTable.remove(key);
                /* Escreve informação no log. */
                LOGGER.info("(Tag Id expired and removed from hashtable)
> "
                                + key + " TimeStamp Remoção > " +
tempoAtual);
            }
        }
    }
}

public synchronized int existsTagExpired(
    ConcurrentHashMap<String, TagObject> _hashTable) {

    int numTagToRemove = 0;
    /*
    * Obter janela temporal do ficheiro de configuração (em minutos) e
    * converter em milisegundos
    */
    janelaTemporal = 10; // configurations.getTimeWindow();
    int janelaTemporalMilSegundos = ((janelaTemporal * 60) * 1000);

```

Sistema Integrado de Localização de Contentores de Correio

```
/* Obter a data e hora atual em milisegundos */
long tempoAtual = new java.util.Date().getTime();

/* Variável para guardar o tempo de cada objeto em milisegundos */
long tempoObjeto = 0;

/* Caso a tabela de hash não se encontre vazia */
if (_hashTable.isEmpty() == false) {
    Enumeration<String> keys = _hashTable.keys(); // key do objeto
    while (keys.hasMoreElements()) {
        String key = keys.nextElement();
        tempoObjeto = _hashTable.get(key).getTimestamp().getTime();
        if (tempoAtual - tempoObjeto > janelaTemporalMilSegundos) {
            numTagToRemove += 1;
            LOGGER.info("(Tag Id expired and removed from hashtable)
> "
                                + key + " TimeStamp Remoção > " +
tempoAtual);
        }
    }
}
return numTagToRemove;
}

public synchronized void tagExpired(
    ConcurrentHashMap<String, TagObject> _hashTable, String _key) {
    _hashTable.remove(_key);
}

public TagObject getTagLastEvent(
    ConcurrentHashMap<String, TagObject> _hashTable) {
    /*
     * Obter janela temporal do ficheiro de configuração (em minutos) e
     * converter em milisegundos
     */
    TagObject tagObject = null;
    janelaTemporal = configurations.getTimeWindow();
    int janelaTemporalMilSegundos = ((janelaTemporal * 60) * 1000);

    /* Obter a data e hora atual em milisegundos */
    long tempoAtual = new java.util.Date().getTime();

    /* Variável para guardar o tempo de cada objeto em milisegundos */
    long tempoObjeto = 0;

    /* Caso a tabela de hash não se encontre vazia */
    if (_hashTable.isEmpty() == false) {
        Enumeration<String> keys = _hashTable.keys(); // key do objeto
        while (keys.hasMoreElements()) {
            String key = keys.nextElement();
            tempoObjeto = _hashTable.get(key).getTimestamp().getTime();
            if (tempoAtual - tempoObjeto > janelaTemporalMilSegundos) {
                /*
                 * Devolver última informação associada ao objeto antes
                 * deste ser removido
                 */
                tagObject = _hashTable.get(key);
            }
        }
    }
    return tagObject;
}

public boolean checkHashTableEmpty(
    ConcurrentHashMap<String, TagObject> _hashTable) {
    if (_hashTable.isEmpty())
        return true;
    else
        return false;
}
}
```

Sistema Integrado de Localização de Contentores de Correio

```
/* *****  
 * Classe que tem a responsabilidade de gerir os estados das ações a  
 * realizar sobre as mensagens: enviar para a base de dados ou descartar. *  
 * ***** */
```

Ficheiro: *MessageController.java*

```
package raspberry.parse.filter;  
  
public class MessageController {  
  
    private java.util.Date timestamp = null;  
    private String msgId = null;  
    private String msgType = null;  
    private String msgBody = null;  
  
    public MessageController() {  
  
    }  
  
    public java.util.Date getTimestamp() {  
        return timestamp;  
    }  
  
    public void setTimestamp(java.util.Date timestamp) {  
        this.timestamp = timestamp;  
    }  
  
    public String getMsgId() {  
        return msgId;  
    }  
  
    public void setMsgId(String msgId) {  
        this.msgId = msgId;  
    }  
  
    public String getMsgType() {  
        return msgType;  
    }  
  
    public void setMsgType(String msgType) {  
        this.msgType = msgType;  
    }  
  
    public String getMsgBody() {  
        return msgBody;  
    }  
  
    public void setMsgBody(String msgBody) {  
        this.msgBody = msgBody;  
    }  
  
}
```

```
/* *****  
 * Classe que implementa a funcionalidade de criação de um ficheiro *  
 * de log de eventos da aplicação. *  
 * ***** */
```

Ficheiro: *SetupLog.java*

```
package raspberry.logging.reader;  
  
import java.io.IOException;  
import java.util.logging.FileHandler;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
import java.util.logging.SimpleFormatter;  
  
public class SetupLog {  
  
    static private FileHandler fileTxt;  
    static private SimpleFormatter formatterTxt;  
  
    static public void setup(String _logFileName) throws IOException {
```

```

// get the global logger to configure it
Logger logger = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);

logger.setLevel(Level.INFO);
fileTxt = new FileHandler("ReaderLogging.txt", true);

// create a TXT formatter
formatterTxt = new SimpleFormatter();
fileTxt.setFormatter(formatterTxt);
logger.addHandler(fileTxt);
}
}

/*****
* Classe que representa uma exceção do tipo ReaderTokensException *
* e utilizada aquando da leitura e parse das mensagens. *
*****/
Ficheiro: RaederTokensException.java
package raspberry.exceptions;

public class RaederTokensException extends Exception {

    private static final long serialVersionUID = 1L;

    public RaederTokensException(String s) {
        super(s);
    }
}

/*****
* Classe que representa uma exceção do tipo SendTagInformationException *
* e utilizada aquando do envio das mensagens para o servidor. *
*****/
Ficheiro: SendTagInformationException.java
package raspberry.exceptions;

public class SendTagInformationException extends Exception {

    private static final long serialVersionUID = 1L;

    public SendTagInformationException(String s) {
        super(s);
    }
}

/*****
* Classe que contém as variáveis - constantes da aplicação e utilizada *
* pela classe MessageController para atribuição dos respetivos valores. *
*****/
Ficheiro: Constants.java
package raspberry.app.constants;

public class Constants {

    public static final String MSGTYPEDBFIRST = "DBFIRST";
    public static final String MSGTYPEDBLAST = "DBLAST";
    public static final String MSGTYPEAPP = "APP";
    public static final String MSGBODYAPP = "DO_NOthing";
    public static final String MSGBODYDBINSERT = "DB_INSERT";
    public static final String MSGBODYDBUPDATE = "DB_UPDATE";
    public static final String FIRSTMESSAGE = "Entrada";
    public static final String MIDDLEMESSAGE = "MIDDLE_MESSAGE";
    public static final String LASTMESSAGE = "Saída";

    public static final String ERRORMSGPARSEHASH = "EXCEPTION";
    public static final String ERRORBODYPARSEHASH = "ERROR_PROCESSING_TAG";
}

```

```
public static final String TAGSTATEWAIT = "TAG_STATE_WAIT";  
public static final String TAGSTATEPROCESSED = "TAG_STATE_PROCESSED";  
}
```

Anexo G – Pacotes e Classes *Middleware*

```

/*****
 * Classe que estende as funcionalidades da classe HttpServlet. Disponibiliza
 * um endereço para ser invocada e receber pedidos de receção das mensagens.
 * Efetua a desserialização destas e envia pedido de inserção para a classe
 * que gere as transações com a base de dados.
 *****/
Ficheiro: ReceiveReaderMessages.java
package reader.rfid.messages;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.StringReader;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.transform.stream.StreamSource;

import server.rfid.dao.MessageStagingDao;
import contentores.rfid.shared.MessageStagingObject;

/**
 * Servlet implementation class ReceiveReaderMessages
 */
@WebServlet("/ReceiveReaderMessages")
public class ReceiveReaderMessages extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        MessageStagingObject messageStagingObject = null;
        MessageStagingDao messageStagingDao = new MessageStagingDao();
        int messageStagingId = 0;
        PrintWriter out = null;
        String inputLine = null;
        StringBuffer sb = new StringBuffer();
        BufferedReader is = null;

        out = response.getWriter();
        response.setContentType("text/text");

        try {

            is = new BufferedReader(new InputStreamReader(
                request.getInputStream()));

            while ((inputLine = is.readLine()) != null) {

```

Sistema Integrado de Localização de Contentores de Correio

```
        sb.append(inputLine.trim());
    }

    /* Deserializa o xml recebido para o objeto java */
    if (sb.length() > 0) {

        JAXBContext jc = JAXBContext
            .newInstance(MessageStagingObject.class);
        Unmarshaller u = jc.createUnmarshaller();

        messageStagingObject = (MessageStagingObject) u
            .unmarshal(new StreamSource(new StringReader(sb
                .toString())));

        messageStagingId = messageStagingDao
            .createMessageStaging(messageStagingObject);

        /*
         * Resposta para o pedido (cliente) do reader. Devolve response
         * status e id da transação
         */
        out.println(messageStagingId);

        out.flush();

    }

} catch (Exception ex) {
    ex.printStackTrace();
    /* Ocorreu erro, devolvo para chamada o código de erro */
    System.out.println("Erro : " + ex.getMessage());
    messageStagingId = -10;
    out.println(messageStagingId);
    out.flush();

} finally {
    /* Close objects */
    is.close();
    out.close();
}

}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
 * response)
 */
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}

}

/*****
 * Classe que representa uma fábrica de objetos da entidade da base de dados *
 * MessageStaging e os disponibiliza no contexto de uma determinada sessão. *
 *****/
Ficheiro: MessageStagingSessionFactory.java
package server.rfid.dao;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class MessageStagingSessionFactory {

    private static final SessionFactory sessionFactory;

    static {
        try {
            /* Create the SessionFactory from hibernate.cfg.xml */
            sessionFactory = new Configuration().configure()
                .addClass(MessageStaging.class).buildSessionFactory();
        }
    }
}
```

```

    } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
    }
}

public static SessionFactory getSessionFactory() {
    return sessionFactory;
}
}

```

```

/*****
* Classe que implementa os mecanismos disponíveis da framework Hibernate para *
* gerir as transações com a base de dados. *
*****/

```

Ficheiro: MessageStagingDao.java

```

package server.rfid.dao;

import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import contentores.rfid.shared.MessageStagingObject;

public class MessageStagingDao {

    static Logger errorLog = null;
    static Logger infoLog = null;

    public int createMessageStaging(MessageStagingObject messageStagingObject) {
        System.out.println("Dentro de createMessageStaging ");
        Session session = MessageStagingSessionFactory.getSessionFactory()
            .getCurrentSession();
        int _messageStagingId = 0;
        MessageStaging messageStaging = new MessageStaging();
        Transaction tx = null;
        errorLog = Logger.getLogger(this.getClass().getName());
        infoLog = Logger.getLogger(this.getClass().getName());

        try {

            tx = session.beginTransaction();

            messageStaging.setMessage_date(messageStagingObject
                .getMessageDate());
            messageStaging.setMessage_stagingstatus(messageStagingObject
                .getMessageStatus());
            messageStaging.setReader_identifier(messageStagingObject
                .getReaderObject().getReaderIdentifier());
            messageStaging.setReader_ipadress(messageStagingObject
                .getReaderObject().getReaderIPAdress());
            messageStaging.setReader_hostname(messageStagingObject
                .getReaderObject().getReaderHostName());
            messageStaging.setTag_type(messageStagingObject.getTagObject()
                .getTagType());
            messageStaging.setTag_id(messageStagingObject.getTagObject()
                .getTagID());
            messageStaging.setTag_batterylive(Double
                .toString(messageStagingObject.getTagObject()
                    .getTagBatteryLive()));
            messageStaging.setTag_barcode(messageStagingObject.getTagObject()
                .getTagBarCode());
            messageStaging.setTag_numsequential(messageStagingObject
                .getTagObject().getTagNumSequential());

            _messageStagingId = (Integer) session.save(messageStaging);

            tx.commit();

```

Sistema Integrado de Localização de Contentores de Correio

```
        infoLog.setLevel(Level.TRACE);
        infoLog.trace(messageStagingObject.getTagObject().getTagID());

    } catch (HibernateException e) {
        System.out.println("Erro: " + e.getMessage());
        errorLog.setLevel(Level.ERROR);
        errorLog.error(e.getMessage());
        if (tx != null)
            tx.rollback();
    } finally {
        if (session.isOpen())
            session.close();
    }

    /*
     * Devolve o id da mensagem introduzida na base de dados - se houver
     * erro devolve um numero negativo correspondente ao erro
     */
    return _messageStagingId;
}

}

/*****
* Classe com anotações que representa a entidade da base de dados no modelo ORM.
*****/
Ficheiro: MessageStaging.java
package server.rfid.dao;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "messagestaging")
public class MessageStaging {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    int id;
    @Column(name = "message_date")
    Date message_date;
    @Column(name = "message_stagingstatus")
    String message_stagingstatus;
    @Column(name = "reader_identifier")
    String reader_identifier;
    @Column(name = "reader_ipadress")
    String reader_ipadress;
    @Column(name = "reader_hostname")
    String reader_hostname;
    @Column(name = "tag_type")
    String tag_type;
    @Column(name = "tag_id")
    String tag_id;
    @Column(name = "tag_baterylive")
    String tag_baterylive;
    @Column(name = "tag_barcode")
    String tag_barcode;
    @Column(name = "tag_numsequential")
    int tag_numsequential;

    public int getId() {
        return id;
    }

    public void setId(int id) {
```

```
        this.id = id;
    }

    public String getMessage_stagingstatus() {
        return message_stagingstatus;
    }

    public void setMessage_stagingstatus(String message_stagingstatus) {
        this.message_stagingstatus = message_stagingstatus;
    }

    public Date getMessage_date() {
        return message_date;
    }

    public void setMessage_date(Date message_date) {
        this.message_date = message_date;
    }

    public String getReader_identifier() {
        return reader_identifier;
    }

    public void setReader_identifier(String reader_identifier) {
        this.reader_identifier = reader_identifier;
    }

    public String getReader_ipaddress() {
        return reader_ipaddress;
    }

    public void setReader_ipaddress(String reader_ipaddress) {
        this.reader_ipaddress = reader_ipaddress;
    }

    public String getReader_hostname() {
        return reader_hostname;
    }

    public void setReader_hostname(String reader_hostname) {
        this.reader_hostname = reader_hostname;
    }

    public String getTag_type() {
        return tag_type;
    }

    public void setTag_type(String tag_type) {
        this.tag_type = tag_type;
    }

    public String getTag_id() {
        return tag_id;
    }

    public void setTag_id(String tag_id) {
        this.tag_id = tag_id;
    }

    public String getTag_baterylive() {
        return tag_baterylive;
    }

    public void setTag_baterylive(String tag_baterylive) {
        this.tag_baterylive = tag_baterylive;
    }

    public String getTag_barcode() {
        return tag_barcode;
    }

    public void setTag_barcode(String tag_barcode) {
        this.tag_barcode = tag_barcode;
    }
}
```

```
    }  
  
    public int getTag_numsequential() {  
        return tag_numsequential;  
    }  
  
    public void setTag_numsequential(int tag_numsequential) {  
        this.tag_numsequential = tag_numsequential;  
    }  
}
```