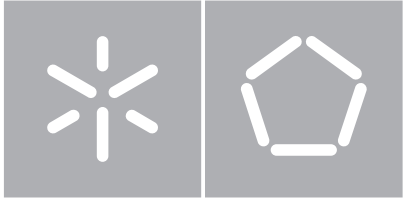
Universidade do Minho
Escola de Engenharia

Nelson Francisco Sousa Cunha

Climawin Demo: protocolo de comunicações, controlador geral e atuação

Outubro de 2014



Universidade do Minho
Escola de Engenharia

Nelson Francisco Sousa Cunha

Climawin Demo: protocolo de comunicações, controlador geral e atuação

Dissertação de Mestrado
Mestrado Engenharia Eletrónica Industrial e Computadores

Trabalho realizado sob orientação de

Professor Jorge Cabral

Outubro de 2014

Agradecimentos

A minha família mais próxima tem tanto mérito como eu nesta dissertação. Portanto, é dedicada ao meu pai e à minha mãe por nunca me ter faltado nada, e ao meu irmão, pela companhia e cumplicidade no dia-a-dia.

Como não poderia deixar de ser, um enorme agradecimento ao meu professor e orientador Doutor Jorge Cabral, por depositar confiança e acreditar em mim.

Esta tese não seria possível sem a ajuda, companheirismo e orientação de colegas no *Embedded Systems Research Group*, a todos, muito obrigado, em especial ao Hugo Gomes por me ter acompanhado ao longo de todo o percurso.

Obrigado a todos, e o que eu mais desejo é que sejam felizes!

Resumo

O *Climawin* [1] [2] é um projeto que visa aumentar a eficiência energética num espaço interior, somando prémios em exposições internacionais. O controlo da entrada e saída do ar numa divisão interior, através de janelas com válvulas e estores, e o pré-aquecimento/arrefecimento do ar são ações consequentes da constante monitorização da temperatura/humidade e níveis de concentração de CO_2. Todo o sistema desenvolvido é *power-aware*, ou seja, controla o consumo de energia e procura minimizá-lo. Este controlo permite melhorar o ambiente no interior de uma habitação, sem que se consuma energia de forma excessiva, como acontece quando, numa habitação mais comum, o utilizador tem total controlo manual da entrada e saída do ar nas divisões de sua casa.

A *cloud* computacional tem crescido nos últimos anos em serviços com variadíssimas aplicações e em diferentes áreas (domótica, industrial, automóvel, militar, etc). No que toca ao utilizador, as vantagens mais imediatas são a flexibilidade, simplicidade de acesso [3] e confidencialidade [4]. Do ponto de vista de produto comercial faz todo o sentido dotar o projeto *Climawin* das características que a nuvem computacional disponibiliza.

O *Climawin* já conheceu o seu primeiro protótipo, desenvolvido por investigadores na Universidade do Minho. No entanto, o acesso do utilizador ao produto não explorava serviços na *cloud*. Desta forma, foi feita uma nova proposta no sentido de desenvolver uma nova versão de demonstração do *Climawin*, o *Climawin demo*. Nesta versão serão utilizados vários componentes, de diferentes camadas de desenvolvimento e aplicação desenvolvidos por uma equipa de investigadores.

Posto isto, esta dissertação, no âmbito do *Climawin demo*, propõe o estudo e o *refactoring* de parte da camada do *Climawin* que acede à *cloud*. Este acesso é feito através de um controlador central, que por sua vez acede a um *gateway* que está ligado à rede. O papel do controlador central, sucintamente, fará parte do encaminhamento das tramas de todos os dispositivos *Climawin* para a *cloud* e vice-versa.

No entanto, o foco do *Climawin* é naturalmente a janela, o *end-device* do sistema. Desta forma é essencial que a mesma se comporte consistentemente de maneira previsível e fiável. Tendo isto em conta, esta dissertação propõe também melhorar o controlo da janela *Climawin*. Quer isto dizer, que a atuação nos estores e nas válvulas será também revista e melhorada.

Resumidamente, a proposta desta dissertação é estudar e utilizar as possibilidades fornecidas pela *cloud* para melhorar e expandir os serviços disponibilizados pelo *Climawin* aos utilizadores e, desta maneira, possibilitar variadíssimos modos de interagir com o sistema. A partir desse momento poder-se-ão desenvolver aplicações para dispositivos móveis, portáteis ou *desktops*, e até um eventual *website* que permita acesso ao sistema. A interação do *Climawin* com diferentes soluções de domótica fica também facilitada e é possível construir sistemas mais complexos ao agrado do cliente. O melhoramento do controlo da janela é uma adição essencial ao objetivo do *Climawin*: produzir a janela para demonstrar em três diferentes locais na Europa (Alemanha, Dinamarca e Irlanda) e eventualmente introduzir o sistema no mercado.

Palavras-chave: *Cloud computing*, domótica, janela inteligente.

Abstract

Cloud based services have been exploited through the years in great numbers in order to develop different domotic, military, automobile or industrial applications. In the user's perspective, the most obvious advantages are flexibility, easy access and safety.

With these properties in mind, it is assumed that *Climawin* should explore the cloud. *Climawin* is a project that aims to improve energy efficiency in a building. The air flow control, achieved through windows, and its cooling and preheating are consequences of the constant monitoring and control of temperature, humidity and CO_2 concentration levels. All the system's components are low-power driven, improving the air quality in the interior without spending unnecessary energy.

This project has already met its first prototype, developed by Universidade do Minho researchers. It played its part, and did everything proposed. However, it did not explore the many possibilities given by the cloud.

So, this thesis proposal is to change or improve the *Climawin* access node to the web in order to prepare it to reply correctly to the new services that will be provided to the user.

The main focus of *Climawin* is the window, though. In order to have a good, consistent and responding window, this thesis also purposes improving its actuation.

Resuming it all, this thesis proposal is to study, analyse and use the cloud's many advantages so that it improves *Climawin's* services to the user. From the moment that's implemented it's possible to develop different applications for mobile equipment, laptops or desktops, or even a website to the user's convenience. The improvement on the window actuation is essencial for the user's satisfaction, too.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	*Application Programming Interface*
ERP	*Enocean Radio Protocol*
ESP	*Enocean Serial Protocol*
GUI	*Graphic User Interface*
IaaS	*Infastructure as a Service*
IoT	*Internet of Things*
LSB	*Less Significant Byte*
MSB	*Most Significant Byte*
PaaS	*Platform as a Service*
RF	*Radio Frequency*
SaaS	*Software as a Service*
SO	*Sistema Operativo*
TWI	*Two Wire Interface*
UI	*User Interface*
VHG	*Virtual Home Gateway*
WHA	*Wireless Home Automation*
WSN	*Wireless sensor network*
ZC	*Zone Controller*

Capítulo 1

Introdução

Através do controlo do fluxo do ar que atravessa janelas controladas eletronicamente, o *Climawin* pretende melhorar a eficiência energética de um espaço interior. Por intermédio de sensores de temperatura, humidade e luminosidade, as janelas controlam estes mesmos fatores naturais de acordo com as condições ideais para habitação. Com válvulas e persianas as janelas atuam de forma a melhorar o ambiente interior, tendo sempre em conta propriedades como o *energy-harvesting* e *low-power*.

Para potenciar a flexibilidade e acessibilidade do produto, a *cloud* computacional foi considerada para o *Climawin*. Nos dias que correm, devido à crescente concorrência, as empresas tencionam desenvolver os seus produtos rapidamente e de forma eficaz, pelo que o *time to market* é considerado como uma das principais condicionantes de sucesso de um produto. Posto isto, a *cloud*, através dos serviços que fornece [8], permite que o mercado seja mais flexível, mais fácil de abordar e com custos reduzidos. Um fornecedor de serviço *cloud* está encarregue de cumprir o contrato feito com o cliente, sendo que este parte do princípio que o bom desempenho e disponibilidade do seu produto estão garantidos.

A abstração dos mecanismos de acesso que um serviço na *cloud* potencia é um dos fatores mais relevantes para o desenvolvimento de sistemas que pretendam fomentar a flexibilidade dos mesmos e a possibilidade da utilização desse mesmo serviço por outras plataformas, desde que tenham acesso à nuvem computacional. Desta forma, um hipotético desenvolvedor de um produto para o mercado tem apenas que se preocupar em permitir que o seu sistema tenha acesso à *cloud* e utilize os seus serviços. A partir do momento que o mesmo tem acesso à rede, a visibilidade e a escalabilidade são propriedades garantidas deste mesmo sistema.

A simplificada forma de acesso a serviços alojados na *cloud* é a propriedade que mais relevo tem para o utilizador, porque este mesmo pode, com o devido meio de acesso ao serviço, usufruir do *on-demand self-service*. Isto é, desde que tenha acesso à *internet*, o utilizador pode interagir com o sistema.

Estas características da *cloud* associadas ao conforto e controlo da domótica, podem despoletar ideias e produtos interessantes. Acesso remoto em qualquer local, boa eficiência energética e conforto são propriedades muito em voga nos países do primeiro mundo, que têm possibilidades de dispensar recursos para usufruir destes serviços.

O *Climawin* é um projeto que tenciona trazer essas mesmas vantagens aos seus clientes. Este sistema encaixa perfeitamente no propósito da união destes dois mundos. O projeto tem como principal objetivo melhorar a eficiência energética de interiores de habitações. Se a esse processo estiverem aliadas as características e as possibilidades que a *cloud* fornece, o produto fica mais flexível e acessível ao cliente. Com a *cloud* o interface com o *Climawin* abrange mais possibilidades e permite o desenvolvimento de diversas aplicações para aceder aos seus serviços.

Como já fora referido, a estrutura utilizada previamente no *Climawin* não permitia que o sistema respondesse a pedidos provenientes da *cloud*. A UI (*User Interface*) desenvolvida corria numa máquina local (PC *desktop* ou portátil), que por sua vez tinha ligação direta ao sistema através do controlador central da zona, ou divisão interior. É portanto necessária uma adaptação do controlador central para que o interface com a *cloud* seja possível. Serão implementados novos serviços no *Climawin* que a *cloud* poderá explorar, e o novo controlador central fará parte do processo.

No que diz respeito à janela, esta também sofrerá alterações e é também proposta desta dissertação rever e melhorar a atuação na mesma, de forma a ficar mais consistente ao agrado do utilizador.

1.1 Objetivos

A proposta de dissertação tem como objetivo remodelar o controlador central do *Climawin*, doravante denominado *Zone Controller* (ZC) e também refazer a atuação na janela. Este estudo e *refactoring* permitirão um melhor e mais conveniente acesso por parte do utilizador às janelas *Climawin* e também um controlo

mais eficaz das mesmas.

Os objetivos serão distribuídos nas seguintes categorias:

- Especificação dos serviços que o *Climawin* fornecerá à *cloud*;
- Respetiva adaptação do ZC de modo a responder adequadamente a estes serviços;
- Implementação de uma rede de ZCs: comunicação entre diferentes controladores;
- Alterações no *software* do ZC de modo a cumprir requisitos funcionais e não funcionais (desempenho);
- Especificação das funcionalidades de atuação da janela;
- Especificação da comunicação entre o módulo de controlo e o módulo de atuação;
- Implementação do software de atuação.

Capítulo 2

Estado da Arte

Neste capítulo irão ser apresentados alguns temas relacionados com a dissertação, tais como os modelos de serviços fornecidos pela *cloud* e os protocolos de comunicação sem fios baseados na tecnologia *Enocean*. Ênfase também para sistemas de domótica já existentes com foco em projetos que tenham implementado acesso à *cloud*, bem como maneiras de conectar diferentes componentes físicos à nuvem computacional. Finalmente, será também abordado o tema de controlo de janelas.

2.1 Conceitos teóricos

Nesta secção apresentam-se alguns conceitos teóricos essenciais para a compreensão da dissertação. São assuntos relacionados com a área de aplicação da tese, direta ou indiretamente.

2.1.1 *Cloud computing*

Cloud computing, ou apenas *cloud*, é um conceito muito em voga na recente década. Classifica-se como um conjunto de recursos de *hardware*, quer a nível de armazenamento quer processamento, a correr serviços de *software*, contratualizados com o cliente, através da *internet*. O crescimento da quantidade de armazenamento e processamento nos últimos anos têm criado bastantes oportunidades para o mercado receber fornecedores deste tipo de serviços. O facto de não ser necessário dispensar recursos em *hardware state-of-the-art*, evitando assim os custos de

manutenção, e a segurança que o fornecedor garante [9] , são aspetos interessantes no que diz respeito a clientes que querem "alugar" *hardware* para fornecer o seu serviço. Através do modelo *pay-per-use* o cliente pode usufruir do serviço quando entender que tem necessidade de recursos, podendo desta maneira, ter melhor controlo sobre os custos a curto prazo. Numa outra perspetiva , não esquecer que o serviço fornecido pode também ser escalado, ou seja, se for necessário mais capacidade (por exemplo em comunicações, armazenamento ou processamento), pode-se comunicar esta necessidade ao fornecedor e facilmente aumentar os recursos que se podem explorar.

Existem três modelos de serviços que a *cloud* pode fornecer: IaaS, PaaS e SaaS. Quantidade de camadas abstraídas e fornecidas pelo servidor em ordem crescente.

- IaaS - Infastructure as a Service

  Este modelo oferece a virtualização do *hardware*, assim como *networking* , armazenamento e servidores. Evita, portanto, custos em *hardware*, manutenção e pessoal para garantir o seu bom funcionamento. Ao mesmo tempo, é uma garantia num eventual crescimento da necessidade do serviço, visto que é garantido que a escala do mesmo é assegurada pelo fornecedor. É vantajoso para, por exemplo, uma companhia que necessite de armazenamento extra numa ocasião em que a procura é grande.

- PaaS - Platform as a Service

  Este modelo cobre todas as camadas da responsabilidade do fornecedor do serviço no modelo IaaS e ainda providencia um ambiente de desenvolvimento. Através do *middleware* [10] é possível desenvolver programas e distribui-los através da *cloud*. Ideal para negócios relacionados com o desenvolvimento de aplicações ou *softwares* que pretendam ser heterogéneos e abstratos da plataforma alvo, pelo que é possível fazer e testar *releases* constantemente.

- SaaS - Software as a Service

  A última camada dos serviços *cloud* convencionais está destinada ao público e é por isso a mais explorada e consumida no mercado [11] [12]. O cliente não tem qualquer controlo sobre a aplicação que está a utilizar. Quer isto dizer, que apenas consome os dados e utiliza o *output* de um *software* já desenvolvido para seu benefício. Utilizadores do serviço tem garantia de compatibilidade visto que todos usam o mesmo *software*.

2.1.2 Redes de comunicações sem fios

As redes de comunicações sem fios estão em todo o lado. Milhões de pessoas em todo o mundo usam esta tecnologia revolucionária que permite milhares de ideias tornarem-se realidade. Existem variadas tecnologias que permitem diferentes dispositivos interligarem-se e criarem redes. Umas mais complexas, outras mais simples, com o propósito de flexibilidade e facilidade de acesso e sempre com o conceito de *low-power* tido em conta.

Uma rede de comunicação sem fios consiste em dispositivos a recolher informação do meio que os rodeia através de sensores, os nós ou *end-devices*, a cooperar e comunicar entre si. Estes nós são normalmente constituídos por uma simples unidade de processamento, memória, bateria e um *transceiver*, para enviar ou receber dados sem fios. Algumas WSN ("Wireless sensor network") tem nós com módulos "energy-harvesting" para aproveitar o ambiente que os rodeia e utilizar diversos métodos de recolha de energia. Um dos maiores problemas das WSN é gerir o consumo destes nós, de forma a aumentar a duração da bateria e consequentemente o tempo de vida de um destes dispositivos. Várias técnicas são utilizadas para tal, como por exemplo pequenas janelas temporais de transmissão ou receção de dados, filtragem de dados, modos *power-down* ou *sleep* e também o controlo do alcance do sinal [13].

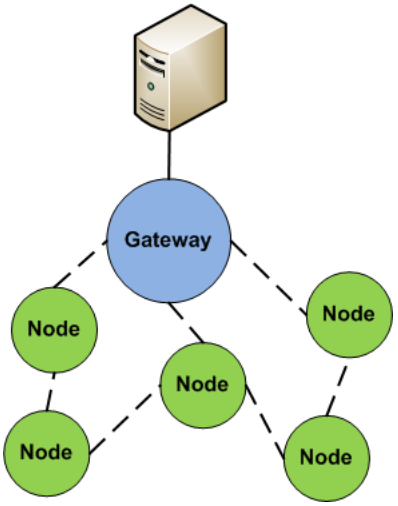


Figura 2.1: Arquitetura tipo de uma WSN

A figura **2.1** representa uma arquitetura típica de uma rede sem fios. Os nós podem comunicar entre si, ou diretamente com o *gateway*. A escolha do destino ou fonte de dados depende, entre outros fatores, da distância e do alcance do

sinal, pelo que se um nó está muito distanciado de o *gateway*, outro nó pode reencaminhar o sinal, ou então um *router* também o poderá fazer. O *gateway* é a ligação da rede de sensores a outras redes ou até à *cloud*. É, de forma sucinta, o ponto de acesso a toda a rede de sensores que este abrange. Se este estiver ligado a um computador, como é o caso da figura **2.1**, ou a um sistema embebido capaz de interpretar os dados, a rede pode ser totalmente controlada a partir dessa ligação. O interface com o sistema pode tomar variadas formas, dependendo da ligação que for escolhida.

O termo *Wireless Home Automation* (WHA) [14] surgiu com o crescimento destas tecnologias e a necessidade de tornar os edifícios mais inteligentes e autónomos, tanto do ponto de vista energético e ambiental, como do ponto de vista da comodidade e segurança. As instalações WHA são flexíveis, com dispositivos "plug-and-play" e menos custosas, ao contrário das instalações conservativas baseadas em cablagem excessiva. As diferentes companhias que apostam neste mercado têm diferentes práticas e princípios, mas normalmente têm em comum a necessidade fornecer ao cliente produtos "low-power", com pequeno impacto no ambiente, e também serviços que requerem pouca, ou quase nenhuma manutenção. Nesta secção serão demonstradas algumas tecnologias e protocolos *wireless* de curto/médio alcance, usualmente aplicados em projetos de domótica.

ZigBee

Protocolo baseado no *standard* IEEE 802.15.4, que define as duas primeiras camadas do OSI (*Open systems interconnection*): as camadas física e de acesso à rede. Este *standard* foi especialmente concebido com o objetivo de criar redes *low-cost*, *low-power* e com taxas de transmissão baixas. Tal como definido neste *standard*, o ZigBee opera nas frequências de 2.4GHz, 915MHz (América do Norte), 920 MHz (Japão) e 868 MHz (Europa) com um alcance entre 10m e 100m [14]. Resulta da *ZigBee Aliance*, que se define como um conjunto de companhias que têm como objetivo uniformizar a criação de diferentes soluções *wireless* para aplicações industriais e residenciais, como o caso da domótica. O *ZigBee* visa criar uma rede com grande densidade de dispositivos (até 64000) com consumos reduzidos e produtos *low-cost*, proporcionando assim uma rede *wireless* mais acessível e com um protocolo simples de implementar. A hierarquia da comunicação está dividida em três diferentes dispositivos: coordenador, *router* e *end device* [15]. O primeiro tem como principal papel ligar a rede de sensores à *internet*. O *router* aumenta o alcance da rede e encaminha as mensagens para os dispositivos requeridos. E finalmente, o *end device* é o dispositivo que obtêm informação do ambiente que

o rodeia recorrendo a sensores, ou então é um atuador que recebe ordens para efetuar alguma alteração. Em certas aplicações pode ter ambos papéis.

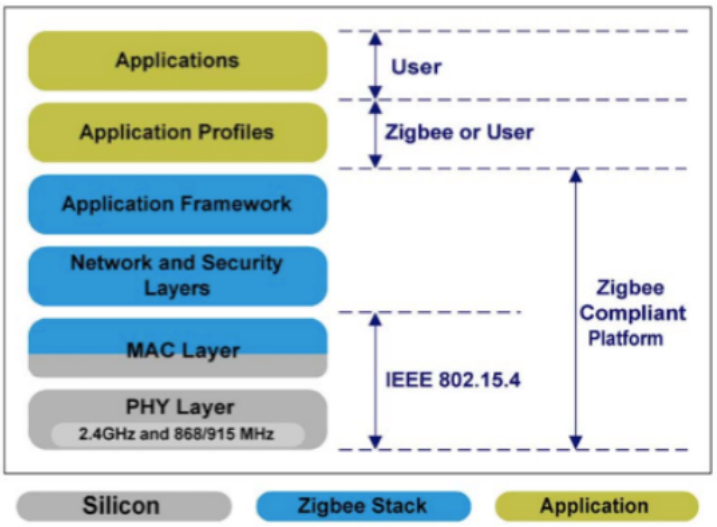


Figura 2.2: Camadas estruturais do *ZigBee* [5]

A figura **2.2** representa o protocolo ZigBee segundo o modelo OSI (*Open systems interconnection*) [5]. Como o protocolo é baseado no *standard* do IEEE, as duas camadas inferiores, física e acesso ao meio, são definidas pelo mesmo. As outras camadas são customizadas pelo protocolo, à exceção da camada de aplicação, que está a cargo do programador/*system designer* ou do vendedor do produto.

Z-Wave

Este protocolo foi criado pela *Z-Wave Alliance* que é composta por diversos fabricadores de produtos de automação residencial, que, em 2005, estariam pouco satisfeitos com a separação e isolamento dos fabricadores no que diz respeito à implementação de protocolos singulares e consequentemente incompatíveis entre diferentes aplicações. Opera a uma frequência de 908MHz (América do Norte) e 868MHz (Europa) e 921.42MHz (Austrália e Japão) e tem um alcance de 30m em interiores e 100m em exteriores [14]. A capacidade máxima de dispositivos da rede é de 232. No entanto, se forem adicionados *gateways Z-Wave*, estes números podem aumentar devido à possibilidade destes *gateways* poderem ligar diferentes redes.

Os componentes da rede estão divididos em duas classes: os controladores e

os escravos (*slaves*). Os escravos têm um papel idêntico ao *end-device* do *ZigBee*: sensor e/ou atuador. Os controladores estão divididos entre os primários e os secundários. O secundário tem sub-redes e é capaz de enviar informações para os dispositivos que lhe estão associados. O controlador primário tem a tabela de *routing*, quer isto dizer que conhece todos os dispositivos da rede e implementa todas as funcionalidades do secundário além de ser capaz de remover e adicionar outros controladores à rede.

Wi-Fi

O *Wi-Fi* é um outro protocolo sem fios geralmente utilizado como meio de acesso à *internet* pelos computadores portáteis. Tal como o caso do ZigBee, este protocolo é definido por um *standard* do IEEE que rege apenas as camadas físicas e de acesso ao meio.

Além da utilização mais comum deste protocolo, a ligação à *internet*, este está a crescer em domótica, principalmente devido à generalidade do mesmo, criando redes complexas de equipamentos ligados entre si e a rede. Ao contrário dos anteriormente referidos, este protocolo está longe de ser *low-power*[14], logo não é o ideal para este tipo de aplicações. No entanto, a maioria das casas tem uma rede wireless *Wi-Fi*, e esta disponibilidade facilita o negócio, pelo que a adição *plug-and-play* destes dispositivos fica facilitada, em deterioramento de outros fatores como o consumo energético. Regido pelo standard 802.11a/b/g do IEEE, operando a uma frequência de 2.4GHz e nalguns casos de 5GHz, esta rede tem um peso significativo no mercado pelos motivos já referidos. Os exemplos mais sonantes serão o *Nest* [16] e o *Revolv* [17].

Enocean

Enocean é uma empresa Alemã que investiga e desenvolve produtos com propriedades *energy-harvesting* (colheita de energia), *low-power* (baixo consumo), auto-suficientes e sem fios. Através da exploração da energia com métodos super eficientes, desenvolvem redes de sensores capazes de comunicar entre si, alguns sem qualquer alimentação, sendo que a própria comunicação é *wireless*.

Para gerir o *standard* e autorizar o uso deste a outras companhias, foi criada a *Enocean Alliance* [18], que consiste num acordo entre diversas empresas tecnológicas que desenvolvem e trabalham em função de promover a monitorização e controlo auto sustentáveis, *low-power* e *wireless* de interiores de edifícios. Domótica eficiente e *low-power*, em suma. Desta forma, é possível formalizar o standard

Enocean e a interoperabilidade entre diferentes dispositivos.

Figura 2.3: Implementação *Enocean*: sensor e controlador [6]

A figura **2.3** reflete a tecnologia *Enocean* e a sua implementação tipo. Do lado esquerdo está representado um módulo *end-device* que normalmente tem um ou mais sensores mas também pode ter um atuador, para, de alguma forma, atuar algo tendo em conta os valores obtidos na leitura dos sensores ou aquando de uma ordem de um controlador. Este módulo *Enocean* provido de um microcontrolador desenvolvido pelos mesmos tem também métodos de *energy-harvesting* para gerar energia suficiente para enviar tramas através do módulo RF. Geralmente este módulo funciona como "*sleep-wake-up*", o que significa que, para poupar energia, está em modo "sleep" grande parte do tempo, acordando para fazer funções básicas de processamento (ler informação do sensor, por exemplo).

Por outro lado, o controlador (lado direito da figura **2.3**) é normalmente alimentado pela rede elétrica da habitação, logo o pequeno consumo deste dispositivo não é muito relevante. Não obstante, o consumo energético é sempre um fator importante, daí a possibilidade de usar o mesmo microcontrolador, que tem mecanismos *low-power*, para os dois diferentes equipamentos. Normalmente, este dispositivo envia ordens aos sensores, quer para fazer uma leitura, atuar de alguma maneira nalguma variável, ou até atualizar o dispositivo com alguma nova informação ou variável que foi alterada enquanto este último estava em modo "sleep".

O *Enocean Radio Protocol* (ERP) [7] é um protocolo de rádio otimizado que garante que os diferentes produtos que tiram proveito da tecnologia *Enocean* comuniquem sem fios entre si, sem comprometer o alto desempenho a baixo consumo. Otimizar a informação a transmitir com o máximo de fiabilidade usando muito pouca energia e assegurando compatibilidade entre diferentes produtos *Enocean* são o motivo pelo qual este protocolo foi criado.

A nível técnico, o ERP funciona com três diferentes frequências, dependendo

da região: 315MHz (Asia), 868MHz (Europa), 902MHz (América do Norte). A densidade da rede, ou número de dispositivos, é praticamente ilimitada, (2^{32}) e o alcance dos sinais é de 30m em interiores e 300m em exteriores.

Layer	Services	Data Units
Application (API)	EnOcean Equipment Profiles (EEP) RPC/RMCC handling	DATA
Presentation	Radio Telegram Processing Encryption	DATA
Session	--- not used ---	--
Transport	Smart Ack Remote Management	TELEGRAM/ MESSAGE
Network	Addressing telegrams (ADT Encapsulation/Decapsulation) Switch telegram conversion (choice/status processing) Repeating (status processing)	TELEGRAM
Data Link Layer	Subtelegram Structure Control Sum Calculation Subtelegram Timing Listen before talk	SUBTELEGRAM
Physical	Encoding/Decoding (inverse bits) Radio reception/transmission	BITS / FRAME

Figura 2.4: Camadas protocolares *Enocean* [7]

A figura **2.4** representa as sete camadas *Enocean* segundo o modelo OSI [7]. O protocolo *Enocean* é proprietário e foi criado com base no OSI 14543-3-10, que é um standard *wireless* para dispositivos *low-power*. A camada física encarrega-se de interpretar os dados em série, de *bit* a *bit*, de forma a descodificar um sub-telegrama. Para cada telegrama *Enocean* enviado, são enviados três sub-telegramas, faseados no tempo, de maneira a garantir a integridade dos dados e a evitar colisões. A segunda camada é responsável por controlar a transmissão e receção destes sub-telegramas e também verificar a sua consistência através de algoritmos de CRC (*Cyclic redundancy check*) e *checksum*. Esta camada tem também a capacidade de funcionar como um dispositivo "*Listen before talk*". Esta funcionalidade opcional faz com que o transmissor verifique se existe algum telegrama no ar, antes de enviar qualquer outro. Se existir de facto alguma atividade, o transmissor espera durante um tempo aleatório antes de verificar novamente se pode transmitir. A camada *network* encarrega-se de garantir a consistência da rede como um todo, de modo a todos os diferentes dispositivos conseguirem comunicar entre si sem problemas. É nesta fase que se implementam as funcionalidades dos repetidores de sinal e o encapsulamento da trama para criar destinos e fornecer informação da origem do telegrama. As seguintes camadas são aplicacionais, que consistem em formas de

comunicação entre dispositivos ("*Smart-Ack*" [19] ou "*Remote Management*" [20]) ou os *profiles*, explicados com mais detalhe na secção **3.2**.

Para além do ERP, for criado o *Enocean Serial Protocol*(ESP) [21]. Este protocolo série é o resultado da necessidade de uma comunicação série, física, entre dois dispositivos. É geralmente utilizado na comunicação entre o *gateway* da rede Enocean e um controlador, de forma a ligar a rede *Enocean* a outros sistemas. O seu propósito está mais destinado a aplicações ou dispositivos que não tenham tanta dependência no *low-power* porque estão ligados fisicamente, normalmente a sua alimentação é proveniente da rede elétrica e o papel destes dispositivos costuma ser de *gateway*, ou seja, sempre à escuta de dados. O ESP é mais complexo que o ERP porque o encapsulamento do telegrama rádio adiciona alguns bytes de dados à troca de informação, como o CRC , que é um teste à integridade dos dados série e também o *header*, que identifica o tipo e tamanho dos dados. Além de ser uma maneira de refletir os telegramas rádio através de uma comunicação série, é também capaz de fazer uma comunicação do tipo pedido/resposta que não é possível implementar via rádio. Através de um identificador que define o tipo de pacote que está a ser transferido, é possível implementar diferentes pedidos.

Tabela 2.1: Tipos de pacotes ESP

Pacotes ESP	
Tipo	descrição
Radio	Telegrama rádio (conversão direta)
Response	Resposta a um pacote
Event	Ocorrência de eventos
Smart_Ack_Command	Comandos "Smart-Ack"
Common_Command	Comandos comuns

Dados como informação do dispositivo, identificação dos dispositivos associados e pedidos de leitura e escrita de memória são algumas das operações possíveis através do ESP.

A tabela **2.1** apresenta os pacotes mais comuns numa comunicação ESP. Entre os quais, o mais usual é o *Radio*, que é o espelho de um telegrama enviado via rádio. O papel principal dos *gateways Enocean* é precisamente encaminhar estes pacotes. *Response* é o tipo de pacote que um dispositivo tem de enviar, na

maior parte dos casos, sempre que recebe um pedido via série através do ESP. É uma garantia de que o equipamento "Enocean" está a responder corretamente aos comandos e pedidos enviados. "Event" é utilizado na ocorrência de mensagens "Smart-Ack" [19], assunto que será abordado mais tarde. Os comandos "Smart-Acknowledge" são utilizados para ordenar o envio de mensagens relacionadas com o processo de aprendizagem e para obtenção de informação relativamento aos dispositivos aprendidos. Finalmente, os "Common_Command" são geralmente utilizados por entidades que pretendem obter informações relativamente ao dispositivo a quem tem ligação série.

No que diz respeito ao controlo e monitorização de um espaço interior, a diversidade de sensores e atuadores existentes é enorme. Esses equipamentos podem ter grande variedade: desde sensores, simples interruptores ou até baterias. Para uniformizar e garantir que diferentes dispositivos ou sistemas interajam entre si, os *Enocean Equipment Profiles* [22] foram criados. Estes perfis estão divididos em categorias de forma a serem o mais universais possível, para os fabricantes de produtos de domótica terem mais facilidade em especificar o componente e o tipo de dados com que o seu produto vai lidar.

Relativamente ao suporte para desenvolvimento de aplicações, a *Enocean* fornece também um conjunto de ferramentas para desenvolver software para os seus microcontroladores através do IDE("Integrated Development Environment") *Keil*, assunto mais detalhado no capítulo **4.3.1**.

Devido a restrições no que concerne ao desenvolvimento do projeto, a tecnologia *Enocean* foi a escolhida para desenvolver o *Climawin*. Todos os dispositivos e ferramentas de desenvolvimento *Enocean* são orientados para o *low-power* e *energy-harvesting*, métricas que facilitam bastante o desenvolvimento de aplicações que tenham os mesmos objetivos. Como o *Climawin* é um projeto de domótica que consiste no controlo de janelas, foi idealizado uma interação para com as mesmas. Os interruptores *Enocean* [23] são o interface mais económico, ergonómico e confortável para atuar em diversas aplicações. Descrição mais detalhada sobre este interruptor no capítulo da arquitetura *Climawin* **3.1.2**.

Os *end-devices* que façam uso do STM300 e da memória *ultra low-power* embutida no mesmo, podem entrar em modo *sleep* durante largos períodos de tempo sem perder informação relevante e reagir de forma rápida em rotinas ou a estímulos. Esta arquitetura de memória em junção com o "Smart-Acknowledge" resulta em sistemas eficazes e consistentes com consumos mínimos, pelo que foi o

último fator que fez a decisão pender para a escolha da *Enocean* como tecnologia e protocolo de comunicação no *Climawin*.

2.2 Ligação *cloud*-domótica

Para um sistema que seja composto por um ou vários elementos aceder à *internet* é necessário ter um *hardware* dedicado para o fazer. Diferentes projetos com diferentes métricas fazem esse acesso de acordo com as suas restrições. Existem duas maneiras de o fazer. Uma consiste em ligar todos os componentes a um só nó, e este é responsável por ligar a rede de dispositivos à *internet*. A outra maneira implica ligar todos os componentes diretamente à rede.

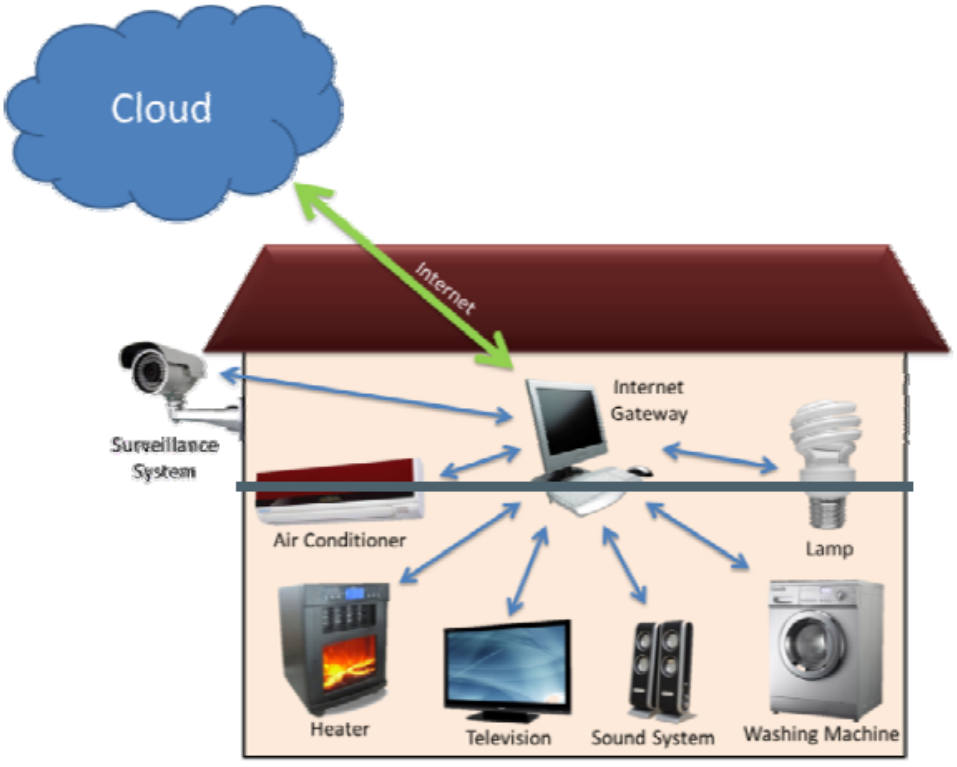


Figura 2.5: Gateway dedicado

A imagem presente **2.5** reflete a implementação mais comum de fazer a conexão entre dispositivos físicos, sejam eles sensores, atuadores, eletrodomésticos ou sistemas acústicos, e a *cloud*. Esta abordagem implica algumas restrições no sentido de que o *gateway* tem de ser um *hardware* dedicado para a aplicação a que se destina. O desenvolvimento de um dispositivo destes é, à partida, exclusivo

para um determinado tipo de aplicações, pelo que não pode ser instalado em diferentes ambientes. Os diferentes interfaces que tem com terminais do utilizador (por exemplo RS232, USB, Bluetooth) estão dependentes da compatibilidade que esse *gateway* suporta para esse tipo de comunicações. O ideal será desenvolver um *gateway* genérico multi-interface [24]. No entanto, nem todos os sistemas necessitam de todos os recursos, pelo que a utilização deste tipo de *gateways* possa induzir algum *overhead* ou custo desnecessário ao sistema.

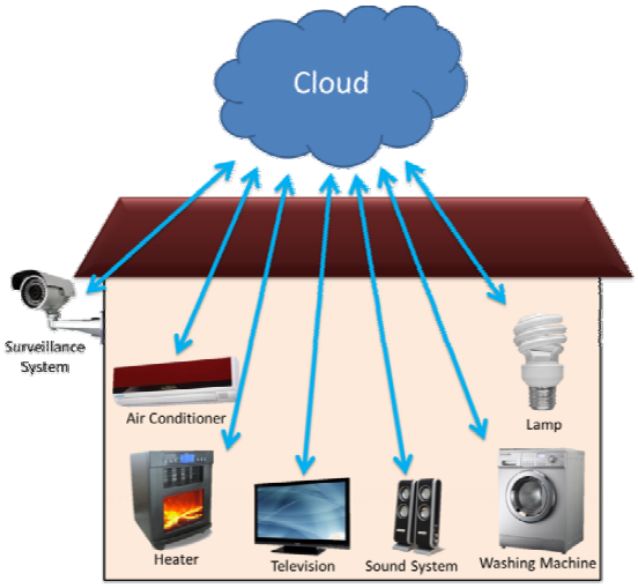


Figura 2.6: Modelo IoT

O modelo IoT é recente e consiste em idealizar uma estrutura formada por diferentes objetos com a representação dos mesmos em forma virtual na *cloud*. Cada objeto tem uma identidade e acesso à *internet* (figura **2.6**).

Com este modelo a troca de informações entre a *cloud* e o objeto é bem mais simples, porque é direta e não é necessária a conversão de protocolos [25]. No entanto, esta abordagem implica que todos os objetos tenham meios de aceder à rede, e isso pode trazer alguns problemas de custo, problemas técnicos e de compatibilidade entre diferentes produtos de diferentes fabricantes [26].

2.3 Sistemas domótica com acesso à *cloud*

Conforto, comodidade e eficiência energética são palavras de ordem em vários interiores, quer seja em ambiente profissional (escritórios, armazéns) ou num ambi-

ente doméstico (habitações). A segurança é também um fator que pesa na decisão do cliente, quer pelo bem estar dos mais próximos, quer pela confidencialidade.

A ligação remota e a garantia da segurança e confidencialidade dos dados, requerem, no entanto, alguma atenção e até alguns procedimentos que podem ser interpretados como desvantagens da *cloud*. Em primeiro lugar, é necessário ter um dispositivo com ligação à internet. Este fator implica logo duas condicionantes: um dispositivo capaz de ligar à rede e também um ISP ("Internet Service Provider"). Como, à partida, o serviço está disponível para variados clientes, e também para a generalidade de aparelhos com acesso à internet, é necessária também uma autenticação, de modo a filtrar os utilizadores fidedignos. Estas desvantagens são bastantes comuns nos dias que correm, pelo que a população que lida com estas condicionantes diariamente já está habituada e consegue ultrapassar estas pequenas nuances muito rapidamente, e por isso mesmo as vantagens da *cloud* compensam.

Não obstante, a domótica tem sido cada vez mais implementada em diferentes meios com diferentes fins. A adição do acesso à *cloud* adiciona flexibilidade a este mercado e um ainda maior conforto ao cliente. As aplicações são variadas, vão desde sistemas que fornecem serviços e métodos de acesso à *cloud* por parte do cliente, até sistemas aplicacionais e específicos para o seu propósito.

"A Platform as a Service for Smart Home" [27] (figura **2.7**) é um PaaS desenvolvido por investigadores da ETRI (*Electronics and Telecommunication Research Institute*) que tenciona mover a localização dos dados produzidos por uma casa inteligente para a *cloud*, e, desta maneira, fornecer posteriormente os dados de uma forma segura e cómoda. Adaptando o elevado fluxo de dados provenientes de sensores para monitorização de variados espaços através de um serviço na *cloud* proporciona mais qualidade e também segurança. Esta migração de informação confidencial ao cliente para a *cloud* deve ser minuciosa e tratada cuidadosamente com a devida atenção. Quando o serviço está na *cloud*, existem diversas ameaças aos dados que supostamente são privados e confidenciais, pelo que este serviço tem mecanismos de proteção e seleção dos dados.

Três tipos de potenciais clientes deste serviço foram identificados: fornecedores da estrutura que disponibiliza o serviço, consumidor final (proprietário da habitação) e fabricantes de aparelhos de domótica. O consumidor final poderá usufruir do controlo, monitorização e conforto que o serviço, juntamente com os aparelhos de domótica, lhe disponibilizarão. O fornecedor do serviço poderá efe-

tuar a manutenção da aplicação, garantir restrições de acesso ao serviço e também aceder ao histórico, ou *log* dos serviços fornecidos. Finalmente, o fabricante de aparelhos poderá atualizar o *firmware* dos dispositivos, obter dados relativamente ao funcionamento dos mesmos e também assistir remotamente o cliente.

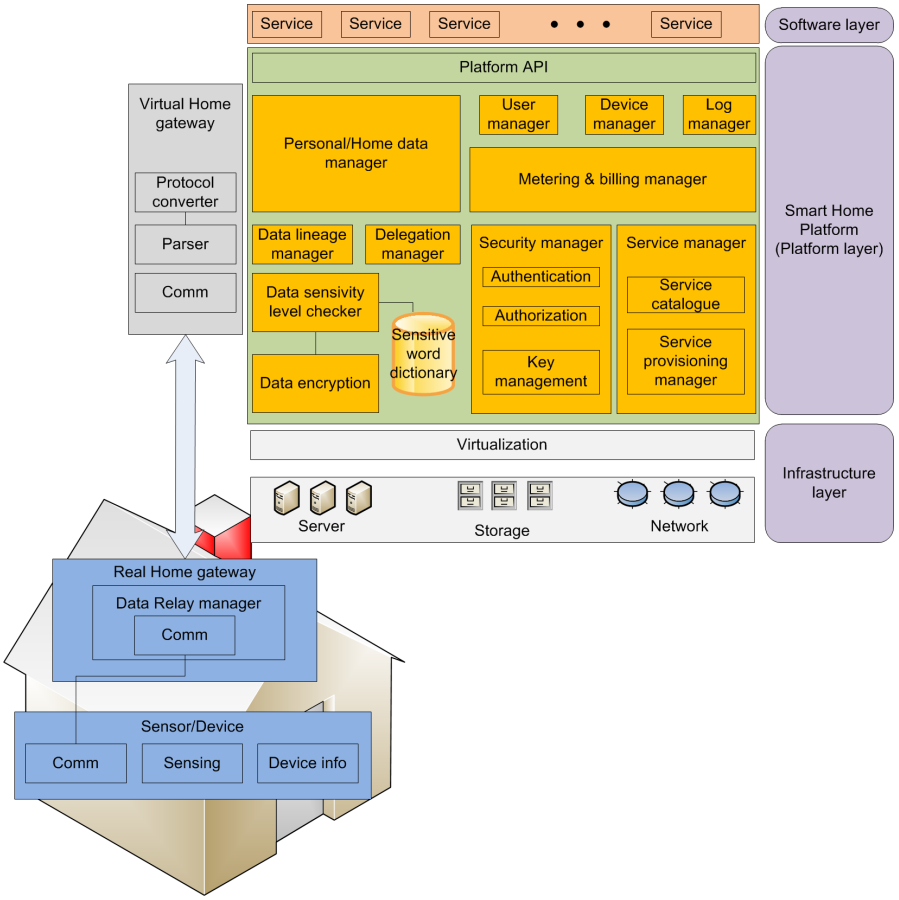


Figura 2.7: Diagrama de blocos de "A Platform as a Service for Smart Home"

Como já referido, a segurança é um dos principais requisitos deste projeto. Tendo isto em conta, a partilha seletiva e encriptação de dados foi implementada. O utilizador pode escolher com que aplicações, serviços ou pessoas pode partilhar os seus dados, oferecendo a possibilidade de utilizar o serviço com diferentes dispositivos. A seleção de dados para encriptação é uma camada adicional à segurança providenciada pela *cloud*. Implementada através de um dicionário com uma *sensivity list*, a encriptação é realizada apenas em dados sensíveis antes de serem guardados na *cloud*.

Quando um dispositivo, ou *end-device*, entende que deve atualizar dados, envia informações ao *gateway* físico instalado na habitação. Esta ligação não é responsabilidade do serviço. No entanto, após os dados serem enviados para o VHG

("Virtual Home Gateway") este deve interpretar os dados e definir a sensibilidade dos mesmos para estes serem depois analisados. No sentido oposto, é este mesmo *gateway* virtual que recebe comandos e envia os mesmos corretamente estruturados para o *gateway* físico posteriormente encaminhá-los para o *end-device*.

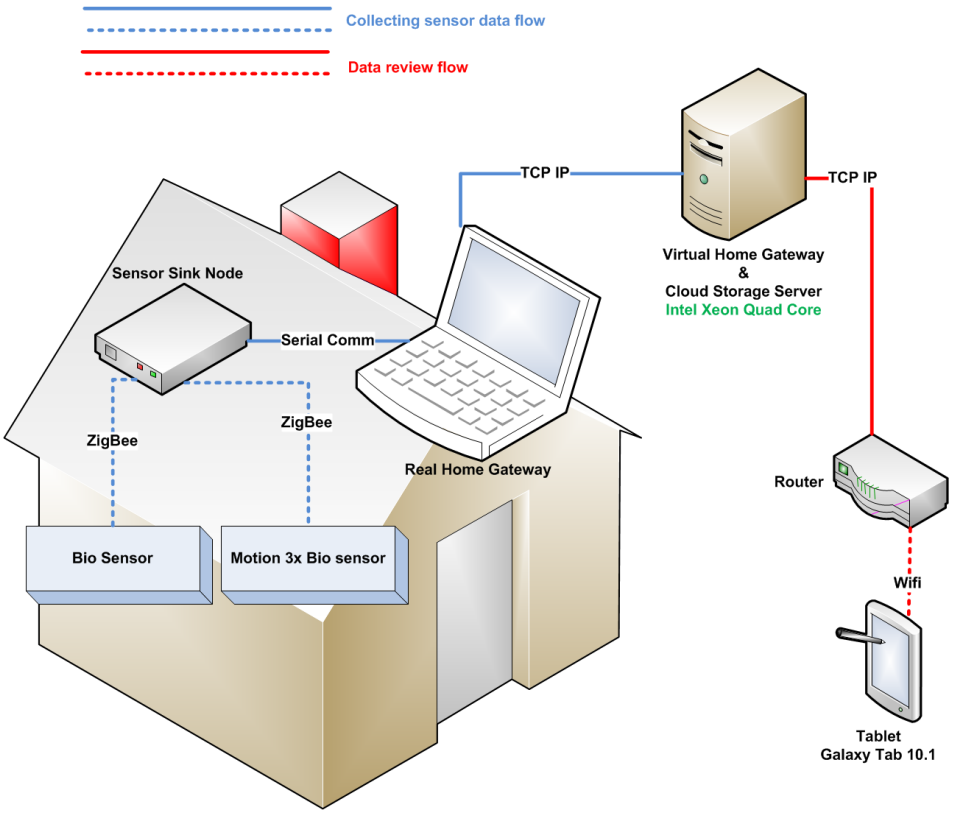


Figura 2.8: Protótipo de "A Platform as a Service for Smart Home"

O protótipo utilizado para demonstração (figura **2.8**) do modelo "A Platform as a Service for Smart Home" (figura **2.7**) envolvia dois sensores biométricos para medir o batimento cardíaco e um acelerómetro de 3 eixos que media as calorias queimadas associadas ao movimento. Como *router*, foi utilizado um computador portátil que, via TCP/IP, trocava dados com o VHG. *Gateway* este que faz parte do serviço alojado na *cloud*. No que toca ao UI, foi desenvolvida uma aplicação Android a correr num *tablet* em que era possível monitorizar os dados dos sensores e efetuar operações de delegação dos dados.

Um outro exemplo da integração da *cloud* com sistemas inteligentes para controlo de uma habitação é o projeto "Home Automation Using Cloud Computing and Mobile Devices" [28] desenvolvido por investigadores do Sinhgad Institute of Technology. Este sistema é bem mais simples que o referido anteriormente, isto porque não oferece nenhum serviço na *cloud* ou soluções para controlar o

acesso através da partilha seletiva de dados, mas sim uma solução de domótica (figura **2.9**). No entanto, é possível controlar os diferentes periféricos da casa através de um controlador local sem fios que tem prioridade sobre os comandos provenientes da *cloud*, adicionando mais robustez e segurança ao sistema. No protótipo desenvolvido era também possível controlar e monitorizar a habitação através de uma aplicação para um telemóvel com SO *Android*.

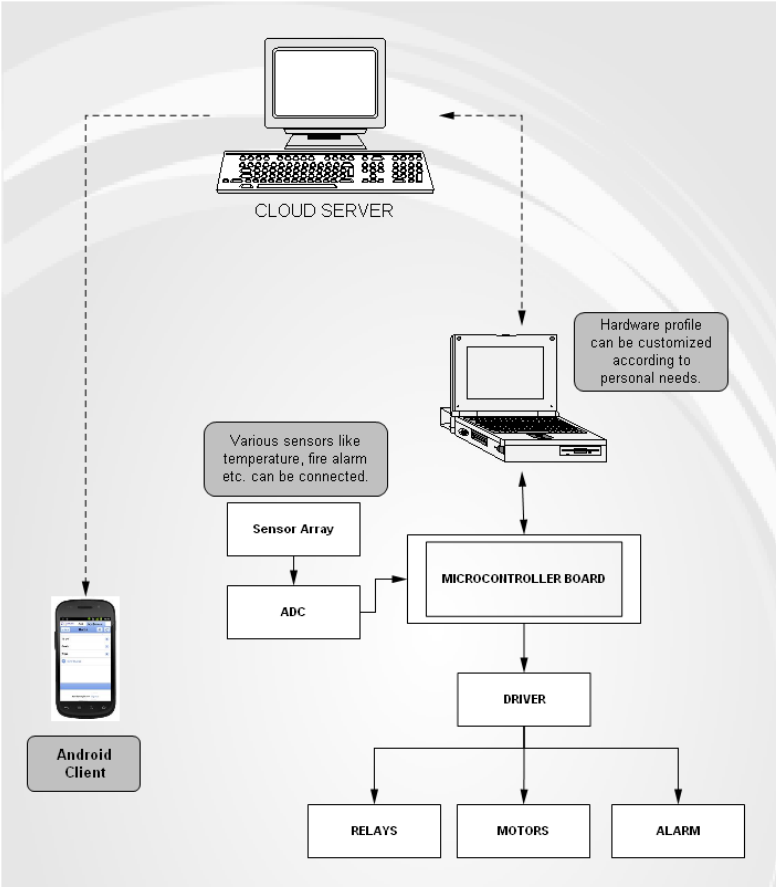


Figura 2.9: Diagrama de "Home Automation Using Cloud Computing and Mobile Devices"

Este sistema poderia integrar no seu protótipo o serviço fornecido por "A Platform as a Service for Smart Home" [27], isto porque utiliza armazenamento de dados na *cloud* para verificar e monitorizar estados de diferentes periféricos instalados em casa, ou para atuar nos mesmos. Também utiliza uma aplicação num dispositivo *Android* que necessita de acesso aos serviços que controlam a habitação, e consequentemente à *cloud*. Ou seja, da camada do *router* para cima (que é referido como "Customizable Hardware") poder-se-ia utilizar um PaaS, tal como o referido em [27].

HAaaS (*Home Automation as a Service*) [25] é um novo conceito de serviços fornecidos pela *cloud* proposto por Anindya Maiti, investigadora da Universidade VIT (*Vellore Institute of Technology*). Tem como propósito conectar todos os sub-sistemas da habitação à *cloud* e, desta maneira, diminuir não só o custo de manutenção mas também a implementação de *gateways* especializados em diferentes contextos.

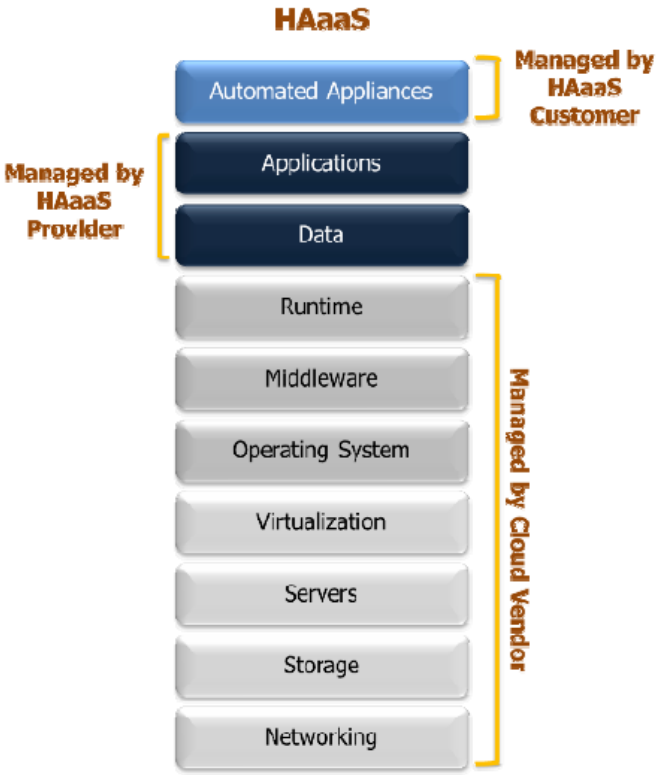


Figura 2.10: Arquitectura de HAaaS

O serviço *cloud* mais utilizado é o SaaS. Neste tipo de serviço o utilizador precisa apenas de correr um software na sua máquina e, nesse instante, o serviço na *cloud* trata de tudo o resto, desde o *hardware* até à aplicação em si. O HAaaS, representado na figura **2.10**, leva este conceito mais longe: por um lado, o fornecedor de serviço poderá utilizar um PaaS que já se encontra no mercado, por outro, é adicionada uma nova camada, que é a dos próprios aparelhos domésticos que o cliente quer controlar, sendo que este último passo é de inteira e exclusiva responsabilidade do mesmo.

Para provar o conceito, foi desenvolvido um cenário de aplicação de um HAaaS desenvolvido por alunos do VIT. A conexão dos diferentes dispositivos ou nodos da casa à *cloud* foi efetuada com uma ligação direta, utilizando o modelo

IoT (**2.2**). Quer isto dizer que todos os objetos relacionados com domótica estão diretamente ligados à *cloud*. Para isto, foi desenvolvida uma placa dedicada para cada nó com o protocolo 802.11n para comunicação sem fios (figura **2.11**). Faz também parte do sistema um nó que faz o papel de *gateway*, ligando à *internet* através de um HSDPA (*High-Speed DownlinkPacket Access*). Através deste nó era possível fazer *upload* de músicas para reproduzir nos diferentes nós.

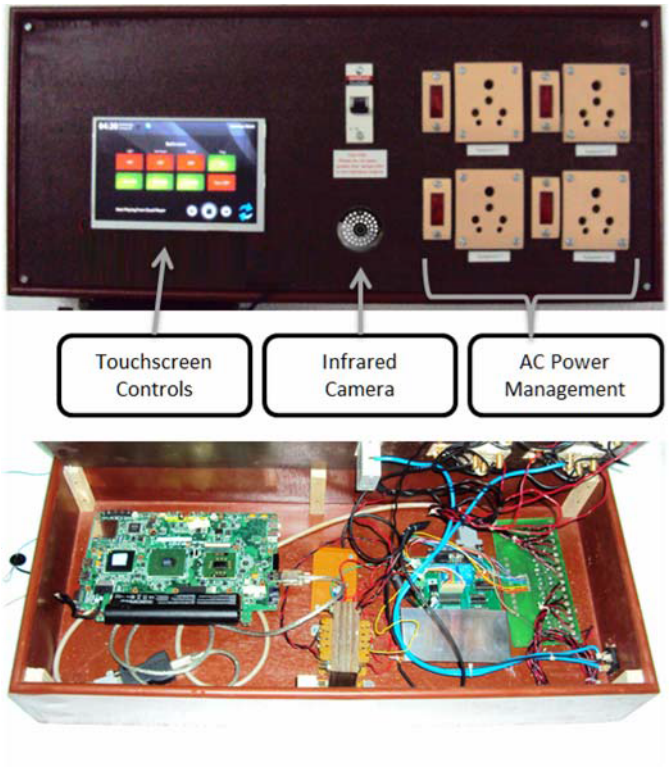


Figura 2.11: Nodo de exemplo aplicativo de HAaaS

Diferentes aparelhos (até quatro) ligam a um nó. Através de um microcontrolador *low-power* são acionados relés para controlar a alimentação a estes aparelhos. De destacar também que cada nó tem um *touch-screen* que tem o papel de *music-player* interativo. Isto porque cada nó contêm suporte para reprodução de som, com colunas internas e opção de extensão para mais aparelhos sonoros, sendo que este projeto tem também bastante ênfase na acústica doméstica, podendo-se reproduzir músicas e ouvir em toda a casa, desde que os nós estejam distribuídos uniformemente.

Para segurança, estão embutidos na placa câmeras de infravermelhos que, após processamento da imagem, são utilizados algoritmos inteligentes para a deteção de intrusão ou anomalias. Após deteção de um evento, o utilizador é imediatamente notificado, através de um sinal enviado para a rede.

Figura 2.12: UI da página web do exemplo aplicacional de HAaaS

Para fornecer o seu serviço, utilizaram um processador Intel Xeon a correr como servidor baseado em Windows Server 2008 R2. Conectado a esse servidor estava um *modem* SMS para enviar notificações para o telemóvel do cliente em caso de eventos mais urgentes, tais como falha de energia ou problemas no acesso à *internet*.

Para interação remota, a partir de qualquer local com acesso à *internet*, foi desenvolvido um *website* (figura **2.12**). Neste interface era possível monitorizar e controlar os diferentes aparelhos ligados aos nós. Após autenticação, era possível navegar em duas diferentes páginas. Uma página tinha como propósito o controlo acústico da casa, apenas para lazer, onde era possível navegar entre diferentes músicas, reproduzir ou eliminar as mesmas. A outra página seria para controlo do consumo energético, em que era possível ligar e desligar objetos que estavam conectados aos nós, visualizar o estado dos mesmos, dar-lhes nomes e adicionar outros tantos.

2.4 Controlo de janelas

As janelas tradicionais com estores ou persianas de correr à mão estão a ser substituídas por complexos sistemas providos de motores elétricos com alguma eletrónica adicional de controlo. O método de atuação é sempre semelhante: um motor elétrico e alguma maneira de controlar o fim de curso através de variadíssi-

mos sensores. Relativamente ao controlo do fluxo de ar, já costumam ser sistemas mais primitivos que recorrem unicamente à mecânica para fazer o seu propósito. No controlo da janela é que as ideias divergem e diferentes aplicações são imaginadas. Através de monitorização, os sistemas podem mudar o estado da janela automaticamente, sem qualquer intervenção do utilizador, deixando estas preocupações quotidianas a cargo de um microcontrolador ou de um sistemas bem mais complexo. Não obstante, é importante a atuação na janela ser bem conseguida, consistente e fiável de modo a ser tolerante a falhas. O mesmo é aplicável à atuação nas válvulas que controlam o fluxo de ar.

2.4.1 Janela inteligente com sistema GSM

Uma janela automática e inteligente pode ser bastante conveniente quando não está ninguém na habitação para controlar a abertura ou fecho da mesma. Com a aquisição de diferentes grandezas através de sensores, pode-se controlar a posição da janela, após atuação, e desta forma criar um sistema inteligente que permita ventilação e controlo de temperatura na habitação.

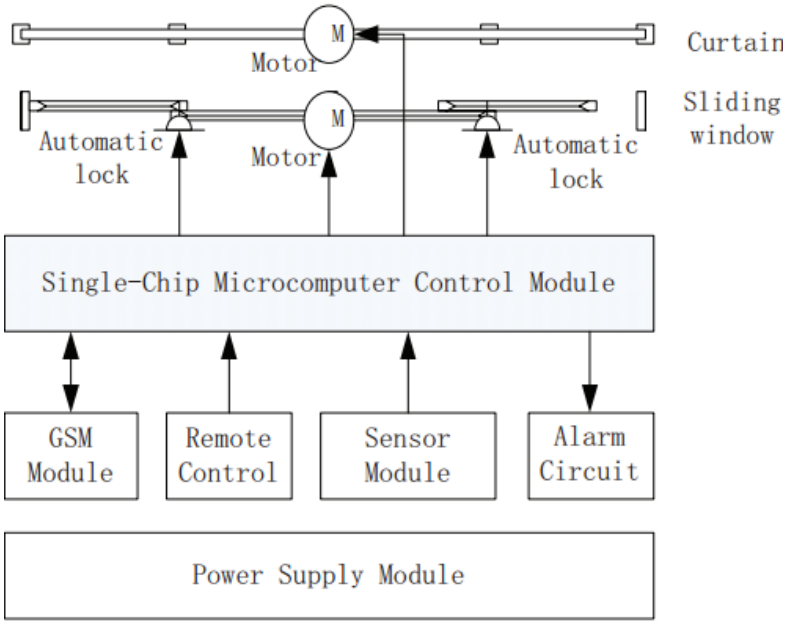


Figura 2.13: Diagrama de blocos de "Design of Smart Window Control System Based on GSM Network"

"Design of Smart Window Control System Based on GSM Network" [29] é um projeto de investigadores a trabalhar para a Universidade Bei Hua (China) que controla uma janela horizontal. Através de diferentes sensores, pretende-se o

controlo automático da janela, de maneira a controlar a entrada de ar, evitando a água da chuva e a elevada concentração de gases.

A janela é composta por um vidro deslizante e uma cortina, cada um controlado por um motor independente (figura **2.13**). Entre as variadas funcionalidades do sistema destacam-se a capacidade de análise ao ambiente exterior e interior à janela e a consequente atuação automática na janela. O módulo GSM (*Global System for Mobile Communications*) serve para alertar o utilizador do estado da janela e dos sensores. Estes alertas ocorrem quando um evento crítico ocorre no sentido de que o utilizador ou o sistema automático devem reagir, como por exemplo a ocorrência de chuva, rajadas fortes de vento ou então fugas de gás.

Um par recetor/emissor de infravermelhos é utilizado como sensor de fim de curso. Colocado numa das extremidades da janela, o recetor, ligado a um pino do microcontrolador, configurado como interrupção externa, tem o papel de comutar o sinal. É então possível detetar a interceção do sinal quando a janela é aberta ou fechada e induzir a sua posição.

Através de um sensor de luz do lado de fora da janela é feito o controlo das cortinas, de acordo com parâmetros ou limites pré-definidos pelo sistema. Um exemplo de atuação é que se a luz do dia for forte ao ponto de poder produzir raios ultravioleta nocivos à saúde, as cortinas fecham. À noite estão sempre fechadas, para manter a privacidade. Um sensor de vento é capaz de medir a intensidade e direção do vento, pelo que se este estiver forte e na direção da janela, esta fecha. Quando o sistema deteta um gás em grande quantidade, através de um sensor dedicado para o efeito, as janelas abrem automaticamente.

2.4.2 Janela para automóvel

Na indústria automóvel o controlo do vidro da janela da porta é também um assunto que merece atenção, porque a circulação de ar é uma propriedade essencial de um veículo, no que toca ao bem estar do condutor e restantes passageiros, e deve ser bastante robusta. Os primeiros carros com este sistema tinham um *design* mecânico que consistia numa manivela para descer e subir o vidro. No entanto, a comodidade é um dos principais objetivos da evolução e da tecnologia, e este caso não foi exceção. "Design and hardware development of power window control mechanism using microcontroller" [30] é uma demonstração de controlo de janela de um carro (figura **2.14**).

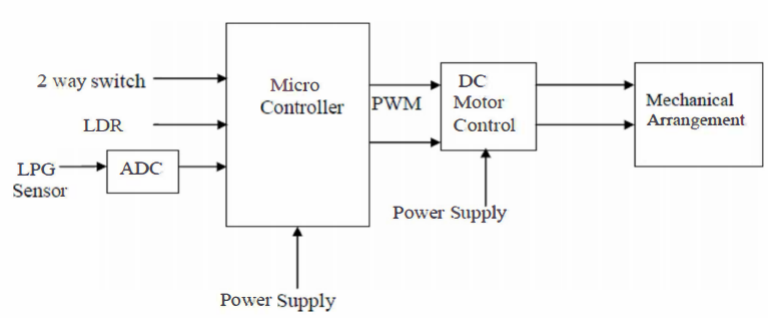


Figura 2.14: Diagrama de blocos de "Design and hardware development of power window control mechanism using microcontroller"

O controlo do motor DC é feito recorrendo a um microcontrolador da Atmel, *ATMEGA 16*. Como elemento de segurança, um detetor de "overheat" separa o motor da alimentação caso detete alguma anomalia no sistema.

Através de dois botões é possível descer e subir a janela e com o *feedback* de sensores, o microcontrolador é capaz de reconhecer as posições totalmente fechada e totalmente aberta como fins de curso. Dois LDR (*Light Dependent Resistor*) que, quando devidamente colocados num circuito comparador dimensionado para o efeito, são capazes de gerar uma interrupção quando a luz incidente nos mesmos varia significativamente. A janela inteligente também é capaz de detetar perigos que possam por em causa a segurança dos ocupantes do veículo no que diz respeito a gases tóxicos. Quando um gás é detetado, através do sensor de LPG (*Liquefied petroleum gas*), que através de um ADC (*Analog to Digital Converter*) envia o valor lido para o microcontrolador, a janela abre na totalidade. Carros movidos através da combustão de gases fariam bom uso desta funcionalidade para adicionar mais segurança. Não obstante, o fogo é também detetado por este sensor através do fumo, funcionalidade útil também em carros mais habituais.

A atuação no motor é feita através de um par *Darlington* que consiste em dois transístores em série (emissor do primeiro na base do segundo). Através deste circuito, é possível acionar um relé que está ligado diretamente ao motor. Um outro transístor e um outro relé são utilizados para mudar a direção do motor.

2.4.3 Janela, temperatura e luminosidade

Numa perspetiva mais científica, "Dual-Mode window covering system responsive to ac-induced flicker in ambient ilumination" [31] é um artigo que explica não só o controlo de uma janela com persianas comuns, mas também a relação

entre a temperatura/luminosidade, e prova também que este assunto já é discutido há mais de uma década, visto que este sistema é datado de 1997. Na altura, as janelas desenvolvidas eram eficazmente controladas tendo em conta diferentes propósitos: controlo de energia através do aquecimento/arrefecimento do interior ou então controlo da luminosidade. Ambos efetuados por intermédio de um objeto que impede ou permite a entrada da luz solar no interior da divisão. O aquecimento/arrefecimento através da luz solar, ou da ausência da mesma, é um conceito básico. Através de parâmetros programáveis, ou definidos pelo utilizador, a janela deixa entrar luz quando pretende aquecer a divisão, ou impede a entrada da luz quando a temperatura está alta. Da mesma maneira, mas para a luminosidade, as janelas podem controlar a intensidade de luz no interior. No entanto, estes dois paradigmas podem entrar em conflito, e é preciso saber encontrar o momento, ou a linha que separa estas duas diferentes abordagens.

Partindo do conceito teórico de outros projetos, foi desenvolvido um que abrange os dois conceitos e que prova que podem co-existir no mesmo sistema.

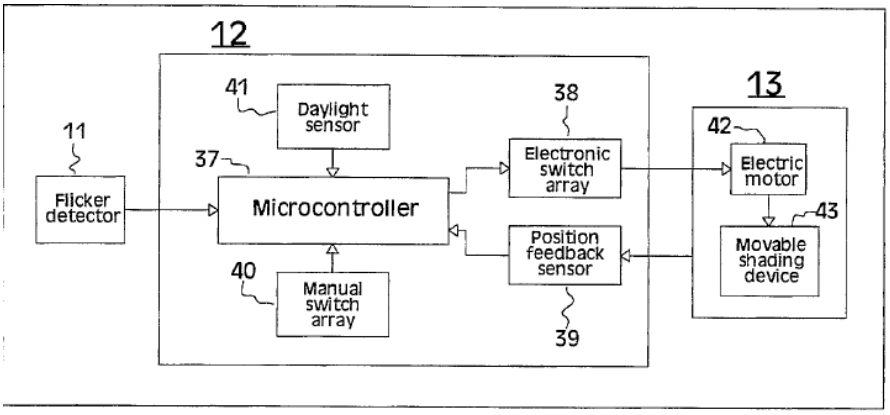


Figura 2.15: Diagrama de blocos de "Dual-Mode window covering system"

O uso de um detetor de luz artificial (*flicker detector*), permite distinguir as diferentes fases de controlo. O sistema parte do princípio que, mesmo num dia com muita luz, no local de trabalho, estão sempre ligadas algumas luzes artificiais. Logo, a quantidade de luz artificial utilizada é naturalmente inversa à quantidade de luz natural que entra na divisão, se se tomar como garantido que as pessoas que estão a usufruir do espaço são moderadas no que diz respeito a gastos energéticos. Quando não existe luz artificial, a janela inteligente conclui que deve começar o controlo de temperatura. De acordo com limites definidos pelo utilizador, a janela sobe ou desce a posição da sua persiana de acordo com a luminosidade. É também responsabilidade do utilizador um eficiente controlo da temperatura. Este deve, de

acordo com a estação do ano, definir as posições de noite/dia de modo ao sistema reagir positivamente consoante a luminosidade. Quer isto dizer, que a persiana deverá estar fechada durante as noites de inverno, de modo a minimizar a perda de calor, e abertas durante o dia, de maneira a aquecer. O contrário é recomendado durante épocas mais quentes, nomeadamente o verão.

A nível técnico (figura **2.15**), o sistema, como todos os outros, tem uma parte de acondicionamento do sinal de sensores. Neste caso, um de luminosidade interior e outro de luz solar. Tem também a componente de controlo, que tem como principal componente um microcontrolador PIC16C63 e na atuação das persianas conta com um motor DC. Para obter a posição da janela com precisão é utilizado um *encoder* que está acoplado à persiana. Através dos pulsos que este origina, o microcontrolador pode induzir a posição da persiana e desta forma saber a sua posição com precisão.

2.4.4 Janela com algoritmo de controlo *fuzzy*

Um outro exemplo de controlo de janela é o "Intelligent Window Based on Embedded System" [32], desenvolvido por investigadores da Universidade de Beijing, China. Este sistema tem como principal curiosidade um controlo "fuzzy" ou indistinto, porque é um controlo que tem como entrada valores analógicos e define um conjunto de gamas de constantes para depois definir a saída do sistema.

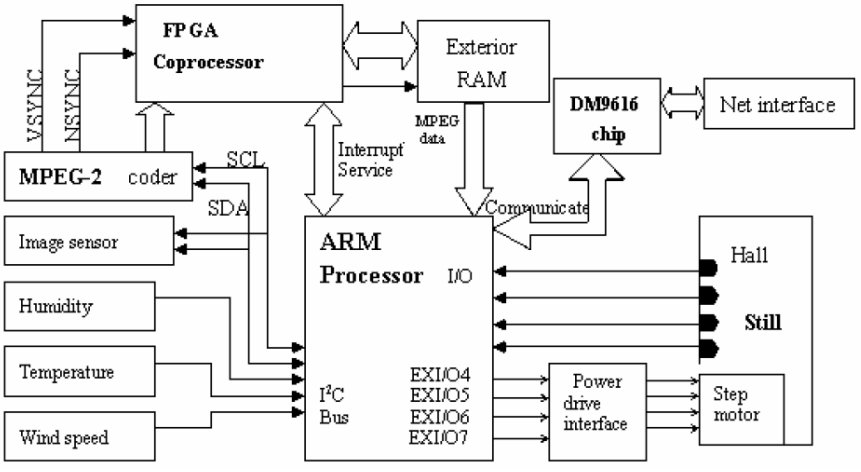


Figura 2.16: Diagrama de blocos de "Intelligent Window Based on Embedded System"

A janela é composta por 3 sensores: humidade, temperatura e velocidade do vento, sendo que o valor da humidade é o que define o comportamento do sistema,

isto é, quando está a chover a janela fecha na totalidade, senão, posiciona-se de acordo com os valores obtidos pelos sensores de temperatura e da velocidade do vento, entrando como argumento do controlo "fuzzy". Depois de definida a tabela das regras do controlo, o sistema atua na janela conforme a posição definida pela mesma tabela. A posição da janela é estimada através de 4 sensores Hall estrategicamente posicionados na moldura da janela. Estes sensores variam a tensão de saída de acordo com a força de um campo eletromagnético, pelo que normalmente são construídos com materiais semicondutores e utilizados juntamente com ímanes.

A janela também é capaz de obter imagens ou fotos tiradas através do módulo OV6630, que consiste numa pequena câmera fotográfica e no SAA6752HS da Phillips que é um microcontrolador capaz de obter a imagem da câmera e configurar a mesma através do protocolo de dois fios I^2C. O *buffering* da imagem e todo o seu processamento é efetuado em paralelo com uma FPGA Xilinx SPARTANIIE, que obtêm os dados da imagem e guarda-os numa RAM externa. Através de uma interrupção externa, o controlador principal da janela, ARM S3C2410, lê a mais recente imagem da RAM. Este microcontrolador é também responsável por ler os valores dos sensores, logo tem também o algoritmo de controlo "fuzzy", e saídas para controlar o motor de passo que move a janela (figura **2.16**).

A janela tem também funcionalidades que podem ser exploradas pelo utilizador através da *internet*, com um módulo DM9161 a funcionar como ponte entre a janela e a rede. O modo como o utilizador opera sobre a janela não é explícito, mas este mesmo pode mover a janela para a posição que desejar através da *internet*.

Todas estas diferentes soluções tem as suas vantagens e desvantagens. Dependendo do seu propósito, todas diferem nos sensores que utilizam, porque simplesmente para diferentes casos diferentes grandezas interessam. Também diferem no modo como atuam, e, principalmente, na precisão da atuação. A janela com módulo GSM e a janela de automóvel confiam o controlo da posição da mesma a sensores de fim de curso. Esta atuação é imprecisa, e é possível induzir com total certeza apenas que a janela está, ou não, numa determinada posição. Este método cumpre as métricas do sistema quando não é necessária uma atuação proporcional, mas sim binária, ou "ON-OFF". Neste tipo de atuação é necessário saber apenas se a janela está aberta ou fechada, e tudo o resto que pode estar indefinido ou, se quisermos, numa zona cega, não importa, pois o propósito é apenas ligar/desligar. Com um controlo proporcional, como nos outros dois casos, é possível saber a posição da persiana e ter um melhor controlo sobre a mesma. Com mais precisão

no caso da janela com o *flicker detector*, pois usa um *encoder* que fornece uma resolução maior do que os quatro sensores Hall estrategicamente posicionados na janela do algoritmo "fuzzy".

Capítulo 3

Visão global e especificação do sistema

Como já referido, a proposta desta dissertação está inserida no âmbito do projeto *Climawin*. Este sistema para domótica visa essencialmente melhorar a eficiência energética de edifícios, tanto para habitações e escritórios, bem como edifícios públicos. Através da abstração à rede elétrica e ao *energy harvesting* por intermédio de painéis solares, as janelas do *Climawin* são capazes de melhorar o ambiente nesses edifícios, isto é, diversos fatores relacionados com a qualidade do ar, como por exemplo a temperatura e a humidade, e ao mesmo tempo são janelas *low-power*, ou seja, consomem pouca energia ao ponto e serem autónomas durante largos períodos de tempo.

Neste capítulo será explicado com mais detalhe a arquitetura do sistema, os seus métodos para atingir os objetivos mas também as restrições impingidas por diferentes parceiros do projeto.

3.1 Arquitetura do sistema

A arquitetura do *Climawin* resume-se a uma rede de janelas, controladas por um ZC, que por sua vez responde a pedidos provenientes de um *gateway*, que pode ser um mini-computador. O *Climawin demo* utiliza uma "Raspberry Pi" para desempenhar este último papel. A partir desta camada, é possível ligar todos os diferentes componentes do sistema à rede.

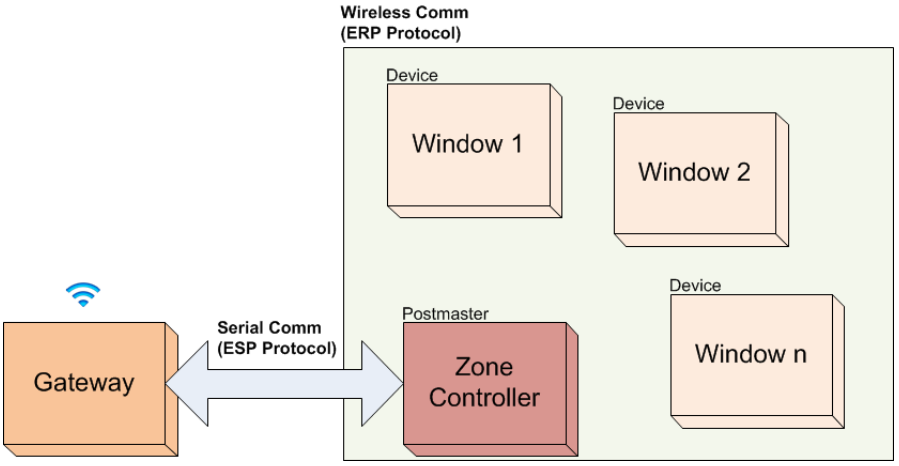


Figura 3.1: Diagrama de blocos da arquitetura do *Climawin*

O *Climawin demo* tem como propósito modelar e implementar estes diferentes componentes e garantir a comunicação entre os mesmos. Dentro de diversas restrições e métricas, destacam-se o *low-power*, autonomia e comunicação sem fios entre diferentes sensores, neste caso, janelas e o controlador central.

As janelas são os dispositivos, ou "devices" e não comunicam entre si. Não é necessário pois nenhuma delas tem informação que seja relevante para outra. O propósito desta comunicação é enviar e receber informações do "Postmaster", o Zone Controller, que tem como principal função reencaminhar mensagens provenientes da *cloud* para as janelas, e vice-versa, ou seja, enviar tramas das janelas para a *cloud*. Esta comunicação entre o "postmaster" e o "device" é regida pelo "Smart-Acknowledge" [19], que é uma comunicação bi-direcional do tipo "master-slave" que é aplicada uma camada acima do ERP.

A troca de informações entre o *gateway* e o ZC é feita través de uma ligação série sob o protocolo ESP. Esta é a ponte que liga toda a instalação à rede, ou seja, é o meio de acesso à mesma. Quer isto dizer que toda a informação que tem a *cloud* como destino ou origem tem de obrigatoriamente passar por este ponto. O *gateway* tem um papel de controlador nesta comunicação bi-direcional, e, nesta nova versão do *Climawin*, pode fazer pedidos ao ZC para variados efeitos.

Este *gateway* pode funcionar sem conexão à *cloud*, e garantir, mesmo assim, um funcionamento consistente da instalação *Climawin* e manter uma base de dados local atualizada. Através desta base de dados local armazenada na sua própria memória pode guardar a informação atualizada da instalação *Climawin*, independentemente da ligação à rede. Um exemplo aplicacional seria uma rede "Wi-fi"

local, criada através de um *hotspot* com ligação a qualquer dispositivo compatível com o *Climawin*, que seriam os pontos de acesso do utilizador ao seu sistema. Estes dispositivos podem ser computadores pessoais, *smartphones* ou *tablets* a correr uma aplicação especialmente desenvolvida para os seus sistemas operativos. Um *website* internamente alojado no *gateway* disponibiliza também a informação da base de dados pelo que através desta é também possível introduzir ou alterar dados no sistema.

Não obstante, a ligação à *cloud* é transparente, pelo que o *gateway* tem o papel de atualizar a base de dados colocada na nuvem computacional através dos novos dados que a base de dados local contém. No sentido inverso, tem também a responsabilidade de atualizar os dados quando é feita uma alteração na base de dados da nuvem de forma a manter ambas em sintonia.

O cenário ideal do *Climawin*, e é sobre essa premissa sobre a qual esta dissertação se apoia, é que o *gateway* esteja ligado à *cloud*, de forma a fornecer um serviço confortável ao cliente. Posto isto, daqui em diante, sempre que a *cloud* for referida entenda-se como um comando proveniente de uma camada superior ao *gateway*, usualmente associado ao cliente ou utilizador, seja da rede local ou efetivamente da *cloud*.

3.1.1 Janela *Climawin*

Em primeiro lugar, a janela *Climawin*. Esta janela é convencional, como muitas outras: tem uma persiana com diversas folhas, ou divisões horizontais que permitem isolar o interior da temperatura e luminosidade exteriores. Através de botões colocados na moldura da janela, o utilizador pode subir e descer as persianas, bem como fazer configurações. No entanto, estas janelas têm mais funcionalidades do que outras, pelo que lhes são acrescentadas alguma complexidade. Têm sensores e são inteligentes ao ponto de atuar conforme as condições ambientais medidas. A sua comunicação constante com o controlador central (ZC), permite um controlo total e monitorização das janelas de uma habitação. Esta comunicação é feita periodicamente, de maneira a minimizar o consumo energético, pelo que a janela não está constantemente a verificar o meio para eventualmente intercetar um telegrama. Uma outra funcionalidade é não só a capacidade de regular a entrada e saída do ar numa divisão, mas também de fazer o pré-aquecimento/arrefecimento do ar, antes do mesmo entrar na divisão. Esta façanha é alcançada através de uma simbiose entre a caixa de ar da janela e as suas válvulas.

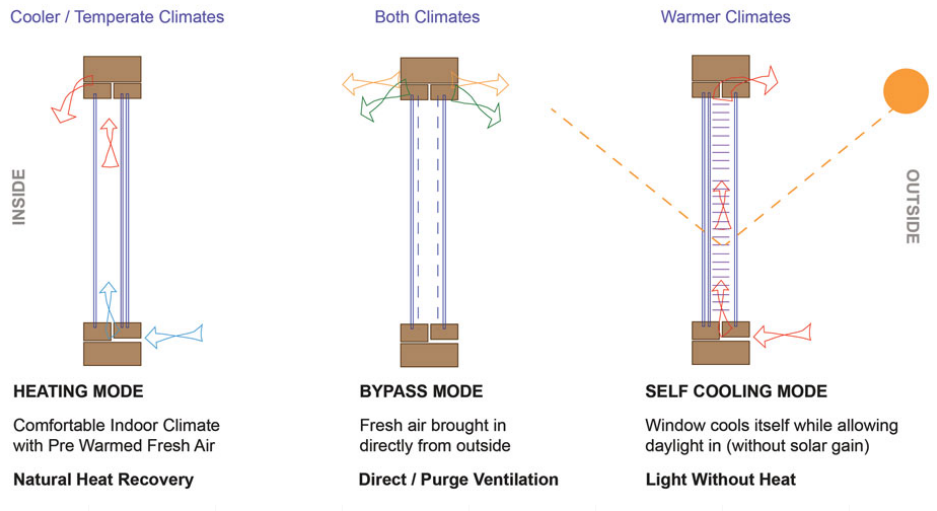


Figura 3.2: Cenários da janela

A figura anterior **3.2** representa diferentes cenários ou estados que são aplicados às válvulas das janelas. Estes cenários são fruto de um algoritmo que tem como entrada os valores de diversos sensores. Desta forma, cada cenário tende a contrariar o ambiente adverso ou menos confortável encontrado na divisão em que a janela se encontra.

O primeiro, "Heating Mode", tende a aumentar a temperatura interior e diminuir a humidade através do pré-aquecimento do ar puro, "fresh air", que entra na caixa de ar da janela. O "Bypass Mode" serve para haver troca de ar livremente entre o exterior e o interior de forma a arrefecer a divisão da habitação. Quando a temperatura exterior é elevada o modo "Self Cooling Mode" é ativado. É barrada a entrada ao ar exterior e a janela arrefece através da circulação.

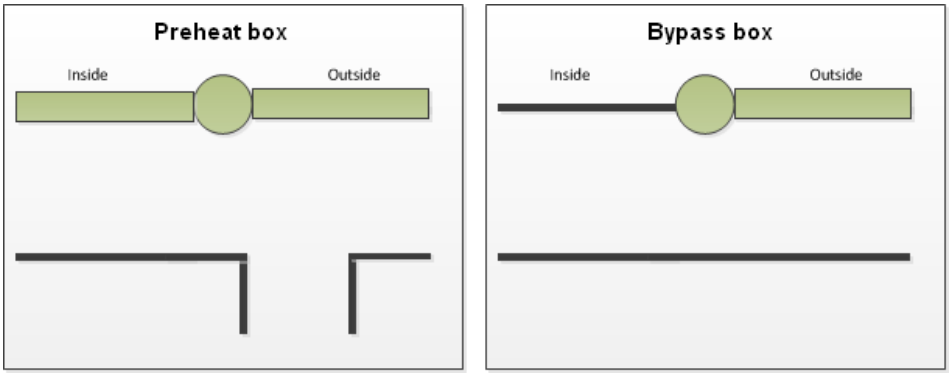


Figura 3.3: Caixa das válvulas colocadas no topo da janela

A presente figura **3.3** é um esquema das válvulas quando ambas estão abertas. As válvulas são compostas por duas caixas divididas por um separador e estão as

duas colocadas lado a lado ao longo do topo da janela. Cada janela tem duas abas(dentro e fora), duas caixas de válvula e uma caixa de ar. Uma das caixas de válvula tem a caixa de ar diretamente por baixo (caixa esquerda da figura **3.3**, "Preheat box"), de maneira a controlar a entrada e saída do ar exterior. A outra caixa de válvula (caixa direita da figura, "Bypass box") é necessária para a extração do ar diretamente para o exterior. Através do movimento das duas abas das válvulas é possível alterar o cenário para a janela se comportar de forma desejável.

Estes diferentes comportamentos das válvulas são necessários para controlar o ambiente interior e serão explicados adiante.

Figura 3.4: Sensor de humidade e temperatura SHT21

Aquando do controlo do ambiente, cada janela tem dois sensores de humidade e temperatura (figura **3.4**) SHT21, da *Sensirion* com interface I^2C, posicionados do lado de dentro e fora da janela. Para calcular o cenário das válvulas, de forma a contrariar as condições adversas, é necessário o utilizador definir parâmetros como limites destes valores obtidos pelos sensores. O utilizador pode fazê-lo através de botões deslizantes colocados na janela.

A concentração de CO_2 também pode entrar na equação, caso o cliente o deseje. Se assim for, a instalação de um sensor de CO_2 *COZIR* [33], poderá adicionar esta camada de controlo à qualidade do ar. Diferente dos outros sensores que estão colocados junto à janela, este sensor de CO_2 é instalado juntamente com o ZC, e tem ligação direta com o mesmo. O ZC é o dispositivo responsável por ler valores do sensor, e, assim que o fizer com sucesso, enviar o valor obtido para as janelas. Esta solução evita custos elevados que seriam implicados por vários sensores CO_2. No entanto, a instalação do ZC convém ser feita de forma a abranger o espaço da divisão em que se pretende controlar o ambiente.

A par dos sensores de temperatura e humidade, as janelas têm também um sensor de luminosidade do lado exterior. Mas o valor obtido por este sensor não entra no algoritmo que calcula o cenário das válvulas. Não obstante, entra no

cálculo do cenário da persiana, que de acordo com o valor da intensidade do sol, move-se para ajudar a encontrar as condições ideais.

Do ponto de vista energético, a janela é alimentada através de quatro baterias Ni-MH de 1.2v [34], que permite maior autonomia e independência energética à janela. O *energy-harvesting* é também um tema muito em voga, devido ao crescimento da ideologia ecológica e da necessidade da humanidade ser menos dependente de combustíveis fósseis. Posto isto, a janela é também capaz de recolher energia e carregar a bateria através da energia solar por intermédio de um painel devidamente colocado do lado de fora da janela. Foi definido um cenário de teste com a persiana a mover-se a cada 30 minutos e com o funcionamento normal da janela, isto é, a calcular cenários através da leitura dos sensores e a mover também as válvulas. Não foi utilizada qualquer recolha de energia por parte da janela, pois o teste foi efetuado no interior, no entanto concluiu-se que a janela teve capacidade para mover a persiana e executar o resto das funções até ficar sem energia durante 250h, ou seja, 10 dias e 10 horas. Outros cenários deverão ser executados e testados para fornecer estimativas mais precisas ao cliente.

Em relação ao estado/cenário que as duas válvulas da janela podem ter, como já fora referido anteriormente, dependem do valor obtido dos sensores. As diferentes grandezas medidas pelos dois sensores têm diferentes pesos no algoritmo que define estes cenários. No total, são feitas quatro medidas, duas por cada sensor, de humidade e temperatura. O valor da concentração de CO_2 é facultativo, no entanto, pode também entrar no cálculo. A temperatura interior é a grandeza natural que mais peso tem no algoritmo, seguido do valor de CO_2, o valor da humidade interior e finalmente a temperatura exterior. A humidade exterior serve apenas para monitorização e informação para o utilizador.

De uma maneira muito racional, quando a temperatura interior é elevada, ou acima do desejado, o que se pretende é a circulação do ar para o exterior, daí o terceiro cenário mostrado na imagem **3.2**. Neste cenário a aba interior está fechada e a exterior aberta. Desta maneira, o ar circula de dentro para fora e o ar que vêm de fora através da caixa de ar não entra na divisão. Presumindo que a temperatura interior está aceitável, o algoritmo entra numa nova fase. A partir do valor de CO_2 obtido (caso não exista sensor é pré-definido um meio-termo), define-se uma percentagem de abertura da aba interior. Esta percentagem é obtida através de um controlo ON-OFF ao longo do tempo, isto porque as válvulas são bi-estáveis(aberta ou fechada). De seguida a humidade interior afeta a percentagem definida pela concentração de CO_2, e aumenta ou diminui de acordo

com o desejado. Por exemplo, se o interior estiver muito húmido, acima do limite definido, o tempo que a aba interior está aberta aumenta, e o contrário acontece se a humidade estiver baixa, abaixo do limite definido. Finalmente, se a temperatura exterior estiver abaixo de uns críticos -5℃ o tempo de abertura também diminui. Com este cenário, é possível aumentar a temperatura interior, através do pré-aquecimento do ar na caixa da janela, tendo sempre em conta a humidade e os valores de CO_2. Esta hipótese está representada no primeiro cenário da imagem **3.2**.

O utilizador, se assim o desejar, através dos botões da janela, pode também abrir as duas abas de maneira a ter uma circulação direta do ar, como demonstra o segundo cenário da imagem **3.2**. Desta forma, o algoritmo não tem qualquer efeito nas válvulas.

Na versão *Climawin* previamente apresentada, a janela tinha subtis diferenças, mas que somadas, prejudicavam o sistema, tanto energeticamente como em termos funcionais. Em primeiro lugar, a janela tinha apenas um microcontrolador STM300 [35] que concentrava em si toda a interpretação dos telegramas *Enocean* bem como a atuação nos motores da persiana e nas válvulas. Este microcontrolador tem um módulo RF com capacidade para enviar sub-telegramas *Enocean* em forma sequencial de forma a poupar energia. Mas a sua principal característica é o reduzido consumo na ordem dos 0.2 μA em "*sleep-mode*", ou modo de poupança. Quer isto dizer que o estado ideal deste microcontrolador é a "dormir". Não obstante, o consumo no envio de uma trama é tipicamente 24mA e a receção na ordem dos 33mA, valores bastante bons que ombreiam com o restante mercado [14]. As janelas *Climawin* "adormecem" periodicamente durante dois segundos, após os quais verifica o que tem a fazer e volta a "adormecer". No entanto, a atuação nas persianas e nas válvulas é um processo moroso e que exige bastante processamento a um microcontrolador que idealmente serve apenas para aplicações simples e rápidas. O controlo nas persianas era feito através da contagem do tempo em que a persiana se movia, prevendo-se a posição da mesma. No entanto, este controlo em malha aberta não é ideal para um sistema que convém ser consistente e preciso. Foi então tomada a decisão de que o controlo da persiana necessitava de um *feedback*. A solução foi a aquisição de um motor DC com um *encoder* já incorporado. Com este sensor é possível contar os passos que que o mesmo dá, obtendo-se então a posição da janela em tempo real. A gestão de interrupções e o manuseamento das mesmas no STM300 não é recomendado, visto que estas mesmas interrupções são utilizadas pela API que a *Enocean* fornece para programar

o microcontrolador *low-power*. Esta fora a razão final para a implementação de um outro microcontrolador *low-power* junto da janela, que ficará responsável pela atuação tanto das persianas como das válvulas.

Finalmente, as válvulas também sofreram modificações. O complexo controlo das mesmas por intermédio de um motor de passo e *feedback* através de um sensor magnético seguido de um ADC não eram os ideais, pois exigia bastante processamento dos dados provenientes do sensor e era uma tarefa um pouco complexa para uma atuação que deveria ser simples. Concluiu-se então que as válvulas não tinham o comportamento desejado e sustentável. Foram então implementadas válvulas bi-estáveis. Através da energização de duas solenoides, uma para cada aba, move num sentido ou noutro, abre ou fecha, de acordo com o terminal escolhido. Neste tipo de atuação não é necessário o *feedback* da posição porque o ato de mover as válvulas é simples e o seu estado é binário: aberto ou fechado. Todos estes processos e funcionalidades serão explicados no capítulo **5**.

3.1.2 *Climawin Zone Controller*

O *Zone Controller* faz principalmente o papel de *gateway* no *Climawin*. É responsável por enviar tramas provenientes da *cloud* para as janelas através de uma *mailbox* (caixa de correio) criada para cada um dos seus dispositivos, neste caso, as janelas. Para cada janela que lhe está associada, o ZC cria uma caixa de correio que guarda a última mensagem destinada à janela. Esta mensagem pode ser proveniente da *cloud*, pode ser ele mesmo a criá-la (para enviar o valor obtido do sensor de CO_2), ou então pode ter origem num interruptor *Enocean* [23]. Ou seja, ao contrário da janela, o ZC está constantemente a monitorizar o meio para intercetar tramas *Enocean*. Por sua vez, cabe-lhe também enviar as tramas no sentido contrário, isto é, das janelas para a *cloud*. Além destas funções, tem também a responsabilidade extra de ligar o exaustor da habitação caso este seja ativado através de um relé *Enocean* quando uma das suas janelas lê uma temperatura interior elevada. Estas comunicações, como mostra a figura **3.1**, são feitas através de duas diferentes tecnologias: série e wireless(rádio), que fazem uso dos protocolos ESP e ERP, respetivamente. Através da comunicação série, o ZC envia os telegramas provenientes da janela para um PC, ou um sistema embebido com a capacidade para comunicar com um *router*, anteriormente denominado como *gateway*, para depois enviar informações para a *internet*.

Um ZC tem cinco zonas ou divisões às quais podem ser associadas três ja-

nelas sendo que o conceito de zona serve apenas para ajudar na configuração e instalação das janelas, mas tem as suas limitações e restrições, provenientes do projeto anteriormente desenvolvido.

- Uma zona tem no máximo três janelas;
- Cada zona pode ter um sensor de CO_2 associado;
- Cada zona pode ter um interruptor *Enocean*.

De maneira a ter um sistema mais uniformizado, o mesmo sensor de CO_2 pode estar associado a mais do que uma zona. O mesmo se aplica aos interruptores *Enocean*. Estes interruptores são dispositivos capazes de enviar telegramas de um só *byte* de informação através da colheita de energia no ato de carregar no botão. Não têm qualquer tipo de bateria ou fonte de energia consumível, o que faz com que seja um objeto peculiar e revolucionário. A partir da energia gerada por um transdutor que transforma energia mecânica em energia elétrica, é gerada energia suficiente para enviar um telegrama RPS, um dos tipos de telegramas *Enocean* [7]. Como já fora referido anteriormente, o sensor de CO_2 é facultativo e os valores de concentração de CO_2 só podem ser obtidos por um ZC, no *Climawin*. Este sensor pode medir até 5000ppm de moléculas CO_2 com uma resolução de 50ppm. A associação de um sensor de CO_2 a uma zona com janelas permite que estas utilizem os valores medidos pelo sensor para criar um cenário de válvulas adequado.

O *Zone Controller* tem também um papel preponderante no que diz respeito ao *feedback* para o utilizador e é também o centro de configuração do *Climawin*. É através de três botões que é possível fazer o *learn*, ou aprendizagem das janelas, associação do sensor de CO_2 e do interruptor a zonas. Com um *display* de 7 segmentos é possível verificar todas estas condições bem como a quantidade de janelas em cada uma das zonas.

Em suma, os serviços já referidos implementados pelo ZC são os que já estavam implementados na versão anterior do *Climawin* à exceção da atuação no relé do exaustor. Nesta nova versão existem dois tipos de *Zone Controller*, o *manager*, ou o mestre, e os *devices*, ou os escravos. Numa instalação *Climawin* só pode haver um *manager*, ao contrário dos *devices*, que podem simplesmente não existir, ou serem vários. O objetivo da distinção é que o mestre terá a possibilidade de fazer um *scan* à rede *Climawin*, com o auxílio dos escravos, e, desta maneira, possibilitar o conhecimento desta mesma rede em níveis mais altos, como na cloud.

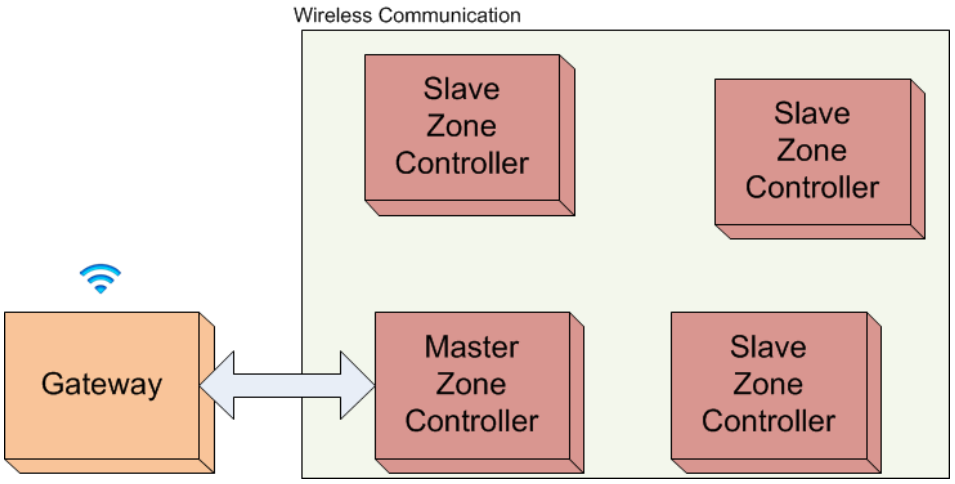


Figura 3.5: Rede de *Zone Controllers*

Todos os ZCs, independentemente da função secundária que tenham, podem ter associadas janelas e desempenhar os processos que já foram referidos anteriormente: controlar janelas, criar tramas para as mesmas e ter zonas para controlar todos estes processos. No entanto, apenas o mestre (figura **3.5**) tem a capacidade de comunicar com o *gateway*, e, por acréscimo, com a *cloud*. Esta comunicação será explicada e detalhada em **4.3.5**.

3.2 Interação entre subsistemas

Neste subcapítulo irão ser mostradas as interações do sistema a um nível aplicacional. A comunicação entre as diferentes partes do sistema, *Zone Controller*, janela e *gateway* são regidas segundo a última camada aplicacional da pilha anteriormente demonstrada na figura **2.4**. Através da API EO3000I, utilizada sobre o microcontrolador STM300, é possível desenvolver diferentes aplicações e criar *profiles* de equipamentos e tramas de acordo com os objetivos estipulados pelo projeto.

Os *profiles*, como já fora referido, servem para identificar o dispositivo que está a enviar a informação. As características técnicas de um dispositivo definem três elementos do seu perfil (ou *profile*):

- O tipo de telegrama ERP (RORG). Este campo define o tipo de telegramas que o dispositivo envia quando tem nova informação sobre os seus sensores.
- Funcionalidade dos dados contidos na trama (FUNC). Campo necessário

para identificar o tipo de dispositivo ou aparelho que está a enviar a trama. Utilizado no processo de aprendizagem.

- Característica ou propriedade do dispositivo (TYPE). Serve este campo para identificação mais específica do que a fornecida pelo campo FUNC. Utilizado no processo de aprendizagem.

Todos os equipamentos *Enocean* que enviam informações de leituras obtidas através de sensores têm um perfil, que pode ser comum entre alguns, se forem aplicados em situações iguais. Não obstante, existe um identificador único para cada um, que os distingue de todos. Este identificador é definido "*hard-wired*" nos microcontroladores *Enocean* no ato de fabrico, logo é responsabilidade da empresa garantir que todos tem um identificador diferente.

Tabela 3.1: Tipos de telegramas *Enocean* utilizados no *Climawin*

Telegrama	RORG	Tipo
RPS	0xF6	Telegrama de interruptor (1 byte de dados)
4BS	0xA5	Telegrama genérico com 4 bytes de dados
SM_LRN_REQ	0xC6	Pedido de learn *Smart Ack*
SM_LRN_ANS	0xC7	Resposta a learn *Smart Ack*
SM_REC	0xA7	Pedido de dados *Smart-Ack*
SYS_EX	0xC5	Telegrama "*Remote Management*"

A tabela presente demonstra os tipos de telegramas utilizados pelos dispositivos *Climawin*. Relativamente ao fluxo de dados, nem todos os equipamentos *Enocean* envolvidos no *Climawin* (ZC, janela, e interruptor) recebem e enviam todos os telegramas. Os interruptores *Enocean* só enviam RPS. As janelas só recebem 4BS, RPS e SM_LRN_ANS e enviam 4BS, SM_LRN_REQ e SM_REC. Por último, os ZCs enviam e recebem 4BS, RPS e SYS_EX, e também utilizam os telegramas "Smart-Ack" para comunicar com as janelas.

Figura 3.6: Conteúdo de telegrama *Enocean* [7]

A figura **3.6** demonstra a estrutura de dados de um telegrama *Enocean* ao

nível fisíco referente à primeira camada do protocolo *Enocean* (figura 2.4). Esta estrutura é referente a telegramas com tamanho maior do que seis bytes, ou seja, representa a maioria dos dados trocados no *Climawin*, no entanto alguns telegramas *Smart-Ack* tem um tamanho menor do que seis bytes.

Em primeiro lugar, o parâmetro "length". Este define o número de bytes de dados que o telegrama contém. Por exemplo, num telegrama 4BS, o conteúdo deste campo é 0x04. O "header" contém informação necessária para endereçar o destino e define o tipo de telegrama (RORG), se for um telegrama isolado. Se o telegrama fizer parte de uma mensagem extensa, que necessite mais do que um telegrama, o tipo de telegrama é definido por um outro campo. Este campo define também a existência, ou não, do campo "Extended Header". Serve este último para fornecer informações sobre o número de vezes que o telegrama fora repetido e também o tamanho dos dados opcionais. De seguida, o "Extended Telegram Type" que, caso o "header" indique que o telegrama é uma extensão, especifica o tipo extensão de que o telegrama faz parte.

A identificação do transmissor e do alvo do telegrama está contida no "Originator ID" e "Destination ID", respetivamente. A origem do telegrama deve ser sempre especificada, ao contrário do destino: quando o transmissor tem como objetivo enviar dados para todos os recetores este identificador é pré-definido como "broadcast". Todos os recetores intercetam aquele telegrama, no entanto, a nível aplicacional o sistema deve responder de acordo com o estado ou as condições adjacentes. Os dados da camada seguinte, "Data Link", são definidos pelo campo adjacente, que o contém . A existência de um "Optional Data" é definida pelo "Extended header". Este campo contém informações relevantes à constituição do telegrama extenso, daí estar contido apenas no "header" extenso. Finalmente, para a garantia de dados consistentes e seguros, o último campo, "CRC", contém o resultado do algoritmo de verificação cíclica de todos os dados do telegrama.

A nível aplicacional estes dados são facilmente acessíveis através de estruturas que dividem os diferentes campos. Desta forma é possível obter toda a informação de um telegrama e filtrar os resultados de acordo com a aplicação.

3.2.1 Comunicação ZC - Janela

O *Zone Controller*, alimentado diretamente através da rede elétrica, tem o recetor RF sempre ligado à escuta de telegramas provenientes de qualquer local.

A janela, por sua vez, é alimentada por uma bateria e, de forma a poupar energia, acorda de 2 em 2 segundos para verificar se existe algo a fazer, quer através de contadores, que vão decrementando com o tempo, quer através de telegramas recebidos pelo ZC. Esta sincronização entre as janelas e o ZC é assegurada pela comunicação bi-direcional "Smart-Acknowledge", abreviado "Smart-Ack" [19]. Através da sincronização é possível ativar o recetor e emissor do dispositivo *low-power* o mínimo de tempo possível, garantindo assim uma troca de informações bastante rentável. Por sua vez, o *postmaster* cria uma *mailbox* para cada um dos seus dispositivos. Quando o dispositivo acorda, faz uma requisição de dados, e o *postmaster* envia o conteúdo da caixa de correio. No *Climawin*, o *postmaster* e o dispositivo *low-power* são respetivamente o ZC e as suas janelas. Esta troca de dados "Smart-Ack" bi-direcional é apenas para o ZC enviar mensagens para a janela de forma a poupar o máximo de energia. Não obstante, a janela também envia telegramas, mas quando o faz não necessita de indicar o destino do telegrama. Todos os dispositivos à escuta podem intercetar e interpretar a mensagem porque estas são de atualização ou "update" de dados, e, neste caso, essa função cabe ao ZC mestre.

Enocean Smart-Ack

Quando interceta um telegrama, o ZC verifica se alguma das suas janelas é o destino do telegrama. No entanto, para fazer de *postmaster* de uma janela, o ZC tem de conhecer as suas janelas, de forma a criar mensagens na caixa de correio para o destino correto. Este "conhecimento", ou "learn", é o processo inicial que garante a comunicação bi-direcional "Smart-Ack" entre um dispositivo e um *postmaster* e é efetuado no momento em que os dois equipamentos trocam informações relevantes ao *profile* do dispositivo *low-power*, neste caso a janela. Serve este procedimento também para registar o *profile* do dispositivo, para assim ser conhecido, a nível aplicacional, o tipo de dados que este está a transmitir. No *Climawin* estas tramas são trocadas quando uma combinação de botões é pressionada na janela e o ZC está em modo de aprendizagem. Para iniciar o modo de aprendizagem "Smart-Ack" no ZC é apenas necessário utilizar o botão de navegação dos menus e selecionar uma das zonas. Neste momento, até três janelas podem ser "aprendidas" pelo ZC. Se por ventura a zona estiver sem vagas, isto é, com três janelas já registadas, o pedido de aprendizagem por parte do dispositivo, neste caso, a janela, é rejeitado.

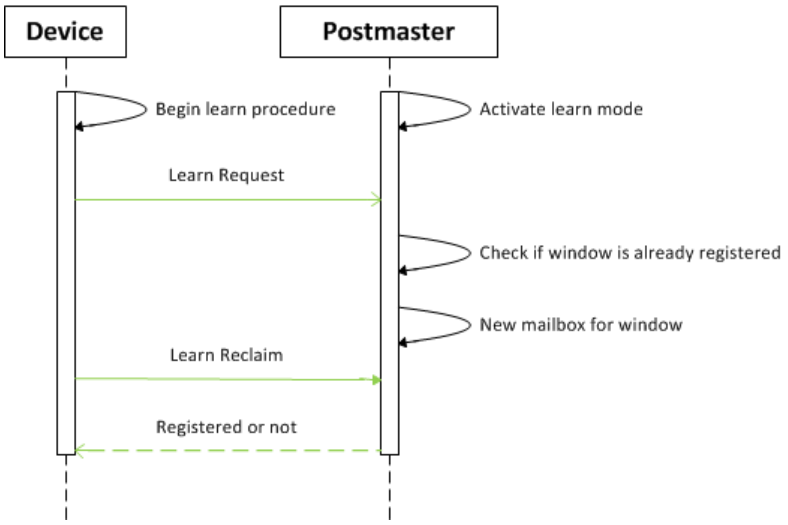


Figura 3.7: Diagrama de sequência Smart-Ack: *learn*

Quando o telegrama de aprendizagem é enviado pelo dispositivo (figura **3.7** "learn request"), este entra num modo *low-power* durante 500 ms, tempo pré-definido para depois enviar um "request" ao *postmaster*. De seguida ativa o seu recetor RF numa janela de tempo muito curta de 20 ms à espera de uma resposta do *postmaster*. Quando recebe efetivamente a resposta, desativa o seu recetor RF e avalia o telegrama recebido, concluindo assim a sequência de aprendizagem.

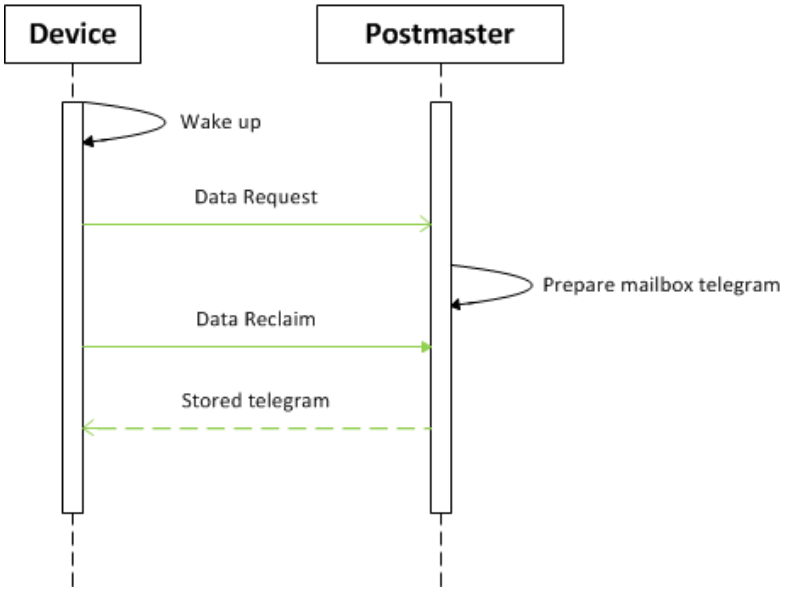


Figura 3.8: Diagrama de sequência Smart-Ack: dados

Quando o dispositivo *low-power* está aprendido, este faz um processo idêntico (figura **3.8**), só que no lugar do telegrama "learn request" e "learn reclaim"

envia "data request" e "data reclaim". De seguida, o *postmaster*, responde com o telegrama que está guardado na caixa de correio previamente criada. No caso do *Climawin*, um "data request" e um "data reclaim" são enviados a cada 2 segundos pelas janelas aos seus respetivos controladores, quando estas "acordam". No caso de não existir nenhum telegrama para ser enviado para a janela, esta recebe um telegrama específico com informação de que a caixa de correio está vazia, e assim entende que não existe nenhuma mensagem para processar.

O telegrama de aprendizagem contém a identificação do *profile* da janela entre os seus campos. Desta forma, através da documentação de perfis *Enocean* [22] é possível identificar o tipo de dispositivo. Como a janela do *Climawin* é um sistema peculiar, porque contém vários sensores e atuadores, foi decidido criar um perfil novo para a janela. No entanto, este está ainda por ser avaliado pela Aliança *Enocean*. Não obstante, o *feedback* recebido pela mesma servirá para melhorar ou alterar este perfil, ou quem sabe, será mesmo adicionado ao portefólio.

Tabela 3.2: *Smart-Ack*: dados do telegrama de aprendizagem

Pedido de aprendizagem	
Tipo de telegrama	0xC6
Tamanho dos dados	10 bytes
Dados	Código do pedido ID do fabricante EEP do dispositivo (RORG, FUNC, TYPE) RSSI (dBm) ID do repetidor
Sub-telegramas	3
Repetido	Não
Origem	Sensor
Destino	N/A

A tabela **3.2** demonstra a estrutura de um telegrama enviado por um dispositivo quando este quer iniciar o procedimento de aprendizagem. O tipo de telegrama é 0xC6 (tabela **3.1**) e o tamanho dos dados é de uns constantes 10 bytes. O código de pedido é também uma constante que é utilizada para efeitos de

verificação de erros. O ID do fabricante contém a identificação do fabricante do equipamento. A Aliança *Enocean* tem também uma lista de fabricadores com a sua devida identificação. O RSSI ("Received Signal Strength Indication") indica a força do sinal recebido em decibéis. O campo EEP contém o *profile* do dispositivo. O ID do repetidor tem a identificação do dispositivo repetidor que enviou a mensagem, caso seja esse o caso, senão o campo é preenchido com um valor por defeito, que é o caso do *Climawin*, que não faz uso de repetidores. O dado "repetido" é consoante o transmissor do telegrama. Se for o dispositivo original, é falso, se for um repetidor é verdadeiro. Nos dados restantes há ainda a identificação da origem do telegrama, o número de sub-telegramas de que é composto o telegrama e o destino que é indefinido, pois um dispositivo *low-power* não tem necessidade de saber com quem está a comunicar pois o reencaminhamento das tramas cabe apenas ao controlador, que neste caso é o ZC.

Perfis *Enocean* no *Climawin*

No que diz respeito aos perfis, como já fora referido anteriormente, foi definido um novo perfil para a janela.

Tabela 3.3: Perfil da janela *Climawin*

EEP da janela *Climawin*	
RORG	0xA5 (4BS)
FUNC	0x15 ("Automated Window")
TYPE	0x00 ("Temperature, humidity and illumination sensors")

A escolha do perfil tem como principal restrição o tipo de telegramas que o dispositivo envia. A escolha do 4BS será justificada mais abaixo. O FUNC escolhido é um dos que estava livre na gama entre 0x00 e 0x3F, que é o máximo. "Automated Window" é o nome escolhido para o campo FUNC porque descreve com generalidade um dispositivo com uma grande gama de aplicações: janelas autónomas. Como é o primeiro tipo da nova função, o valor TYPE escolhido foi zero. "Temperature, humidity and illumination sensors" são características mais aprofundadas do que as definidas pelo FUNC, e as principais unidades medidas pela janela *Climawin*.

Tabela 3.4: Estrutura de um telegrama 4BS

Telegrama 4BS			
RORG	Dados	Origem	Estado
0xA5	DB (4 bytes)	ID (4 bytes)	Contagem repetidor (1 byte)

Segundo a especificação dos *profiles Enocean* [22], um dispositivo deve enviar telegramas de *update* de dados de sensores do tipo RORG do seu perfil. Assim sendo, a janela envia somente telegramas do tipo 4BS (tabela **3.4**). O *payload* do telegrama 4BS é de 4 bytes. Este foi o principal motivo para a escolha do 4BS como RORG do perfil da janela porque estes 4 bytes permitem enviar bastante informação num só telegrama e a escolha de um telegrama com menor carga de dados implicaria escolher um telegrama com apenas 1 byte de dados pois é a escala inferior entre as diferentes possibilidades, o que seria insuficiente. Para identificação do dispositivo, o ID de origem também faz parte do telegrama. O último byte conta o número de vezes que o telegrama foi repetido até chegar ao recetor, caso sejam utilizados repetidores na aplicação. Dados como o ID do destino, ou a força do sinal são comuns entre diferentes telegramas, daí não estar demonstrado na tabela 4BS da documentação *Enocean*.

Outra propriedade de telegramas 4BS é a possibilidade destes mesmos servirem para telegramas "Teach-in". Estes telegramas são enviados pelos dispositivos quando estes recebem um pedido para tal. Nestes telegramas são enviadas informações relativamente ao perfil do dispositivo, e não dados dos sensores/atuadores dos mesmos. Para identificar um telegrama normal de um telegrama "Teach-in" é utilizado o 3º bit do byte 0 de dados do telegrama 4BS. Quando este byte tem o valor negado (zero), significa que o telegrama é de "Teach-in", pelo que os dispositivos têm sempre de comutar este bit para um sempre que enviam um dado. O mesmo acontece com a janela *Climawin*.

A tabela **3.5** detalha o perfil da janela, isto é, os telegramas que a mesma envia em funcionamento normal. Estes telegramas podem ser enviados sequencialmente num curto espaço de tempo, quando a janela faz um "update" ao seu estado, ou então pode enviar os valores isolados quando existe uma alteração no atuador (posição da persiana ou alteração do cenário), dados sempre importantes e interessantes para o *feedback* enviado ao utilizador.

Tabela 3.5: Perfil detalhado da janela *Climawin*

Detalhe do perfil da janela *Climawin*					
Descrição	Byte 3	Byte 2	Byte 1	Byte 0	Unidade
Estado da válvula e posição da persiana	0x80	Posição da persiana	Cenário da válvula	Rotação da persiana	%
Manual ou automático	0x81	Estado Manual	N/A	N/A	Booleano
Bateria	0x01	Carga	N/A	N/A	%
Intensidade da luz	0x02	LSB	MSB	N/A	Lux
Temperatura Exterior	0x03	LSB	MSB	N/A	℃
Humidade Exterior	0x04	LSB	MSB	N/A	‰
Temperatura Interior	0x05	LSB	MSB	N/A	℃
Humidade Interior	0x06	LSB	MSB	N/A	‰

O byte 3 do telegrama identifica os dados que são enviados, entre os quais estão os últimos valores medidos da humidade e temperatura, interiores e exteriores, o valor percentual da bateria e até a intensidade da luz exterior. No que diz respeito ao atuador, também são enviados dados para o utilizador poder monitorizar a posição em tempo real, tanto da posição da persiana, a rotação das folhas da mesma e até o cenário das válvulas. Este último é um enumerado, e cada possível valor define um cenário(figura **3.2**).

No que diz respeito à estrutura de dados, quando é necessário um valor que tem a possibilidade de ultrapassar os 8 bits (1 byte, até 255) o telegrama é formatado em *Big Endian* [7]. Quer isto dizer que o byte mais significativo (MSB) aparece em primeiro lugar no telegrama, neste caso, no byte 1. Assim sendo, e visto que 16 bits são suficientes para abranger todos os valores, o byte menos significativo (LSB) está contido no byte 2. Os valores da temperatura e humidade poderiam ser enviados com um só byte, porque a humidade é normalmente apresentada em percentagem e a temperatura dificilmente ultrapassa os 40℃. Estas gamas de valores são menores do que 255, que é o valor máximo de um byte, no entanto entendeu-se que estes valores deveriam ter a precisão de casas decimais. Daí serem

enviados os valores de humidade em permilagem e temperatura em décimas logo são necessários 2 bytes para o efeito.

A opção "Manual ou automático" descreve o estado de uma propriedade da janela. Como a janela calcula um cenário para as persianas, estas podem-se mover sozinhas. No entanto, seria um pouco redundante o utilizador mover a persiana, e, de seguida, ela mover-se por si própria em função do resultado do algoritmo de controlo. Assim sendo, o utilizador tem a opção de definir o estado manual, em que o cenário da persiana não tem qualquer efeito sobre a sua posição. Este estado é também enviado para a *cloud*, para efeitos de monitorização.

O Zone Controller por sua vez, não é "aprendido" por nenhum dispositivo, no entanto a definição de um perfil para este equipamento é necessária porque o ZC obtém o valor de um sensor e envia telegramas com dados do mesmo. Neste caso, foi selecionado o melhor perfil para um sensor CO_2 *Enocean* e foi escolhido o "Pure CO_2 Sensor" [22].

Tabela 3.6: Perfil do ZC *Climawin*

EEP do ZC *Climawin*	
RORG	0xA5 (4BS)
FUNC	0x09 ("Gas Sensor")
TYPE	0x08 ("Pure CO_2 Sensor")

Com este perfil é possível enviar valores de concentração de CO_2 até 2000ppm, o que é suficiente para o sistema *Climawin*. A partir dos 1500ppm o ar de uma divisão é já considerado desfavorável, portanto, acima desse valor é irrelevante até para o algoritmo de controlo das válvulas.

Sequência de comunicação

A sequência de telegramas e dados trocados permite entender o funcionamento do sistema de uma maneira generalizada e abstrata. A comunicação entre o ZC e as suas janelas é efetuada sempre por intermédio de sinais RF. Não obstante, estas comunicações podem ser encadeadas por dados série provenientes da *cloud* ou até de um interruptor *Enocean*. Depois, através do *Smart-Ack*, quando uma janela "acorda" recebe e interpreta o telegrama.

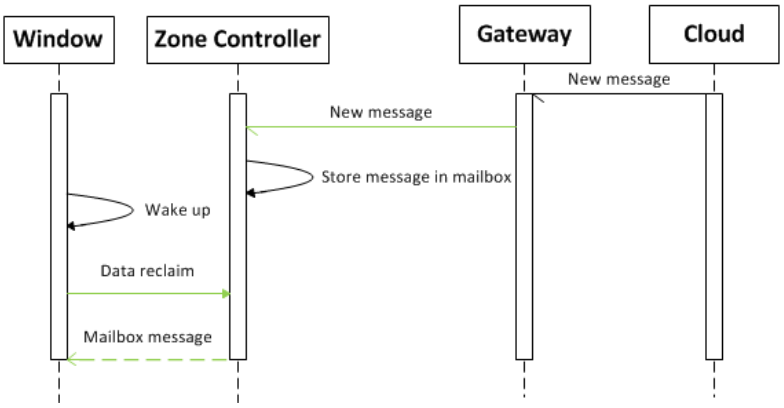


Figura 3.9: Diagrama de sequência da comunicação entre ZC e uma das suas janelas

A figura **3.9** representa o comportamento do sistema e o fluxo de informação do mesmo quando é necessário enviar uma mensagem a uma determinada janela, para esta atualizar dados ou atuar tanto na persiana, como nas válvulas. As mensagens a verde são protocolo *Enocean*, quer sejam radio ou série. Neste exemplo, o ZC recebe um pacote série ESP, por exemplo, para abrir a janela. Este pacote é criado pelo *gateway*, que recebe uma mensagem genérica da *cloud*, que terá sido acordada entre as duas partes *à priori*. Depois de os serviços estarem especificados, o *gateway* tem de traduzir o que recebe da *cloud* para ESP, de forma ao ZC analisar as mesmas de acordo com a especificação do protocolo *Enocean*.

Depois de receber o pacote série, o ZC mestre verifica o destino da mensagem, e, caso seja uma das suas janelas, guarda-a na caixa de correio da janela correspondente. Caso contrário, envia o telegrama via rádio, e todos os ZC escravos intercetam-no e interpretam-no. Se o destino for uma das suas janelas, guardam a mensagem na caixa de correio, caso contrário, ignoram a mensagem.

Neste exemplo é possível ver a obtenção de dados por parte da janela através do "Smart-Acknowledge". Estes dados serão um telegrama *Enocean*, devidamente convertidos pelo *gateway*, após receber uma mensagem da *cloud*, estruturados de acordo com o perfil da janela de forma à mesma interpretar a trama corretamente.

O ponto de partida que gera esta cadeia de dados pode não ser a *cloud*, porque um interruptor *Enocean* pode iniciar processo semelhante. No entanto, neste caso, o *gateway* não entra no esquema, porque o ZC interceta o telegrama diretamente do ar. Quando isto acontece, verifica se o interruptor está associado a uma zona, e cria mensagens de correio na *mailbox* das janelas destinadas.

É com base neste diagrama de sequência que o utilizador/cliente de um

produto *Climawin* pode, em qualquer local, aceder à sua conta, através dos serviços fornecidos pela *cloud* e atuar em qualquer janela das suas habitações.

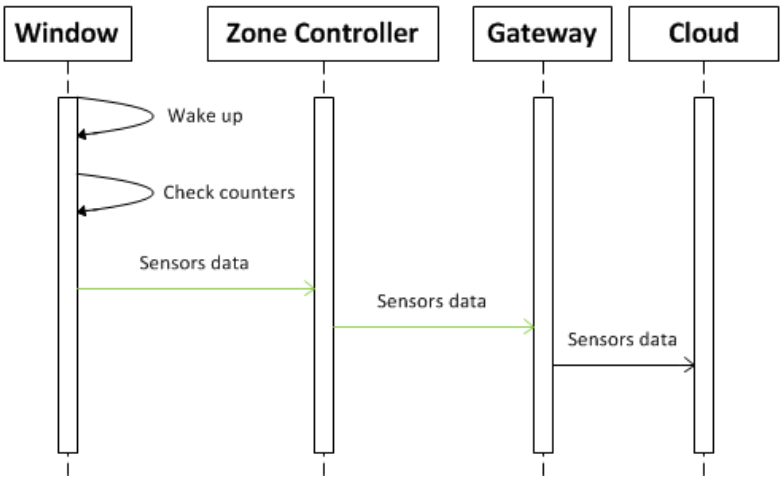


Figura 3.10: Diagrama de sequência da comunicação de uma janela e o ZC

No sentido oposto, quando uma janela necessita de fazer um "update" à base de dados do utilizador, esta envia diferentes dados dos sensores que contêm. Quando a janela acorda, verifica se está na altura de processar diferentes funções: ler dos sensores, atuar nas válvulas ou persiana ou até executar o algoritmo de controlo para definir cenários. Por intermédio de contadores é possível definir janelas temporais para executar certas funções. Como a janela acorda com um "watchdog timer", múltiplos destes valores podem definir maiores espaços de tempo, e é assim que a janela consegue temporizar todas as suas funções. No que diz respeito a este diagrama de sequência (figura **3.10**), a cadeia de dados que representa é periódica e serve apenas para o utilizador ter uma constante informação dos dados que as suas janelas estão a recolher. A partir do momento que os dados estão na *cloud*, estes podem ser utilizados para diferentes aplicações, como por exemplo monitorização ou manutenção.

Relativamente à sequência de dados, o ZC, que neste caso tem de ser o ZC mestre, apenas reencaminha as tramas de qualquer janela, intercetando-as no ar, e convertendo-as para ESP. De seguida cabe ao *gateway* traduzir estes telegramas para atualizar a informação na *cloud*.

3.2.2 Comunicação Gateway - ZC

Uma *Raspberry Pi* com SO Linux é o portal entre a *cloud* e o *Climawin*. Este pequeno sistema embebido é responsável por traduzir as mensagens provenientes da rede e por sua vez transformá-las em pacotes ESP e enviar as mesmas ao ZC mestre (figura **3.5**). Para construir estas mensagens, o *gateway* recorre ao "EOLink", uma biblioteca fornecida pela *Enocean* para compor e decompor telegramas série de uma forma simplificada e rápida em sistemas embebidos com ambiente Linux.

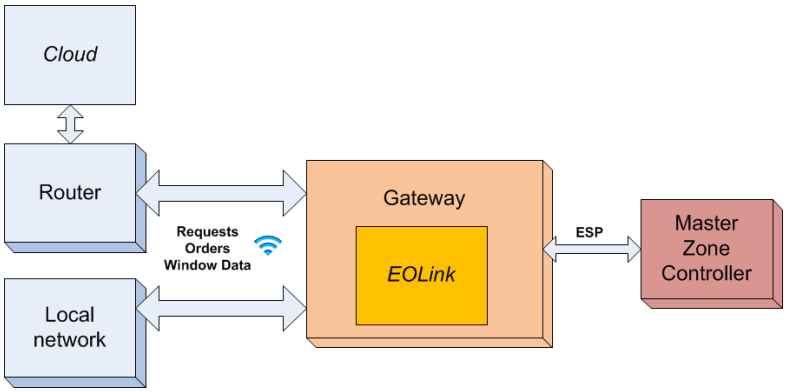


Figura 3.11: Diagrama da comunicação entre *gateway* e ZC

Após a receção de uma mensagem, seja esta proveniente da *cloud* ou da rede local, o *gateway* transforma essa mesma mensagem em pacotes série *Enocean*. O destino destes telegramas é sempre uma janela, porque no fundo os "end devices" do *Climawin* são estas mesmas. Portanto, estes telegramas construídos pelo *gateway*, e prontamente enviados via rádio pelo ZC, são regidos pelo perfil da janela. Quer isto dizer, que a estrutura dos seus dados é equivalente aos telegramas enviados pelas janelas. Desta forma, uma janela que receba um telegrama proveniente da *cloud* interpreta o mesmo tal como o envia, uniformizando e simplificando o sistema como um todo.

O *gateway* deve ser capaz de criar diferentes pacotes *Enocean*, não apenas traduzir mensagens rádio para série e vice-versa. Isto porque o ESP contém vários tipos de pacotes que não estão diretamente relacionados com telegramas. Contém também diferentes comandos que permitem interagir com o dispositivo ao qual está ligado. Este dispositivo é geralmente um controlador da rede, neste caso, o ZC mestre. Estes comandos permitem, entre outros, gerir a rede de sensores e obter informações da mesma. Na secção **4.2** do respetivo capítulo serão especificados os pedidos, comandos ou serviços da *cloud* no *Climawin*. Portanto, nesta comunicação

série o dispositivo controlado é o ZC mestre, pelo que o *gateway* é o controlador.

3.2.3 Comunicação ZC mestre - ZC escravo

A nova versão do *Climawin* terá dois tipos de *Zone Controllers*, como já fora prontamente referido anteriormente. O tipo de comunicação entre os mesmos é uma relação comum de muitos escravos para um mestre. Os telegramas rádio trocados entre os escravos e o mestre são também regidos segundo o protocolo *Enocean*, mas, tal como o "Smart-Ack", fazem uso de uma camada aplicacional (figura **2.4**). Este procedimento é chamado de "Remote Management" [20]("ReMan" abreviado). Consiste num conjunto de funções executadas através de telegramas rádio em que existe a possibilidade de controlar uma rede de sensores a partir de um só ponto de acesso. Tal como as funcionalidades do ESP, mas *wireless*, com o "ReMan" é possível adicionar e remover nós a uma rede e fazer variados pedidos aos diferentes nós. Como esta comunicação é sem fios, existem métodos de bloqueio de segurança e pedidos em que é possível fazer o "scan" à rede, através de difusão de telegramas "Broadcast". A implementação desta funcionalidade será detalhada na secção **4.3.5**.

3.2.4 Comunicação Interruptor - ZC

Depois de associados a uma zona, os interruptores são capazes de mover as persianas das janelas dessa mesma zona. Este relacionamento consiste na verificação da fonte do telegrama enviado: quando o ZC verifica que o telegrama recebido é de um certo interruptor, cria na *mailbox* das devidas janelas um telegrama para esta mesma mover a sua persiana. Este telegrama depois recebido pela janela, aquando do seu "despertar" (figura **3.8**), tem a estrutura do seu perfil, de forma a uniformizar o conjunto de telegramas rádio que circulam no sistema *Climawin*.

3.3 Foco da dissertação no âmbito do *Climawin*

Posto tudo isto, e tendo em conta que o *Climawin* é um projeto com bastantes componentes e diferentes módulos, a proposta desta dissertação é desenvolver duas diferentes partes deste sistema.

Em primeiro lugar, o *Zone Controller*. Este equipamento já se encontrava desenvolvido e pronto para responder às mais primárias necessidades de um simples sistema numa divisão de uma casa. No entanto, não era possível, com as condições de outrora, obter informações de diferentes ZCs numa só instalação. Consequentemente, não existia a possibilidade de enviar telegramas a janelas desses ZCs, porque o sistema não estava desenhado para o fazer. Esta tese tem como objetivo tapar essas lacunas, que são um entrave para o desenho do sistema quando o *Climawin* tem como principal objetivo ser controlado a partir da *cloud*. Não obstante, foram implementadas novas características ao sistema, como a funcionalidade extra do controlo do exaustor de uma casa.

No que diz respeito à janela, não se trata de uma questão de carência de funcionalidades da versão anterior, mas sim da sua consistência e fiabilidade. Como já se referiu em **3.1.1**, o controlo da persiana e das válvulas não era o melhor, pelo que se decidiu adicionar um novo microcontrolador à janela e alterar toda a atuação. O desenvolvimento do software de atuação e a especificação da comunicação entre os dois microcontroladores da janela também fazem parte desta dissertação.

Em suma, esta dissertação tem dois diferentes planos de trabalho, um que diz respeito à implementação de novas funcionalidades e adaptação à *cloud*, e outro que melhora a consistência do sistema, pelo que a estrutura desta dissertação está dividida em duas diferentes partes. A primeira, que diz respeito ao ZC, no capítulo **4**, e a segunda, que foca na atuação da janela, no capítulo **5**.

Capítulo 4

Zone Controller

Neste capítulo será feita a análise aos requisitos da nova versão do novo Zone Controller, bem como a sua implementação. Será efetuada também uma explicação das funcionalidades que já estavam implementadas. De seguida, o estudo das novas funcionalidades que o ZC deve ter, tendo em conta as necessidades de uma aplicação que corra com serviços *cloud* e algumas alterações que o *software* do mesmo foi sujeito. Finalmente, uma das principais finalidades da nova versão do ZC, o controlo remoto, "Remote Management", será também abordado.

4.1 Análise da versão anterior

O *software* anterior do ZC fora desenvolvido tendo em conta algumas restrições e funcionalidades. Com a crescente influência da *cloud* no mercado, e todas a flexibilidade que fornece, o *Climawin* necessitava de algumas alterações para albergar este novo serviço. O ZC e o *gateway* entre o *Climawin* e a *cloud*, neste caso, a *Raspberry Pi*, foram os principais alvos desta reestruturação. Estando o desenvolvimento do *software* do *gateway* ao encargo de um outro responsável, o comportamento do ZC teve de ser alterado.

Recordando um pouco da secção **3.1.2** que trata este assunto, os requisitos funcionais que foram implementados foram os seguintes:

- Conector entre *gateway* e janelas: o ZC era um dispositivo controlador, único no sistema, que tem o papel de fazer a ligação entre o *gateway*, e consequentemente a *cloud*, e as janelas;

- Capacidade para controlar até 15 janelas. Através de 5 diferentes zonas, o ZC consegue registar em cada uma delas 3 janelas, comunicando com as mesmas através do protocolo bi-direcional "Smart-Acknowledge";
- Interruptor *Enocean* controlava a persiana das janelas. Após associação à zona, o ZC é capaz de intercetar o telegrama de um interruptor, e criar mensagens para as janelas;
- Leitura dos valores de CO_2. Por intermédio de um sensor fisicamente acoplado ao ZC, este consegue ler o valor de concentração de CO_2 do ambiente que o rodeia e envia o valor para as janelas.
- Interface para o utilizador controlar todos os equipamentos associados às diferentes zonas: janelas, interruptores e sensor CO_2.

Estes diferentes objetivos foram todos realizados com distinção. O Zone Controller realiza todas as tarefas corretamente sem comprometer o resto do sistema. Foi a partir desta base que começaram a surgir novas ideias e a fixação de que uma instalação *Climawin* deveria ter a capacidade para ter mais ZC e mais janelas. Esta nova restrição implicaria dois tipos de controladores diferentes: um com ligação física, direta ao *gateway*, e outro isolado, apenas com comunicação rádio. Esta solução abstêm os vários controladores de uma instalação terem todos um *gateway* acoplado para se ligarem à *internet*. Basta um ter acesso físico ao *gateway* para conectar toda a rede de dispositivos *Climawin* à *cloud*. No entanto, seria necessário desenvolver um suporte para fazer o *scan* e reconhecer todos os elementos da rede: todos os Zone Controllers e as suas janelas. Este *scan* é propício para fazer um mapa da rede de dispositivos que estão instalados na habitação de um cliente, e, desta forma, fornecer toda a informação deste mapa ao mesmo. A *Enocean* fornece a camada aplicacional para o fazer: "Remote Management", assunto tratado na secção **4.3.5**.

4.1.1 Módulos do ZC

O ZC é composto por diferentes módulos em *hardware* que desempenham diferentes funções auxiliares ao microcontrolador. No que diz respeito à característica de *gateway Enocean* do ZC, entram em funcionamento conjunto o módulo interno RF do STM300 e o módulo de comunicação série. Estes dois elementos são responsáveis por fazer a ligação do sistema *Climawin* ao *gateway* e, consequentemente, à *cloud* por intermédio do ESP.

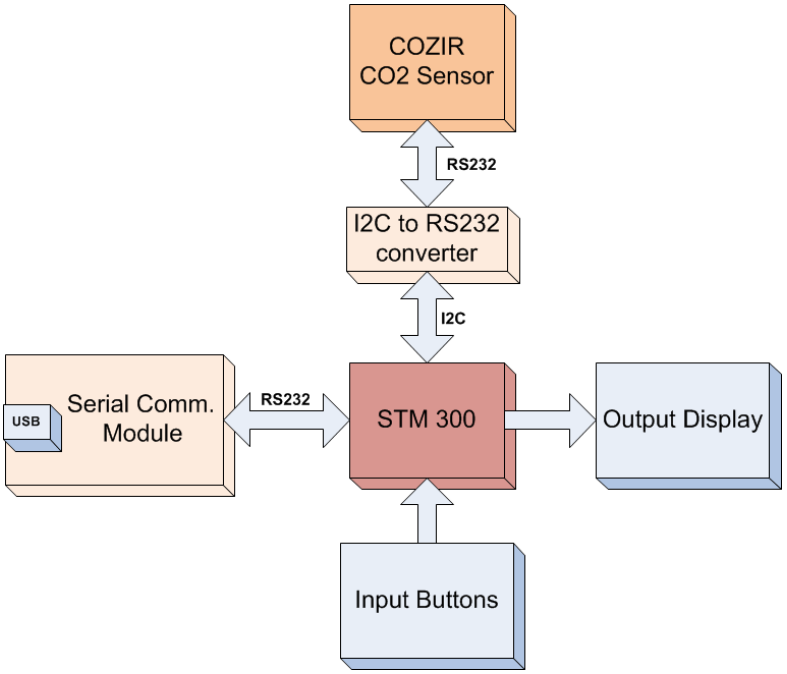


Figura 4.1: Diagrama dos componentes de um Zone Controller

A figura **4.1** demonstra os principais elementos de um ZC. Tal como a janela, o ZC têm um microcontrolador STM300 [35] e todos os outros elementos são módulos de suporte ao mesmo. O módulo de comunicação série consiste num tradutor do protocolo RS232 para USB. Através do módulo UART ("Universal asynchronous receiver/transmitter"), embutido no microcontrolador, é possível utilizar o protocolo RS232 com o ESP da *Enocean* como camada aplicacional. Com a ligação do tradutor é possível obter pacotes ESP em qualquer computador genérico com porta USB. Este módulo é usado apenas no ZC mestre, porque apenas este tem ligação para o *gateway*. Nos restantes ZC, os escravos, o módulo UART não é utilizado para evitar processamento desnecessário.

Para comunicar com o sensor de CO_2 COZIR [33] é necessária uma ligação série UART também, porque este comunica também através do protocolo série RS232. No entanto o STM300 tem apenas um módulo UART apenas, pelo que foi preciso encontrar uma solução para contornar este problema. Para implementar o protocolo *master-multi slave* I^2C são necessários apenas duas ligações. Partindo desta regra, utilizou-se um tradutor I^2C-RS232 para comunicar com o sensor. Assim sendo, a obtenção da concentração de CO_2 da divisão onde o ZC se encontra instalado é obtida através de dois pinos do microcontrolador que, por *bit-banging*, executam o protocolo I^2C. Se o cliente optar por não utilizar um sensor de CO_2 num determinado ZC este tradutor não tem qualquer utilidade, e, por isso mesmo,

o ZC determina dinamicamente a existência, ou não, de um sensor.

Para a criação de um menu foram utilizados botões e um display de 7 segmentos. Os primeiro são dispositivos de entrada, e o segundo de saída.

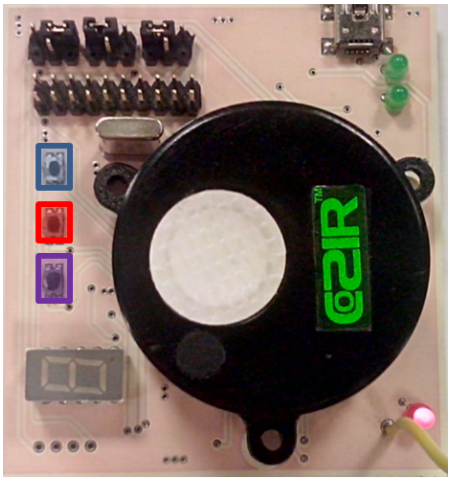


Figura 4.2: Placa do ZC

Como se pode observar, existem 3 botões e um *display*. O botão roxo permite iniciar o modo de aprendizagem *Smart-Ack* (figura **3.7**). É também possível, com este mesmo botão, navegar entre as cinco diferentes zonas e, com o auxílio do botão vermelho, iniciar o processo de aprendizagem do relé do exaustor, bem como iniciar o processo do "ReMan". O botão vermelho permite analisar as zonas: após a escolha de uma destas, pode-se observar o número de janelas, e a associação ou não de um interruptor e do sensor de CO_2 a essa mesma zona. Através do ponto do *display* e de sequências comutadoras do estado dos segmentos do *display* (ligado/desligado) é possível deduzir todos os elementos de uma zona. Este botão permite também apagar toda uma zona, de maneira a libertar o espaço ou a facilitar a manutenção/remoção do sistema. Finalmente, o botão azul permite associar o sensor de CO_2 a uma zona. Após a escolha de uma através do botão roxo, o pressionamento do botão azul "carimba" o sensor de CO_2 nessa zona.

Também visível na imagem está o *display* que é o dispositivo de *output* do ZC, para fornecer diversas informações ao utilizador, bem como um auxiliar na navegação dos menus. Os *leds* vermelho e verdes, são para a indicação da alimentação e ligações série, respetivamente. Uma porta *mini-USB* é necessária para ligar o ZC a um computador ou *gateway*, estabelecendo assim a ligação série entre ambos. No centro encontra-se o sensor de CO_2 *COZIR* e do seu lado oposto, no lado reverso da placa, está o microcontrolador bem como os restantes componentes

de regulação de potência e de auxílio de comunicações.

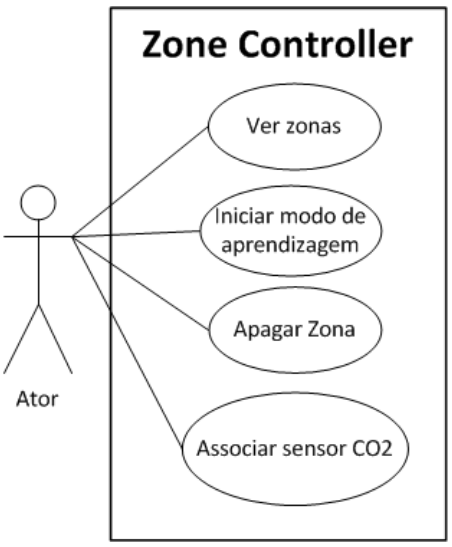


Figura 4.3: Casos de uso do ZC

Na anterior figura **4.3** estão representadas as operações exequíveis no ZC, anteriormente referidas, de uma forma sucinta. O ator representado no diagrama é o utilizador comum do *Climawin*, ou então o instalador do sistema na habitação. Não existe qualquer identificação ou forma de os distinguir, pelo que o utilizador pode também fazer modificações no *Climawin*. No entanto, o ZC será instalado numa zona remota e, à partida, pouco ou nada visível ao cliente, logo este deverá utilizar este interface poucas ou nenhuma vez, uma vez que este *feedback* tem apenas utilidade no momento da instalação.

4.1.2 Sequência de código

Nesta divisão serão apresentados fluxogramas que demonstram a sequência de código executada pelo microprocessador do STM300. Estes diagramas representam o código já desenvolvido, sendo que a sua representação serve para análise da sequência e da lógica do *software* já implementado, servindo assim de base para a nova implementação. Será demonstrado o principal ciclo do código, bem como algumas comparações e sequências essenciais que representam os diferentes módulos presentes no ZC.

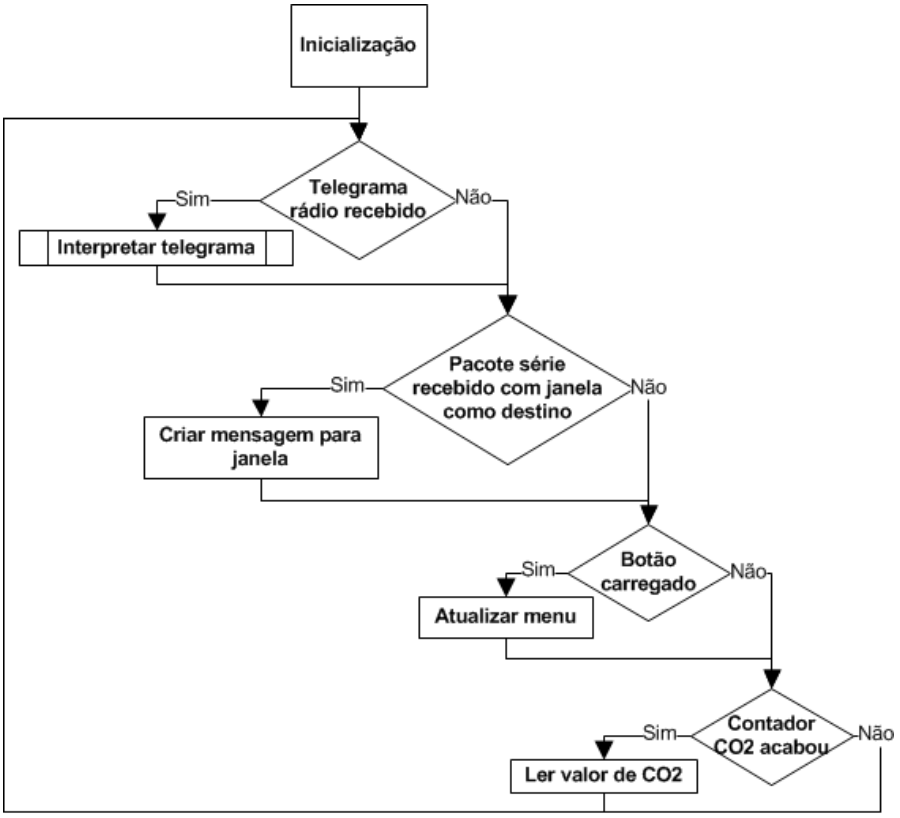


Figura 4.4: Fluxograma da iteração do código *main*

O microcontrolador está constantemente a correr esta sequência de código. Quer isto dizer, que a verificação da existência de algum telegrama ou pacote série nos respetivos "buffers" não é propriamente paralela. O STM300 contém um 8051, um processador "single-core", pelo que o paralelismo é uma propriedade impossível. A gestão de tarefas, ou processos é também um assunto fora de questão, pois é necessário um SO para gerir esse tipo de paradigmas, e o STM300 está longe de ter suporte para tal robustez. No entanto, os módulos RF e UART são periféricos em *hardware*, e, naturalmente tem maneiras de acumular o último dado recebido, portanto é praticável a verificação constante dos mesmos. Os "application notes" da *Enocean* demonstram isso mesmo. As funções que são fornecidas pela API constroem telegramas e pacotes série através deste método iterativo e fornecem informações relativamente a todo o processo: se está em construção, se não existe nada, ou se o processo está finalizado. A figura **4.4** demonstra isso mesmo. Quando um telegrama, ou um pacote série é finalmente recebido, segue-se a sequência definida pelo utilizador.

No que diz respeito ao fluxograma presente, antes de o ZC iniciar o processo

iterativo principal, realiza uma iniciação e configuração dos diferentes periféricos do microcontrolador. Desde o módulo RF, UART, leitura de variáveis da memória flash, temporizadores e "Smart-Ack", tudo é configurado neste primeiro passo com a ajuda dos ficheiros gerados pelo "Dolphin Studio" (capítulo **4.6**).

O ZC está sempre à escuta de telegramas que possa intercetar no ar, e, quando o faz inicia uma interpretação do mesmo. Esta análise será detalhada mais adiante. Por outro lado, quando recebe um pacote série, do *gateway*, o ZC verifica se o destino da mensagem do pacote é uma das suas janelas. A informação das mesmas está guardada numa posição específica na memória *flash*, isto para manter informação sobre as mesmas aquando de uma falha de energia ou algum problema de alimentação. Se o destino da mensagem proveniente da ligação série for uma janela, o ZC cria uma mensagem na *mailbox* específica, tal como está indicado no "Smart-Acknowledge" (figura **3.9**). O efetivo envio da mensagem via rádio, é efetuado automaticamente utilizando uma função "Smart-Ack" da API da *Enocean*, que se encarrega de verificar se existe alguma mensagem e envia-la para a janela, tal como demonstrado no processo da figura **3.8**.

De seguida é feita a verificação dos *inputs* do utilizador. Quando o utilizador pressiona uma tecla, é feita uma verificação do estado do menu, e se a tecla estiver habilitada, é efetuada a devida alteração. Para gerir o menu foi implementada uma máquina de estados que corre uma diferente sequência de código para cada estado. Estas diferentes sequências de código têm a ver com o *feedback* fornecido ao utilizador pelo *display* e o controlo de outras variáveis que são necessárias alterar para indicar o início de outros processos, como por exemplo, o modo de aprendizagem.

Finalmente, um temporizador ajuda a verificar se chegou a altura de ler o valor da concentração de CO_2. Quando o temporizador sinaliza "time out", o ZC inicia o processo de leitura através do protocolo I^2C, que de seguida é traduzido para RS232 para comunicar com o sensor. Se o sensor de CO_2 estiver associado a zonas, o ZC guarda na *mailbox* das respetivas janelas, uma mensagem que contém informação relativamente ao valor do CO_2. Este telegrama é estruturado segundo o perfil do ZC (tabela **3.6**). Para calibrar o sensor, é utilizado um algoritmo com "zero point". O ZC obtém amostras do CO_2 durante um grande período de tempo, cerca de um dia, e no final, estabelece o ponto mínimo, ou o "zero point". Este valor é o menor de todas as leituras que o ZC fez durante esse período, e é guardado de maneira a estabelecer uma referência nas leituras de CO_2. Após o calculo desta referência, todos os valores obtidos vão ser subtraídos ao "zero point",

estabelecendo-se assim um padrão para qualquer local em que o ZC seja instalado. Nem todas as habitações ou divisões tem os níveis de CO_2 equivalentes, e este algoritmo faculta a instalação do ZC em qualquer local, pois este "adapta-se" e define referências diferentes para qualquer situação.

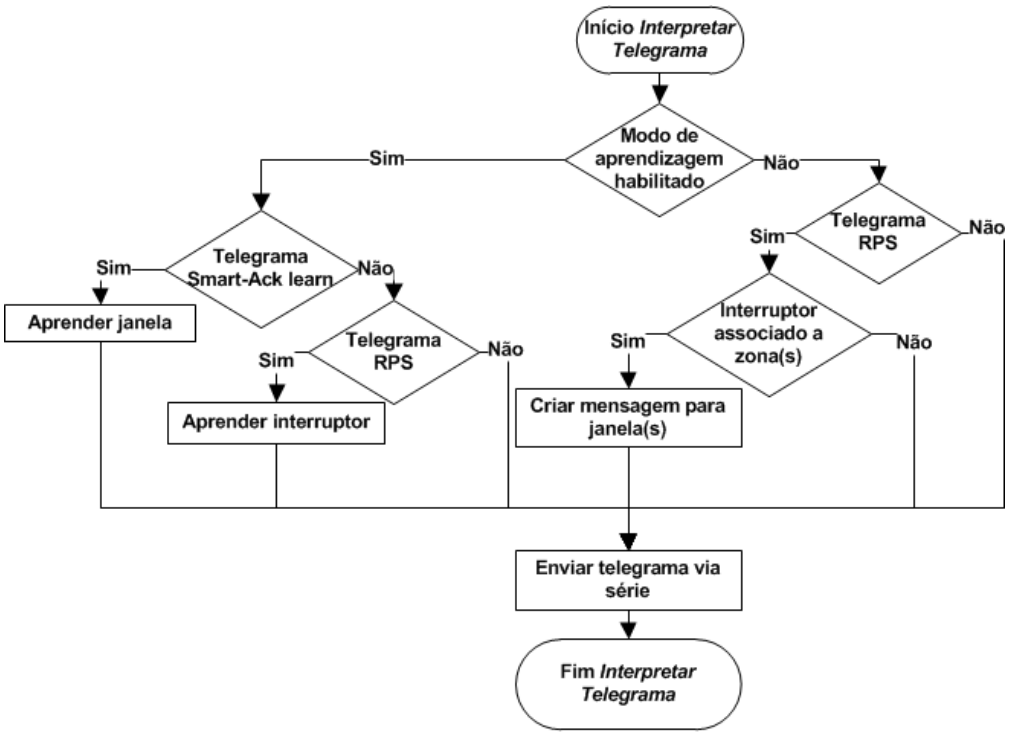


Figura 4.5: Fluxograma de interpretação de telegrama recebido pelo ZC

Este sub-processo representado na figura **4.5** faz parte da iteração principal da *main* (figura **4.4**) e é chamado em "Interpretar telegrama". Como já fora referido, uma função da API fornece informação sobre o estado do telegrama que hipoteticamente o microcontrolador está a receber. Quando o telegrama é efetivamente recebido e está pronto a ser avaliado pelo utilizador, a função retorna uma indicação para tal. Aquando da receção de um telegrama, o ZC filtra os dados recebidos tendo em conta o estado em que está. Logo, nem todos os telegramas são processados, apenas os utilizados no sistema *Climawin* estão sujeitos a avaliação. Não obstante, todos os telegramas recebidos são reencaminhados via série. A avaliação dos mesmos é feita *à posteriori* pelo *gateway* ao qual o ZC está ligado. Não existe qualquer filtro nesta transformação rádio/série pois existe a possibilidade de adicionar novas funcionalidades ao sistema, podendo isso refletir-se através de novos e diferentes telegramas.

No que diz respeito à sequência de código, a primeira condição a verificar é se

o modo de aprendizagem está ou não habilitado. Este modo pode ser ativado pelo utilizador, aquando do pressionamento do botão de aprendizagem (figura **4.2**), para aprender janelas, interruptores *Enocean* ou até o sensor de CO_2 acoplado ao ZC. Se o modo de aprendizagem estiver habilitado, o ZC entra num modo em que interpreta todos os telegramas que recebe como telegramas de aprendizagem. Quer isto dizer que o ZC está neste momento à espera que um dispositivo lhe vai enviar dados para este ser conhecido ou adquirido pelo ZC. Para aprender uma janela, e iniciar o processo de aprendizagem do "Smart-Ack" (figura **3.7**), o ZC ativa uma função especial de "Smart-Ack" para receber estes telegramas. Esta função é também fornecida pela API e retorna um determinado valor quando o dispositivo recebe um telegrama "Learn request". Quando a função assim o dita, a comparação presente no fluxograma guia o código para "Aprender janela", e neste momento o ZC guarda o ID da janela na memória *flash* da zona selecionada pelo utilizador, e conclui o processo de aprendizagem do "Smart-Ack". Se, por ventura, a janela já estiver aprendida, um processo semelhante, mas reverso, acontece: o ZC apaga a janela da sua memória e o telegrama de resposta, depois do "Learn reclaim" enviado pela janela, indica um "learn-out".

Por outro lado, se o ZC receber um telegrama RPS e estiver no modo de aprendizagem, não cabe ao "Smart-Ack" da *Enocean* fazer o "handling" deste telegrama, pois estes telegramas devem ser regidos e controlados pela aplicação apenas, e no caso do *Climawin*, mais especificamente, o ZC, um RPS é a indicação de que alguém pressionou um botão do interruptor *Enocean*. Quando isto acontece, o ZC guarda a identificação do interruptor na memória flash, associando assim o ZC à zona escolhida.

Da mesma forma que o utilizador pressiona o botão do interruptor para aprendizagem do mesmo, este mesmo botão pode ser pressionado para enviar mensagens à janela. No entanto, para o fazer, o ZC não pode estar em modo de aprendizagem, e assim o utilizador pode mover as janelas associadas ao interruptor como lhe aprouver.

Tanto para o caso deste fluxograma como no anterior, o processo "Criar mensagem para a janela" apenas guarda um telegrama na caixa de correio da janela desejada, não a envia. Não obstante, o envio do telegrama é feito quando uma janela acorda e o processo de obtenção de dados do "Smart-Ack" inicia (figura **3.8**). Quando o ZC é configurado para responder a pedidos "Smart-Ack", a API encarrega-se de todo este processo de envio de dados do "postmaster" para o "device". Fica assim explicada a ausência de um processo que se encarregue desta

comunicação nos anteriores fluxogramas.

Estes processos demonstram as principais funcionalidades do ZC: é um *gateway* e "postmaster" do *Climawin*, e também é capaz de obter valores de concentração de CO_2 do meio que o rodeia. No entanto, podem-se apontar algumas lacunas nestes fluxogramas. A primeira e mais relevante é o facto de o ZC não enviar telegramas para o ar quando recebe pacotes série. Desta forma os ZCs escravos não recebem ordens da *cloud*, isto porque a primeira versão do *Climawin* não tinha ZCs remotos completamente abstraídos da ligação série com o *gateway*. Outra lacuna é a limitação de o ZC não criar mensagens para as janelas quando recebe telegramas rádio. Novamente, isto deve-se ao facto de o sistema ter sido pensado para apenas um ZC, e, desta forma, o único ZC presente na instalação criava mensagens 4BS para a janela apenas quando recebia pacotes série ESP. As mensagens 4BS devem ser estruturadas segundo o perfil da janela, e, desta forma, aumentar o número de comandos que o utilizador pode enviar à mesma. Na anterior versão, o utilizador podia apenas mover as persianas através dos telegramas RPS enviados pelo interruptor *Enocean*. Outro defeito é o facto de este dispositivo não responder a pedidos "Common_Command" (tabela **2.1**) provenientes da comunicação série. Estes comandos permitem, entre outros, identificar o dispositivo ao qual o *gateway* está ligado, e desta forma, arbitrar uma função para a ligação que efetuou. É também, a partir destes comandos, que o *gateway* pode pedir informação relativamente aos "devices Smart-Ack" que um "postmaster" tem registados, funcionalidade essencial para criar o mapa de rede.

Em suma, na anterior versão:

- ZC não reencaminha pacotes série para telegramas rádio;
- ZC não cria mensagem para a janela (guarda novo telegrama na caixa de correio) quando recebe telegrama rádio diferente de RPS;
- Aquando da receção de um pacote série, o ZC envia para a janela apenas telegramas RPS, além do valor do CO_2;
- Comunicação do ZC com controlador série, neste caso o *gateway*, é insuficiente para um bom interface entre os dois dispositivos.

4.2 Especificação das novas funcionalidades

Aquando da decisão de adaptar o *Climawin* para corresponder às exigências da *cloud*, o ZC foi um dos dispositivos do sistema que mais teve de ser adaptado, a par do *gateway*. Aumentar o número de janelas, e consequentemente ZCs, é o objetivo primário, de forma a criar sistemas maiores em habitações que assim o exijam. Para implementar o sistema da figura **3.5** é necessário que o *gateway* consiga obter informação de todos os componentes da rede. Isto porque, como já fora mensurado, um serviço na *cloud* está completamente abstraído da complexa ligação da rede de sensores, e, para o utilizador ter acesso a todos, é necessário implementar uma maneira de "descobrir" os mesmos. Em primeiro lugar, foi estipulada uma maneira de fazer o descobrimento da rede. Neste procedimento, o ZC mestre faz um "Broadcast" para todos os escravos e todos os que responderem são adicionados às referências do ZC mestre. Quando um serviço na *cloud* necessita de adquirir informação acerca de todos os ZCs e das suas janelas faz um pedido ao ZC mestre, que por sua vez encarrega-se de obter essa informação e enviá-la para o *gateway*.

Para aumentar o leque de opções de configuração do utilizador, foram também adicionados novos filtros aos telegramas recebidos pelas janelas. Desta forma é possível configurar os limites do algoritmo de controlo de válvulas (secção **3.1.1**) ou optar por escolher um cenário e ignorar o algoritmo e também definir uma posição específica das persianas e das suas folhas. Esta implementação, responsabilidade de um outro elemento da equipa, demanda uma alteração no ZC. Como já fora referido, o ZC não criava mensagens diferentes de RPS na caixa de correio das janelas (figura **4.5**), fator que limitava a quantidade de telegramas que as janelas poderiam receber. Posto isto, o ZC cria agora mensagens 4BS, previamente estruturadas segundo o perfil da janela, para esta mesma receber novos comandos.

Entendeu-se também que seria necessária uma melhor e mais eficaz extração do ar interior quando o ambiente está com elevada temperatura, humidade ou até níveis de concentração CO_2 demasiado altos. Posto isto, numa habitação usual, é necessário utilizar o exaustor embutido normalmente na cozinha para aumentar o fluxo de ar extraído. Um relé *Enocean* é ideal para o fazer, pois consegue comutar o estado de uma linha elétrica de tensão doméstica de 230v/50Hz recebendo telegramas que lhe indicam para o fazer [36]. Estes relés tem a capacidade de entrar num modo de aprendizagem e obter a identificação de dispositivos que lhe enviam telegramas RPS. A partir do momento em que estes dispositivos estão aprendidos,

estes conseguem comutar o estado do relé. No caso do *Climawin*, o ZC consegue abrir/fechar o relé, e por conseguinte, ligar/desligar o exaustor enviado telegramas RPS quando uma das suas janelas está num determinado cenário em que é necessário um elevado fluxo de ar.

Em suma, os requisitos da nova implementação do ZC são os seguintes:

- Reencaminhamento de telegramas rádio aquando da receção de um pacote série por parte do ZC mestre;
- Adição de novos ZCs: implementação de uma rede de controladores. Adaptação do código do ZC mestre para comunicação com os escravos sem comprometer as funcionalidades anteriormente implementadas. Adaptação do código dos escravos para estes responderem e manterem as funcionalidades iniciais;
- Resposta a pedidos ESP provenientes da ligação série do ZC mestre como informação do dispositivo e mapa de rede;
- Criação de mensagens 4BS na caixa de correio das janelas, sejam elas provenientes da ligação série ou de telegramas rádio;
- Controlo do acionamento de um exaustor;
- Criação de novas entradas no menu de navegação.

4.3 Implementação de novas funcionalidades

Nesta secção será demonstrado o trabalho desenvolvido no *software* do ZC no âmbito da dissertação. Esta parte será dividida em 4 diferentes assuntos, além da especificação das ferramentas utilizadas: a criação novas mensagens para a janela, o controlo do exaustor, a resposta a um pedido específico ESP, e finalmente a comunicação entre os diferentes ZCs. Todas estas novas funcionalidades são essenciais para a implementação de um ZC preparado para a *cloud* e todas as exigências que uma ligação à mesma demanda.

4.3.1 Ferramentas utilizadas

A Enocean fornece um *framework* que consiste num conjunto de aplicações e bibliotecas para desenvolver software para o microcontrolador STM300. O desen-

volvimento do software a nível aplicacional recai no IDE *Keil*. Para fazer debug ou configurar módulos do microcontrolador, estão disponíveis um conjunto de programas para utilizar num computador pessoal com SO *Windows*.

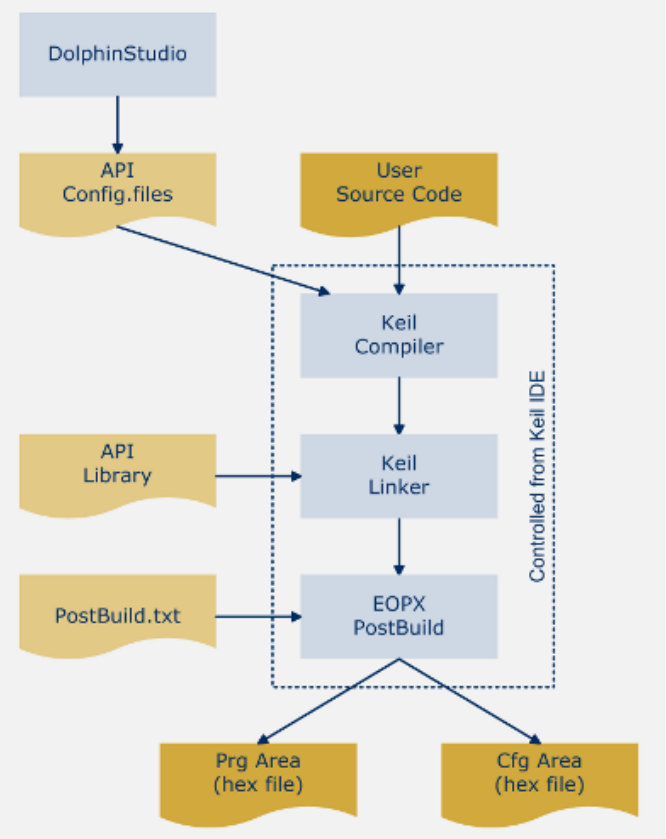


Figura 4.6: Esquema de desenvolvimento de Software *Enocean*

A figura **4.6** representa o processo de construção de código para o microcontrolador STM300.

O "Dolphin Studio" é um programa com interface gráfico que gera ficheiros de configuração da API (figura **4.6**, "API Config.files") de acordo com as escolhas do utilizador. Existe a possibilidade de definir a função dos diferentes pinos do microcontrolador, como funções alternativas, tipo módulos UART ou SPI, ou até PWM ("Pulse-width modulation") e entradas analógicas. Também se pode configurar a inicialização, ou não, de modos aplicacionais *Enocean*, como o "Smart-Acknowledge" ou o "Remote Management". Através da API "EO3000I", uma biblioteca pré-compilada, todas as funcionalidades do *chip* são abstraídas e podem ser utilizadas com facilidade e segurança.

Após a implementação do código no Keil, o compilador de linguagem C *C51*, encarrega-se de transformar toda a solução do utilizador em código legível para o

8051, microprocessador do STM300 [35], o microcontrolador de eleição da *Enocean*, que tem todo o suporte para desenvolvimento. O "postbuild", como o nome indica, é um ficheiro utilizado no final da geração do código e serve para configurar o *firmware* da memória Flash e calcular os códigos CRC da mesma. Com o auxílio da ferramenta "EOPX", que também é utilizada para fazer o *download* do código para o *chip*, este ficheiro gera os dois códigos binários (ficheiros .hex) tanto para a área de configuração da Flash, como a área de código.

Para monitorização e controlo, foi também desenvolvido um programa chamado "Dolphin View" que permite observar com facilidade pacotes ESP no computador pessoal. Ligando um *gateway Enocean* ao computador, este recebe os telegramas via série, que, após serem interpretados pelo programa, são apresentados no ecrã com os dados separados e fáceis de analisar. Esta ferramenta é também ideal para fazer o *debug* de telegramas e entender se está tudo correto.

4.3.2 Criação de novas mensagens para a janela

Com o propósito de aumentar o leque de comandos que uma janela pode receber e interpretar, o filtro que o ZC impunha aos telegramas que recebia foi alterado de forma a aumentar o número de comandos que uma janela poderia receber. Foi adicionado então uma nova condição ao filtro de telegramas recebidos, para criar mensagens na *mailbox* das janelas quando estes telegramas fossem do tipo 4BS. Os telegramas são, à partida, estruturados segundo o perfil da janela, para esta mesma conseguir interpretá-los da mesma maneira que os envia. Se não forem estruturados segundo o perfil da mesma, a janela não os consegue interpretar, portanto, ignora-os. A janela pode então obter os dados de uma forma simplificada, pois as variáveis estão endereçadas de maneira igual. É também uma forma de uniformizar o sistema e de não aumentar o tipo de telegramas utilizados pelo mesmo.

Todas as situações aqui referidas partem da norma de que os telegramas/pacotes série são endereçados ou tem como destino janelas aprendidas pelos ZC.

À parte das mensagens de CO_2, que são automáticas e enviadas periodicamente, ambos os tipos de ZCs devem criar mensagens na *mailbox* das suas janelas quando recebem comandos do utilizador (figura **3.9**). Estes comandos do utilizador podem ter duas fontes: a *cloud* ou um interruptor *Enocean*. Para ambos os ZCs, as ordens provenientes de um telegrama originado por um interruptor são

interpretados da mesma maneira: o telegrama é do tipo RPS, então é criada uma mensagem na *mailbox* (figura **4.5**).

O problema está aquando da receção de uma ordem da *cloud*. No caso do ZC mestre as ordens da *cloud* manifestam-se quando recebe um pacote série do tipo "Radio" (tabela **2.1**), que fora devidamente criado pelo *gateway* e originado através de uma ordem da *cloud*. Segundo o fluxograma da figura **4.4** facilmente se entende que para criar mensagens para as janelas quando se recebe um pacote série não se aplica nenhum filtro. Quer isto dizer que todos os pacotes série do tipo "Radio" que o ZC mestre recebe são guardados na caixa de correio das janelas. Relativamente ao ZC escravo, a situação é diferente. Uma ordem proveniente da *cloud* não se manifesta de maneira idêntica. Quando o ZC mestre recebe um pacote série em que o destino não é uma das suas janelas, este envia o telegrama via rádio, porque, à partida, o telegrama tem como destino uma janela que não a sua. Recebendo este telegrama, o ZC escravo tem de criar a mensagem para a sua janela. No entanto, segundo o fluxograma da figura **4.5** um telegrama do tipo 4BS, do tipo do perfil da janela, não tem seguimento para a janela. A condição testada é apenas se o telegrama é do tipo RPS. Para adicionar as novas funcionalidades e comandos à janela é necessário criar mensagens do tipo 4BS. Quer isto dizer que agora o ZC escravo cria também mensagens 4BS para as suas janelas, além das mensagens de CO_2.

Numa perspetiva mais minuciosa, poder-se-ia então aplicar outra camada no filtro do telegrama 4BS. Visto que o telegrama que o ZC guarda na caixa de correio é para a janela, este telegrama deve ser estruturado segundo o perfil da mesma. Segundo o perfil da janela (tabela **3.5**) o byte 3 identifica ao quê que diz respeito os dados dos seguintes bytes. Logo, o ZC poderia filtrar e ignorar tudo o que não abrangesse esses valores possíveis para o byte 3 segundo o perfil da janela. No entanto entendeu-se que eventualmente, numa nova versão, o perfil da janela poderia ser alterado, e, consequentemente, a mesma poderia receber uma maior variedade de telegramas.

Em suma, o filtro que dita se um telegrama é guardado na caixa de correio de uma janela é se o destino é correspondente, como não poderia deixar de ser, e também se o tipo de telegrama é RPS ou 4BS.

Numa perspetiva mais técnica a criação de mensagens na *mailbox* das janelas é um processo muito simples para quem implementa o *software*. Uma função da API da *Enocean* trata de criar a mensagem na caixa de correio tendo como

argumento apenas o telegrama recebido.

4.3.3 Controlo do exaustor

A ideia que justifica esta nova implementação é que a extração do ar do interior de uma casa diminui a temperatura, humidade e níveis de concentração de CO_2 da mesma. A razão que levou à escolha do ZC ser responsabilizado por esta funcionalidade reside no facto de o ZC estar mais próximo de um exaustor e a facilidade de configuração do mesmo.

Quando uma janela define, por algoritmo, o estado das suas válvulas envia o cenário via rádio num telegrama estruturado segundo o seu perfil (tabela **3.5**). Se for necessária uma ventilação direta do ar (figura **3.2**) é necessário um auxílio para as mesmas, e o ZC encarrega-se de ligar o exaustor. Quando o cenário é qualquer um dos outros, o ZC desliga o exaustor. Logo, indiretamente, todas as janelas do ZC tem a capacidade de ligar/desligar o exaustor.

No novo *Climawin* é possível haver mais do que um ZC. Fica então a questão de qual ZC é que vai controlar o(s) exaustor(es) da casa. O critério fica ao entender do instalador do sistema. Se se entender que as janelas de um determinado ZC devem ser capazes de ligar o exaustor porque estão mais próximas do mesmo, ou então porque é uma zona da casa que necessite de elevada ventilação por outros motivos quaisquer pode-se configurar o ZC para tal. Parte-se então do princípio que as janelas que têm esta capacidade estão colocadas na mesma fachada da habitação para as mesmas não se contradizerem através dos seus algoritmos de controlo de válvulas. Se isto acontecer, a janela que enviar o cenário por último vai sobre-escrever o cenário da outra, e o ZC vai comutar o estado do relé do exaustor sem qualquer influência nesta disputa, pois é um elemento passivo na definição do estado do mesmo.

Antes de mais, o relé *Enocean* [36] tem dois botões para configurar os seus dispositivos. Como já fora referido, o relé é acionado quando recebe um telegrama RPS de um interruptor *Enocean*. Estes telegramas tem um byte de dados, e é este byte que define se o relé abre ou fecha. Através de um dos botões do relé é possível ativar o modo de aprendizagem. Neste modo todos os telegramas RPS que o relé receber vão servir para guardar o identificador do dispositivo de origem pelo que também é possível dois diferentes ZCs controlarem o mesmo exaustor. No modo de aprendizagem o relé está constantemente a comutar o estado da sua saída, pelo

que é recomendável que não se ligue o exaustor ao mesmo antes de o ZC estar configurado como dispositivo associado ao relé. O motivo pelo qual isto acontece deriva do facto de haver necessidade de indicar ao utilizador de que o relé detetou e aprendeu o um dispositivo *Enocean* a partir de um telegrama recebido. Quando o relé efetivamente recebe um telegrama e guarda o identificador do emissor deixa de comutar o estado da saída durante 4 segundos dando sinal de que a operação foi concluída com sucesso. Quando se desliga o modo de aprendizagem, o relé entra em funcionamento normal e todos os dispositivos que este "aprendeu" são capazes de o controlar. O outro botão serve apenas para apagar o registo de todos os dispositivos.

Para configurar o ZC controlador do exaustor foi criada uma nova entrada no menu navigacional para o efeito. Através de um dos botões (figura **4.2**, botão roxo) pode-se não só navegar através de zonas e iniciar o modo de aprendizagem, bem como entrar no menu "fan". Este menu é representado pelo *display* com um simples ponto a piscar para o utilizador entender que está prestes a configurar o relé. Depois de carregar num outro botão (botão vermelho), o ZC simula um interruptor, que no fundo consiste em enviar um telegrama RPS estruturado segundo o perfil de um destes dispositivos.

Tabela 4.1: Perfil de interruptor *Enocean*

EEP de um interruptor *Enocean*	
RORG	0xF6 (RPS)
FUNC	0x02 ("Rocker Switch, 2 Rocker")
TYPE	0x01 ("Light and Blind Control - Application Style 1")

Os interruptores usados no *Climawin*, para associar a zonas e controlar as persianas, têm este perfil. Este interruptor tem 4 botões. O byte de dados do mesmo indica quais botões foram pressionados, ou então libertados. Para registar o ZC no relé foi simulado o botão de abrir quando pressionado, segundo o perfil. Basicamente consiste em enviar um RPS com o dado igual a 0x10. Após o processo de aprendizagem, do lado do relé, estar concluído, pressiona-se um botão no mesmo, e a aprendizagem finaliza. A partir desse momento, sempre que o ZC envia um RPS com dado 0x10 o relé fecha e o exaustor liga. Quando manda um dado igual a 0x30 que, segundo o perfil do interruptor, é o valor do byte de dados quando o botão de fechar é pressionado, o relé abre, e o exaustor para.

Desta maneira, quando um ZC recebe informação de que uma das suas janelas está num cenário em que é necessária extração de ar, o ZC liga o exaustor. No sentido oposto, quando uma das suas janelas indica que as condições ambientais não necessitam de circulação do ar, o ZC desliga o exaustor.

4.3.4 Resposta a pedido de versão

O protocolo ESP, além de ser um excelente auxiliar a um módulo de *gateway*, devido à simples conversão de telegramas em pacotes série, consiste também em ordens do controlador e respostas do dispositivo ao qual este está ligado via série. Estes pedidos e respostas servem para obter dados do dispositivo controlado, bem como dar início a diferentes procedimentos. À exceção de alguns comandos, todos devem ter uma resposta para indicar o controlador, neste caso o *gateway*, que tem uma ligação série ao dispositivo, de que o sistema está operacional.

Existe, no entanto, um comando comum a todos as aplicações *Enocean* que consistem num computador a controlar um dispositivo *Enocean* via comunicação série (RS232). Este comando é o pedido de versão [21]. É um simples pacote série do tipo "Common command" que tem apenas um byte de dados que identifica que o pedido, que neste caso é o "read version".

Nos exemplos aplicacionais do "EOLink" este comando é sempre utilizado inicialmente de forma a verificar a existência de um dispositivo *Enocean* numa determinada linha série e eventualmente identificar a sua aplicação. Depois de enviar o comando, o controlador pode esperar pela resposta, pelo que é um comando bastante pertinente em sistemas que tenham diversas ligações e queiram verificar se alguma delas está de facto relacionada com a *Enocean*. No caso do ZC mestre, é necessário identificá-lo como sendo um dos aparelhos *Climawin*.

Os pacotes ESP podem ter vários tipos, e existe um campo para o identificar. No fluxograma referente à versão anterior do *Climawin* (figura **4.4**) não é referida nenhuma verificação ao tipo de pacote. Nesse caso apenas se reencaminha o pacote para a janela, guardando uma mensagem na caixa de correio, partindo do princípio que este é do tipo rádio. Nesta nova versão, os pacotes recebidos vão ter diferentes tipos, pelo que terão de ser filtrados e analisados. Desta forma, um dos tipos analisados será o "Common command" que, entre outros, tem o comando "Co_Rd_Version" abreviação para "Controller read version" [21], que é precisamente o pedido de versão. Outros tipos serão também analisados, no entanto isso

é assunto para outra secção deste capítulo.

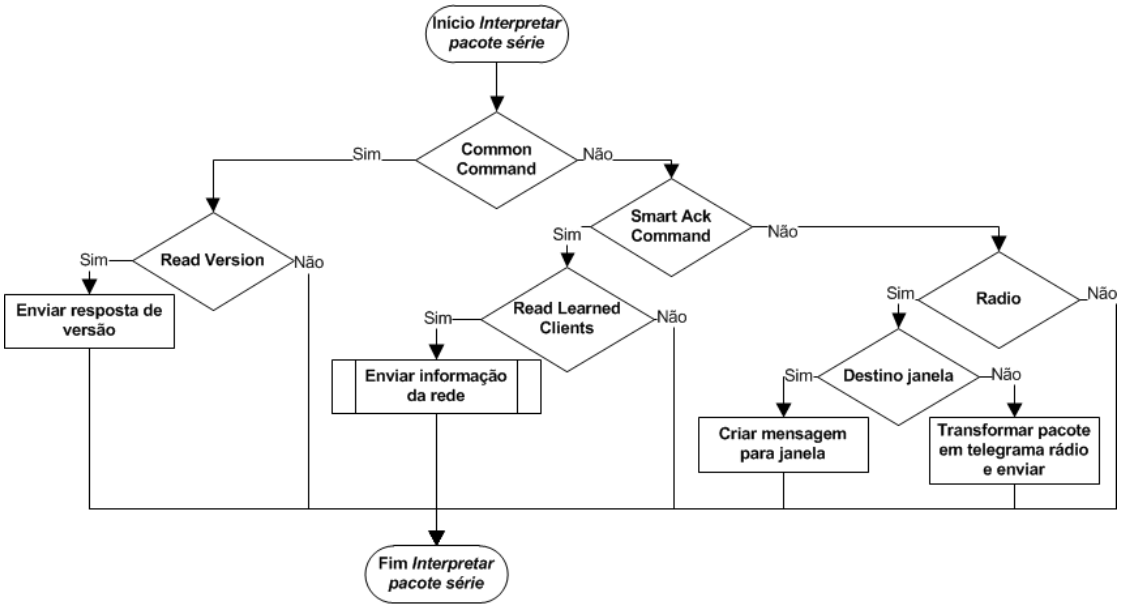


Figura 4.7: Fluxograma de interpretação de um pacote série recebido

Este novo fluxograma representado na figura **4.7** toma o lugar da comparação e a consequente ação feita no fluxograma referido anteriormente, em que verifica se o pacote série tem como destino a janela (figura **4.4**). O primeiro campo verificado é o tipo do pacote série. Se for rádio e o destino for uma janela, é criada uma mensagem para a mesma. No entanto, se o destino não for nenhuma das suas janelas, o destino pode ser a janela de um outro ZC. Neste momento o ZC mestre está a desempenhar o papel de encaminhador de telegramas, e se isto acontecer, o ZC mestre transforma o pacote série num telegrama rádio, através de uma função fornecida pela API EO3000I, e envia o telegrama para o ar. E então desta forma que através de um comando proveniente da *cloud* que os ZCs escravos recebem telegramas.

Além do tipo rádio são verificados mais dois tipos de pacotes série: o "Common Command" e o "Smart Ack Command". O último é assunto para a próxima secção deste mesmo capítulo, e a principal implementação de *software* neste novo ZC. Em ambos é verificado o primeiro dado do pacote série que identifica o sub-comando do tipo de comando. O ZC responde apenas a um sub-comando do tipo "Command Command" que é precisamente o pedido de versão. Caso o sub-comando tenha outro valor, não é enviada qualquer resposta.

A resposta ao pedido de versão consiste em enviar informação referente ao

aparelho *Enocean* conectado e também à versão *software* que este apresenta. São também enviadas as versões da API *Enocean* por questões de compatibilidade e também o identificador único embutido no chip do microcontrolador associado ao dispositivo.

Para o fazer, é necessário criar uma descrição da aplicação e da versão na memória configurável da *flash* do microcontrolador. Através de um vetor de caracteres configuráveis e com algumas restrições o utilizador pode descrever de maneira intuitiva o nome da aplicação e também a versão da mesma.

Listagem 4.1: Estrutura de versão *Climawin* ZC

```
1 code uint8 VERSION_APP[] =
2 {0xE0,
3 'V','E','R','S','I','O','N',
4 2,0,0,0,
5 'A','C','L','I','M','A','W','I','N',' ','Z','C',
6 0x00,0xE0};
```

Segundo a API, é necessário criar um "array" com a estrutura representada na listagem **4.1** para guardar a informação de maneira correta na memória de maneira à mesma ficar disponível para posterior avaliação. Depois de um inteiro pré-definido com o valor de 0xE0 é necessário introduzir os caracteres "Version" com a ordem correta. De seguida vem a versão da aplicação, com dois inteiros a definir a versão 2.0 do *Climawin* Zone Controller. Os dois restantes inteiros tem o propósito de definir a versão com mais detalhe, pelo que no Zone Controller não se encontrou essa necessidade e foram colocados valores nulos. De seguida introduz-se o caráter 'A' seguido da descrição da aplicação, que no caso é "CLIMAWIN ZC". O inteiro 0x00 é o caráter terminador que precede novamente o 0xE0, terminado assim o vetor.

Este "array" de caracteres ASCII serve apenas para guardar a informação na memória *flash*. Para enviar a resposta ao controlador que está do outro lado da ligação série é necessário aceder a estes dados. Através de uma estrutura fornecida pela biblioteca EO3000I e inicialização de uma instância da respetiva estrutura os valores são facilmente obtidos por campos separados em que, além dos dados do "array" previamente inicializado, também se pode obter a versão da API. Posto isto, é necessário criar um pacote ESP do tipo resposta para enviar os dados ao controlador, neste caso, o *gateway* ligado ao ZC. Após a criação do pacote e o devido preenchimento dos dados do mesmo, de acordo com a estrutura referida na

documentação ESP [21] do pacote de resposta a um "Co_Rd_Version", os dados são enviados via série através de uma função que se encarrega de enviar pacotes via UART com protocolo RS232 requerendo apenas o pacote previamente criado.

4.3.5 Comunicação entre diferentes ZCs

A comunicação entre diferentes controladores da rede *Climawin*, o mestre e o escravo, tem como único propósito obter a tabela de janelas que o escravo contém como aprendidas. Esta comunicação tem de ser do tipo pedido-resposta de maneira a ser assíncrona e que esteja disponível sempre que necessário. A figura **3.5** demonstra os vários Zone Controllers numa rede. Um conectado ao *gateway* e os restantes isolados, apenas comunicando via rádio. Todos estes ZCs funcionam como encaminhadores de mensagens para as suas janelas e podem ter sensores de CO_2 acoplados, no entanto, o mestre, que está ligado ao *gateway*, é diferente de todos os outros. Este ZC vai ser o controlador remoto de todos eles, pelo que tem o papel de controlador no processo "Remote Management", que se avalia como uma comunicação sem fios do tipo pedido-resposta para controlar vários dispositivos *Enocean*, neste caso, os ZC escravos.

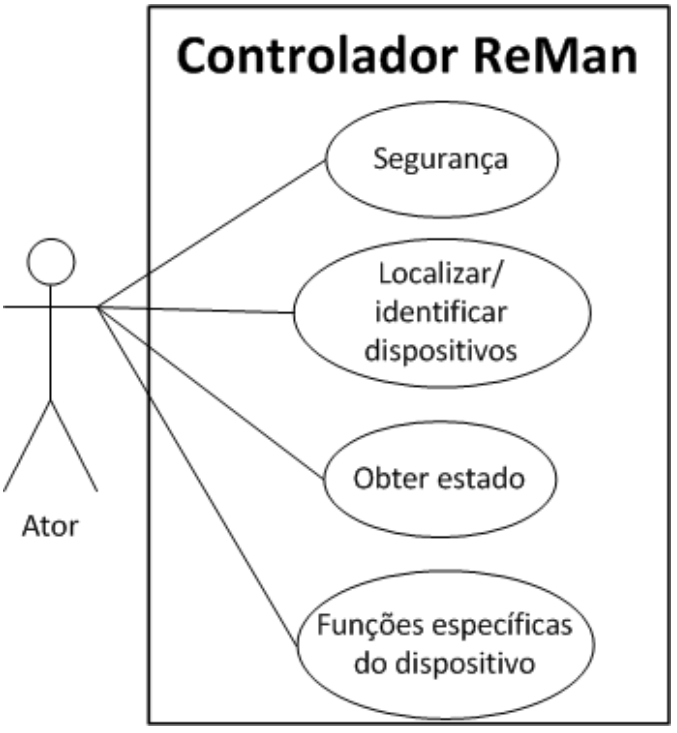


Figura 4.8: Casos de uso de controlador *ReMan*

Com o controlador *ReMan* podem-se fazer 4 diferentes ações. As operações de segurança visam bloquear e desbloquear o(s) dispositivo(s) quando o controlador assim o desejar, de forma a evitar que outros controladores possam aceder a informações da rede. Estes dispositivos estão sempre em um de dois estados: bloqueados ou desbloqueados. Só respondem a determinados pedidos caso estejam desbloqueados, e para o fazer, é necessário enviar códigos de desbloqueio por parte do controlador. No caso do *Climawin*, foram definidos códigos de bloqueio únicos, de forma a outras aplicações *Enocean* não conseguirem aceder à informação do *Climawin* nem a misturarem sistemas com propósitos completamente diferentes. Foi também desenvolvido um método de proteção de dados contra outras instalações *Climawin* vizinhas que possam interferir em sistemas próximos. As operações de localização e descoberta de dispositivos são a principal justificação de utilização do "Remote Management". Como já fora referido, é essencial, através da *cloud*, o utilizador ter acesso à informação e localização das janelas e a que zonas e ZCs estão associadas, de forma a criar um mapa da rede. Desta forma o utilizador pode identificar a fonte dos telegramas que recebe e interpretar os dados da maneira que entender. A natureza destas operações de descoberta é de difusão a todos os dispositivos, ou seja, "broadcast". Pode-se também obter o estado do dispositivo, isto é, se está ativo, qual o último pedido que foi efetuado e também operações de *ping*, para verificar se este responde ou não. Por último, podem-se fazer também funções específicas do fabricador ou da aplicação para enviar respostas únicas que diferem de outras aplicações.

Tabela 4.2: Funções do controlador *ReMan* utilizadas no *Climawin*

Funções *ReMan* no *Climawin*	
UNLOCK	(0x01) Desbloquear dispositivo
LOCK	(0x02) Bloquear dispositivo
SET_CODE	(0x03) Enviar código de bloqueio/desbloqueio para o dispositivo
QUERY_ID	(0x04) Pedir identificação e perfil ao dispositivo
PING	(0x06) Verificar se o dispositivo está ligado
WINDOW_SWEEP	(0x0A) Função específica para o *Climawin*

As funções são enumerados que especificam o tipo de pedido ou resposta que foi recebido. A grande maioria das funções têm todas resposta, no entanto existem exceções. Os enumerados das respostas são normalmente formados por uma soma

entre a função de um pedido mais um *offset* constante, maneira de simplificar as funções e o próprio *software* que as utiliza.

A tabela presente (**4.2**) demonstra todas as funções utilizadas pelo controlador *ReMan*, o ZC mestre, para fazer diferentes operações de segurança, descobrimento, identificação e obtenção da tabela de identificador de janelas que os dispositivos, os ZC escravos, têm aprendidas. A função é um campo de um tipo de telegrama especial utilizado apenas em comunicações *ReMan*: SYS_EX [20]. Após a função do telegrama, aparecem os campos com os dados. No caso do "Unlock", "Lock", e "Set_Code" estes dados são os códigos de desbloqueio, bloqueio e o novo código que o dispositivo deve gravar, respetivamente. Estas primeiras três operações não requerem qualquer resposta. São como ordens que não requerem qualquer satisfação por parte do dispositivo controlado. São funções de segurança, pelo que o controlador que as utiliza tem de ter a certeza que as utiliza sequencialmente de maneira correta, porque em caso contrário, pode bloquear o dispositivo. As duas seguintes funções, "Query_id" e "ping" já exigem resposta por parte do dispositivo. A primeira é essencial e serve de descoberta dos dispositivos após estes estarem desbloqueados. O pedido "Query_id" pode ser usado como "broadcast" e a resposta por parte do dispositivo é faseada temporalmente, de forma às diferentes respostas não colidirem e serem interpretadas corretamente por parte do controlador. Por outro lado, o "ping" é incondicional, isto é, o dispositivo responde quer esteja bloqueado ou não e serve apenas para verificar se este está ativo pelo que a resposta é um simples sinal para o indicar.

Quando um controlador desbloqueia, ou tenta desbloquear, um dispositivo com um código errado, este bloqueia-se automaticamente durante 30s, independentemente do tipo de funções ReMan que receba, mesmo sendo uma função "Unlock" com o código correto. Por outro lado, quando um dispositivo recebe um telegrama com a função "Lock" com o código errado nada acontece. Se um dispositivo se encontrar bloqueado não vai obedecer nem responder a quaisquer pedidos do controlador, a não ser o "ping". Esta característica permite que diferentes controladores comuniquem com os seus dispositivos de forma segura e sem interferências externas.

Listagem 4.2: Configuração do módulo *ReMan*

```
1 //! EEP Profile definition
2 #define EEP_ORG 0xA5
3 #define EEP_FUNC 0x09
4 #define EEP_TYPE 0x08
```

```
5  #define DEFAULT_MANUFACTURER_ID 0x7FF
6
7  //! REMAN Buffer allocation
8  #define RM_BUFF_SIZE 75
9  #define RM_CODE_ADDR 0x7E00
10
11 extern uint8 xdata u8gRmDataBuffer[RM_BUFF_SIZE];
12
13 extern uint16 code reman_param[];
14
15 //! User defines
16 #define CLIMAWIN_MAN_ID 0x00C9
```

Esta porção de código demonstrada na listagem **4.2** é a inicialização do módulo *ReMan* configurada pelo programa "Dolphin Studio" segundo os parâmetros introduzidos. Pode-se observar as constantes do perfil do dispositivo e também o código do fabricador. Este último é um valor pré-definido para todas as aplicações. De seguida, o tamanho do *buffer* de dados utilizado nos telegramas SYS_EX para a comunicação *ReMan*. 75 foi o tamanho escolhido devido à função personalizada "Window sweep", que será detalhada adiante. A constante seguinte define a posição de memória de código *flash* em que o código *ReMan*, utilizado para bloquear e desbloquear o dispositivo, vai ser guardado. Este valor é definido por defeito. A variável do *buffer* e dados vem a seguir, definida como externa. Isto porque aquando da receção de dados, o *buffer* é alterado, pelo que a API tem a necessidade de alterá-lo para providenciar os dados ao utilizador. A variável seguinte é um vetor que tem como conteúdo as funções compatíveis com o dispositivo, para este responder aos pedidos do controlador, logo é apenas utilizada pelo ZC escravo. O ZC mestre não necessita de configurar as funções compatíveis pois este não responde. Esta variável é inicializada no ficheiro ".c" correspondente, também configurado pelo "Dolphin Studio". Finalmente, as constantes definidas pelo programador para a aplicação em si. Nos processos *ReMan* deve ser feita a verificação do identificador do fabricante do dispositivo e para o *Climawin* foi escolhido o número 0xC9.

Descobrimento de ZC

Antes de obter informações das janelas para enviar para o *gateway*, o ZC mestre necessita primeiro de guardar o identificador dos ZCs escravos para comunicar com eles, e aí sim, pedir-lhes o identificador das janelas. Este procedimento

é garantido através de um *scan* à rede por intermédio de um telegrama ReMan com a função "Query_id" com difusão "broadcast".

Para fazer este "broadcast" e obter resposta por parte dos ZCs escravos, o ZC mestre necessita que estes estejam desbloqueados, senão não lhe respondem. O código de bloqueio/desbloqueio é um inteiro de 4 bytes de dados. Por definição este código é nulo, ou seja, zero. Para evitar que outras instalações *Climawin* interfiram na instalação em causa, é necessário utilizar um código personalizado. Foram utilizados três códigos diferentes, um para comunicar unidirecionalmente com o dispositivo, outro para evitar que outros sistemas desbloqueiem e acedam aos ZC escravos, e um outro código para iniciar a sequência de descobrimento aquando de um *input* do utilizador.

Listagem 4.3: Códigos de bloqueio/desbloqueio *ReMan* utilizados no *Climawin*

```
1 char const reman_code_0[4] = {'J','C','2','7'};
2 char const reman_code_1[4] = {'H','G','8','8'};
3 char const reman_code_2[4] = {'N','C','2','1'};
```

Os códigos consistem num vetor de inteiros com 256 possibilidades de valores. Logo, a probabilidade de uma outra instalação *Enocean* conseguir desbloquear um ZC escravo num determinado momento é de $1/255^4$, pelo que tem de acertar com exatidão o código do dispositivo. Por ventura, a *Enocean Alliance* poderia também definir perfis para este tipo de comunicação, de forma a uniformizar e haver mais transparência no que diz respeito a estes códigos para, de uma forma mais exata e controlada, evitar que diferentes instalações *Enocean* interfiram umas com as outras.

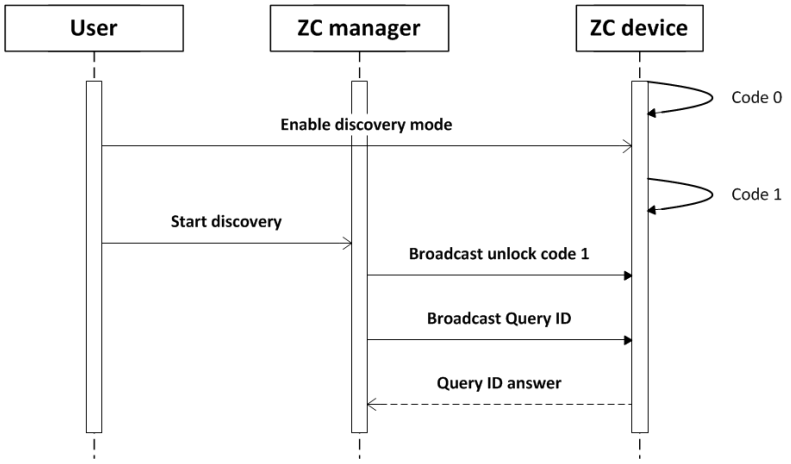


Figura 4.9: Diagrama de sequência do descobrimento de ZC

No *Climawin*, ambos os tipos de ZCs conhecem os três diferentes códigos. Estes códigos, bem como o código de bloqueio/desbloqueio que está no momento associado aos processos *ReMan*, são guardados na memória *flash*.

A figura **4.9** representa um ZC mestre e apenas um ZC escravo. Não obstante, neste diagrama poderiam estar representados mais ZC escravos, no entanto o processo seria repetitivo. O utilizador teria de habilitar um número de vezes igual ao número de escravos presentes na sua instalação, isto é, o utilizador deveria executar o "Enable discovery mode" n vezes para n escravos. O processo de configuração exige que a habilitação seja feita antes da iniciação.

Neste modo de descobrimento, o ZC mestre envia apenas telegramas "broadcast", pois está num modo de aprendizagem de dispositivos e desconhece toda a rede de Zone Controllers *Climawin* presente na instalação.

O ZC escravo ao iniciar o módulo *ReMan* verifica qual o código presente, se este código for diferente do código 2, este grava o código 0, que é o caso deste diagrama. O código 2 é o código enviado pelo ZC mestre aos ZC escravos quando estes estão aprendidos, de seguida é explicado o motivo.

O utilizador que deverá ser o instalador do sistema, ou então um cliente que tenha conhecimento do procedimento que deverá seguir, tem um papel preponderante neste reconhecimento. A habilitação dos ZCs escravos ("Enable discovery mode") é a camada de isolamento perante outras instalações *Climawin* vizinhas que poderão estar instaladas nas proximidades. O código de descobrimento é o código número 1, isto porque o "unlock" que o precede tem o primeiro código. Outros dispositivos que não tenham este código configurado no seu módulo *ReMan* não irão responder ao "query", que envia informações relevantes a si próprios. Desta forma foi conseguida uma maneira de definir uma janela temporal para os dispositivos *ReMan*, neste caso, os ZCs escravos, responderem ao "query" do controlador. Através de uma nova entrada no menu do Zone Controller, o utilizador pode habilitar a descoberta do dito dispositivo, e desta forma, permitir que este seja adicionado à rede. Este menu é representado através de o número '8' piscando no *display*. O menu pode ser acedido através da navegação do botão roxo (figura **4.2**) e a habilitação de descoberta iniciada pelo botão vermelho. Quando o ZC escravo tiver sido descoberto e concluído o processo este vai apagar o "display", disponibilizando desta forma, *feedback* ao utilizador. Voltando ao momento em que o utilizador habilita o ZC escravo, este último grava internamente o código 1 de forma a ser desbloqueado e descoberto pelo ZC mestre quando inicia o

procedimento de descoberta.

Quer isto dizer que, no caso do ZC mestre, é necessário iniciar o procedimento de descoberta, para enviar os telegramas "broadcast" e obter todas as respostas dos "querys". Para iniciar este processo, o utilizador deve fazer exatamente igual ao ZC escravo: navegar com o botão roxo, ir à entrada do menu respetiva, e pressionar o botão vermelho. A diferença é que neste caso dá inicio ao procedimento, em vez de o habilitar. Para diferenciar o mestre do escravo, a o display que representa a entrada do menu "ReMan" tem uma ligeira diferença nos dispositivos: no caso do mestre, além de piscar o número '8', o ponto também o acompanha.

Pode-se induzir através da ausência de um "lock" neste diagrama de que o processo não está ainda concluído. A cada resposta do "query" que o ZC mestre recebe, este guarda num *buffer* temporário o identificador único do ZC escravo. Como as respostas são desfasadas num tempo, tempo este definido por um valor aleatório entre 0 e 2s, o ZC mestre tem capacidade para interpretar os telegramas de resposta que recebe e acumular estes identificadores. Como já fora referido anteriormente, este processo é feito a um ou mais dispositivos, pelo que a sequência que vai ser representada de seguida é apenas dedicada a uma comunicação "unicast", isto é, os telegramas enviados pelo ZC mestre têm destino de apenas um ZC escravo.

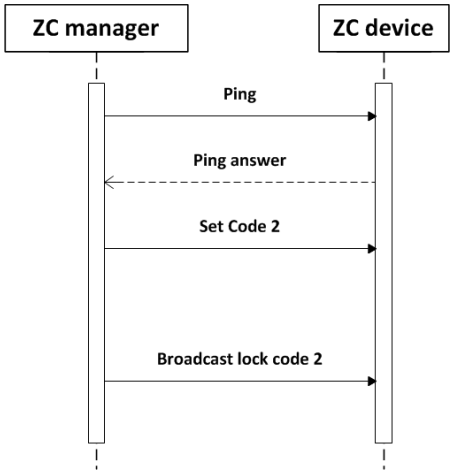


Figura 4.10: Diagrama de sequência da continuação do descobrimento de ZC

Após o identificador de todos os ZCs escravos que responderam estar guardado no *buffer* temporário, é altura de comunicar um a um para fazer verificação de consistência. Para o efeito é utilizada a função *ReMan* "ping". São efetuadas três tentativas para cada ZC escravo, se este eventualmente não responder o iden-

tificador é eliminado e o ZC mestre passa ao próximo dispositivo. Se, por outro lado, responder, é-lhe enviado o código 2, que será mais tarde utilizado para a função personalizada "Window sweep". Este segundo código é uma outra camada de segurança adicionada ao sistema. Este código ficará guardado permanentemente no ZC escravo a não ser que o utilizador o altere através da habilitação de descoberta. O propósito deste código 2 é finalizar o processo de descobrimento deste ZC escravo e prepará-lo para a função "Window sweep" que é o único processo utilizado pelo ZC mestre fora o descobrimento. Após este novo código estar guardado no módulo *ReMan* do escravo, este será dado pelo ZC mestre como aprendido e adicionado à rede que este poderá mais tarde aceder e obter dados. Quando o ZC mestre necessitar de utilizar a função "Window sweep" para obter os dados das janelas dos ZCs escravos é efetuada uma comunicação unidirecional dedicada para cada ZC escravo. Desta forma outros ZCs mestres de outras instalações *Climawin* vizinhas não conseguirão aceder a ZCs escravos que não lhes dizem respeito porque *à priori* o ZC mestre não tem o identificador dos escravos, apesar de o código de desbloqueio para o "Window sweep" ser igual e independente da instalação.

No final de todo o processo estar concluído, isto é, após serem enviados os novos códigos a todos os ZCs escravos, um a um, o ZC mestre grava os identificadores dos ZCs escravos na memória *flash*, e envia um "broadcast lock" para todos fecharem comunicações. Como já referido, apenas o controlador da mesma instalação, neste caso, o ZC mestre, poderá no futuro aceder a estes dispositivos. Podem existir até um máximo de cinco escravos Zone Controller. Deve-se esta limitação ao facto da necessidade de armazenar a identificação dos escravos na memória de código (memória FLASH) do ZC mestre e do facto de a memória estar quase toda esgotada. Para uma eventual futura implementação de um outro serviço, seria complicado devido ao reduzido espaço.

Para implementar todo este controlo de dispositivos e as respostas dos mesmos, foi implementada uma máquina de estados no *software* do ZC mestre. A função que chama o "handler" do estado é utilizada no final da sequência iterativa principal do código "main" (figura **4.4**). Grande parte do tempo está no estado "Idle", mas este estado pode ser alterado através da iniciação do processo de descobrimento, ou então, do processo de obtenção da tabela das janelas de todos os ZCs. Para interpretar os telegramas recebidos pelo ZC mestre foi adicionado um filtro aos telegramas recebidos via rádio (figura **4.5**). Após a chamada de uma função da biblioteca "EO3000I" que indica se o telegrama recebido é *ReMan*, os telegramas são interpretados tendo em conta o estado atual da máquina e o estado

da mesma alterado.

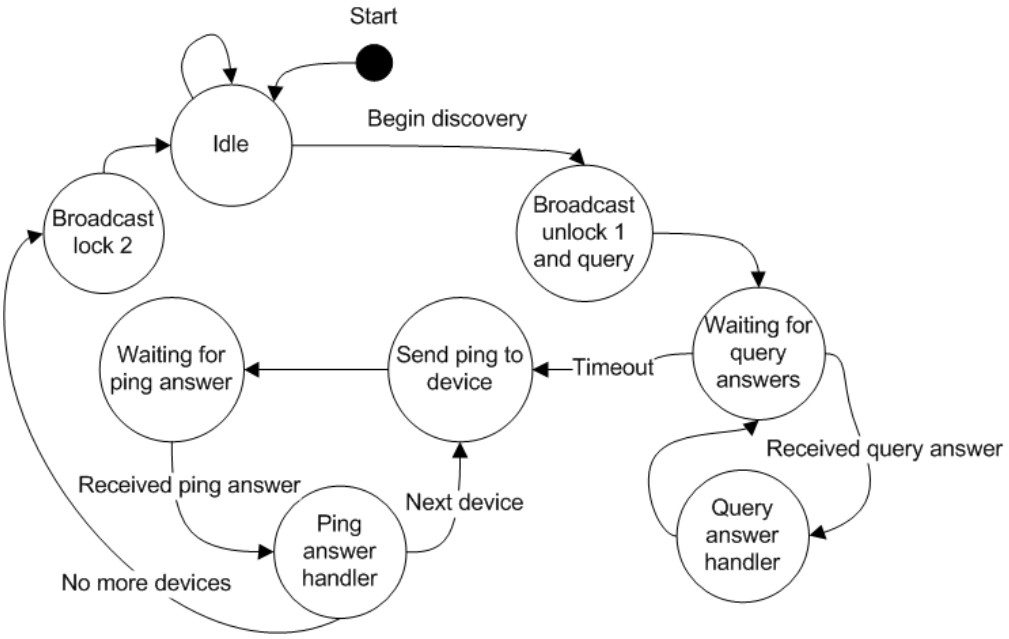


Figura 4.11: Máquina de estados do descobrimento do ZC mestre

Na presente figura **4.11** está representada, de uma maneira simplista, a máquina de estados implementada no *software* do ZC mestre para a gestão de respostas dos dispositivos após o início do descobrimento.

Como já fora referido, em primeiro lugar é enviado um "unlock" e um "broadcast query". De seguida o controlador fica 4 segundos à espera de respostas. Durante este tempo este guarda os identificadores dos dispositivos que lhe respondem. Após 4 segundos o controlador, ZC mestre, começa a enviar "pings" e a esperar pelas respostas. O controlador faz 3 tentativas de "ping", o que não está representado nesta máquina de estados. Não obstante, está presente o estado em que é feito o "handling" da resposta do "ping" em que o controlador guarda permanentemente na memória *flash* o identificador do ZC escravo e também lhe envia um novo código. Quando não há mais dispositivos o ZC mestre envia um "broadcast lock" e o processo fica concluído.

No que diz respeito ao *software* do ZC escravo, relativamente a este processo de descobrimento *ReMan*, sofreu menos alterações do que o seu controlador. As repostas às funções *ReMan* enviadas pelo controlador são efetuadas automaticamente pela biblioteca, isto no que diz respeito a funções não personalizadas, como o caso do "query" e do "ping". Quando o dispositivo recebe um telegrama *ReMan* e o utilizador faz a devida identificação do mesmo através da chamada de uma

função da API, a resposta é construída por defeito. É necessário apenas enviar o telegrama de resposta através de uma função chamada recursivamente que não recebe qualquer parâmetro. Para enviar telegramas com funções personalizadas esta função também é utilizada, no entanto é necessário alterar o *buffer* de dados, que é uma variável externa definida e utilizada pela API, e também usar uma outra função da biblioteca para definir a função *ReMan* a ser enviada como resposta. No caso do *Climawin*, mais precisamente, no ZC escravo, a função personalizada será "Window sweep answer".

Foi também efetuada uma gestão de códigos *ReMan* no ZC escravo. Como podemos observar na figura **4.9** o ZC escravo guarda códigos internamente sem receber qualquer comando "set code", daí ambos os ZCs terem acesso a todos os códigos. Este controlo é necessário para a instalação ser à prova de interferências externas, como já foi explicado.

Pedido de mapa de rede

Segundo a figura **4.7** o ZC mestre responde também a um outro pacote série do tipo "Smart-Ack Command" e comando "Read Learned clients". Os pedidos da *cloud* refletem-se no ZC mestre sobre a forma de um pacote série, e este não é exceção à regra. Quando a aplicação que está utilizar o serviço *cloud* do *Climawin* tem a necessidade de obter o identificador de todas as janelas de todos os ZCs da instalação e as suas respetivas zonas, para fazer um mapa com mais detalhe, recorre a este pedido. Doravante, o ZC mestre tem a responsabilidade de enviar todas as janelas via RS232 num ou mais pacotes de resposta ESP. Foi escolhido o tipo "Smart-Ack Command" para o pacote série porque a relação janela-ZC está regida segundo o "Smart Acknowledge" e a janela é um dispositivo aprendido pelo ZC. O "Read Learned clients" fala por si mesmo, em que o "postmaster" que recebe este pedido deve enviar uma resposta com informação relativa aos dispositivos que este tem aprendidos. Desta forma este comando despoleta o início do processo. A resposta a este pedido tem no máximo 15 janelas, que é o máximo de "devices" para um "postmaster", logo a estrutura de dados do pacote de resposta está orientada com este propósito. No entanto, no caso do *Climawin* a estrutura dos dados de resposta foi personalizada e será detalhada adiante.

Após o ZC mestre ter feito o descobrimento tem acesso "Remote Management" direto aos ZCs escravos pois tem o código de desbloqueio dos mesmos e também o identificador único destes. Desta maneira o ZC mestre consegue comunicar com um de cada vez e pedir-lhes informações relativas às suas tabelas de

janelas aprendidas.

Em primeiro lugar, o ZC mestre deve enviar o identificador das suas janelas, através de um pacote série do tipo "response" ou resposta (tabela **2.1**). A cada ZC existente, a aplicação controladora que corre do outro lado da ligação série, vai receber um pacote ESP do tipo resposta com os dados estruturados da mesma maneira. Estes dados serão constituídos pelo identificador único do ZC, seguido dos identificadores das janelas e da zona à qual a mesma pertence. Para cada janela os dados repetem-se no mesmo pacote.

Listagem 4.4: Estrutura de dados para resposta ao pedido do mapa da rede

```
1 typedef struct
2 {
3  uint32 window_id;
4  uint32 Controller_id;
5  uint8 zone_id;
6 } window_data_to_send;
```

Foi escolhida esta estrutura um pouco redundante devido à simplicidade da mesma. Através desta redundância é mais fácil para a aplicação construir um mapa dos dispositivos. Concluindo, o tamanho de um pacote série contendo a informação das janelas de um ZC que tem "x" janelas é de $(4+4+1)*x$ em que os dois primeiros quatro correspondem ao tamanho de um identificador único de um dispositivo *Enocean*, neste caso o ZC e a janela, respetivamente. O *byte* seguinte diz respeito à zona da janela, que pode variar entre um e cinco. No caso do ZC mestre, preencher os dados do pacote série é relativamente simples: tem a informação na sua tabela de dispositivos, ou janelas, guardada na memória *flash*. De seguida é só copiar os dados para um vetor da referida estrutura (listagem **4.4**) e usar uma função personalizada para enviar os dados. Esta informação é a primeira que o ZC mestre envia. No entanto, se por ventura o ZC mestre não tiver qualquer janela aprendida, não há necessidade de enviar a resposta que diz respeito às suas janelas. Se assim for, o controlador *gateway* não recebe qualquer resposta referente a janelas do ZC mestre. Não obstante, de seguida tem a responsabilidade de criar novamente pacotes série de resposta estruturados da mesma forma, devido às janelas dos ZCs escravos. Para obter essa informação é utilizada a função "Window sweep" através do *ReMan*.

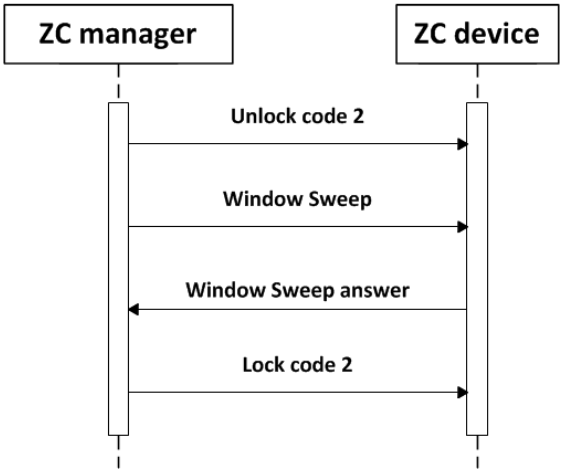


Figura 4.12: Diagrama de sequência da função "Window Sweep"

O processo é idêntico a todos os outros que utilizem o "ReMan". Comunicação bi-direcional entre o controlador e o controlado, em que o primeiro deve desbloquear o segundo antes de lhe fazer pedidos. O segundo deve ser sempre bloqueado após a troca de dados ser dada como concluída. O ZC mestre deve fazer este processo com todos os ZCs escravos que responderam ao descobrimento explicado anteriormente. E para cada resposta obtida, deve usar a mesma estrutura de dados (listagem **4.4**) para enviar o pacote série para a aplicação a ser executada no *gateway*. Para o fazer, o ZC mestre copia os dados do telegrama recebido através de um preenchimento de um "array" da estrutura **4.4** e constrói o pacote de maneira idêntica aquando do envio das suas próprias janelas.

Relativamente ao telegrama *ReMan* "Window Sweep answer" enviado pelo ZC escravo, o tamanho máximo é de 75 bytes. Isto porque um ZC pode ter no máximo 15 janelas, 3 para cada zona. $15 * 3 = 60$. Mais um byte dedicado a cada janela para indicar a zona e resulta em $60 + 15 = 75$. O tamanho do telegrama é portanto dinâmico. Quer isto dizer que depende do número de janelas que um ZC tem aprendido. Poder-se-ia então fixar o tamanho em 60 bytes, e enviar a tabela de aprendizagem toda. As entradas que estivessem preenchidas de maneira correta indicariam uma janela, e as entradas que não estivessem preenchidas teriam um valor por defeito. Desta forma a aplicação que recebesse os dados via série, após devida conversão pelo ZC mestre, saberia incrementar o índice dos *bytes* e deduzir as zonas de cada janela. No entanto dificilmente um ZC terá todas as janelas preenchidas e entendeu-se que esta seria a melhor solução para evitar um elevado tráfico de dados desnecessários.

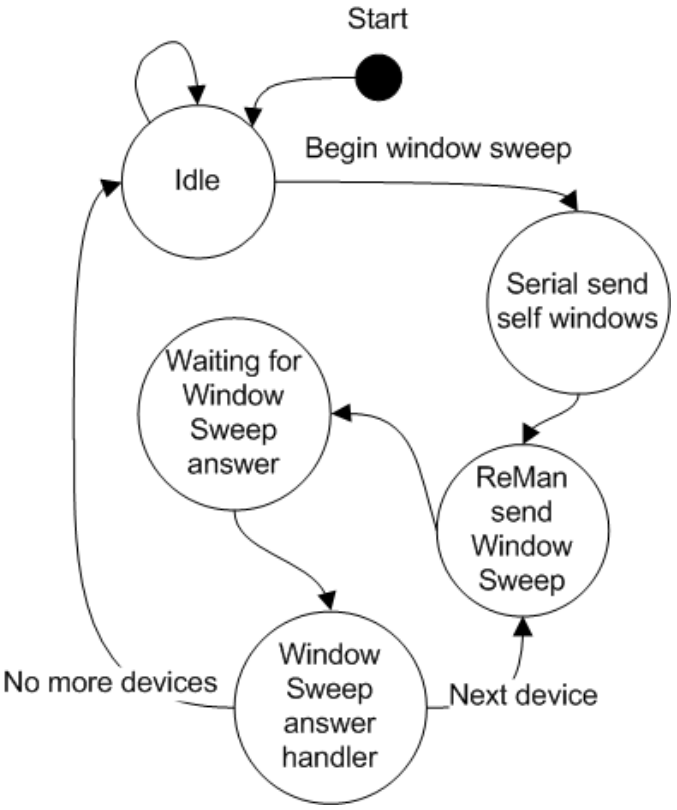


Figura 4.13: Máquina de estados do "Window Sweep"

Na presente figura **4.13** o estado "Idle" é o mesmo encontrado na máquina de estados da figura **4.11**. A iniciação de uma sequência ou de outra dependem apenas do utilizador, quando faz o descobrimento, ou então do pedido da *cloud*, quando faz o "window sweep". A primeira ação é enviar as próprias janelas para o *gateway*, caso exista alguma. De seguida envia o "Window Sweep" para cada ZC escravo e espera pela resposta. Neste "handler" da resposta do escravo, o ZC mestre constrói e envia o pacote ESP de resposta contendo a informação das janelas e também envia o código de bloqueio para o ZC escravo. O processo finaliza quando a comunicação com o último dispositivo, um ZC escravo, acaba.

Capítulo 5

Atuador da janela

Neste capítulo serão abordados os temas associados à atuação da janela *Climawin*. A versão anterior será analisada e uma nova solução será também estudada e implementada. O detalhe da comunicação entre o módulo de controlo e o módulo de atuação será analisado e detalhado, tendo em conta as novas funcionalidades da janela, entre as quais, a sua capacidade de receber novos telegramas provenientes do ZC. Finalmente, a implementação do software e todo o suporte para a mesma será detalhado e explicado.

5.1 Análise da versão anterior

A anterior janela consistia no microcontrolador STM 300 da *Enocean* a interpretar telegramas, o teclado da janela, em que o utilizador pode mover a persiana, e também a atuar no motor. Estas funções revelam-se muito pesadas para um microcontrolador que idealmente apenas interpreta e envia telegramas. A adição de elevado processamento adiciona um *overhead* muito custoso para a janela como um todo, isto porque energéticamente e funcionalmente não é ideal. Como já fora referido no capítulo **3.1.1** a conjugação de todos os pequenos problemas que a janela tinha resultavam num controlo inconsistente tanto das persianas, como nas válvulas e o microcontrolador estava demasiado tempo acordado a controlar as mesmas, cenário que não é o mais ideal.

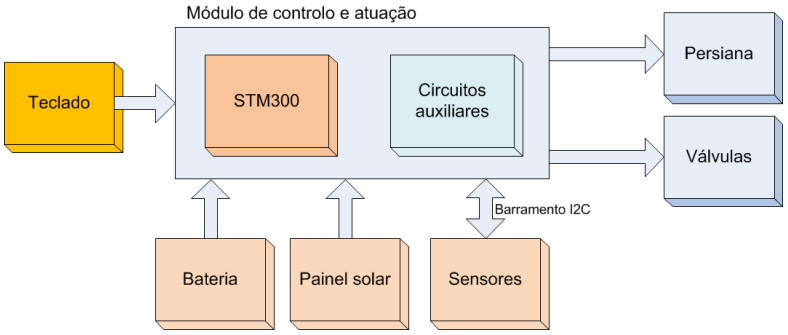


Figura 5.1: Diagrama de blocos da versão anterior da janela

Todos estes elementos apresentados na figura **5.1** foram já detalhados e as suas funções analisadas. De destacar apenas o papel central que o STM300 tem na janela, tanto como controlador, que contém os algoritmos e a interpretação dos telegramas bem como o *input* do utilizador através do teclado, mas também a atuação, para mover a persiana e as válvulas. Os circuitos auxiliares fazem parte da mesma placa em que o STM300 está contido. Entre outros tantos destacam-se os reguladores "step-down" e "step-up", para alimentar o microcontrolador e o circuito de atuação nas persianas e válvulas, que consistiam em pontes "H" para comutar o sentido dos motores. O teclado também está embutido na mesma placa, no entanto não faz parte do controlo nem da atuação, pois é apenas um dispositivo de entrada de comandos por parte do utilizador. A bateria e o painel solar também eram auxiliados através de circuitos de regulação e carregamento, respetivamente, para fornecer energia ao sistema de maneira sustentável. Os sensores ligavam ao módulo principal através de dois fios para comunicar e outros dois para a alimentação e a referência.

Em modo de síntese, os problemas com o atuador anterior:

- Controlo da persiana em malha aberta: não havia *feedback* para a posição da mesma;
- Atuação nas válvulas era inconsistente e indesejável (capítulo 3.1.1);
- O único microcontrolador da janela (STM300 da *Enocean*) estava sobrecarregado.

5.2 Especificação do atuador

Os problemas anteriormente ditados requerem uma solução eficaz em que consiste em separar os módulos de controlo e atuação em duas diferentes partes, independentes uma do outro em que ambas tenham poder de processamento.

A responsabilidade do controlo deverá caber ao mesmo microcontrolador STM300, pois este é essencial ao sistema para interligar a janela a uma rede *Enocean* e é também programável para desenvolver o *software* necessário para o seu propósito.

O novo módulo do atuador terá componentes essenciais para comportar as novas funcionalidades da nova janela *Climawin*. Novos comandos provenientes da *cloud* e configurações no teclado necessitam de um atuador consistente e eficaz. Desta forma foi adicionado um novo microcontrolador à janela: *ATTiny* 1634 da Atmel.

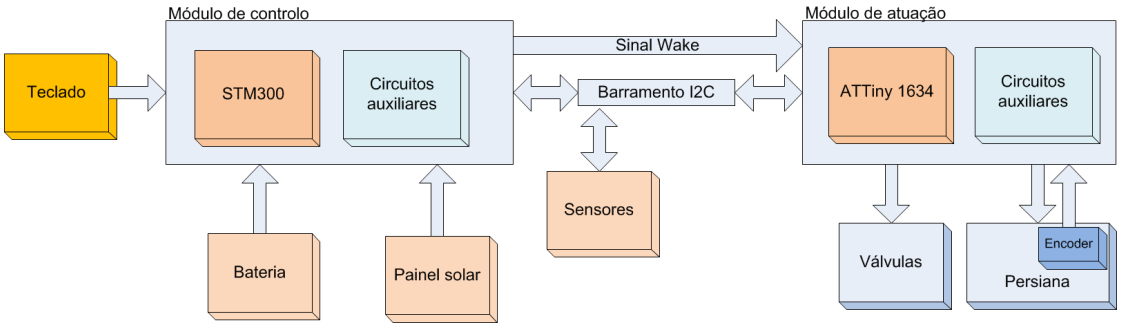


Figura 5.2: Diagrama de blocos da atual janela *Climawin*

A figura **5.2** representa a atual janela *Climawin*. O módulo de controlo que contém o STM300 mantém todas as ligações que possuía anteriormente, à exceção dos motores de atuação. Toda a interpretação e envio de telegramas ou obtenção de valores dos sensores continua a ser efetuado neste módulo, com um aproveitamento do trabalho já desenvolvido somado a umas alterações no *software*. Estas alterações, a cargo de um outro investigador envolvido no projeto *Climawin*, resumem-se na remoção na atuação, implementação da interpretação de novos comandos provenientes do ZC, indiretamente da *cloud* e outras melhorias no desempenho das variadas funções. Estas alterações efetuadas neste microcontrolador ditam requisitos no novo atuador, sobre a forma de comandos, configurações e possibilidades de controlo. Posto isto, aquando das necessidades da janela, foi efetuada uma ligação das linhas de comunicação partilhadas pelo microcontrolador e pelos sensores para

o módulo de atuação. Esta linha consiste em dois fios regidos segundo o protocolo I^2C. Assim sendo, aproveitar-se-ia uma comunicação já implementada com sucesso para comunicar também com o ATTiny da Atmel. A nível de *hardware* mantiveram-se nesta placa os reguladores para tensões nominais de alimentação dos diferentes componentes de ambas os módulos e removeram-se os circuitos de atuação, estando agora naturalmente do lado da atuação. Foi também adicionado um sinal chamado de "wake", que consiste num único fio que tem como única funcionalidade acordar o ATTiny do modo "power down" e indicar que um novo comando será recebido via I^2C.

A escolha do ATTiny 1634 da Atmel [37] deve-se à sua simplicidade e características *low-power* e às ferramentas fornecidas pela companhia para desenvolver com suporte e facilidade variadíssimas aplicações para os seus produtos.

Este microcontrolador de registos de 8 bits têm 16KBytes de memória de código *flash* programável, o que permite o desenvolvimento de aplicações com algum grau de complexidade. 256 bytes de memória EEPROM para recorrer em momentos de falta de energia, recurso auxiliar bastante útil em aplicações com bateria. Dois temporizadores e um módulo TWI escravo com interrupções programáveis para implementar comunicações protocolares de dois fios, o que é o caso do I^2C, ideal para a comunicação entre o STM300 e o ATTiny1634. Tem também interrupções externas programáveis para diferentes *triggers* ideal também para o sinal de o *encoder* que o motor fornece. No que diz respeito às características elétricas, já foi anteriormente referido que é um microcontrolador *low-power*, tem um oscilador interno de 8MHz, que quando habilitado e alimentado a 3.3v, tem consumos na ordem dos 0.1 μA em estado de "power down" em que todos os periféricos auxiliares estão desligados.

No que diz respeito ao *hardware* este módulo de atuação tem agora os circuitos "step-up" para alimentar as pontes "H" para atuar no motor de 24v da persiana e também nas solenoides bi-estáveis das válvulas bem como o circuito do acondicionamento do sinal do *encoder*. Para movimentar as folhas da persiana é necessário mover a persiana em si, alterando a posição em que a mesma se encontra no momento. Não obstante, a posição das folhas pode ser controlada partindo do princípio que as mesmas, quando estão na posição inicial, se encontram totalmente fechadas. Desta forma, e com o *feedback* fornecido pelo *encoder*, é possível controlar a abertura das folhas horizontais e consequentemente a luminosidade do interior.

Posto isto, o novo atuador deverá implementar as seguintes funções:

- Novos movimentos na persiana como suporte às novas funcionalidades da janela;
- Controlo da posição da persiana bem como das folhas horizontais;
- Comutação do estado das válvulas bi-estáveis;
- Resposta a pedidos provenientes do módulo de controlo;
- Guardar informações relevantes numa eventual falha de energia(tamanho da janela, posição da persiana e das folhas horizontais);

5.3 Comunicação entre controlador e atuador

Entre o módulo de controlo a atuação existe uma ligação com fios para os comandos do utilizador ou o resultado dos algoritmos de controlo se refletirem fisicamente na janela. Esta comunicação é uma consequência do *input* do utilizador ou então de um algoritmo interno, funcionando como uma extensão ao controlo da janela. Com a implementação de novas mensagens 4BS com a janela como destino, é agora possível definir um conjunto de diferentes comandos como definir a posição da persiana ou das suas folhas horizontais ou então definir um cenário manualmente à escolha do utilizador. A aplicação que corre na *cloud* pode enviar dados ao *gateway* para este criar uma mensagem estruturada segundo o perfil de uma janela, que de seguida é enviada para o ZC mestre, para a mover ou então mudar o cenário. Outros comandos são também possíveis, como por exemplo a alteração dos limites do algoritmo das válvulas, mas que não estão diretamente relacionados com uma atuação imediata. Todos estes comandos e possibilidades espelham-se no interface entre ambos os módulos. Deve ser uma comunicação rápida e eficaz, pois o módulo de atuação tem de processar instruções constantemente quando recebe um comando e ao mesmo tempo oode estar a mover a persiana. Para o utilizador convém também que a janela, como um todo, responda rapidamente aos seus comandos, de forma ao interface homem-máquina ser o mais eficaz possível.

No entanto, não só a *cloud* pode mover a janela, existem outras duas fontes para a mesma se mover: o teclado e o interruptor *Enocean*. O teclado pode mascarar todas as funções que o interruptor é capaz de fazer, como mover a persiana e as folhas da janela. Para o atuador, não existe qualquer diferenciação entre o

movimento induzido pelo teclado ou pelo interruptor porque a camada de comunicação entre o módulo de controlo e o atuador, que dá a iniciação ao processo do movimento, é utilizada pela janela após processar o carregamento de um botão do teclado ou receber uma mensagem do ZC aquando do carregamento de um interruptor. Portanto, nestes casos, o comando enviado pelo módulo de controlo para o módulo de atuação é o mesmo. Não obstante, o teclado tem mais funcionalidades que o interruptor. Através de uma combinação de teclas (4 no total) é possível configurar o tamanho da persiana, alterar o cenário das válvulas e também forçar um "update" dos diferentes dados da janela para serem enviados para a *cloud*. Alguns destes comandos do teclado exigem a colaboração do atuador, pelo que estas combinações influenciam necessitam de ser prolongadas até ao atuador.

Tabela 5.1: Comandos entre módulo de controlo e atuação

Comando	**Descrição**
BLIND_UP	Mover persiana para cima
BLIND_DOWN	Mover persiana para baixo
BLIND_POSITION_RECEIVE	Obter posição da persiana
BLIND_POSITION_SEND	Definir nova posição da persiana
BLIND_WORKING	Obter estado da persiana (movendo ou não)
BLIND_LENGTH	Definir tamanho da persiana
BLIND_MANUAL_CONFIG	Definir tamanho da persiana manualmente
BLIND_CONFIG_FINISH	Finalizar processo de configuração manual
BLIND_UP_CONFIG	Subir janela independentemente da posição atual
FLAP_UP	Mover folhas para cima
FLAP_DOWN	Mover folhas para baixo
FLAP_POSITION_RECEIVE	Obter posição das folhas
FLAP_POSITION_SEND	Definir nova posição das folhas
VALVE1_OPEN	Abrir aba interior
VALVE1_CLOSE	Fechar aba interior
VALVE2_OPEN	Abrir aba exterior
VALVE2_CLOSE	Fechar aba exterior
VALVES_STATE	Obter estado das válvulas

Posto isto, é necessário refletir estes novos comandos para a comunicação I^2C entre os dois módulos. O módulo de controlo contém o STM300 que tem o papel de mestre nesta comunicação. Outros dispositivos, como o ATTiny 1634 e

os sensores tem o papel de escravo. Através de um endereçamento definido pelo protocolo I^2C, é possível comunicar com um escravo isoladamente e , desta forma, evitar colapso na transmissão de dados.

É por intermédio destes comandos que o módulo de atuação inicia qualquer um dos seus processos. Este módulo é um escravo, quer isto dizer que não toma iniciativa para fazer qualquer ação, está sempre pronto e à espera de um pedido do módulo de controlo, e aí sim, atuar na persiana ou na válvula, ou então responder-lhe com alguma informação.

A comunicação foi definida segundo um enumerador para escolher o comando. O primeiro dado da comunicação I^2C define o comando. Este comando pode ser isolado, isto é, não é seguido por nenhum dado. Pode também ser seguido por um dado, em que a sua função é quantificar o comando, como por exemplo, definir a posição da persiana. Finalmente, o comando pode ser seguido de um dado mas no sentido contrário: enviado pelo módulo de atuação, tecnicamente falando, pelo ATTiny. Serve este último tipo de comando para reportar para o módulo de controlo dados relativos à atuação. Estes dados podem ser posteriormente usados pelo módulo de controlo para *feedback* do utilizador ou então para verificar se a persiana se está a mover para dar seguimento a outros processos, ou então esperar que esta pare.

Os comandos representados na tabela anterior **5.1** têm associados um número inteiro único para cada um e estão definidos através de um enumerador no *software*. A tabela encontra-se dividida em diferentes partes devido à natureza dos comandos. Os primeiros dizem respeito ao movimento da persiana e os segundos servem para configurar o tamanho da mesma. A terceira divisão representa comandos relacionados com as folhas das janelas e finalmente, a quarta e última divisão diz respeito às válvulas.

Os comandos "up" e "down" são os mais comuns e podem ter duas fontes de iniciação dos mesmos: o teclado ou o interruptor. Como já fora referido, para o atuador não há distinção. Este tem apenas de mover a persiana para cima ou para baixo, de acordo com o comando recebido. Os comandos cujo nome é sucedido de "Position", bem como o "Blind length" e o "Blind working" têm um segundo dado que quantifica o comando. No caso dos "Position" é enviada a posição desejada ou atual tendo em conta o comando recebido. Se for "Receive" o módulo de controlo deseja obter a posição atual da persiana ou das folhas, pelo que o atuador responde com essa informação. Se, por outro lado, for "Send", o atuador deverá reposicionar

o dito objeto na posição definida pelo dado sucedido. O comando "Blind length" precede também de um dado para definir o tamanho da persiana, para efeitos de configuração. Este dado é um inteiro que irá endereçar um vetor que define um conjunto de tamanhos constantes pré-definidos para a janela. Ou seja, este comando deverá ser utilizado apenas quando a janela e o seu tamanho já estão documentados e registados como janela *Climawin*. Neste momento existe apenas um tamanho pré-definido. Finalmente, o comando "Blind working" serve para reportar o módulo de controlo sobre o estado do motor da persiana: se este se está a mover ou não.

Os comandos "Blind Manual Config" e "Blind config finish" servem para colmatar a lacuna deixada pelo comando "Blind length" quando a janela tiver um tamanho não definido ou documentado. Aquando do pressionamento de uma combinação de botões no teclado, é enviado um "Blind Manual Config" para o atuador, e este inicia o processo de configuração manual. A persiana começa a mover-se e este guarda o tamanho da janela quando receber um "Blind config finish". O utilizador que fizer este procedimento deverá ter cautela devido à estrutura da persiana que não deverá dar a volta e voltar a subir. De qualquer maneira, um temporizador foi implementado para, ao fim de um minuto, caso o utilizador não finalize o processo, a configuração pare e a janela volte à posição inicial.

O ponto de calibração da janela é na posição zero quando esta tem a persiana totalmente subida. Tendo isto em conta e que também podem ocorrer erros, o comando "Blind up config" é um extra necessário para evitar problemas e calibrar novamente a janela. Este comando, iniciado sob ordem do utilizador quando este pressiona o botão de subir durante dois segundos, sobe a janela independentemente da sua posição. É possível saber que a mesma chegou ao ponto zero através do abrandamento do sinal do *encoder*, a melhor solução encontrada na ausência de um fim de curso.

Finalmente, os comandos relacionados com as válvulas. Estes comandos são simples e não necessitam de um segundo dado, à exceção do último, "Valves State". Este comando serviria para enviar a posição das válvulas caso o atuador tivesse forma de obter a posição das mesmas, mas como estas são bi-estáveis, e a sua atuação reside num "disparo" de milissegundos atuar no sentido desejado, não há forma de obter realmente o estado das mesmas, pelo que este comando é residual e poderá ser implementado para, por exemplo, enviar informação relativa à última atuação nas válvulas.

Quatro comandos no total para mover as válvulas porque cada aba fornece dois diferentes pontos de conexão, cada um para enviar um sinal e movê-las num determinado sentido pelo que esta comunicação é a mais simples e mais rápida, ao contrário da utilização de um comando com um segundo dado. Os sinais enviados às válvulas são pequenos pulsos de elevada energia que movem as mesmas. Enviar um pulso uma segunda vez no mesmo sentido não tem qualquer efeito. Com estes comandos, o módulo de controlo, após definir um cenário através do algoritmo de controlo, ou então receber um comando para tal, muda ambas as válvulas para a posição que desejar, com comandos independentes.

5.4 Implementação da atuação

Nesta secção concentram-se os principais métodos e implementações do software que corre no ATTiny 1634. Foram desenvolvidas várias bibliotecas fazendo o papel de HAL ("Hardware Abstraction Layer") para utilizar os diferentes periféricos do microcontrolador. Posto isto, serão demonstradas os diferentes periféricos e como foram utilizados na globalidade da aplicação em que o propósito final é controlar a persiana e as válvulas.

5.4.1 Ferramentas utilizadas

A Atmel disponibiliza o IDE "Atmel Studio" para desenvolver aplicações para os seus microcontroladores com suporte de "debug", configuração dos periféricos do micro como a frequência do oscilador interno ou a habilitação do temporizador "Watchdog". O ambiente de desenvolvimento de código é muito amigável e tem um interface fácil para o programador. Para este atuador foi desenvolvido código em linguagem C. A programação em microcontroladores da Atmel consiste, em primeiro lugar, configurar os registos dos diferentes periféricos para posteriormente utilizar os mesmos para desenvolver a aplicação e encontrar as métricas da mesma. O acesso ao registo bem como a documentação fornecida é muito completa e rápida de interpretar, facilitando e acelerando o processo de desenvolvimento do *software*.

Posto isto, a escolha do ambiente de desenvolvimento recaiu sobre o "Atmel Studio" pois é fornecido pela empresa que desenha a arquitetura dos seus microcontroladores e o suporte para desenvolvimento é o ideal.

5.4.2 Noções e restrições da implementação

Uma das grandes métricas da janela é o *low-power*. O STM300 tem métodos em que minimiza os consumos desabilitando os seus próprios periféricos do microcontrolador através do modo "sleep" bem como os componentes do módulo de controlo. O ATTiny 1634 tem também os seus métodos. Existem 4 modos de diminuição de consumo de energia. Diferem uns dos outros pela quantidade de energia que consomem, resultado da habilitação de diferentes periféricos do microcontrolador. No caso da janela *Climawin* é utilizado o modo que menos energia consome, chamado de "Power-Down". Neste modo, apenas a comutação do estado dos pinos, ou interrupção externa, o temporizador "Watchdog" e os módulos TWI, USART e USI ("Universal Serial Interface") podem tirar o microcontrolador deste estado.

Os módulos de "Watchdog", USART e USI não são utilizados. O primeiro porque a sua utilização traduziria-se em maiores consumos. Os outros dois porque são desnecessários para o módulo de atuação. Restam as interrupções externas e o TWI.

Tendo em conta que o atuador deve começar a agir quando recebe uma ordem do módulo de controlo, poder-se-ia concluir que o módulo TWI, responsável pela comunicação, iniciaria todo o processo e "acordaria" o microcontrolador. O módulo TWI tem uma opção programável para habilitar uma interrupção de "Address match". Quer isto dizer que quando o módulo detetasse um endereçamento da comunicação I^2C para si mesmo, este interromperia o microcontrolador. Acontece que, por razões aplicacionais, esta solução seria inviável. O ATTiny, enquanto está a atuar nas válvulas dificilmente recebe um comando por parte do mestre. Isto porque as válvulas necessitam apenas de um pequeno pulso temporal para serem movidas. No entanto, a persiana pode demorar uns razoáveis segundos a mover-se pelo que é bastante plausível que o ATTiny possa receber um outro comando. Isto implicaria alterar a biblioteca do TWI e misturá-la com o nível aplicacional, que, neste caso, seria alterar uma variável que indicasse que existia um novo comando prestes a ser recebido. A biblioteca desenvolvida para o TWI abstrai o programador do *hardware*, é apenas necessário ler ou escrever para o barramento I^2C, daí não ser conveniente alterar a sequência de código na interrupção do TWI. Esta biblioteca será detalhada mais adiante. Outra explicação para esta solução foi o facto de os pinos do STM300 terem um comportamento irregular quando este envia telegramas *Enocean* através do seu módulo RF e também quanto entra em

modo "sleep". Verificou-se que os pinos definidos como saídas do microcontrolador eram puxadas ao nível baixo quando eram enviados telegramas, e isto é uma condição grave para o I^2C. Este protocolo necessita de ter os níveis das linhas altas quando não estão a ser utilizadas, e a comutação das linhas para o nivel baixo inicia a comunicação. No STM300, como foi implementado através de *bit-banging* esta situação provoca várias interferências nos dispositivos escravos. No caso dos sensores não há problema, pois estes são desabilitados quando não são utilizados, através da alimentação, para poupar energia. Mas o ATTiny deve ser sempre alimentado para atuar na persiana.

Posto isto, a interrupção externa foi a solução indicada responsável por acordar e indicar, ou se já estiver acordado, apenas indicar, que o módulo de atuação está prestes a receber um novo comando da tabela **5.1**. Assim sendo, o módulo de controlo deve então enviar um sinal ascendente para um determinado pino do ATTiny para avisar ou alertar o início de uma nova sequência de comunicação.

Dois pinos do ATTiny foram configurados para atuar no motor da persiana. Estes pinos são a entrada da ponte "H" que tem as saídas diretamente ligadas ao motor. Quatro pinos estão ligados a quatro outras pontes "H", duas para cada aba das válvulas. Estas pontes "H" estão divididas em dois circuitos integrados idênticos, ou seja, cada um tem duas pontes, e cada um está dedicado a uma aba da válvula. São necessárias duas pontes para cada aba porque é necessária bastante corrente para mover as mesma. As saídas das duas pontes "H" em cada um dos circuitos integrados está ligada ao mesmo ponto, que por sua vez estão ligados aos terminais de uma das abas da válvula. Só esta junção de duas pontes "H" permite energizar as solenoides, porque uma ponte apenas não conseguia fornecer corrente suficiente. Estes circuitos não têm como propósito movimentar um motor, até porque as suas saídas só estarão habilitadas durante 100ms, o tempo necessário para atuar nas válvulas. Servem somente para dar um pulso com elevada potência energética. Quando o pulso é enviado no sentido para abrir ou levantar a aba, a solenoide retrai um pistão e a aba sobe. Por outro lado, se o pulso tiver o sentido contrário, a solenoide relaxa o pistão e a aba desce.

Em qualquer momento não pode funcionar mais que um dos circuitos das pontes "H", quer sejam as das válvulas ou a da persiana, porque isso reflete-se num elevado consumo de corrente que a bateria da janela não pode fornecer, podendo até comprometer o correto funcionamento dos circuitos auxiliares.

Uma outra saída do microcontrolador foi utilizada para habilitar os "step-

ups" da placa de circuitos. Esta saída habilita a ponte de 24v, as pontes das válvulas e também a alimentação do encoder. Desta forma o microcontrolador consegue limitar os consumos energéticos e habilitar os circuitos quando necessário, neste caso, quando move a persiana ou uma das válvulas.

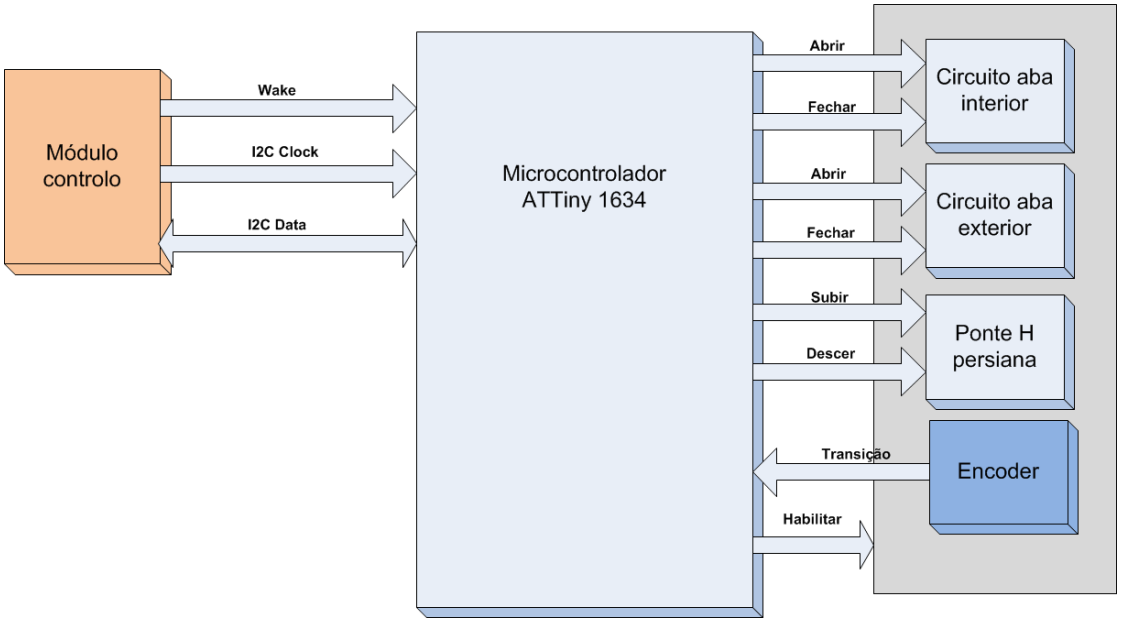


Figura 5.3: Diagrama de blocos do *pinout* do microcontrolador do módulo de atuação

A figura **5.3** demonstra as ligações e o consequente campo de ação do ATTiny1634 colocado no módulo de atuação. Nenhuma das setas representa um barramento, isto é, todas as setas são apenas uma ligação e o sentido das mesmas representa a orientação dos pinos, isto é, se estão definidos como entrada e/ou saída. De destacar a comunicação I^2C proveniente do módulo de controlo e o sinal de "Wake", com a mesma fonte, bem como o pino de habilitação dos circuitos, que é necessário ativar quando qualquer um dos componentes de atuação está a ser utilizado.

5.4.3 Reposicionamento das folhas horizontais da persiana

Um dos requisitos do atuador é o reposicionamento das folhas horizontais da persiana. Sempre que a persiana para, as folhas devem ser orientadas de acordo com um valor definido. Este controlo define a inclinação das folhas que é essencial para a luminosidade. A inclinação das mesmas pode ser proveniente do utilizador, que assim o definiu, ou então do resultado do algoritmo das persianas, calculado no módulo de controlo.

Para implementar a inclinação das folhas da persiana foi definida uma zona "cega". O tamanho da zona é igual para qualquer janela e consiste no número de passos do *encoder* que a janela, numa posição intermédia, necessita para virar totalmente as folhas. Através de vários testes conclui-se que o número de transições do *encoder* necessárias para isto acontecer era de 276. A zona está posicionada portanto, entre o início ou a posição zero, que é a posição de calibração, e 276. Entre as janelas testadas, esta zona cobria apenas 10% do tamanho total da janela mais pequena. A limitação que esta zona implica é que não é possível posicionar a persiana entre qualquer área que a zona abrange. Esta condicionante não afeta significativamente o desempenho ou os requisitos do cliente pois a janela só se encontra nesta zona quando está a subir para a posição zero.

A justificação para esta limitação é a tipologia da persiana. Quando a persiana começa a mover-se da posição zero, esta move-se no sentido descendente. A partir deste momento parte-se do princípio que as folhas estão já na posição máxima, ou de 100%. Através de um *offset* para a posição da persiana é então possível reposicionar as suas folhas para a posição desejada. Para posicionar as mesmas na posição desejada, a persiana tem de primeiro parar, numa posição qualquer, e finalmente ajustar as folhas. Partiu-se do princípio que para uma posição alvo da persiana, era sempre adicionada a inclinação das folhas à posição final como *offset*. Posição_alvo = Posição_persiana + Inclinação_folhas.

Em jeito de exemplo, o utilizador define a posição da persiana como 50%. Esta percentagem é então convertida para um valor exato de transições de *encoder*, através de uma regra de três simples que tem como referência o tamanho máximo da janela. A persiana começa a mover-se e só para quando atinge os 50% mais a inclinação das folhas. Isto porque após paragem, é necessário reposicionar as folhas, que até então induziram um excesso na posição da persiana. Após o reposicionamento das folhas, está então subtraído o valor o valor de inclinação à posição alvo, ficando então a persiana na posição desejada, com as folhas também posicionadas corretamente. Esta inclinação das folhas pode ser definida a qualquer altura pelo utilizador, quer no teclado, quer na aplicação da *cloud*. Este valor é guardado, e quando a persiana para, é utilizado para reposicionar as folhas da mesma. No que diz respeito à zona "cega", este reposicionamento das folhas após paragem da persiana poderia induzir o utilizador em erro, e a própria aplicação, quando a posição da persiana fosse igual ou menor à inclinação das folhas, daí nesta zona "cega" não ser possível parar a persiana.

5.4.4 Estados do atuador

O atuador pode ser representado em dois genéricos estados, acordado e "power-down". O *software* do microcontrolador ATTiny gira também à volta desta restrição necessária para poupar energia e também para um bom desempenho da janela.

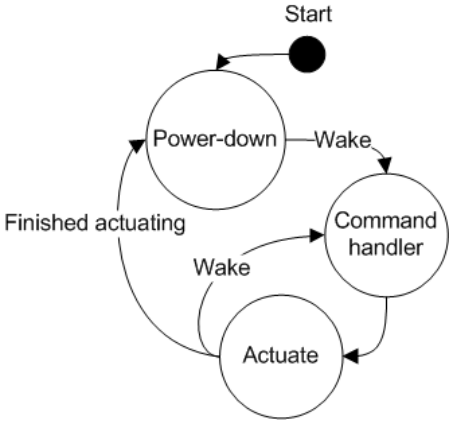


Figura 5.4: Máquina de estados genérica do atuador

Num nível mais abstrato, o atuador pode ser representado pela máquina de estados presente na figura anterior **5.4**, daí ser genérica. Quando está no modo "Power down", apenas a interrupção externa está ativa e à escuta. Quando recebe um sinal, que na figura está representado por "wake", o microcontrolador altera o seu estado e habilita o módulo TWI para ficar à escuta de um novo comando. Quer um comando seja recebido, quer ocorra um "timeout" é o seguinte estado que se encarrega de proceder. Este estado é o resultado um conjunto de interrupções e do principal estado do atuador, gerido por uma variável global que define o mesmo, que dita se a ação que sucede o comando está ou não concluída. Se o estiver, o microcontrolador volta ao estado de "Power-down", senão continua a atuar até chegar ao seu objetivo. Se por acaso, entretanto é recebido um novo comando, este é interpretado como todos os outros e de seguida é executada a ação requerida tendo em conta o estado anterior.

5.4.5 Módulo TWI

O módulo TWI é um periférico do microcontrolador do ATTiny 1634. Este periférico só pode ser utilizado como escravo na comunicação I^2C. O papel do

escravo nesta comunicação é passivo: deve ficar à espera de um endereçamento e de seguida interpretar o pedido do mestre, que neste caso é o STM300 do módulo de controlo. Existem dois tipos de operações no I^2C, a leitura e a escrita. Um bit no final do endereçamento indica qual das operações foi escolhida.

Foi desenvolvida uma máquina de estados, a principal do atuador, que faz o "handling" do comando recebido via I^2C. Logo, toda a implementação do atuador gira à volta deste comando que é um dado resultante da comunicação entre os dois módulos. A partir do comando recebido, a aplicação utiliza outros periféricos, como os temporizadores e também as interrupções externas para mover e parar a janela nas posições desejadas.

Para desenvolver uma biblioteca com bom interface para o utilizador foi necessário criar uma função de configuração do módulo e também uma outra função de habilitação do mesmo. Posto isto, a função de configuração recebe como parâmetro o endereço que o utilizador deseja definir para o dispositivo. Após escrever sobre alguns registos de configuração do periférico, a função define o endereço I^2C do escravo. Com a utilização da função de habilitação, é então possível habilitar e desabilitar o periférico de forma a poupar energia ou evitar que este interfira no barramento I^2C.

As interrupções do periférico, que são despoletadas quando ocorre um "Start Condition", "Stop Condition" ou aquando da troca completa de um byte, são implementadas na biblioteca. Quer isto dizer que o utilizador não tem acesso às mesmas, daí a abstração do *hardware*. Nestas interrupções são guardados os dados provenientes do barramento num *buffer*, no caso da leitura, ou então é escrito o registo do dado do barramento, caso seja uma escrita. Uma máquina de estados em *software* interna à biblioteca ajuda as funções que o utilizador pode utilizar, as funções de leitura e escrita, a monitorizar o estado atual da troca dos dados e assim definir um "time-out" para a comunicação ou então obter ou escrever os dados diretamente do registo do TWI.

A função de leitura tem como argumento um apontador para um *byte*, isto porque os dados podem ser mais do que um e esta mesma função retorna o número de leituras efetuadas. Por outro lado, a função de escrita tem como argumento o apontador para um byte, caso se deseje enviar mais do que um dado, e também o número de bytes expectados. O escravo na comunicação I^2C não define o número de vezes que vai escrever para o barramento, o mestre é que o define. Após cada dado, o mestre indica ao escravo se quer, ou não, outro. Portanto, esta função

retorna o número de operações efetuadas, que, há partida, serão as que foram definidas por argumento.

5.4.6 Sinal Wake

O sinal "wake" (figura **5.4**), proveniente do módulo de controlo, mais precisamente do STM300 da *Enocean*, tem a função de acordar o microcontrolador e indicar que vai começar uma nova sequência de dados I^2C com o ATTiny como destino. Como já fora referido, pode não acordar o microcontrolador se este já estiver acordado a atuar na persiana, mas apenas iniciar o processo de comunicação. O "handling" da interrupção é portanto personalizado para a interrupção e altera o estado da máquina de estados principal do atuador, a tal máquina que também é alterada quando se recebe um comando, para, *à posteriori*, ser feita a operação de leitura do barramento I^2C e interpretar o novo comando.

A configuração desta interrupção é bastante intuitiva pelo que a alteração de um único registo habilita a interrupção para um único pino. Qualquer transição do pino escolhido gera uma interrupção, logo é sempre necessária uma função de habilitação da interrupção para escolher qual das transições se quer para a aplicação. No caso do atuador, foi escolhida a transição do nível baixo para o nível alto como sinal "wake". Portanto, após a trama I^2C estar completa espera-se pelo sinal do pino vir ao nível baixo para habilitar novamente a interrupção.

5.4.7 Sinal do *encoder*

O sinal do *encoder* proveniente do motor é o único *feedback* posicional que o mesmo fornece. Não há qualquer sinal de fim de curso, apesar de existir um botão no topo da persiana para esta parar quando chega ao limite. O sinal consiste em transições de nível, e não pulsos positivos nem negativos. Quer isto dizer que o *encoder* não gera pulsos, apenas comuta o nível do sinal, existindo a possibilidade de o motor parar e o sinal do *encoder* ser baixo ou alto.

Foi então definida uma interrupção de transição de nível para contar os pulsos provenientes do *encoder*, e desta forma monitorizar a posição da persiana e das suas folhas através de variáveis globais. A interrupção é configurável através de um registo, tal como o sinal do "wake". Foi criada uma função para o efeito e também uma função de habilitação. No entanto, ao contrário do sinal "wake",

esta interrupção está sempre habilitada e porque todas as transições devem ser contadas e devidamente controladas.

Em vez de uma interrupção externa de transição de nível poder-se-ia ter utilizado um contador externo ligado a um pino, e a cada transição aumentava-se o contador. Ao invés de estar constantemente a interromper a sequência principal de código, o microcontrolador era interrompido apenas quando o contador chegasse a um valor de destino, que neste caso seria uma posição da janela. No entanto, as opções de contador externo que este microcontrolador fornece são limitadas: são contadas apenas as transições ascendentes ou descendentes, nunca as duas. Desta forma, metade dos sinais do encoder seriam perdidos, logo perder-se-ia resolução e precisão na posição da persiana. De qualquer maneira, como será possível observar, as interrupções externas constantes, que ocorrem quando a persiana se move, não irão afetar o desempenho do sistema.

Em velocidade nominal a frequência das transições está na ordem dos 286 Hz, ou seja, um período de 3.5ms.

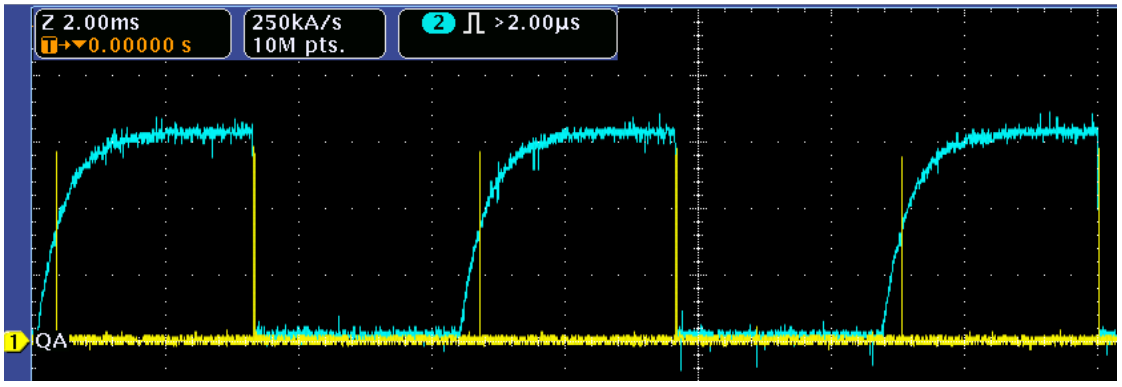


Figura 5.5: Amostra de sinal do *encoder*.
Azul: Sinal do encoder após acondicionamento.
Amarelo: Pulso da interrupção do ATTiny.

Analisando a figura **5.5** conclui-se que o ATTiny tem tempo suficiente para processar informação relativamente às suas variáveis internas, e fazer os devidos cálculos para perceber se a persiana deve parar, antes de ocorrer uma outra interrupção. Neste momento, o microcontrolador verifica se o contador que contém o valor de pulsos do *encoder*, uma variável de inteiros, atinge um valor de comparação, de forma a interpretar corretamente quando a posição alvo foi atingida. O pulso amarelo do gráfico representa o início e o fim da função de atendimento à interrupção externa gerada pelo *encoder*. A diferença entre os dois sinais é significativa, logo o ATTiny pode perfeitamente interpretar as variáveis e parar o

motor antes de outro qualquer sinal de interrupção ocorrer, evitando assim perder alguma transição e consequentemente a posição da persiana.

O sinal do *encoder*, após acondicionamento do mesmo, contém algum ruído, no entanto pode-se observar a interpretação do ATTiny ao sinal. Sempre que há uma transição o microcontrolador consegue detetá-la corretamente.Não obstante, para uma adicional segurança e qualidade do sinal foi adicionado um *Schmitt trigger*.

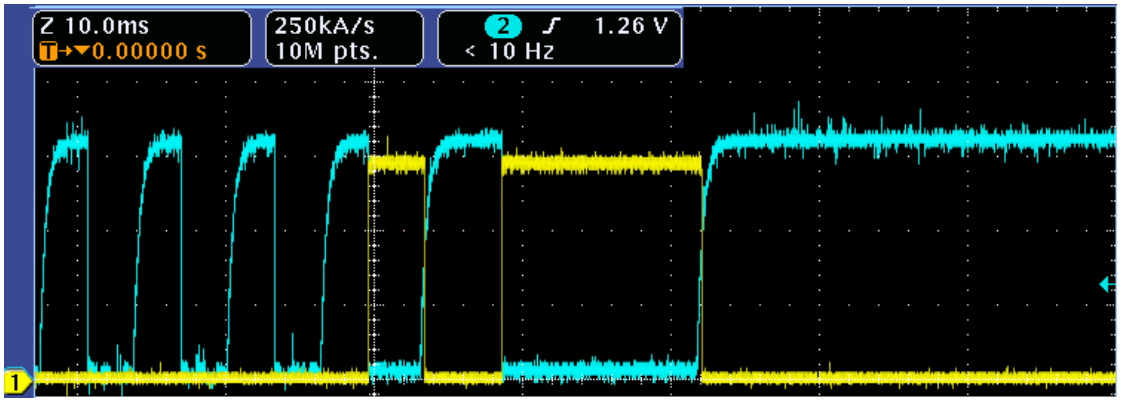


Figura 5.6: Demonstração da inércia do motor através do sinal do encoder.
Azul: Sinal do encoder após acondicionamento.
Amarelo: Pulso da interrupção do ATTiny após deixar de atuar no motor.

Este gráfico representado na figura **5.6** demonstra a inércia do motor. Após o micro desligar a atuação, isto é, após o motor deixar de receber os 24v num dos seus terminais, o motor continua a mover-se e o *encoder* demonstra isso mesmo. As interrupções foram ativadas apenas após a paragem do motor, ou então após o microcontrolador deixar de enviar o sinal para a ponte "H" cortar a alimentação ao motor e neste caso pode-se verificar que ocorreram 4 transições antes de o motor finalmente parar. Como o número de pulsos após ordem de paragem não é constante, foi utilizado um temporizador para dar um *time-out* à contagem. Entre o momento em que a ponte "H" deixa de induzir o motor e o temporizador definir que o motor definitivamente parou, as transições continuam a ser contadas através da interrupção externa que incrementa/decrementa a variável que contém o número de passos do *encoder*. Após este tempo parte-se do princípio que o *encoder* não irá comutar mais o sinal de interrupção. Este tempo foi definido após um conjunto de amostras do sinal logo foi escolhido um tempo que abrangesse o pior caso possível com alguma margem de erro.

5.4.8 Temporizadores

Os dois temporizadores disponíveis no microcontrolador foram utilizados na atuação da janela. O primeiro, de 8 bits, foi utilizado para controlar o arranque e a paragem da persiana. Com intervalos de 100ms é feita a verificação do *encoder*. A variável global que conta os pulsos do *encoder* reporta o movimento, ou não, da persiana, através da sua alteração. Se houver uma diferença em relação ao valor anteriormente guardado, significa que a persiana está-se a mover, se, pelo contrário, o valor for igual, significa que a persiana está parada. Estas comparações podem ajudar a perceber se o motor está com problemas de arranque, se a persiana parou após o movimento induzido pela inércia, ou então se a persiana chegou à posição inicial de calibração. É este temporizador que arbitra se o motor da persiana finalmente parou e que também permite o microcontrolador voltar a adormecer quando isto acontece.

O outro temporizador, de 16 bits, é utilizado para o modo de configuração do comprimento da persiana. É apenas utilizado como um modo de segurança. Após o início da configuração o temporizador começa a contar. Quando chega a um minuto a configuração é cancelada e a persiana volta a subir à posição zero. Se a configuração for bem sucedida o temporizador para e o atuador guarda o comprimento da mesma com o valor que se encontra na variável que conta o número de transições do *encoder*.

5.4.9 Memória EEPROM

A memória EEPROM ("Electrically-Erasable Programmable Read-Only Memory") permite ler dados após o corte na alimentação. Fácil de programar, o microcontrolador pode aceder a esta memória em qualquer momento, após configurada, e escrever ou ler valores da mesma. Esta foi a solução encontrada para a janela *Climawin*: guardar informações relevantes na EEPROM, e ler caso haja um "reset" ao microcontrolador. Portanto a primeira tarefa do ATTiny, quando inicializa o sistema após alimentação, é ler da EEPROM o tamanho da persiana, a sua posição e também a inclinação das suas folhas.

Foi então criada uma função de configuração da EEPROM, em que apenas se escreve par um registo, e também as funções de leitura e escrita de um só byte de cada vez. No que diz respeito à aplicação, foram criadas funções de inicialização das variáveis a partir da EEPROM, leituras basicamente, e também

funções de gravação das mesmas. Em dois casos diferentes a janela guarda na memória EEPROM variáveis: quando o tamanho da persiana é configurado e sempre que a janela para de se mover. Desta forma sempre que um *reset* ocorra, seja por falta de energia na bateria, ou então porque houve um problema no *hardware*, a posição da persiana será sempre mantida. Pode no entanto ocorrer a situação de haver um problema quando a persiana estiver a mover, e nesse caso a posição não será guardada. De qualquer maneira a falta de carga na bateria dificilmente comprometerá a validade dos dados, pois a janela não move a persiana quando a bateria está com carga baixa, por ordem do módulo de controlo.

5.4.10 Interação de módulos a nível aplicacional

Após a inicialização de todos estes módulos, e a abstração dos mesmos, utiliza-se as funções criadas para construir a aplicação do atuador, que se resume em mover as persianas e as válvulas. Nesta parte deste capítulo serão explicados os processos do *software* ATTiny bem como a relação entre os diferentes periféricos e as funções de cada um.

O atuador está constantemente a utilizar um apontador constante para funções. Estas funções atendem o estado atual da máquina que controla os comandos provenientes do I^2C. O início desta sequência de código é despoletado pelo sinal "wake". Após interpretação do comando são testadas sempre duas condições: a posição da persiana, pois pode influenciar a consequência do comando, e também se a persiana está em movimento ou não. O único comando que é independente do estado do atuador é o "Blind working", que deve sempre responder ao módulo de atuação o estado da persiana. Nos outros casos, quer sejam ordens que ditam o movimento da persiana, quer sejam leituras para obter informações da mesma, são sempre testadas estas condições.

Quando é uma ordem para movimentar a persiana ou a válvula, é necessário saber qual a posição da persiana ou se a mesma está em movimento, isto porque pode ser necessário para-la para calcular nova posição alvo, ou então para mover a válvula. Se for uma outra ordem de leitura, para obtenção de dados por parte do módulo de controlo, é necessário parar a persiana para fazer cálculos. Todas a leituras, à exceção do "Blind working" são feitas com valores percentuais. É portanto necessário converter as variáveis, através de regras de três simples, para esta gama de valores. De forma a não comprometer o cálculo a persiana para, para evitar interrupções, e finalmente a resposta é enviada.

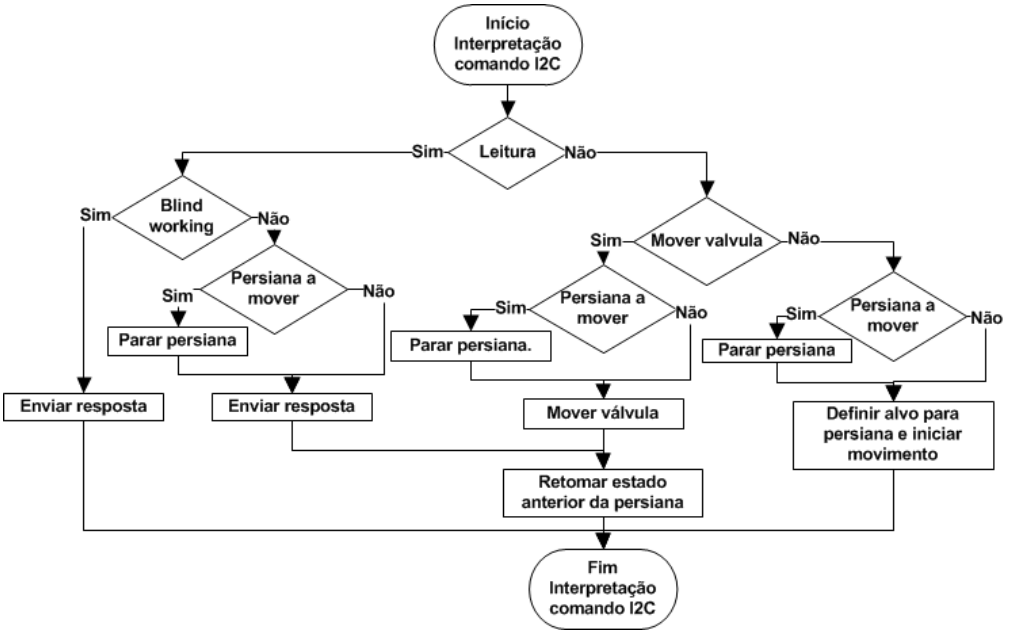


Figura 5.7: Máquina de estados de interpretação de comandos I^2C

O fluxograma representado na figura anterior não reflete o software na sua íntegra. É apenas uma maneira simplificada de representar a máquina de estados presente no *software*. As condições testadas no fluxograma no que diz respeito aos comandos provenientes do I^2C na realidade não são testadas. A diferenciação dos comandos é feita através de um "array" de apontadores para funções que é indexado através da variável que contém o comando I^2C. Utilizando o apontador para função é efetuado o atendimento correto ao respetivo comando, seja ele de escrita ou leitura referente às válvulas ou à persiana.

As operações podem ser de leitura e escrita, e como já fora referido. no caso das leituras, à exceção do "Blind working" a persiana tem de parar para fazer cálculos. Após o cálculo, o movimento reinicia.

Quando é uma escrita pode ser uma ordem para movimentar as válvulas ou a persiana. Para ambos os casos, se a persiana se estiver a mover, esta para e o comando é executado. No caso dos comandos das válvulas, após atuação, a persiana retoma o movimento, já no caso dos comandos da persiana é ligeiramente diferente. Se for um comando de configuração não é definido um alvo, antes pelo contrário, a persiana começa a mover-se e o utilizador é que define o limite do movimento, configurando neste caso o tamanho da persiana. Se, por outro lado, for um comando "Blind up" ou "Blind down" cujo segundo dado é inexistente, a atuação é feita de imediato. Se a persiana estiver a movimentar-se no sentido

que o comando ordena, para e reposiciona as suas folhas de acordo com o valor anteriormente guardado. Se estiver a movimentar-se no sentido oposto do comando a persiana inverte o sentido. Se por ventura o comando for "Blind position send", a persiana para, são feitos os cálculos para definir a nova posição através das transições do *encoder* e o movimento é retomado. Finalmente, no caso do "Flap position send", o dado recebido, que contém a inclinação percentual das folhas, é gravado numa variável temporária. Quando a persiana parar, é efetuado o cálculo e as folhas são reposicionadas de acordo com o seu resultado. Se por ventura a persiana estiver parada e o mesmo comando for recebido, o reposicionamento das folhas é efetuado de imediato.

Em qualquer um dos casos, seja leitura ou escrita, o microcontrolador, após a saída deste fluxograma, não entra obrigatoriamente no estado de "Power-down". O atuador só deixa de ser necessário quando não está a atuar nem na persiana nem nas válvulas, e aí sim pode entrar num modo de redução de consumo de energia. Um dos estados principais do atuador, o que é alterado através da comunicação I^2C, verifica se o motor está parado. Este estado é atendido apenas quando não há qualquer comunicação I^2C, isto é, quando o módulo de controlo não tem qualquer troca de informação com o módulo de atuação. Não obstante, este estado tem a interrupção da linha de "wake" ativa, de forma a escutar e dar inicio a uma nova troca de dados. Outra função deste estado é verificar se o motor finalmente parou, para iniciar o modo "Power-down" e reduzir consumos, através de uma outra variável global que indica se o motor está parado.

Para fazer todo este controlo é necessária uma sintonia entre os diferentes periféricos do microcontrolador que conseguem estão sincronizados devido a estas variáveis globais que indicam o o estado principal do atuador, ou da comunicação I^2C, o estado do motor e até o estado dos temporizadores. É desta forma que um dos temporizadores consegue verificar se a persiana está em movimento ou parada. A interrupção externa que recebe os sinais do *encoder* incrementa ou decrementa a variável posicional da persiana bem como das folhas e verifica se um alvo foi atingido. Estes alvos são definidos aquando do inicio do movimento e podem ser tanto para a persiana como para as suas folhas. Quando o valor da variável posicional é igual ao alvo, a persiana para e as folhas são reposicionadas, com um alvo também, que está guardado *à priori* e também pode ser alterado pelo utilizador com os comandos "Flap".

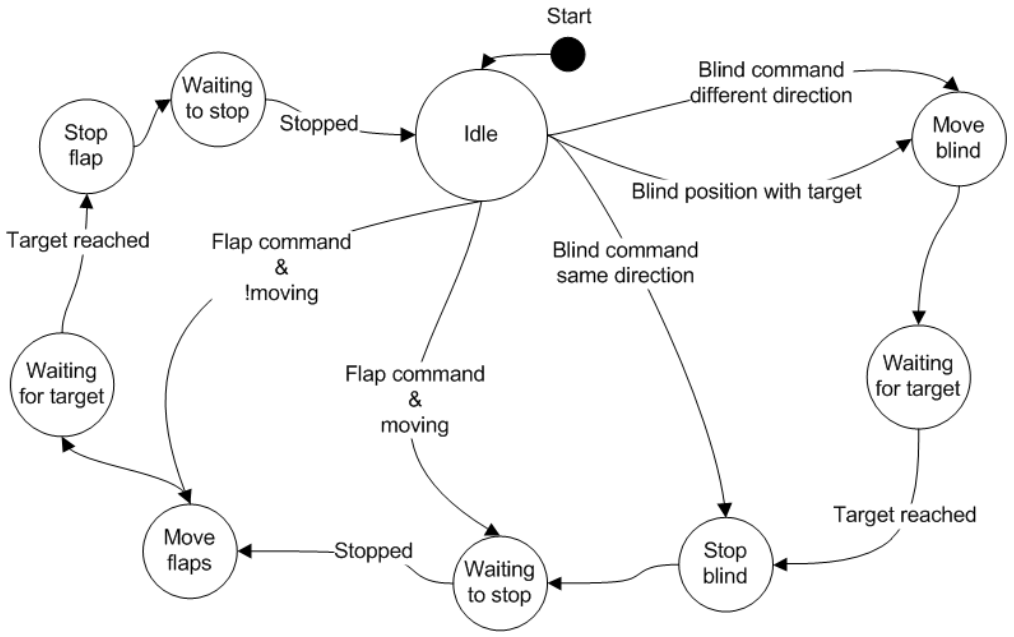


Figura 5.8: Máquina de estados de movimento da persiana e reposicionamento das folhas horizontais

Quando a persiana para de mover, ou por indicação do utilizador ou então porque o alvo posicional foi encontrado, as folhas horizontais são reposicionadas tendo em conta a inclinação percentual previamente indicada pelo utilizador. A figura **5.8** demonstra a máquina de estados que representa o movimento da persiana e o reposicionamento das folhas e as condições testadas, e já referidas, quando é recebida uma nova ordem. Esta máquina é no fundo uma representação com estados da parte da escrita I^2C do fluxograma demonstrado na figura **5.7**. Os estados "waiting for target" são controlados pela interrupção do *encoder*, momento este em que é feita a verificação dos limites da persiana bem como os alvos, se estes existirem. Por outro lado, os estados "waiting to stop" são controlados pela interrupção do temporizador, que basicamente espera que o *encoder* pare de transitar o sinal que fornece.

Os comandos de escrita da persiana, que são basicamente ordens para movimentar a mesma, podem ser "Blind down" ou "Blind up" que são representados neste diagrama por "blind command" e o "Blind position send" que neste diagrama é o "Blind position with target". No caso dos "blind command", a primeira questão a resolver é se é necessário movimentar ou parar a persiana, porque no "blind position with target" a persiana move-se sempre, a não ser que já esteja na posição desejada. Como já fora referido anteriormente, se a persiana estiver a movimentar-se no mesmo sentido que o recebido pelo comando, para. Se pelo

contrário, a persiana estiver parada ou o comando for o sentido oposto, a persiana movimenta-se no sentido desejado. Partindo do princípio que está parada, a persiana começa a movimentar-se, quer seja um comando com ou sem alvo. De seguida, a persiana para, porque, caso não haja um alvo definido pelo utilizador, como o caso dos "blind command", a persiana tem de parar nos limites mínimo ou máximo. Esses limites são os "target" que estão a ser testados constantemente pela interrupção externa do sinal do *encoder*, teste este representado no estado "waiting for target". Quando o limite é atingido, ou o alvo, caso seja um comando "blind position with target", é dada a ordem de paragem. No entanto, e devido à inércia, o temporizador é que define quando há efetivamente uma paragem. Esta espera, induzida pelo temporizador, é representada pelo estado "Waiting to stop". Depois de parado está na altura de reposicionar as folhas de acordo com o valor previamente guardado e desta forma é iniciado um ciclo idêntico ao anterior: a persiana começa a mover, neste caso para movimentar as folhas, chega ao alvo é dada a ordem para parar e espera-se que finalmente pare.

No que diz respeito aos comandos das folhas da persiana é outro processo muito semelhante. A única exceção é que se a persiana estiver em movimento é guardado num *buffer* temporário a posição desejada e quando a persiana estiver realmente parada as folhas são reposicionadas.

Capítulo 6

Testes e Validação

Neste capítulo serão apresentados os testes efetuados às diferentes implementações desenvolvidas, tanto na janela, bem como no ZC. Como o propósito da dissertação é melhorar o *Climawin*, adaptando-o às exigências da *cloud*, será também efetuado um teste de integração de todos os módulos, de forma a desmonstrar o correto funcionamento de todo o sistema.

6.1 Resultados e testes ZC

Nesta secção apresentam-se os resultados obtidos nos testes executados no ZC. As diferentes novas funcionalidades serão testadas e aprovadas de forma a demonstrar um correto funcionamento do dispositivo. Alguns dos testes foram efetuados com a ajuda de um computador de desenvolvimento Linux com a API "EOLink", doravante designado de PC. Com esta API é possível fazer os pedidos série característicos do ESP para testar as respostas do ZC e portanto simular o papel do *gateway*. Foi então desenvolvida uma simples aplicação utilizando o "EOLink" para testar o ZC fazer a validação ao *software* desenvolvido.

6.1.1 Controlo do exaustor

Para controlar o exaustor é necessário enviar apenas dois diferentes telegramas, uma para abrir o relé, e outro para o fechar. Para o relé ser comutado, é necessário este aprender ou obter o identificador do ZC correspondente através da

receção de um telegrama RPS equivalente ao telegrama de abertura do relé. No teste demonstrado de seguida, este processo de aprendizagem já teria sido efetuado.

Listagem 6.1: Telegramas escutados no processo de controlo do exaustor

```
1 Sep 10 15:52:56 | Data: 80 64 00 FF    SrcID: 010297AE
        DstID: 00861D26
2 Sep 10 15:53:06 | Data: 80 64 FF 3E    SrcID: 00861D26
        DstID: FFFFFFFF
3 Sep 10 15:54:04 | Data: 80 FF 05 FF    SrcID: 00861D26
        DstID: FFFFFFFF
4 Sep 10 15:54:04 | Data: 10      SrcID: 010297AE    DstID:
      FFFFFFFF
5 Sep 10 15:54:38 | Data: 80 FF 03 FF    SrcID: 00861D26
        DstID: FFFFFFFF
6 Sep 10 15:54:38 | Data: 30         SrcID: 010297AE
     DstID: FFFFFFFF
```

Estes telegramas são o resultado de um "sniffer" *Enocean*. Este dispositivo pode ser qualquer microcontrolador STM300 que tem a capacidade de intercetar telegramas *Enocean* e reencaminhá-los via série através do protocolo ESP. Neste caso foi utilizada uma pen USB300, que é um aparelho *Enocean* que é compatível com todo o protocolo ESP e pode ser utilizada como um *gateway*. Através do "EOLink" é possível desenvolver uma aplicação que guarda todos os telegramas intercetados num ficheiro de *log*, disponibilizando assim um histórico de tudo o que foi apanhado no ar.

No *log* **6.1** foram intercetados tramas de comunicação entre uma janela e um ZC. Os dois primeiros telegramas nada tem a ver com o exaustor, são apenas uma demonstração da comunicação entre um ZC e uma janela aprendida através do "Smart-Acknowledge". O primeiro telegrama é uma ordem do ZC, após o pedido de dados da janela (figura **3.8**), para mover a persiana para uma determinada posição (tabela **3.5**), daí o 0x80 como o primeiro dado. 0x64 indica a posição desejada, ou seja, 100%: persiana totalmente fechada. Os seguintes dados são ignorados como comando pois a janela interpreta corretamente os seus valores, quer isto dizer que o telegrama é apenas para mover a persiana para a posição indicada. Uns segundos depois a janela para e responde com a posição da persiana bem como da inclinação das suas folhas. Após a receção da informação via I^2C, a janela constrói um telegrama 4BS e envia-o sem destino, ou como "broadcast". Não obstante, o propósito do telegrama é um *update*, e como a fonte do telegrama é identificável

os dados podem ser interpretados corretamente. Neste caso, a persiana parou na posição 100% e as suas folhas encontram-se com uma inclinação de 30%.

Estes dois telegramas servem apenas para demonstrar que a janela está aprendida pelo ZC. De outra forma esta janela não se moveria após a receção de um telegrama deste mesmo controlador. Os quatro telegramas que vêm de seguida demonstram o processo de abertura e fecho do exaustor.

Para efetuar este teste foram forçados diferentes cenários através da combinação de botões da janela de forma a obter resultados. Quando a janela enviou um telegrama indicando o cenário 0x05, que é um dos cenários que necessita elevada circulação de ar, o ZC imediatamente abriu o relé através de um telegrama RPS com o dado 0x10. Por outro lado, quando o ZC intercetou um telegrama da janela indicando que a mesma alterou para um cenário em que a circulação de ar não é uma prioridade, o ZC fechou o relé, daí o telegrama com o dado 0x30. Estes telegramas são construidos e estruturados segundo os telegramas de interruptores *Enocean*, daí serem RPS e os dados terem estes valores constantes. Posto isto, este teste valida o comportamento do ZC aquando da receção de um telegrama de uma das suas janelas indicando alteração de cenário. Após interpretação do telegrama verifica se é necessário abrir ou fechar o relé.

6.1.2 Resposta a Read version

O "Common command" com o sub-comando "read version" serve para obter o nome da aplicação do dispositivo Enocean e a versão do *software*. Posto isto, após uma pacote série *Enocean* ser recebido, o ZC mestre deve responder corretamente. Através de um programa que consiste num simples terminal série foi introduzida a sequência hexadecimal do pacote "read version". Através da especificação do ESP [21] é possível construir o telegrama byte a byte, pois este é constante.

Figura 6.1: Resposta a pedido "Read version"

A amostra na figura **6.1** é uma resposta ao pedido de versão do ZC. O cabeçalho indica-nos o tamanho do telegrama, que é de 0x0021 (33 bytes). O "Optional Data" é zero, e o tipo de pacote série é 0x02, que é resposta. 0x26 é o resultado do CRC do cabeçalho. De seguida, o primeiro dado do pacote é zero. Isto

é apenas uma indicação de que o pedido foi corretamente interpretado. A versão do *software* da aplicação vem de seguida, com 2.0.0.0, seguida da versão da API "EO3000I", atualmente em 2.4.4.2. O identificador único do ZC mestre testado é 0x10297AE e finalmente a descrição da aplicação "CLIMAWIN ZC" seguida de um carácter terminador. O último byte é o resultado do algoritmo CRC com os dados todos do telegrama como entrada.

Este teste foi também efetuado com o "EOLink", simulando verdadeiramente a aplicação que é executada no *gateway Climawin*. No entanto, ao contrário do exemplo anterior, em que uma pen USB300 estava ligada ao PC que executava um código desenvolvido com a ajuda da biblioteca "EOLink", neste caso era o ZC que estava diretamente ligado ao PC, de forma a receber comandos diretamente do mesmo via série pelo protocolo ESP. O resultado foi obtido utilizando o "debugger" do IDE *NetBeans* para sistemas operativos Linux.

```
176
177 if(myGateway->commands.ReadVersion(version)==EO_OK)
178 {
    printf("%s %i.%i.%i.%i, ID:0x%08X on %s\n",
180                 version.appDescription,
181                 version.appVersion[0], version.appVersion[1], version.appVersion[2], version.appVersion[3],
182                 version.chipID,
183                 SER_PORT);
184
Output
EXEMPLO1 (Build, Debug) ×  EXEMPLO1 (Debug) ×
Opening Connection to USB3
CLIMAWIN ZC 2.0.0.0, ID:0x010297AE on /dev/ttyUSB3
```

Figura 6.2: Resposta a pedido "Read version" obtida no terminal do PC

A figura **6.2** ilustra o resultado do pedido efetuado pelo PC e impresso no terminal da aplicação. Esta é a forma que os próprios exemplos aplicacionais do "EOLink" testam o dispositivo aos quais estão ligados e também será a maneira ideal de detetar o ZC mestre quando este estiver ligado ao *gateway Climawin*, como demonstrado na figura **3.1**.

6.1.3 Descobrimento de ZCs

O ZC mestre deve ser capaz de descobrir ZCs escravos da mesma instalação aquando de um *input* do utilizador. Foram executados dois diferentes testes para dois diferentes casos. Para o efeito foram monitorizados três diferentes terminais série de três diferentes dispositivos *Climawin*: um ZC mestre e dois ZCs escravos. Nestes testes foram avaliadas a habilitação dos ZCs escravos bem como o tempo

de resposta dos mesmos assim como a obtenção dos identificadores dos devidos escravos por parte do ZC mestre.

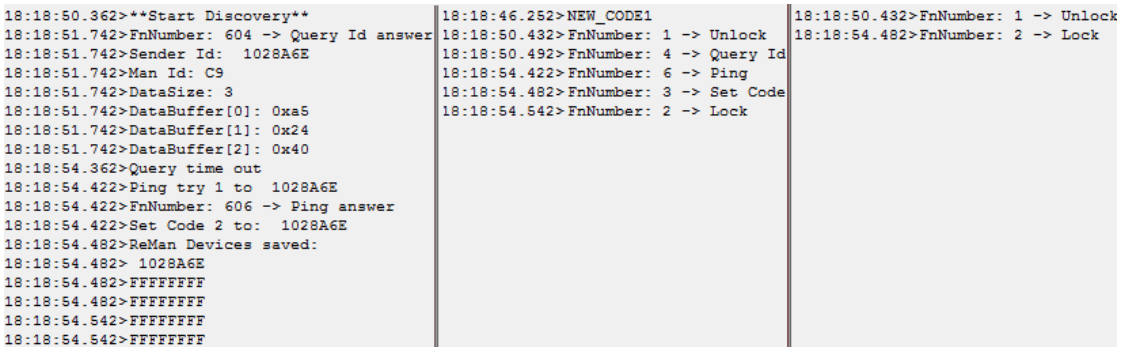


Figura 6.3: Demonstração do descobrimento de ZCs através de terminais série.
Esquerda: ZC mestre
Centro: ZC escravo 1, habilitado
Direita: ZC escravo 2, não habilitado

Neste primeiro exemplo apenas um dos escravos está habilitado, isto é, em apenas um deles o utilizador optou por iniciar o processo do descobrimento, que, como fora referido anteriormente, necessita de ser habilitado antes de o ZC mestre começar a procurar dispositivos. O escravo 1 é habilitado às 18h18m46s, como é possível observar pela gravação do código 1 como código de bloqueio/desbloqueio *ReMan* em "NEW_CODE1" (diferentes códigos *Climawin* na listagem **4.3**). Neste momento o escravo 1 responde a pedidos que sejam provenientes de um ZC mestre que execute um descobrimento. Por outro lado o escravo 2 não está habilitado que hipoteticamente poderá ser um ZC escravo de uma outra instalação vizinha.

Todas as mensagens *ReMan* são caracterizadas pelo número da sua função. Sempre que um dispositivo recebe uma mensagem, imprime a função e a sua descrição. Às 18h18m50s o ZC mestre inicia o procedimento de descoberta, consequência direta do *input* do utilizador. Todos os ZCs escravos ao seu alcance recebem o "unlock", mas apenas um é desbloqueado, o escravo 1. A partir do momento em que está desbloqueado recebe e responde a qualquer pedido do mestre. No final o escravo 2 deteta um "lock", no entanto não tem qualquer efeito, pois o código é errado e este já se encontra bloqueado.

O primeiro passo é enviar o "query broadcast" e esperar pelas respostas (figura **4.9**). A cada resposta o ZC verifica o EEP do dispositivo que lhe respondeu, bem como o "Manufacturer ID". O identificador do fabricante é obtido através de uma função do "EO3000I" que fornece o dado relativamente à última resposta obtida. O *payload* de um "query answer" é de três *bytes* e os seus dados contém o

EEP do dispositivo. O primeiro *byte* contém o RORG, o segundo contém parte do FUNC e do TYPE e o terceiro contém parte do TYPE. Após transformação dos *bytes*, segundo o telegrama 4BS "teach-in" [22], verifica-se que o RORG é 4BS, o FUNC 0x09 e o TYPE 0x08, precisamente o perfil do ZC.

Exatamente quatro segundos depois de enviar o "query", o ZC mestre inicia o processo de verificação dos escravos que lhe responderam, neste caso apenas 1, que consiste em "ping" e a espera de uma resposta, seguido de um novo código (figura **4.10**). Para efeitos de avaliação, foi também impresso no terminal os identificadores dos ZCs descobertos. Neste caso fora apenas o escravo 1, que tem como identificador 0x1028A6E. Os restantes quatro não existem e estão preenchidos com o valor por defeito.

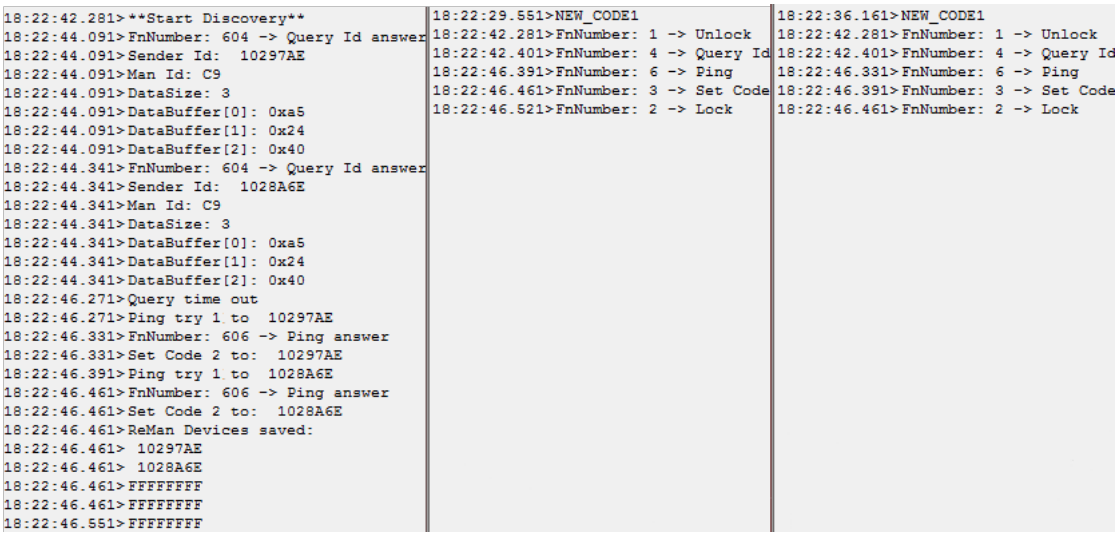


Figura 6.4: Demonstração do descobrimento de ZCs através de terminais série.
Esquerda: ZC mestre
Centro: ZC escravo 1, habilitado
Direita: ZC escravo 2, habilitado

Neste segundo exemplo o mesmo procedimento é executado, no entanto o utilizador, antes de iniciar o processo no ZC mestre, ativou ambos os escravos. Consequentemente, ambos os dispositivos são desbloqueados e ambos respondem ao "broadcast query" desfasados no tempo, neste caso com uma diferença de 250ms. O resultado é a obtenção de duas respostas ao "broadcast query" por parte do ZC mestre seguidas de dois procedimentos de confirmação enviando o "ping" e esperando pela resposta de um escravo de cada vez, guardando entretanto o novo código. Na tabela final verifica-se que dois ZCs escravos diferentes foram guardados na memória do ZC mestre e ambos estão preparados para eventualmente

responderem a um pedido "window sweep" para o ZC mestre obter o identificador de as janelas associadas aos dois escravos.

6.1.4 Obtenção do mapa de rede

Partindo do teste anterior, ou seja, um ZC mestre com dois diferentes escravos registados na sua memória, foi avaliada a capacidade do mestre em obter a lista de dispositivos na rede, isto é, janelas. O ZC deve interpretar corretamente o pedido série enviado pelo controlador e iniciar o processo de criação do mapa de rede com a contribuição do "ReMan" para depois construir telegramas de resposta e enviar toda a informação de volta para o *gateway*. Este teste foi também executado através de um pedido série feito através da aplicação desenvolvida com a utilização da biblioteca "EOLink" a correr no PC.

```
Source  History
34  eoReturn ZoneController::sendSmackLearnTables()
35  {
36      if(this->myGateway != NULL)
37      {
38          eoPacket myPacket;
39
40          myPacket.type = PACKET_SMART_ACK_COMMAND;
41          myPacket.dataLength = 1 ;
42          myPacket.optionalLength = 0;
43          myPacket.data[0] = SA_RD_LEARNEDCLIENTS;
44
45          this->state = WAITING_FOR_TABLE_ANSWERS;
46
47          timer = eoTimer::GetTickCount() + WINDOW_TABLE_ANSWER_DELAY;
48
49          return myGateway->stream->Send(myPacket);// only send, don't wait for answer
50      }
51      return EO_ERROR;
52  }
```

Figura 6.5: Código de teste desenvolvido para fazer o pedido "Read Learned Clients"

Como já fora referido, o pedido "Smart Ack Table" (figura **4.7**) inicia o procedimento da obtenção do mapa de rede. Esta porção de código desenvolvida em ambiente Linux envia esse mesmo pedido via série sob o protocolo ESP para obter a resposta do ZC mestre. Em primeiro lugar é construído o pacote, com o tipo "Smart Ack Command" seguido do comando, "read learned clients". As seguintes atribuições servem apenas para definir o estado da máquina da aplicação, pois esta tem de esperar quatro segundos no máximo pelas respostas do ZC mestre. Finalmente o pacote é enviado através de uma função que tem a responsabilidade de interpretar os dados do mesmo e enviá-lo efetivamente via série e retorna um enumerado que indica se o processo foi concluído com ou sem sucesso.

De seguida é só entrar num ciclo com a ligação série aberta à espera de respostas. Após a receção da mesma, é verificado o tipo, neste caso, se é do tipo resposta, e também se é divisível pelo número inteiro nove. Se o for, significa que o ZC mestre construiu corretamente o telegrama e os dados do mesmo contém informação sobre as janelas de um determinado ZC do sistema.

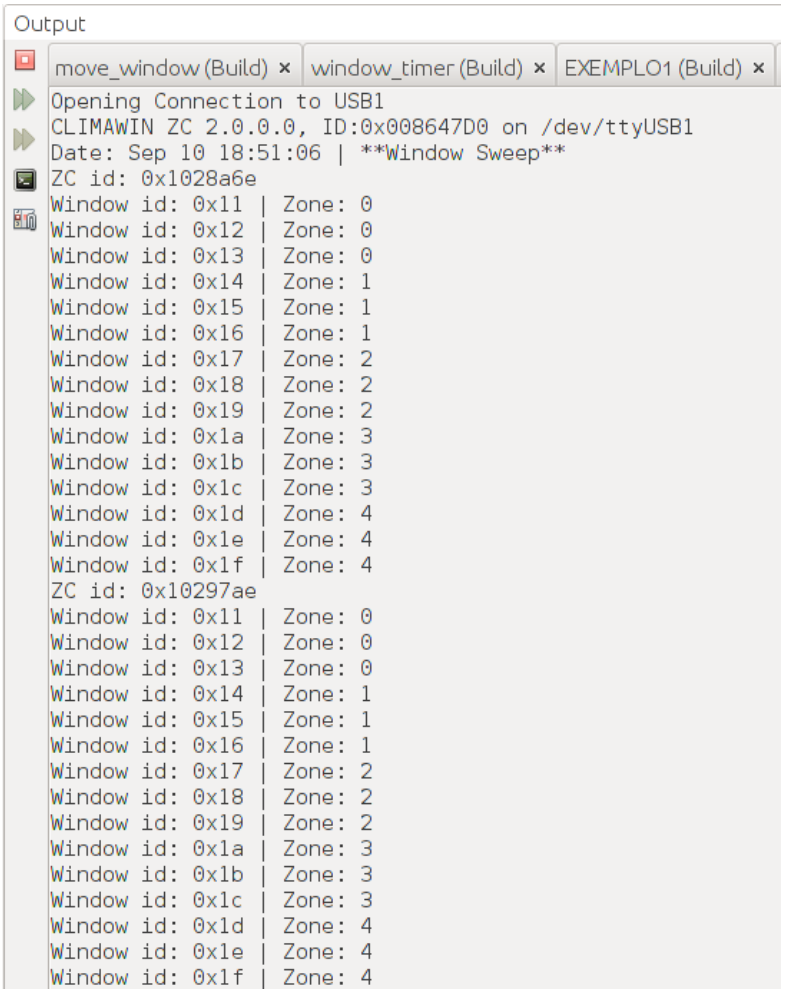


Figura 6.6: Respostas obtidas após envio do pacote "Read Learned Clients"

Esta informação filtrada e representada na figura **6.6** é adquirida sobre forma de um, ou mais, pacotes de resposta provenientes do ZC mestre. Após a introdução de um comando via teclado, o processo inicia e o código na sequência **6.5** é executado. Depois inicia a espera pelas respostas.

Para cada janela são enviados 9 *bytes*, no entanto a informação demonstrada no ecrã apresenta o identificador do ZC que tem a janela aprendida apenas uma vez, de modo a não repetir dados. Os identificadores únicos dos ZCs apresentados são os ZCs escravos obtidos no processo de descobrimento: 0x1028A6E e 0x10297AE.

Ambos os ZCs têm 15 janelas aprendidas e os identificadores as mesmas são iguais porque foi utilizado um software de teste nos ZCs escravos. Também se pode concluir que o ZC mestre não tem qualquer janela aprendida.

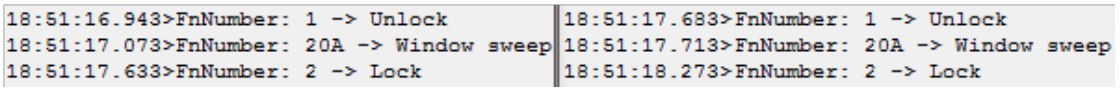


Figura 6.7: Demonstração do "Window sweep" através de terminais série.
Esquerda: ZC escravo 1.
Direita: ZC escravo 2.

A reação dos ZCs escravos está representada nesta figura **6.7**. Um de cada vez, o ZC mestre faz o "unlock", "window sweep", espera pela resposta e envia o "lock" sequencialmente. Pelos tempos, pode-se observar que o processo é feito separadamente, ou seja, um ZC escravo de cada vez, e que não há telegramas "broadcast".

6.2 Resultados e testes do Atuador

Nesta parte do capítulo dos resultados serão explicados os testes executados na janela como um todo. Quer isto dizer que a avaliação será feita a toda a janela, ou seja, a ambos os módulos de controlo e atuação pois é necessário testar ambos para que o produto final seja consistente. Como este trabalho tem como foco o módulo de atuação, serão executados alguns testes com a ajuda do módulo de controlo, bem como dos restantes periféricos do *Climawin* para provar o correto funcionamento da atuação.

6.2.1 Demonstração da atuação na persiana e válvulas

Como se trata de uma implementação que resulta em movimento e perceção visual, ao contrário do ZC - resultados da implementação através de imagens do computador - esta secção demonstrará diferentes posições dos objetos atuados sobre forma de fotografias.

Após a receção um telegrama proveniente do ZC a janela pode alterar a posição da persiana, a inclinação das folhas e também o cenário, pelo que o utilizador pode-o fazer através da *cloud*. Neste caso foi desenvolvida uma outra aplicação recorrendo à biblioteca "EOLink" para criar telegramas estruturados segundo o

perfil da janela, e então, mover os objetos da forma pretendida. O interface criado não está numa aplicação desenvolvida a pensar no utilizador, com interface gráfico agradável, como seria de esperar de uma aplicação que recorresse aos serviços da *cloud*. Não obstante, a aplicação permite definir qualquer posição para a persiana e as suas folhas e também cenários das válvulas através da introdução de códigos na linha de comandos do computador que está ligado à pen USB300, que posteriormente envia o telegrama para o ZC.

Figura 6.8: Demonstração de diferentes posições da persiana da janela *Climawin*. Seguindo ordem de leitura: totalmente aberta, 33%, 66% e totalmente fechada.

Pela primeira vez é apresentada a janela *Climawin*. Toda a sua estrutura é desenhada e construída pela empresa alemã Rauh, que, com a colaboração da equipa *Climawin*, desenhou uma estrutura que suporta todos os componentes eletrónicos da janela. À esquerda da janela temos o painel frontal, com o teclado que está diretamente ligado ao módulo de controlo (figura **5.2**). A parte de cima da estrutura da janela contém uma cavidade para colocar a caixa das válvulas. Toda esta parte superior é depois tapada por uma calha de velcro e alumínio de forma a esconder e proteger as válvulas sem comprometer a circulação do ar.

Nesta sequência de fotografias na figura **6.8** estão apresentadas diferentes posições da persiana, algumas das demais pelas quais o utilizador pode optar. Todas estas posições foram definidas através da aplicação que está a ser executada

no PC e envia o telegrama para o ZC através da pen USB300. Notar que a orientação das folhas, ou a sua inclinação, já se encontra definida, pois a persiana já teria parado e reposicionado as folhas antes de tirar a fotografia. Neste caso as folhas encontravam-se com uma inclinação zero, isto é, em posição horizontal.

Figura 6.9: Demonstração de diferentes inclinações das folhas da persiana da janela *Climawin*.
De cima para baixo: inclinadas para o lado de fora, horizontal e inclinadas para o lado de dentro.

A anterior figura **6.9** demonstra as três diferentes posições que o utilizador

pode escolher através do teclado. A primeira impede a entrada da luz solar dentro da divisão, pois as folhas estão totalmente inclinadas contra o sol. A segunda e terceira posições permitem entrada de luz, mais ou menos, dependendo da posição do sol. Estas são as únicas posições que podem ser definidas através do teclado colocado na moldura frontal da janela, no entanto outras inclinações podem ser definidas através da *cloud*, ou neste exemplo, através da aplicação desenvolvida a correr no PC. A posição mínima é a representada pela primeira imagem da figura anterior, em que as folhas estão viradas para o lado de fora. Esta posição corresponde a uma inclinação de 0% para o utilizador. Desta forma, 50% é a posição horizontal e 100% com as folhas viradas totalmente para dentro. A resolução da inclinação das folhas é de um ponto percentual, tal como na definição da posição da persiana, logo o utilizador pode inclinar as folhas com uma precisão que corresponde a qualquer situação.

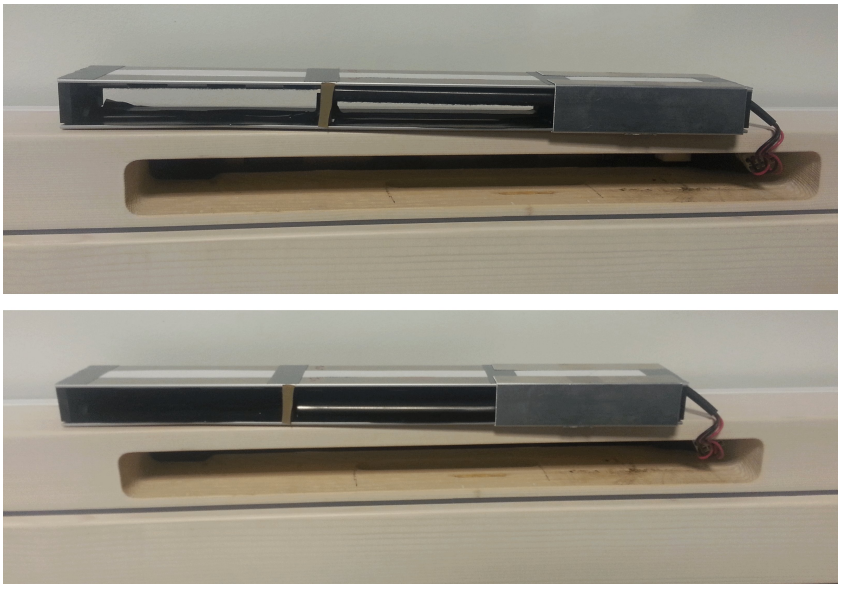


Figura 6.10: Demonstração das duas diferentes caixas das válvulas.
Primeira foto com as duas abas abertas e segunda foto com as duas abas fechadas.

Nesta figura 6.10 estão representados dois dos cenários possíveis após o cálculo do algoritmo de controlo. A primeira foto representa um cenário em que seria necessária a circulação do ar, e o segundo o seu oposto, em que ambas as abas estão fechadas e não há circulação alguma. Em ambas as fotos a caixa da esquerda das válvulas é a "Bypass box" e a caixa do lado direito é a "Preheat box" (secção **3.1.1**). Por este mesmo motivo a caixa da esquerda, a do "bypass", tem apenas uma aba do lado de fora, isto porque a caixa de ar presente na janela, entre os vidros, após a colocação das válvulas na cavidade da janela, encontra-se por baixo da

caixa de "preheat". Como já fora referido cabe a esta caixa do "preheat", através da aba que se encontra do lado de dentro, controlar a entrada do ar pré-aquecido na janela.

O propósito principal destas fotos é demonstrar a atuação sobre as válvulas e a comutação do seu estado. Estas posições das abas definem cenários. Cenários estes que foram escolhidos e enviados para a janela através da pen USB300, após passar pelo ZC.

Quando a janela recebe um cenário das válvulas, dado proveniente de uma mensagem da *cloud*, o algoritmo de controlo deixa de ter efeito, pois assim o utilizador desejou. Nesse momento, o contador que decrementa e define o momento em que o algoritmo deve ser executado novamente reinicia, dando prioridade à escolha do utilizador, pelo menos durante um certo período de tempo. Tempo este que é constante e de 90 minutos, que é o tempo definido pelo módulo de controlo para o cálculo de novos cenários.

6.2.2 Teste de integração

A previsão de cenários do utilizador permite estimar a duração da bateria da janela bem como o comportamento do mesmo quando se encontra no mercado ou nas mãos dos utilizadores. Foi portanto definido um cenário para estimar a duração da bateria da janela, bem como avaliar o comportamento dos equipamentos ZC e janela do *Climawin*.

O cenário de teste consiste em três diferentes dispositivos *Enocean* a comunicar entre si, sendo que apenas dois deles fazem parte do *Climawin*, não excluindo a possibilidade de desenvolver uma versão compatível com o terceiro. Os componentes *Climawin* são um ZC escravo com uma janela aprendida pelo mesmo. O restante é uma pen USB300 *Enocean* ligada a um computador com SO Linux a correr uma aplicação com a biblioteca "EOLink" como principal suporte.

De 30 em 30 minutos o ZC recebe um telegrama proveniente da pen e estruturado segundo o perfil da janela para esta mover a persiana para a posição máxima ou mínima. A janela tem também as devidas funcionalidades ativadas, isto é, efetua a leitura de sensores para calcular o cenário das válvulas e move as mesmas. A leitura dos valores dos sensores e o cálculo do cenário das válvulas é efetuado de 90 em 90 minutos e o período de controlo das válvulas é de 30 minutos. Este período de controlo da abertura ou fecho das válvulas serve para, em alguns

cenários, comutar a posição das mesmas de forma a regular a entrada e saída do ar frequentemente. Como as persianas estão constantemente a ser movidas através de ordens provenientes do PC, o resultado do algoritmo para mover persianas não tem qualquer influência.

Este cenário antevê uma utilização frequente e constante por parte do utilizador e permite estimar a duração da bateria. Diferentes testes concluem uma duração da bateria, com a janela em funcionamento normal, isto é, a desempenhar todas as funções, de cerca de 10 dias e 10 horas. Em poupança de energia a janela faz todas as suas funcionalidades com a exceção do movimento da persiana, pois a prioridade é comutar as válvulas tendo em conta o cenário calculado através de sensores. Este modo de poupança de energia é ativado quando a bateria está muito baixa, abaixo de 4% de carga. Neste teste em específico, a janela continuou a comutar as válvulas e a enviar dados para o ZC, através dos telegramas de "update", durante mais 3 dias e 18 horas até finalmente ficar sem energia para fazer qualquer função. Gráficos que ilustram os resultados deste teste no anexo **A**.

Em suma, estes testes permitiram avaliar de forma positiva o desempenho do atuador, pois durante os testes a estes cenários o módulo de atuação correspondeu corretamente às expectativas e desempenhou a sua função sem comprometer o correto funcionamento tanto da persiana como das válvulas.

Por outro lado, no que diz respeito ao ZC, a nova funcionalidade que consistia em criar novas mensagens para a janela *Climawin* está testada e aprovada aquando do sucesso destes testes. Isto porque o comando para mover a janela para a posição mínima e máxima, por exemplo, é um novo comando que na versão anterior não estava implementado e que o ZC não tinha capacidade de reencaminhar para a janela através da *mailbox* do "Smart-Ack". Neste teste em específico foi referido um ZC escravo que recebia telegramas do PC através de uma pen USB300 *Enocean*. Quer isto dizer que o ZC cria as mensagens de correio através de telegramas provenientes do PC, e indiretamente, da *cloud*. Portanto, além do desenho da arquitetura *Climawin* que está demonstrada na figura **3.1** é também possível desenvolver uma instalação em que não existe qualquer ZC mestre, sendo que o papel de *gateway* entre a *cloud* e o *Climawin* pode ser desempenhado apenas pelo PC, ou Raspberry Pi, com a pen USB300 em vez de o PC com o ZC mestre.

Não obstante, foi também efetuado um teste com um ZC mestre apenas, ligado ao PC, a controlar uma janela. Os resultados foram semelhantes aos do teste anterior pois do ponto de vista da janela a ligação *Climawin* - PC é indiferente. A

única diferença é que o ZC neste caso recebe ordens da ligação série sobre forma de pacotes ESP para posteriormente criar mensagens para a janela.

Capítulo 7

Conclusão

Neste capítulo serão apresentadas as conclusões resultantes do desenvolvimento da dissertação bem como algumas propostas para expandir e melhorar o produto.

7.1 Conclusão

O *Climawin Demo* tinha como principais métricas melhorar a consistência do produto bem como preparar o sistema para a ligação à *cloud*. A adaptação do controlador central, o ZC, foi um ponto essencial para fazer esta ligação à rede. É então possível instalar diferentes ZCs remotos, ou escravos. Esta nova característica permite a expansão da instalação, tanto em termos espaciais, bem como em quantidade de *end-devices* ou janelas. No entanto, a principal implementação no que diz respeito ao controlador central é a capacidade de enviar informação de toda a rede de *end-devices* e os respetivos controladores para o seu controlador, o *gateway*, aquando de um pedido do mesmo. Esta nova implementação é essencial para o utilizador ter um mapa de todos os dispositivos instalados nas habitações que lhe dizem respeito. Estão então fornecidas as bases para desenvolver aplicações para o *end-user* desfrutar do seu produto simples e confortavelmente.

No que concerne a janela, as modificações e melhorias fornecem um *end-device*, que é essencialmente o cerne de todos os sistemas de domótica, melhor e mais consistente. Em conjunto com as alterações efetuadas no ZC, no que diz respeito à criação de novas mensagens para a janela, e também no módulo de controlo da mesma, a nova atuação permite uma maior gama de controlos por

parte do utilizador. O *feedback* fornecido pelo *encoder* é o maior trunfo desta nova janela, pois com a implementação do mesmo são agora exequíveis comandos para posicionar a persiana em qualquer posição, bem como as folhas horizontais para controlar a luminosidade no interior. As válvulas estão também bem mais simples e eficazes com um controlo bi-estável para controlar a circulação do ar na janela.

Todos os requisitos inicialmente propostos foram cumpridos. O *Climawin Demo* está pronto para a *cloud* e a janela está pronta para ser apresentada. Uma enorme quantidade de possibilidades está agora aberta, e o *Climawin* pode ser explorado para serem desenvolvidas aplicações em variadíssimas plataformas.

7.2 Trabalho futuro

Apesar de a proposta da dissertação ter sido cumprida, o *Climawin* pode sofrer algumas alterações sofre forma de melhorar o produto em si. Serão apresentadas de seguida algumas ideias para melhorar o *Climawin*.

Em primeiro lugar, é necessária uma melhor gestão da relação controlador/espaço. Como referido anteriormente, um ZC remoto pode ter associadas varias janelas para funcionar como "postmaster" na comunicação "Smart-Ack". Para instalar janelas em divisões diferentes e distanciadas é necessário instalar um novo ZC escravo, que por sua vez necessita de uma comunicação direta com o ZC mestre. Estas restrições implicam custos que com um repetidor de telegramas *Enocean* talvez fossem suprimidas. Posto isto, o estudo e a implementação de repetidores na rede *Climawin* é uma possibilidade para o futuro.

Outra questão no que diz respeito à gestão de recursos e espaço no *Climawin* é o sensor de CO_2, que é um produto dispendioso. Como este está acoplado ao ZC só as janelas deste ZC podem receber informação relativamente ao valor da concentração do gás. O ideal seria um sensor CO_2 independente de qualquer ZC, que poderia ser aprendido pelos controladores através de um método que poderia ser despoletado pelo instalador assim que o desejasse. Desta forma seria possível ter um sensor apenas, para variados ZCs, e consequentemente janelas. Posto isto, a ideia do desenvolvimento de um sensor destes, bem como a adaptação dos ZCs para aprender estes novos sensores, são propostas futuras.

Por outro lado, no âmbito do outro foco desta dissertação, a atuação pode também sofrer algumas modificações. Como já fora referido, o motor da persiana

fornece um *feedback* da posição do mesmo. Contando os pulsos do *encoder* o microcontrolador consegue estimar a posição da persiana. No entanto, para efeitos de segurança seria conveniente um sinal de fim de curso, para ter a certeza de que a persiana está na posição inicial. Apesar de o *software* desenvolvido ter a capacidade de prever que a persiana está na posição zero, através de um temporizador, seria uma outra camada de consistência e fiabilidade, caso o *encoder* deixasse de funcionar.

Anexos

Anexo A

Gráficos de testes à janela

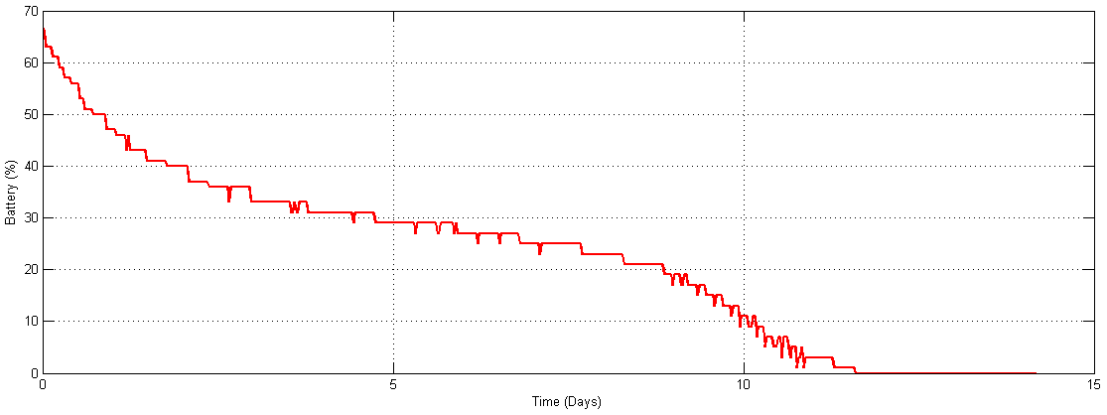


Figura A.1: Gráfico da percentagem da capacidade da bateria da janela em função dos dias.

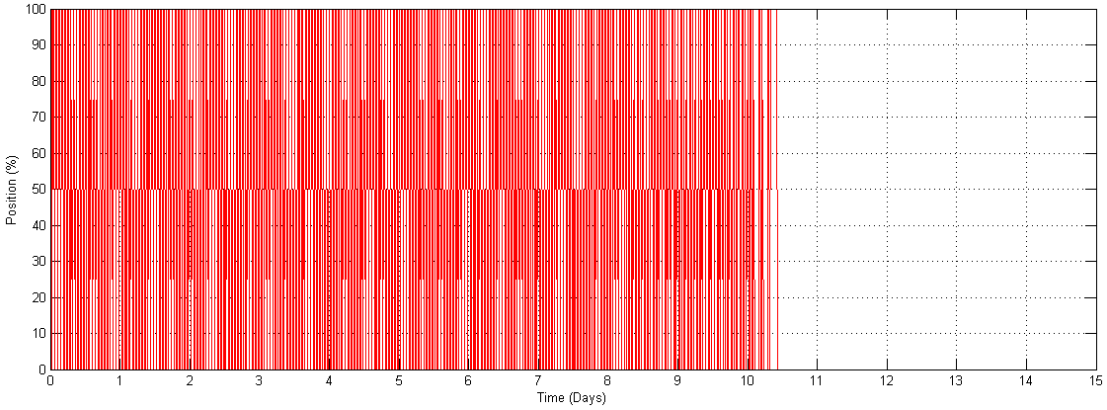


Figura A.2: Gráfico da posição percentual da persiana da janela em função dos dias.

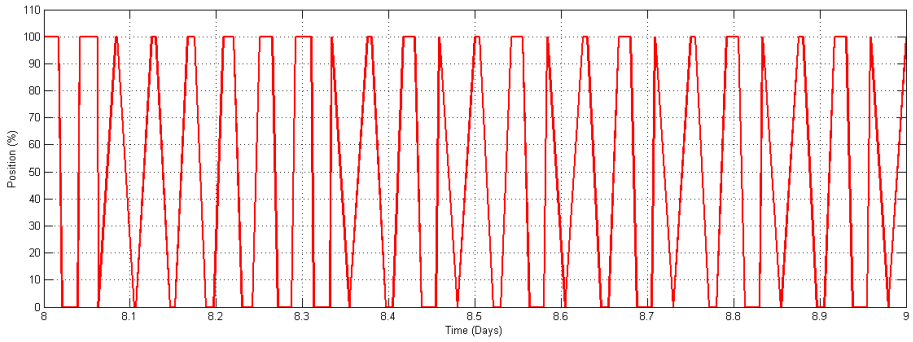


Figura A.3: Gráfico da posição percentual da persiana da janela em função dos dias, representativo de um só dia.

Referências bibliográficas

[1] N. Brito, "Climawin Final Report," vol. 19, no. 6 Muscle Disease, pp. 1495–6, Dec. 2012.

[2] Climawin, "Climawin | The Intelligent Ventilation Window." [Online]. Available: http://climawin.eu/

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," 2011.

[4] P. Rubens, "5 Reasons Why the Cloud Should Be More Secure Than Your Data Center," 2011. [Online]. Available: http://www.serverwatch.com/trends/article.php/3931976/5-Reasons-Why-the-Cloud-Should-Be-More-Secure-Than-Your-Data-Center.htm

[5] G. Benediktsson, "Lighting control - possibilities in cost and energy-efficient lighting control techniques," Ph.D. dissertation, 2009.

[6] E. Alliance, "EnOcean Dolphin - System Architecture."

[7] E. Gmbh, "EnOcean Radio Protocol 2," pp. 1–19, 2013.

[8] Kristen Hemmings, "3 Types of Cloud Service Models," 2012. [Online]. Available: http://blog.appcore.com/blog/bid/168247/3-Types-of-Cloud-Service-Models

[9] M. B. Mollah, K. R. Islam, and S. S. Islam, "Next generation of computing through cloud computing technology," *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–6, Apr. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6334973

[10] P. A. Bernstein, “Middleware An Architecture for Distributed System Services,” 1993.

[11] J. Manciocchi, “Global SaaS Market Growing-Driven by CRM, Mobile Deployments,” 2011. [Online]. Available: http://saasmarkets.com/report-global-saas-market-growing-driven-by-crm-mobile-deployments/

[12] C. Vouillon, “10 Key Facts About The Growing SaaS Market,” 2013. [Online]. Available: http://efounders.co/infographic-saas-is-beautiful-10-key-facts-about-the-growing-saas-market/

[13] M. Ali and S. K. Ravula, “REAL - TIME SUPPORT AND ENERGY EFFICIENCY IN,” Ph.D. dissertation, 2008.

[14] A. J. D. Rathnayaka and V. M. Potdar, “Evaluation of Wireless Home Automation Technologies,” vol. 5, no. June, pp. 76–81, 2011.

[15] J. Tammilehto, “Wireless building automation in Finland,” Ph.D. dissertation, 2013.

[16] Nest, “Nest.” [Online]. Available: https://nest.com/

[17] Revolv, “Revolv | The Universal Smart Home Automation Hub and App.” [Online]. Available: http://revolv.com/

[18] Enocean Alliance, “The wireless standard for sustainable buildings.”

[19] E. Gmbh, “Smart Acknowledge,” pp. 1–49, 2010.

[20] ——, “Remote Management 1 . 9,” pp. 1–43, 2013.

[21] E. Alliance, “EnOcean Serial Protocol 3 (ESP3),” vol. 3, 2013.

[22] ——, “Enocean Equipment Profiles,” 2011.

[23] E. Gmbh, “Pushbutton Transmitter Module,” no. November, p. 60669, 2012.

[24] G. Song, Y. Zhou, W. Zhang, and A. Song, “A multi-interface gateway architecture for home automation networks,” *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1110–1113, Aug. 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637595

[25] A. Maiti, “Home Automation as a Service,” vol. 2, no. 3, pp. 421–427, 2012.

[26] P. Moorhead, "The Problem With Home Automation's Internet Of Things (IoT) - Forbes," 2013. [Online]. Available: http://www.forbes.com/sites/patrickmoorhead/2013/09/26/the-problem-with-home-automations-iot/

[27] B. Eom, C. Lee, C. Yoon, H. Lee, and W. Ryu, "A Platform as a Service for Smart Home," *International Journal of Future Computer and Communication*, vol. 2, no. 3, pp. 253–257, 2013. [Online]. Available: http://www.ijfcc.org/index.php?m=content&c=index&a=show&catid=39&id=434

[28] P. M. B. Salunke, D. Sonar, N. Dengle, and S. Kangude, "Home Automation Using Cloud Computing and Mobile Devices ." vol. 3, no. 2, pp. 35–37, 2013.

[29] S. Gao, Y. Qian, F. Cao, and L. Wang, "The design of smart window control system based on GSM network," *2011 International Conference on Multimedia Technology*, pp. 1297–1299, Jul. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6002782

[30] R. Kumar and A. Kumar, "Design and hardware development of power window control mechanism using microcontroller," pp. 361–365.

[31] P. P. Popat, "Dual-Mode window covering system," 1997.

[32] J. Tian, M. Gao, and J. Li, "Intelligent Window Based on Embedded System," *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, pp. 395–399, Jul. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4287716

[33] G. S. Solutions, "Ultra Low Power Carbon Dioxide Sensor," pp. 4–7.

[34] Saft Systems, "Super High Energy Series VH Cs 3200," pp. 0–1.

[35] E. Gmbh, "Scavenger Transceiver Module STM 300 / STM 300C / STM 300U," 2013.

[36] ——, "Universal switching receiver," p. 250.

[37] Atmel, "ATTIny 1634 Datasheet," 2013.