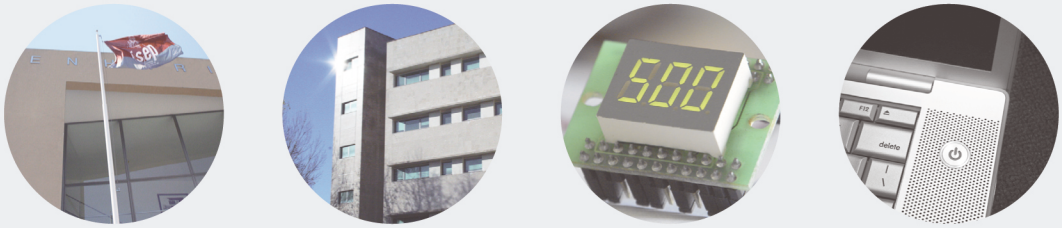
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Controlador de Domótica Z-Wave

MANUEL LUÍS MENESES MONTENEGRO VISEU SANTOS
Julho de 2019

POLITÉCNICO
DO PORTO

Controlador de Domótica Z-Wave

Manuel Luís Meneses Montenegro Viseu Santos

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2019

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Manuel Luís Meneses Montenegro Viseu Santos, Nº 1141133, 1141133@isep.ipp.pt
Orientação científica: Nuno Filipe da Fonseca Bastos Gomes, nbg@isep.ipp.pt
Empresa: Central Casa Supervisão: Carlos Silva, carlos.silva@centralcasa.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2019

Resumo

A domótica é o termo usado para referir à automatização de um edifício. Um sistema deste tipo pode controlar iluminação, climatização, sistemas de segurança, entre outros. A domótica tem como objetivo aumentar o conforto e segurança, assim como, melhorar a eficiência energética das casas. Na área da domótica existe uma vasta gama de tecnologias disponíveis, contudo, o Z-Wave destaca-se graças ao seu desempenho e à interoperabilidade entre produtos de diferentes fabricantes. Este, é um protocolo de comunicação por radiofrequência dedicado à transmissão de mensagens de controlo. Dada a, cada vez maior, relevância do protocolo Z-Wave na domótica, este projeto foi realizado com o intuito de estudar este protocolo, perceber as suas capacidades e o quão acessível, em termos de disponibilidade e custos, é desenvolver sistemas com este protocolo.

O projeto realizado consiste no desenvolvimento de um controlador de domótica Z-Wave que, para além de conseguir atuar, configurar e recolher informações de dispositivos que usem este protocolo, incorpore sensores de forma a medir determinadas grandezas e que seja capaz de controlar estas grandezas atuando dispositivos Z-Wave. O sistema desenvolvido tem como elemento central uma placa de desenvolvimento, que será responsável por gerir a rede Z-Wave, obter e efetuar o controlo PI das grandezas medidas pelos sensores e onde irá estar alojada uma página *web* que servirá de interface gráfica. À placa de desenvolvimento será acoplada uma antena Z-Wave que adicionará ao sistema a capacidade de comunicar através deste protocolo.

Para validar o correto funcionamento do sistema foi utilizado um analisador de rede Z-Wave, que permite visualizar as mensagens transmitidas na rede com a qual este está emparelhado. Com este, foi possível observar as mensagens enviadas pela rede aquando da atuação de um dado dispositivo através da interface gráfica, assim como, os *reports* enviados pelos dispositivos Z-Wave para o controlador. Para além disso, foram realizadas várias experiências para validar os controladores PI implementados. Estas, consistiram na variação dos ganhos proporcionais e integrais, de forma a ver a influência de cada um deles e comparando com o que seria de esperar teoricamente.

Palavras-Chave: Domótica, Automação, Z-Wave, Protocolo, Controlador.

Abstract

Home automation is a term used to refer to the automatization of a building. A system of this kind can control lightning, temperature, security systems, among others. Home automation has the objective of increasing confort and security, as well as, improve home's energy efficiency. In the home automation field there's a vast range of technologies available, however, Z-Wave stands out thanks to its performance and interoperabilty between products of different manufacturers. Z-Wave is a communication protocol, that uses radiofrequency, dedicated to the transmission of control messages. Given the growing relevance of the Z-Wave protocol in home automation, this project was done with the intention of studying this protocol and understand its capabilities and how accessable, in terms of availability and costs, it is to develop systems with this protocol.

This project consists in the development of a home automation Z-Wave controller that, besides being able to actuate, configure, and get informations about devices that use this protocol, it incorporates sensors to measure given physiscal quantities and that is able do control these, actuanting Z-Wave devices. The system developed has as central element a development board, that will be responsible for managing the Z-Wave network, get and do the PI control of physical quantities measured by the sensors and where it will be hosted a webpage that will work as graphical interface. To the development board it will be connected a Z-Wave antenna that will add to the system the ability to communicate through this protocol.

To validate the system it was used a Z-Wave network analyser, that allows to see the messages in the network with wich the analyser is paired. With this, it was possible to see the messages sent through the network when a given device is actuated through the webpage, as well as, the reports sent by the Z-Wave devices to the controller. Beside that, it was made various tests to validate the PI controllers implemented. These tests, consisted in varying the proportional and integral gain, to see the effect of each one and compare it with what it was expectable theoretically.

Keywords: Home, Automation, Z-Wave, Protocol, Controller.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

ADC *Analog-to-Digital Converter*

AJAX *Asynchronous Javascript*

API *Application Programming Interface*

CIT *Certified Installer Toolkit*

CSS *Cascading Style Sheets*

EEPROM *Electrically-Erasable Programmable Read-Only Memory*

EOF End of Frame

GPIO *General Purpose Input/Output*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

IEEE *Institute of Electrical and Electronics Engineers*

IoT Internet of Things

IP Internet Protocol

I2C *Inter-Integrated Circuit*

JSON *JavaScript Object Notation*

LDR *Light Dependent Resistor*

LED *Light Emitting Diode*

MAC *Medium Access Control*

MISO *Master In / Slave Out*

MOSI *Master Out / Slave In*

NDA *Non-Disclosure Agreement*

NIF *Node Information Frame*

OEM *Original Equipment Manufacturers*

PAN *Private Area Network*

PCB *Printed Circuit Board*

PHP *Hypertext Preprocessor*

PI Proporcional e Integral

PID *Proporcional, Integral e Derivativo*

RAM *Random Access Memory*

RF radiofrequência

RFC *Request For Comments*

SoC *system on chip*

SOF *Start of Frame*

SPI *Serial Periphal Interface*

SS *Slave Select*

SSH *Secure Shell*

SVG *Scalable Vector Graphics*

TCP/IP *Transmission Control Protocol/Internet Protocol*

USB *Universal Serial Bus*

WLAN *Wireless Local Area Network*

XML Extensible Markup Language

6LoWPAN IPv6 *over Low power Wireless Personal Area Network*

Capítulo 1

Introdução

Neste capítulo é feita uma introdução ao projeto, realizando um breve enquadramento sobre a domótica e de seguida são identificados os objetivos do projeto. Por fim, é apresentada a calendarização do mesmo e a organização da dissertação.

1.1 Contextualização

O presente relatório descreve todo o trabalho realizado no âmbito da unidade curricular de Tese/Dissertação do 2º ano do Mestrado em Engenharia Eletrotécnica e Computadores - Automação e Sistemas, do Departamento de Engenharia Eletrotécnica, do Instituto Superior de Engenharia do Porto. O projeto foi realizado durante um estágio curricular na empresa Central Casa, especializada em domótica.

A domótica é o termo usado para referir à automatização de um edifício. Um sistema deste tipo pode controlar iluminação, climatização, sistemas de segurança, entre outros. A domótica tem como objetivo aumentar o conforto e segurança, assim como, melhorar a eficiência energética das casas. Dentro da domótica existem várias tecnologias, no entanto, atualmente, o Z-Wave destaca-se graças ao seu desempenho e à interoperabilidade entre produtos de diferentes fabricantes. Este, é um protocolo de comunicação por radiofrequência dedicado à transmissão de mensagens de controlo. Dada a, cada vez maior, relevância do protocolo Z-Wave na domótica, este projeto foi realizado com o intuito de estudar este protocolo, perceber as suas capacidades e o quão acessível, em termos de disponibilidade e custos, é desenvolver sistemas com este protocolo.

1.2 Objetivos

O objetivo deste projeto centra-se na criação de um controlador de domótica com capacidade de comunicação Z-Wave, que incorpore sensores cujas grandezas

medidas são controladas por dispositivos Z-Wave. Para além disso, deverá dispor de uma interface gráfica que apresente todas as informações acerca do sistema, assim como, permita atuar os dispositivos Z-Wave. O sistema deverá ser desenvolvido recorrendo a elementos, componentes e *software* de fácil acesso e de baixo custo. Os objetivos deste projeto podem ser sumarizados nos seguintes pontos:

- Estudo acerca da domótica e do protocolo Z-Wave;
- Análise de alternativas de implementação da comunicação Z-Wave;
- Análise de alternativas de implementação de interface gráfica;
- Desenvolvimento do *software* para comunicação Z-Wave;
- Desenvolvimento do *software* para leitura dos sensores incorporados;
- Desenvolvimento do *software* para controlo das grandezas medidas pelos sensores;
- Desenvolvimento da interface gráfica;

1.3 Calendarização

O projeto foi desenvolvido de forma contínua com uma duração de 19 semanas, em que o tempo de cada uma das etapas deste projeto está descrita na Figura 1.1. Este projeto começou com uma fase de investigação, principalmente acerca do protocolo Z-Wave, passando-se por uma análise de requisitos e de seguida ao desenvolvimento em concreto, terminando com a elaboração do relatório.

	Março				Abril				Maio					Junho				Julho	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Estudo do protocolo Z-Wave	■	■	■																
Análise de requisitos		■																	
Estudo/testes inciais com OpenZWave				■															
Desenvolvimento *software* para com. Z-Wave					■	■	■	■	■										
Desenvolvimento *software* para leitura dos sensores										■									
Desenvolvimento *software* de controlo										■	■								
Desenvolvimento da interface gráfica						■	■	■	■	■	■								
Elaboração do relatório												■	■	■	■	■	■	■	■

Figura 1.1: Calendarização do projeto

1.4 Organização da Dissertação

No capítulo 1, é feita uma introdução ao projeto, destacando-se o porquê de se realizar este. é realizado um breve enquadramento acerca da domótica e de seguida são identificados os objetivos do projeto. Por fim, é apresentada a calendarização do mesmo.

No capítulo 2, é abordada a domótica e, principalmente, o protocolo Z-Wave incluindo a sua estrutura, como é realizada a comunicação através deste protocolo e algumas vulnerabilidades encontradas em sistemas de domótica Z-Wave. Por último, são apresentados alguns controladores de domótica Z-Wave comercializados, assim como, alternativas *open source*.

No capítulo seguinte, 3, é apresentada a arquitetura do sistema a desenvolver, descrevendo cada uma das partes que o constitui. Estas, serão descritas em termos da sua funcionalidade, dos componentes necessários e o porquê da sua escolha.

No quarto capítulo, é descrita a implementação dos vários módulos apresentados no capítulo anterior, de forma a obter um sistema funcional.

No capítulo 5, são apresentados os testes, e resultados obtidos destes, para validar o correto funcionamento do sistema quer a nível de comunicação Z-Wave, quer a nível de controlo das grandezas medidas pelos sensores incorporados.

No último capítulo, 6, são apresentadas as conclusões do projeto e são idealizadas possíveis melhorias no trabalho concretizado.

Capítulo 2

Estado da Arte

Neste capítulo é abordada a domótica, com foco especial nas tecnologias existentes nesta. Além disso, é apresentada a estrutura do protocolo Z-Wave, aspetos relativos à segurança deste e como desenvolver produtos que usem este protocolo. Por fim, são apresentados alguns controladores de domótica Z-Wave comercializados, assim como, alternativas *open source*.

2.1 Domótica

Domótica é o termo usado para referir à automatização de várias atividades domésticas. Um sistema de domótica pode controlar a iluminação, a climatização, eletrodomésticos, sistemas de segurança, como por exemplo, alarme, videovigilância, deteção de fumo ou de inundação, entre outros. A automatização destes sistemas têm como objetivo aumentar o conforto, a segurança, a eficiência energética, ou até mesmo melhorar a qualidade de vida das pessoas, incluindo pessoas idosas ou com problemas motores [9] [10].

Por norma, o controlo é efetuado de forma centralizado num equipamento, que envia ordens a outros equipamentos para que estes realizem uma dada ação. Um sistema de domótica pode ser conetado à Internet através de um computador, denominado de *gateway*, para permitir controlar de forma remota todos os equipamentos da habitação [9]. No momento atual, em que a Internet se tornou uma parte integral da vida diária, praticamente todos os sistemas de domótica têm a capacidade de se ligar à Internet, tornando-se assim num componente da *Internet of Things* (IoT) [11] [12].

A escolha do sistema de domótica a implementar numa casa depende muito desta. Numa casa em construção, a escolha do sistema é menos restritivo, podendo esta ser projetada de forma a acomodar a tecnologia pretendida, como por exemplo, para colocar cablagem específica para o sistema de domótica. Em casa já construídas são normalmente usados sistemas *wireless*, que usam radiofre-

quência (RF), sinais infravermelhos, ou que usam a instalação elétrica existente (*powerline*) para comunicarem entre si [9].

O termo casa inteligente é usado muitas vezes para descrever casas automatizadas, e em [10] são propostos três gerações de casas inteligentes. Na primeira, é monitorizada a atividade da casa, e dos seus ocupantes, e permite controlar os dispositivos nesta. Este é o modelo mencionado anteriormente, em que dispositivos comunicam entre si de forma a realizar as ações necessárias, havendo um dispositivo capaz de interligar o sistema com a Internet. A segunda geração de casas inteligentes introduz inteligência artificial no processo de automatização. Contudo, tecnologia deste tipo existe maioritariamente em laboratório. Um exemplo disso é o iSpace, na Universidade de Essex, onde o sistema aprende os comportamentos dos utilizadores e dá sugestões de acordo com as necessidades destes. A nível mais comercial, o Amazon Echo é visto como um primeiro nível nesta segunda geração de casas inteligentes. Este produto apenas aceita instruções do utilizador, através da voz, para controlar os dispositivos da casa, assim como, responder a perguntas realizadas pelo utilizador. Para além disso, tem a capacidade de aprender à medida que interage com o utilizador. A terceira geração de casas inteligentes faz uso de robôs equipados com inteligência artificial que interagem com os utilizadores, através da voz, e que se deslocam livremente pela casa. Estes, não são vistos apenas como robôs que controlam os dispositivos na casa, mas também como um elemento de "companhia" para os humanos na forma como interagem com estes [10].

No entanto, independentemente da geração de casa inteligente e da tecnologia usada, o principal objetivo da domótica é de melhorar a qualidade de vida dos utilizadores [10].

2.1.1 Internet of Things

A *Internet of Things* é um termo introduzido em 1999 por Kevin Ashton, que tem crescido bastante nos últimos anos. Este termo refere-se a dispositivos conetados através da Internet, sendo que por dispositivos entendem-se sensores, atuadores e recursos computacionais em Cloud [13]. Desta forma, o conceito da IoT é criar uma conexão virtual, entre uma rede e dispositivos eletrónicos, que permite controlar e monitorizar estes dispositivos. Aplicações IoT têm-se tornado cada vez mais populares sobretudo devido à evolução da Internet e dos *smartphones* [11]. É esperado que a IoT venha a ter grande importância na vida diária e atualmente é usado para implementar cidades inteligentes, carros inteligentes e casas inteligentes [13].

Contudo, atualmente o mercado IoT, incluindo a domótica, está bastante fragmentado em termos de plataformas, assim como, em termos de protocolos, não existindo um *standard*. Isto, torna a escolha de uma plataforma ou protocolo

difícil, não só para o consumidor final, assim como para os desenvolvedores que também têm de escolher para os seus projetos. E, embora existam várias propostas para solucionar este problema ainda nenhuma destas se destacou [14] [15] [16] [13].

2.1.2 Tecnologias de Domótica

Atualmente existem várias tecnologias dedicados à domótica, como por exemplo, o X10, o KNX, o Z-Wave e o Zigbee [17]. Para além destas, mais recentemente foi publicado um novo protocolo dedicado à domótica denominado de Thread [13]. Neste subcapítulo é apresentada uma breve análise destas tecnologias.

X10

O X10 foi o primeiro protocolo de comunicação dedicado à domótica. Foi lançado em 1975 e continua disponível no mercado apesar da concorrência de novos protocolos. Uma das vantagens do X10 é que tanto pode funcionar através de *powerline* como através de comunicação rádio. Contudo, uma das desvantagens é que a transmissão de mensagens ocorrem à vez, o que significa que mensagens transmitidas em simultâneo podem levar a problemas na descodificação destas e, assim, perder estas. Para além disso, a comunicação não é bidirecional o que significa que não é feito o *acknowledgedment* das mensagens. A taxa de transmissão destas é relativamente baixa, variando entre os 20 e os 200 bits por segundo. Os dispositivos de radiofrequência X10 têm um alcance de cerca de 30 metros. Tem um custo baixo e existem muitos dispositivos disponíveis [17].

KNX

A tecnologia KNX foi apresentada em 1999 com a fundação da Associação KNX [18]. Esta tecnologia é a única norma global para controlo de residências e edifícios, sendo aprovada como Norma Europeia, Internacional, Chinesa e EUA [19]. O KNX é um sistema de barramento digital que dispõe de vários meios de comunicação: Par Entrançado (TP), Radiofrequência, Linha de Potência (*powerline*) e Internet Protocol (IP) (Ethernet). O par entrançado permite uma taxa de transmissão de 9600 bits/s enquanto que usando *powerline* é possível comunicar com uma taxa de 1200 bits/s. Os dispositivos que utilizam radiofrequência transmitem os sinais na banda de frequência de 868 MHz, com uma taxa de transmissão de 16 384 kbps. Para além disso, as mensagens KNX podem ser transmitidas encapsuladas em telegramas IP, utilizando a rede LAN e a Internet como meio de comunicação [20]. A grande vantagem do KNX é que todos os produtos no mercado passam por um processo de certificação, de forma a garantir a compatibilidade e interoperabilidade entre sistemas independentemente do fabricante [19].

Z-Wave

O Z-Wave é um protocolo de comunicação, cuja transmissão realiza-se por radiofrequência, lançado em 2001 pela Zensys. Este é um protocolo *mesh*, o que significa que cada dispositivo na rede funciona como repetidor, aumentando o alcance das mensagens a transmitir. Contudo, existe um número limitado de saltos que cada mensagem pode efetuar, sendo que o Z-Wave definiu esse limite como sendo quatro. Embora o Z-Wave tenha um alcance de 100 metros, em campo aberto, de dispositivo para dispositivo, numa instalação real, tendo em conta as paredes e outros obstáculos, o alcance é de aproximadamente 200 metros, já com os quatro saltos de retransmissão [17] [21]. A frequência de trabalho depende da região, sendo que na Europa é de 868,42 MHz e tem uma taxa de transmissão é de 40 kbps. A comunicação é bidirecional e os comandos podem ser confirmados (*acknowledged*). A principal vantagem do Z-Wave é a compatibilidade entre dispositivos de diferentes marcas [17].

Zigbee

O Zigbee é um *standard* Institute of Electrical and Electronics Engineers (IEEE) 802.15.4, usado em domótica, que foi publicado em 2004 pela Zigbee Alliance, cuja transmissão é realizada através de radiofrequência. Tal como o Z-Wave, forma uma rede *mesh*, em que cada mensagem pode realizar até seis saltos e, sendo que cada dispositivo tem um alcance de cerca de 10 metros, este protocolo tem um alcance máximo de cerca de 60 metros. A taxa de transmissão pode ser até 250 kbps, contudo, a versão europeia do Zigbee tem uma taxa de apenas 20 kbps. A comunicação neste protocolo é bidirecional e as mensagens enviadas podem ser confirmadas (*acknowledged*). Este é um protocolo que apresenta como sua principal vantagem o baixo consumo energético por parte dos dispositivos. Por outro lado, alguns problemas de incompatibilidade entre dispositivos de diferentes marcas tem evitado o seu crescimento a nível de mercado [17].

Thread

Tal como mencionado anteriormente, o protocolo Thread é bastante recente, tendo sido publicado em 2015 pela Thread Group, Inc. Este grupo foi fundado por sete empresas entre as quais, a ARM, a Nest Labs (Google), a Samsung e a Silicon Labs. Também é uma rede *mesh* capaz de realizar até 36 saltos, sendo que cada salto tem um alcance de 30 m, e cada rede pode ter mais de 250 dispositivos. A taxa de transmissão deste protocolo é de 250 kbps. Atualmente, ainda não existem dispositivos Thread disponíveis para consumidores, embora alguns dispositivos Zigbee e o termóstato Nest sejam compatíveis com Thread [13].

O Thread é um protocolo que usa IPv6 *over Low power Wireless Personal Area Network* (6LoWPAN) e é baseado num *standard* existente, IEEE 802.15.4,

e em vários *standards* propostos (*Request For Comments* - RFC) [13]. Assim, em termos de avanços tecnológicos não propõe nada de novo mas, em termos de standardização pode ser um passo importante nesta direção. O Thread procura ganhar momento para fazer com que todos os outros sistemas proprietários se unam num único protocolo. Tendo isto em conta, Thread e Zigbee anunciaram que a versão Zigbee 3.0 vai suportar Thread, com a esperança de que este tipo de consolidação continue até um único protocolo IoT [13].

Apesar disto, atualmente, na área da domótica o Z-Wave destaca-se. Este, é o protocolo mais usado e amplamente reconhecido, oferecendo boa fiabilidade e estabilidade de rede [17].

2.2 Z-Wave

O Z-Wave foi desenvolvido pela Zensys, uma empresa privada sediada nos EUA e na Dinamarca, e trata-se de um protocolo de comunicação, cuja transmissão se faz por radiofrequência, usado em domótica. A Zensys começou por introduzir no mercado, em 2001, um sistema de controlo de iluminação que evoluiu até um protocolo de rede *mesh* residencial, implementado num *system on chip* (SoC) proprietário. O Z-Wave tornou-se rapidamente num protocolo de domótica popular, principalmente devido ao facto de oferecer aproximadamente as mesmas funcionalidades que o Zigbee mas com menos problemas de interoperabilidade e uma banda de frequência não licenciada (868 MHz na Europa). Esta banda, é vista como menos problemática do que a banda de 2,4 GHz, já muito sobrecarregada. Em 2008, a Zensys foi adquirida pela SIGMA Designs, que por sua vez vendeu a tecnologia Z-Wave à Silicon Labs, em 2018. Esta, é quem atualmente fornece todo o serviço técnico para projetar, desenvolver e certificar produtos Z-Wave. Todos os fabricantes, *Original Equipment Manufacturers* (OEMs), de produtos Z-Wave pertencem à Z-Wave Alliance, que promove o protocolo [22] [23].

Este protocolo, é *half duplex* de baixa largura de banda, cujo principal objetivo é a transmissão de mensagens de controlo pequenas, de forma fiável, de uma unidade de controlo para um, ou mais, nós na rede e com custos baixos. O protocolo especifica quatro camadas (Figura 2.1): a camada *medium access control* (MAC) que controla o meio RF, a camada de transferência que controla a transmissão e receção de tramas, a camada de *routing* que controla o *routing* de tramas na rede, e a camada de aplicação que interpreta os dados recebidos e transmitidos pelas tramas [1].

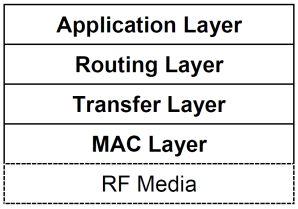


Figura 2.1: Camadas do protocolo Z-Wave [1]

2.2.1 Tipos de Nós Z-Wave

Uma rede Z-Wave consiste em dois tipos de nós, os controladores e os *slaves*. Os primeiros, são capazes de calcular rotas e são estes que iniciam e enviam comandos de controlo para os outros nós. Os *slaves* respondem e executam estes comandos e também são capazes de reencaminhar os comandos para outros nós, formando assim uma rede *mesh*. Uma rede Z-Wave pode ter até 232 nós, controladores ou *slaves* [1] [24]. Sendo recomendado que haja um dispositivo a cada 10 metros, embora, tal como já foi referido, o Z-Wave tenha um alcance de 100 metros em condições ideais [21].

Controladores

Os controladores mantêm um mapa completo da topologia da rede e, por isso, podem calcular rotas para alcançar qualquer nó na rede, assim como, também podem enviar estas rotas aos *slaves*. O controlador usado para incluir o primeiro nó é configurado para atuar como controlador primário. Este, é capaz de incluir e excluir nós da rede, gere a alocação dos IDs do nós (*node* IDs) e recolhe informação acerca de quais nós cada um consegue alcançar de forma direta. Qualquer controlador pode ser primário mas apenas pode haver um em cada rede. Quaisquer outros controladores que sejam adicionados à rede são denominados de controladores secundários. Estes, obtêm a informação acerca da rede a partir do controlador primário [22] [24]. Nos controladores podem ser identificadas algumas variantes destes:

- Controlador Portátil - tal como o nome indica, é portátil e, por isso, alimentado a baterias. Tem a possibilidade de descobrir a sua posição na rede quando precisa de comunicar com outros nós. Dado o facto de este dispositivo poder mover-se na rede, é usualmente usado para incluir, ou excluir, nós [24].

- Controlador Estático - é fixo numa localização, alimentado pela rede elétrica e está sempre a escutar as mensagens na rede. Isto permite que outros nós transmitam tramas para este quando necessário [24]. Neste tipo controlador podem ser identificadas outras duas variantes:

  - Controlador de *Update* Estático (SUC - *Static Update Controller*): Quando existe um controlador deste tipo na rede, o controlador primário envia automaticamente as atualizações da rede para este, por exemplo, quando um novo nó é adicionado na rede, este irá aparecer automaticamente no mapa da topologia do SUC. Outros controladores podem pedir individualmente informações sobre a rede ao SUC. Este, é capaz de criar um novo controlador primário caso o original desapareça ou caso tenha um funcionamento defeituoso. Só pode haver um SUC por cada rede Z-Wave [24].
  - SUC ID *Server* (SIS): Quando um SUC é configurado como um servidor de IDs de nós faz com que todos os outros controladores incluam e excluam nós. Quando criado, o SIS torna-se automaticamente o controlador primário na rede e só pode existir um SIS em cada rede Z-Wave [24].

- Controlador Ponte - é um controlador estático, que tem a capacidade de emular dispositivos de redes de outro tipo, como KNX ou X10, como nós virtuais na rede Z-Wave. Permitindo assim, por exemplo, controlar dispositivos KNX a partir de um controlador Z-Wave e vice-versa [24].

Slaves

Os nós *slaves* não contêm tabelas de *routing* e podem atuar como repetidores desde que esteja à escuta. Para tal, tem de estar num estado ativo, o que quer dizer que tem de ser alimentado pela rede elétrica, visto que os dispositivos alimentados a baterias passam grande parte do tempo num estado de suspensão, serão raras as vezes que estes funcionarão como repetidores. Desta forma, estes dispositivos são ignorados pelo controlador quando calculando rotas [24]. Existem vários tipos de *slaves*:

- *Slave* - este tipo de nós recebe comandos e executa uma ação de acordo com o comando recebido. Não enviam informação para outros *slaves* ou controladores a não ser que seja pedido num comando. Este tipo de nó consegue receber tramas e replicá-las se necessário [24].

- *Routing Slave* - tem a mesma funcionalidade que o tipo de *slave* anterior, mas com a diferença de que consegue enviar mensagens para outros nós mesmo que não tenha sido solicitado. Este armazena rotas para chegar a outros nós, denominadas de "*Return Routes*". O *routing slave* é normalmente usado em dispositivos que precisam de reportar alarmes [24].

- *Frequently Listening Routing Slave* (FLiRS) - este é um caso especial de um *routing slave* alimentado a baterias, que é configurado para periodicamente

escutar por um sinal de *wakeup*. Isto permite que outros nós acordem este e enviem-lhe uma mensagem [24].

- *Enhanced Slave* - trata-se de um *routing slave* com uma EEPROM para armazenar dados da aplicação.
- *Enhanced Slave* 232 - este tem as mesmas funcionalidades que um *enhanced slave* e também consegue guardar *return routes* até 231 destinatários [24].

2.2.2 Home ID e ID do Nó

O *home* ID é um identificador de 32 *bits*, que é programado em todos os controladores, usado para separar as redes Z-Wave umas das outras. Os *slaves*, inicialmente têm o *home* ID igual a zero e, quando é incluido numa rede, o controlador irá atribuir-lhe um *home* ID igual ao da rede. O ID do nó é um valor de 8 *bits* que identifica um nó numa dada rede. Este ID é único dentro de cada rede Z-Wave [24].

2.2.3 Protocolo

Camada MAC

A camada MAC controla o meio RF e é independente deste, da frequência e do método de modulação. Esta camada necessita de acesso à trama de dados quando recebida ou ao sinal inteiro de forma binária. A informação é codificada segundo a codificação de Manchester e consiste num preâmbulo, um delimitador de início de trama (SOF - *start of frame*), a trama de dados e um delimitador de fim de trama (EOF - *end of frame*). Esta informação é transmitida em tramas de 8 *bits*. O protocolo Z-Wave segue as especificações ITU-T G.9959 para as camadas física e MAC [1] [22].

A camada MAC inclui um mecanismo de prevenção de colisão (*collision avoidance*) para evitar que nós transmitam quando outros já estão a transmitir. Este mecanismo consiste em verificar o meio antes de transmitir e no caso de estar ocupado, a transmissão é adiada por um número aleatório de milissegundos [1] [22].

Camada de Transferência

A camada de transferência do protocolo Z-Wave controla o envio e receção de dados entre dois nós. Isto incluí retransmissão, verificação de *checksum* e *acknowledgements* (ACK). Esta camada contém quatro tipos de tramas, contudo, todos seguem o formato ilustrado na Figura 2.2.

- Trama *Singlecast* - são enviadas para um nó em específico e podem incluir uma rota até ao destino. Também contém uma *flag* que indica se se pretende

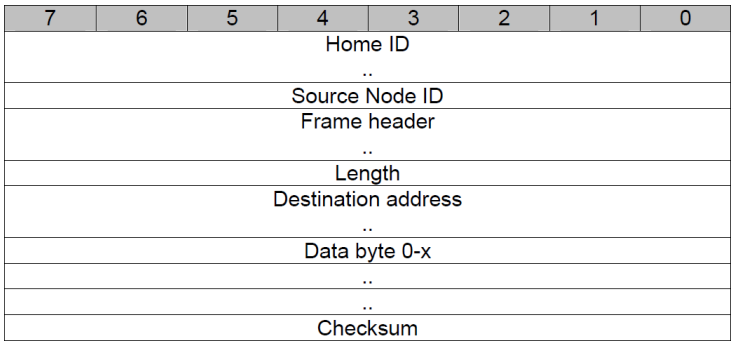


Figura 2.2: Formato da trama Z-Wave [1]

acknowledgment. Estes, são tramas *singlecast* sem dados e geradas em cada salto da rede *mesh*. Caso não seja recebido um ACK, a trama é retransmitida [1] [22].

- Trama *Multicast* - é enviada de um nó para um grupo de nós que pode variar de 1 até 232 dispositivos. É usada para endereçar vários nós sem ter que enviar uma trama por cada nó. Este tipo de trama não suporta *acknowledgement* e, por isso, não deve ser usado caso se pretenda comunicação fiável [1] [22].

- Trama *Broadcast* - é direcionada a todos os nós na rede Z-Wave com o *home* ID especificado e não é confirmada a sua receção (*acknowledged*). É usada, por exemplo, para transmitir a trama de informação do nó (NIF - *node information frame*) [1] [22].

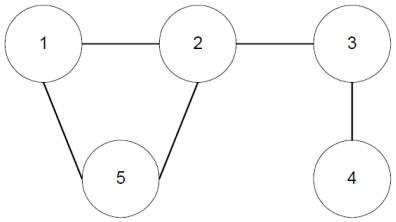
- Trama *Explorer* - é um caso especial da trama de *broadcast* em que todos os nós, no alcance direto do nó que gerou a trama, recebem uma trama *explorer*. O tratamento desta trama depende do endereço de destino (um único nó ou todos) e de algumas *flags* de configurações [1] [22].

Camada de *Routing*

Esta camada controla o *routing* dos pacotes na rede e tanto controladores como *slaves* podem participar no *routing* dos pacotes. As redes Z-Wave usam um mecanismo de *source-routing*, o que significa que é o criador do pacote que gera uma rota completa até ao destino final, passando por vários nós caso o remetente e o destinatário não estejam ao alcance direto um do outro. A rota consiste numa sequência de IDs que é colocada na trama. Esta pode ser denominada de trama *routed singlecast*, quando se destina a apenas um nó, com *acknowledgement*, e que contém informação de repetição de pacote. Um nó que recebe este pacote repete-o retirando o seu ID da rota até ao nó destino. Outro tipo de trama é a trama *routed acknowledge* que tem a mesma estrutura que a anterior mas sem

quaisquer dados, para confirmar ao controlador que a trama *routed singlecast* chegou ao destinatário [1] [22].

Para ajudar nesta tarefa de *routing*, o controlador guarda a informação acerca da topologia da rede Z-Wave numa tabela denominada de tabela de *routing*. Esta assinala que nós estão ao alcance direto dos outros. A Figura 2.3 mostra, à esquerda, o mapa de uma rede e, à direita, a respetiva tabela de *routing*.



	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	0	0
5	1	1	0	0	0	0
6	0	0	0	0	0	0

Figura 2.3: Mapa de uma rede Z-Wave (à esquerda) e respetiva tabela de *routing* (à direita) [1]

Camada de Aplicação

A camada de aplicação tem como função analisar os dados dos pacotes, descodificar e executar comandos numa rede Z-Wave [1]. Na Figura 2.4 é apresentado o formato da trama da camada de aplicação.

7	6	5	4	3	2	1	0
Single/Multi/Broadcast frame header							
Application command class							
Application command							
Command parameter 1							
Command parameter 2							
...							
Command parameter x							

Figura 2.4: Formato da trama da camada de aplicação [1]

Para garantir a interoperabilidade entre as várias aplicações, o protocolo Z-Wave define uma estrutura de comandos *standard* [22]. Cada comando encapsulado numa trama Z-Wave é composto por:

- Identificador da *command class* - este identificador especifica a que *command class* um dado comando pertence, sendo que existem *command classes* definidas para o protocolo Z-Wave e para as aplicações Z-Wave. Este identificador trata-se de um número que vai de 0x00 até 0x1F, no caso das reservadas para o protocolo, enquanto que os números de 0x20 até 0xFF identificam as reservadas à aplicação [1] [22]. Um exemplo de uma *command class* de aplicação é a COMMAND_CLASS_SWITCH_BINARY identificada pelo número 0x25 [8].

- Identificador de comando - especifica a ação, dentro da *command class* a executar [1]. Por exemplo, a *command class* dada como exemplo anteriormente, terá como um dos comandos possíveis o SWITCH_BINARY_SET [8].

- Lista de parâmetros do comando - esta lista contém os parâmetros associados com um determinado comando. O número de parâmetros depende do comando e podem ser acrescentados mais com novas versões, sendo que dispositivos com versões anteriores ignoram estes novos parâmetros [1] [22]. Por exemplo, o comando dado como exemplo anteriormente terá como um dos parâmetros o Value, que guardará um valor a atribuir ao dispositivo [8].

Em [8] é exposto em detalhe toda a informação das *command classes* e respetivos comandos e parâmetros.

Os dispositivos indicam as *command classes* que suportam durante o processo de inclusão através do *Node Information Frame* (NIF). Este, é enviado como *broadcast* quando o botão de ação deste é pressionado ou como resposta a um controlador que tenha requerido tal [1] [22].

Alguns dispositivos implementam mais do que uma funcionalidade, isto é, sensores para várias grandezas físicas, múltiplos *switches*, entre outros. O que implica que enviar apenas um comando para um nó pode não ser suficiente para realizar a ação desejada. Para resolver este problema, o Z-Wave utiliza um comando denominado de *multi-instance command encapsulation* que acrescenta um parâmetro que indica a instância [22].

O Z-Wave dispõe de uma funcionalidade. à qual dá o nome de associações, que permite que dispositivos interajam diretamente sem necessitarem da ajuda do controlador. Isto melhora a responsividade do sistema e elimina a possibilidade de haver uma falha no controlador. Z-Wave define uma associação como sendo a operação pela qual um dispositivo é configurado para controlar um dispositivo ou um conjunto destes. Esta funcionalidade é alcançada através de algo denominado de grupo. Cada dispositivo controlador deverá guardar localmente um ID de grupo por cada evento que consegue gerar, ao qual irá associar os IDs dos nós que deverão ser controlados aquando da geração do evento. As associações seguem um modelo de "*publish/subscribe*" e são tratadas ao nível da aplicação do código do dispositivo [22].

Outra funcionalidade disponível são os cenários Z-Wave, que permitem configurar vários dispositivos de forma a possam ser atuados simultaneamente. Para além disso, evitam efeitos que podem acontecer quando, por exemplo, atuando vários dispositivos ao mesmo tempo que irão ligar luzes, e que deveriam ligar

todas ao mesmo tempo, estas ligarão uma de cada vez. Os cenários são normalmente usados para predefinir situações comuns, como por exemplo, regular a iluminação de uma divisão da melhor forma para ver televisão, jantar, etc [22].

Num controlador podem ser definidos vários cenários. Cada um destes define uma lista de IDs de nós que pertencem ao cenário e um parâmetro de 8 *bits* a atribuir ao nó. Depois de configurado, um cenário pode ser ativado através do comando SCENE_ACTIVATION_SET, que pode ser *multicast*, e especificando o ID do cenário [22].

2.2.4 Segurança do Z-Wave

Tal como em qualquer outro protocolo de comunicação, a segurança do Z-Wave é uma preocupação e alvo de várias análises, de como sistemas que usem estes protocolos podem ser comprometidos. Um sistema deste tipo pode ser comprometido de três formas diferentes. Uma delas é realizando o ataque à *Cloud* onde estão os dados e onde é realizado o processamento destes. Outra alternativa é realizando o ataque à *gateway* que liga o sistema à Internet. E, por último, o ataque pode ser realizado à *Private Area Network* (PAN). PAN é a rede local *wireless* que consiste em todos os sensores, atuadores e dispositivos que comunicam e uma *gateway*, neste caso, a rede Z-Wave [25].

O Z-Wave desde do início que disponibiliza encriptação, contudo, tem dado aos fabricantes a opção de escolher usar ou não. Como nem todos os fabricantes usavam encriptação, houve várias investigações a mostrar esta vulnerabilidade [25]. E, mesmo quando implementada, esta nem sempre era implementada da melhor forma. Em [26] é apresentado o caso de uma fechadura de porta, um dos dispositivos em que a segurança é mais crítica, que é comprometida devido ao facto de a implementação do protocolo de troca de chave de encriptação não ser o melhor. Neste, foi detetada uma vulnerabilidade que permitia a um dispositivo atacante redefinir a chave de encriptação da fechadura para um valor conhecido. Com isto, é possível enviar qualquer comando para a fechadura, como por exemplo, abri-la ou mudar o PIN de desbloqueio.

Contudo, a forma mais comum de uma rede Z-Wave ser comprometida é através da sua *gateway*. Esta, é facilmente acedida através da *Wireless Local Area Network* (WLAN) a que está ligada, tendo em conta que, por norma, estas redes estão configuradas de forma pouco segura e, assim, a dificuldade de penetrar nestas redes é relativamente baixa. Com acesso à rede e determinando o IP da *gateway*, obtém-se acesso à interface gráfica do controlador, sem qualquer tipo de autenticação, sendo possível atuar todos os dispositivos. Em [27] é demonstrada esta vulnerabilidade nos controladores VeraEdge, RaZberry Pi e Almond+.

Para além disso, a *gateway* em si apresenta várias vulnerabilidades que ficam disponíveis a quem tiver acesso a esta. Todas as *gateways* usam pedidos *Hyper-*

text Transfer Protocol (HTTP) do tipo `POST` e `GET` para enviar comandos para o seu servidor, que irá retornar informação ao *chip* Z-Wave que irá transmitir estes comandos. Estes pedidos HTTP são passíveis de serem capturados, modificados e enviados para o servidor sendo aceites de forma legítima, como se fossem provenientes da interface gráfica. Permitindo assim atuar todos os dispositivos na rede, incluindo dispositivos que usem encriptação. Isto permite que, por exemplo, capturando pedidos para atuar um dispositivo de iluminação, estes podem ser modificados para atuar uma eletroválvula. Outra vulnerabilidade das *gateways* tira partido do ficheiro de *backup* que é facilmente descarregado. No caso do VeraEdge, neste é guardada informação sobre o sistema e *passwords*, incluindo a da WLAN e a para acesso *Secure Shell* (SSH). No caso do RaZberry Pi este ficheiro apenas contém informação acerca dos dispositivos Z-Wave [27].

Ainda em [27] é apresentado como criar uma conexão permanente a todos os dispositivos Z-Wave mais discreta, e em que apenas é necessário acesso à WLAN aquando da introdução de um dispositivo intruso. Esta vulnerabilidade tira partido do facto de alguns controladores permitirem colocar o controlador em modo de inclusão através de *software*, sem ser necessário pressionar nenhum botão físico. O dispositivo intruso trata-se de um controlador RF que é incluído na rede mas que, como explicado em [27] é possível fazer com que não apareça na interface gráfica. Este controlador intruso, após entrar na rede, consegue comunicar diretamente com todos os dispositivos Z-Wave e, ao contrário dos exemplos anteriores, passa despercebido sem ser detetado pela *gateway*. Para além disso, caso a *gateway* perca conexão à Internet ou até mesmo caso perca energia, o dispositivo intruso continua a ter acesso à rede Z-Wave e a poder controlar os dispositivos nesta [27].

Em 2017, a Z-Wave Alliance anunciou um novo *framework* de segurança, denominado S2. Este é obrigatório para todos os dispositivos de forma a passar o processo de certificação. As medidas de segurança fornecidas por este *framework* abrangem os dispositivos Z-Wave, os controladores e as *gateways* [25] [28]. O *framework* de segurança S2 é baseado em encriptação AES-128 para troca de dados e ECDH para troca da chave. Este *framework* resolve vulnerabilidades a ataques do tipo *man-in-the-middle*, reforçou a comunicação IP e eliminou o risco dos dispositivos serem *hackeados* enquanto estão incluídos na rede através de um código no próprio dispositivo [28].

O S2 divide a rede em três classes de segurança: S2 *Access Control*, S2 *Authenticated* and S2 *Unauthenticated*. Dispositivos que sejam adicionados na rede devem pertencer a uma destas classes. Cada uma destas tem a sua chave, o que significa que existem três chaves AES-128 por rede. A classe *Access Control* é a mais segura de todas e é usada em dispositivos como fechaduras. A segunda classe mais segura é a *Authenticated* e a menos segura a *Unauthenticated*. Um

dispositivo pode pertencer a mais do que uma classe mas, nesse caso, deverá usar a chave apropriada para o dispositivo com o qual vai comunicar [25].

Este novo *framework* também introduziu encriptação ECDH usada para trocar a chave AES-128 de forma segura. Para um dispositivo e o controlador puderem trocar a chave da rede, deve ser estabelecida uma conexão segura. Contudo, como ainda não foi trocada a chave de encriptação, tal não pode ser alcançado. Assim, o Z-Wave usa uma técnica denominada de ECDH que permite a troca de uma chave temporária para estabelecer uma comunicação segura. Com esta, a chave AES-128 pode ser trocada de forma segura. De realçar que, esta chave, nos dispositivos da série 500 e seguintes, é guardada na memória *flash* que é apagada quando um novo *firmware* é carregado, o que evita que a chave seja lida [25].

Este *framework* veio tornar as redes Z-Wave mais seguras, não através do protocolo em si, mas através da implementação efetuada pelos fabricantes de produtos Z-Wave. Sendo que não são conhecidas falhas de segurança no protocolo em si [29]. E, apesar de os desafios em termos de segurança em sistemas IoT sejam cada vez mais, o Z-Wave tem mostrado capacidade em mudar e adaptar-se a estes desafios de segurança [25].

2.2.5 Desenvolver Produtos Z-Wave

O desenvolvimento de dispositivos Z-Wave tem-se tornado mais acessível, quer a nível de custos quer a nível de acessibilidade. Até 2016, para ter acesso ao *hardware*, *software* e especificações do Z-Wave, era necessário comprar um *System Development Kit* (SDK) que, dependendo da versão, custava entre $1500 e $3550. E, para além disso, para ter acesso a toda a documentação, exemplos de códigos de *software* e outras ferramentas, era necessário assinar um *Non-Disclosure Agreement* (NDA) em que o fabricante compromete-se a manter toda a informação como confidencial [30].

Em 2016, a Sigma Designs lançou publicamente as especificações da camada de aplicação do Z-Wave. Isto permitiu que qualquer pessoa tivesse um conhecimento mais aprofundado do protocolo mais facilmente. Contudo, continua a ser necessário ser membro da Z-Wave Alliance para poder certificar os produtos, mas qualquer um pode usar o SDK e a documentação para desenvolver aplicações ou serviços que interajam com dispositivos Z-Wave [31] [32]. Atualmente, a Silicon Labs é a responsável por disponibilizar todas as ferramentas necessárias, sendo que o *kit* de iniciação para a série 700 custa $379. Este incluí todo o *software* e documentação de apoio ao desenvolvimento e, em termos de *hardware*, destaca-se a placa principal que permite programar e fazer *debug* do módulo rádio Z-Wave (ZGM130S), também incluído no *kit* [33]. A placa principal disponibilizada pode ser visualizada na Figura 2.5.

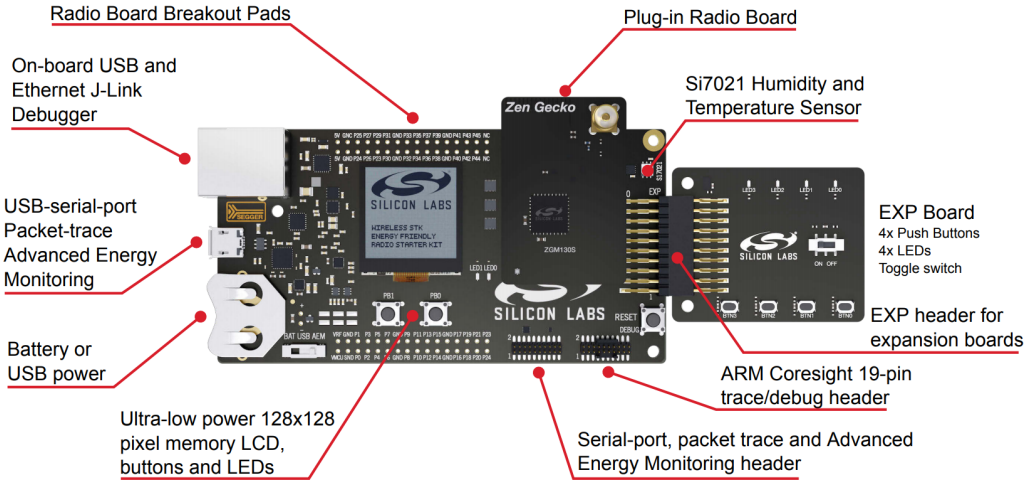


Figura 2.5: Placa principal do *kit* de iniciação série 700 (UG373) [2]

O módulo Z-Wave (ZGM130S) incluí um microcontrolador de 32 *bits* ARM Cortex-M4, possui 32 *General Purpose Input/Output* (GPIO), conversor analógico-digital (ADC - Analog-to-Digital Converter), comunicação série, *Serial Periphal Interface* (SPI), interrupções externas, entre muitas outras funcionalidades [3]. O diagrama de blocos do módulo pode ser visualizado na Figura 2.6.

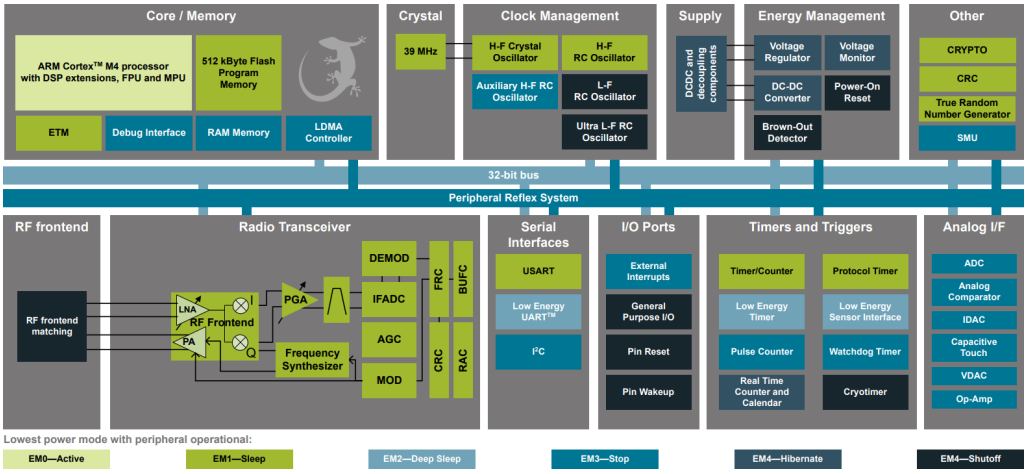


Figura 2.6: Diagrama de blocos do módulo Z-Wave ZGM130S [3]

De realçar que, como o Z-Wave usa diferentes frequências em diferentes regiões do mundo, é fabricado *hardware* específico para cada região.

Tal como já foi referido, qualquer produto Z-Wave de forma a poder ser comercializado tem de passar por um processo de certificação. Esta garante aos consumidores de que todos os produtos Z-Wave funcionam uns com os outros independentemente da marca e da versão. Esta interoperabilidade é, desde há muito, um dos grandes objetivos e um ponto diferenciador do Z-Wave. O processo

de certificação inclui testes técnicos, programas de uniformidade e aplicação dos padrões de certificação [34].

Mais recentemente, foi apresentado o programa de certificação Z-Wave Plus, desenvolvido para os consumidores identificarem os produtos que tiram proveito de quinta geração de *hardware* Z-Wave, conhecida como a série 500. Esta nova geração de *hardware* traz novas vantagens e funcionalidades entre as quais [35]:

- Menor consumo de energia em dispositivos alimentados a baterias, aumentando a duração desta em 50%;
- Aumento do alcance em 67%;
- Aumento da largura de banda em 250%;
- Mais três canais RF para melhorar a imunidade a ruído;
- Nova funcionalidade de inclusão *Plug-n-Play*;

Atualmente, este programa de certificação Z-Wave Plus é mantido em paralelo com a sua nova versão: Z-Wave Plus v2. Este introduz novas especificações para as *command classes* e é o programa aplicado para dispositivos que usem a série 700 do *chip* Z-Wave, enquanto que à série 500 é aplicada o programa Z-Wave Plus [36].

2.3 Controladores de Domótica Residencial Z-Wave

O protocolo Z-Wave tem evoluído ao longo dos anos, e os produtos associados a este protocolo acompanham esta evolução, incluindo os controladores. Este subcapítulo apresenta várias opções de escolha de controladores Z-Wave disponíveis, sendo que estes são divididos em dois tipos: os controladores comercializados e os controladores *open-source*.

2.3.1 Controladores Z-Wave Comercializados

Por controladores comercializados compreende-se controladores que estão à venda e que cuja compra inclui todo o *hardware* e *software* necessários para o funcionamento do controlador. Neste sector a oferta é vasta, contudo, alguns fabricantes destacam-se como a Vera, a Fibaro, a Zipato e também a Samsung.

Vera

A Vera é umas das marcas mais conhecidas na área da domótica, e atualmente dispõe de três controladores diferentes, o VeraEdge, o VeraPlus e o VeraSecure que custam $89,95, $119,95 e $269,95. Estes diferem essencialmente nos protocolos

que incluem, sendo que todos suportam Z-Wave Plus e Wi-Fi, a versão Plus e Secure suportam também Zigbee e Bluetooth. A versão Secure para além destes, suporta RF, a 345 e 433 MHz, e 3G, assim como também pode funcionar como central de alarmes [4]. A versão mais básica, VeraEdge, é capaz de suportar até 220 dispositivos numa rede e todas as versões permitem o controlo remoto através de um navagador de Internet ou através da aplicação para *smartphone* [37].

Uma das grandes vantagens deste dispositivo é que o controlo é realizado no próprio controlador em vez de uma Cloud, o que significa que funcionará à mesma caso não haja Internet [38]. Para além disso, permite a realização de cenários mais complexos do que o SmartThings, se necessário através de *scripts* em LUA. Também dispõe de vários *plugins* que permitem adicionar funcionalidades, como por exemplo, obter informação meteorológica detalhada através de *Application Programming Interfaces* (APIs) externas. Contudo, a qualidade do *software* apresenta-se como uma desvantagem com problemas de estabilidade ao apresentar bastantes erros [38] [37]. Na Figura 2.7 são apresentados o VeraEdge, à esquerda, e o VeraPlus, à direita.

Figura 2.7: VeraEdge (à esquerda) e VeraPlus (à direita) [4]

SmartThings

Ao contrário do Vera, o SmartThings apenas apresenta uma versão com um custo de $69,99. Esta, para além de Z-Wave, também suporta Zigbee e Wi-Fi, colocando-se ao nível de um VeraPlus em termos de protocolos mas a preço bastante inferior [5]. Um dos pontos positivos deste controlador é o facto de ser compatível com o Amazon Echo e o Google Home, permitindo assim controlar os dispositivos através da voz. Para além disso, o aspeto apelativo quer do controlador em si quer da aplicação também são pontos atrativos, assim como, o facto de poder operar a bateria. Contudo, apresenta algumas desvantagens a começar por apenas permitir controlar o sistema através da aplicação para dispositivos móveis, não sendo possível controlar, ou configurar nenhum aspeto, através de um computador. Para além disso, este é um sistema baseado em Cloud, o que significa que é necessária uma ligação à Internet para comunicar com esta, assim como com a aplicação. Em caso de falha de Internet, o sistema fica inoperável [38]. Embora seja um sistema mais simples que um Vera, e sem uma grande diferença

de preço, o facto de ser fabricado por uma marca conhecida, dá-lhe uma grande projeção. Na Figura 2.8 é apresentado o *hub* SmartThings da terceira geração.

Figura 2.8: SmartThings [5]

Estes são apenas alguns exemplos de controladores Z-Wave dos muitos que existem no mercado, sendo o Vera um exemplo de um controlador já bastante consolidado no mercado da domótica, enquanto que o SmartThings é um controlador mais simples.

2.3.2 Controladores Z-Wave Open Source

Por controladores Z-Wave *open-source* compreende-se *software open-source* que está disponível de forma gratuita, e acessível para qualquer pessoa montar o seu controlador tendo adquirido o *hardware* necessário. Sendo que o *hardware* necessário trata-se de um computador e uma antena Z-Wave, que normalmente estão disponíveis sobre o formato de uma *pen Universal Serial Bus* (USB). Neste tipo de *software* destacam-se o Home Assistant, o OpenHAB e o Domoticz.

Home Assistant

O Home Assistant foi lançado em 2013 e tem-se tornado no *software open source* para domótica mais popular. Este corre em Python 3 e normalmente é instalado num Raspberry Pi para formar um sistema de baixo custo. Permite que qualquer pessoa desenvolva *scripts* em Phyton para adicionar mais funcionalidades ao sistema. A partilha destes *scripts* faz com que as funcionalidades suportadas pelo Home Assistant seja vasta, suportando vários protocolos de comunicação, assistentes de voz, integração com outras plataformas de automação e muitos outros *plugins*. Os principais protocolos suportados pelo Home Assistant são o Z-Wave, Zigbee, Wi-Fi, Ethernet, USB e IFTTT, contudo, permite instalação de *plugins third-party* que alarga a gama protocolos. Para além disso, permite a integração com a Alexa da Amazon e o Google Assistant, com APIs externas, como por exemplo para obter a meteorologia, e até mesmo integração com

Arduinos ou ESP8266 para criação de dispositivos personalizados. As funcionalidades Z-Wave do Home Assistant são implementadas através de uma biblioteca *open source*, desenvolvida em C++, chamada OpenZWave [39] [40].

OpenHAB

O OpenHAB é outra solução bastante popular na área da domótica, tendo sido lançado em 2010. É baseado em Java e suporta vários dispositivos e protocolos e, tal como o Home Assistant, permite ao utilizador personalizar e criar funcionalidades. Em termos de protocolos, o OpenHAB suporta Z-Wave, Bluetooth, HTTP, Insteon, MQTT, RFXCOM, entre outros. Para além disso, é possível a instalação de *plugins* que permitem integrar o OpenHAB com IFTTT, Alexa da Amazon e várias APIs. Apesar de ser bastante parecido com o Home Assistant em termos de funcionalidade, é visto como não sendo tão fácil de usar como este [39] [40].

Este tipo de solução dá ao utilizador uma maior versatilidade e, por vezes, com um menor custo em relação aos controladores comercializados. Contudo, apresenta uma desvantagem, comum a todos os *software*, que é de requerem que o utilizador tenha algum conhecimento, quer a nível de computadores, quer a nível dos protocolos utilizados, para uma melhor integração.

Capítulo 3

Projeto

Ao longo deste capítulo será apresentada a arquitetura do sistema a desenvolver, descrevendo cada uma das partes que o constitui. Estas, serão descritas em termos da sua funcionalidade, dos componentes necessários e o porquê da sua escolha.

3.1 Arquitetura

De forma a poder especificar a arquitetura do sistema e todos os elementos que constituem este, é necessário definir primeiro o objetivo do sistema e quais as suas funcionalidades. E, tal como já foi referido anteriormente, este deve fazer uso de placas de desenvolvimento de forma a implementar um controlador de domótica residencial com capacidade de comunicação Z-Wave, de forma a interagir com outros dispositivos que implementem o mesmo tipo de protocolo. Para além disso, o controlador deve integrar sensores, conectados fisicamente à placa de desenvolvimento, e controlar as variáveis medidas por estes, atuando dispositivos Z-Wave. Por fim, o sistema deverá dispor de uma interface gráfica que apresente os valores medidos pelos sensores, informações acerca dos dispositivos Z-Wave e que permita a atuação destes. Os sensores que se definiu usar foram um de temperatura e um de luminosidade, realizando-se assim o controlo de um sistema térmico e de um sistema de luminosidade. Deste modo, o sistema poderá ser dividido nos seguintes módulos:

- Antena Z-Wave;
- Placa de desenvolvimento;
- Conversor analógico-digital;
- Sistema térmico;
- Sistema de luminosidade;

- Dispositivos Z-Wave;

A arquitetura do sistema é apresentada na Figura 3.1.

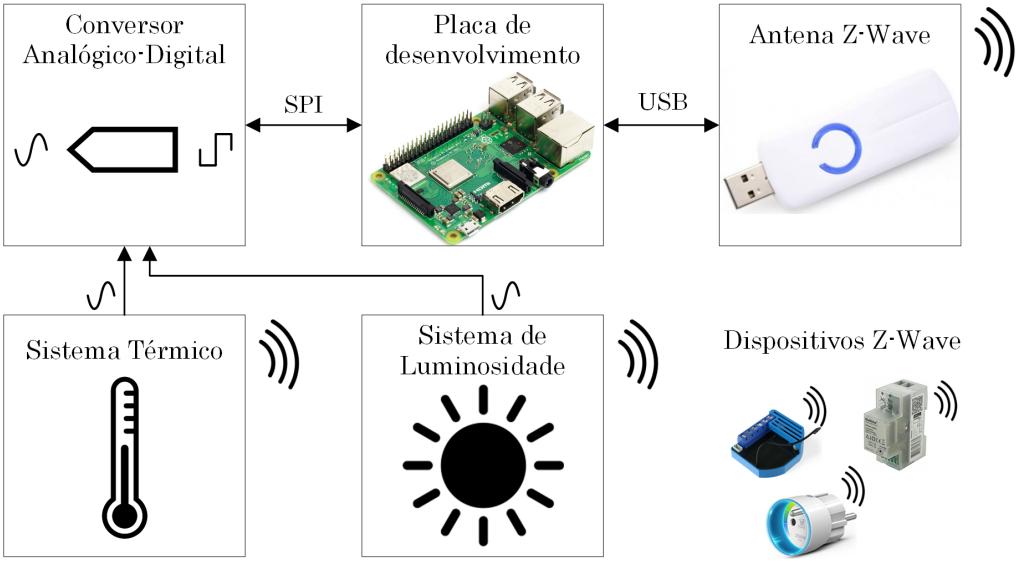


Figura 3.1: Arquitetura do sistema

3.2 Antena Z-Wave

A antena Z-Wave será o elemento que permitirá ao sistema comunicar através deste protocolo, neste caso, foi usada a Z-Stick Gen5 da AEOTEC, representada na Figura 3.2. Esta trata-se de uma *pen* USB que pode ser ligada a qualquer computador com sistema operativo Linux, Windows ou Mac [41].

Figura 3.2: AEOTEC Z-Stick Gen5

Esta possui um botão que permite ao utilizador colocá-la em modo de inclusão ou de exclusão, de forma, a incluir ou excluir dispositivos da rede Z-Wave. Esta, é uma funcionalidade útil numa fase inicial de implementação pois, permite adicionar e remover dispositivos sem que o *software*, que irá colocar a *pen* nestes

modos, esteja implementado, sendo assim possível testar outras funcionalidades. Contém também um botão na parte traseira que permite fazer o *reset* à *pen*, sendo que perde todas as configurações incluindo os dispositivos emparelhados.

Pode ser utilizada tanto com *software* profissional, como o Home Seer ou o Axial Control, assim como com *software open source*, como o Home Assistant ou o OpenHAB [41].

3.3 Placa de Desenvolvimento

Como placa de desenvolvimento, foi escolhido o Raspberry Pi, este é um computador de baixo custo e pequeno, muito utilizado para desenvolver todo o tipo de projetos, incluindo na área da IoT. Para além do seu baixo custo, outra característica importante na sua escolha para este projeto é o facto de possuir pinos GPIO com os quais será possível obter os valores medidos pelos sensores. Estes pinos também permitem comunicar através de vários protocolos como SPI, *Inter-Integrated Circuit* (I2C) e comunicação série [42]. Para além disso, é capaz de correr Linux, um dos sistemas operativos compatíveis com a *pen* USB de Z-Wave. Neste caso, foi usada a distribuição Linux Raspbian Stretch Lite, que é uma distribuição minimalista e sem qualquer tipo de interface gráfica, que foi desenvolvida especificamente para usar em Raspberry Pi.

Existem vários modelos disponíveis sendo que o utilizado foi o 3 B+ que possui um processador Cortex-A53 de 64 bits e uma frequência de 1,4 GHz, com uma arquitetura ARMv8. Tem 1 GB de *Random Access Memory* (RAM) disponível, *Bluethoot* 4.2, *Wi-fi* IEEE 802.11.b/g/n/ac de 2,4 GHz e 5 GHz e uma porta de Ethernet, sendo que, pelo menos uma destas duas últimas é um requisito obrigatório, para que o servidor com a página *web*, que servirá como interface, fique acessível [42]. Na Figura 3.3 é apresentado um Raspberry Pi 3 B+.

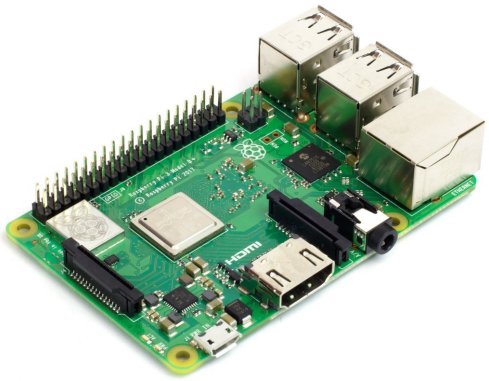


Figura 3.3: Raspberry Pi 3 B+

Este será o elemento central do sistema, onde se encontra todo o *software*

responsável por processar os dados, ou seja, este será o controlador. E, neste, podem ser identificados três elementos ao nível do *software* como representado na Figura 3.4.

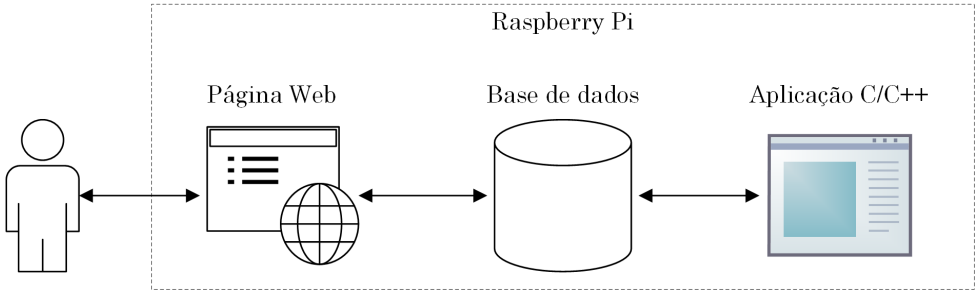


Figura 3.4: Diagrama do *software* constituinte do sistema

O primeiro elemento trata-se de uma aplicação, desenvolvida em C e C++, na qual é realizada a leitura dos sensores, controlo das variáveis medidas por estes e onde se comunica com os dispositivos Z-Wave, de forma a atuá-los e interpretar as mensagens recebidas destes. A comunicação Z-Wave é realizada recorrendo a uma biblioteca, em C++, *open source* chamada OpenZWave. Esta, foi criada com o intuito de facilitar a implementação deste protocolo a pessoas sem acesso ao SDK, e possui uma vasta lista de dispositivos compatíveis, incluindo a *pen* Z-Stick Gen5 da AEOTEC. Esta biblioteca tira partido da API de comunicação série desenvolvida pela Sigma Devices para os módulos Z-Wave [30]. O OpenZWave foi desenvolvida originalmente através de engenharia inversa, sendo que a partir de 2016, com o lançamento público das especificações do Z-Wave, tem sido atualizada para seguir esta especificações [43]. Para interagir com os pinos GPIO foi utilizada a biblioteca WiringPi, escrita em C, que permite efetuar leituras e escritas digitais nos pinos GPIO e utilizar os protocolos de comunicação disponíveis nestes.

A informação recolhida por esta aplicação será apresentada ao utilizador através de uma página *web* alojada no Raspberry Pi, num servidor Apache. Esta, irá trocar mensagens com a aplicação, em ambas as direções. Para efetuar esta comunicação existem várias possibilidades, como por exemplo, através de pedidos POST e GET, escrever as mensagens em ficheiros, *JavaScript Object Notation* (JSON) ou *Extensible Markup Language* (XML), a que ambas as partes têm acesso, escrever as informações numa base de dados a que ambas têm acesso, entre outras. A escolha recaiu sobre a última opção, visto que, também se pretende apresentar gráficos dos valores medidos pelos sensores ao longo do tempo e, para tal, é mais adequado uma base de dados que armazene os valores. Esta página *web* será desenvolvida recorrendo a três linguagens de programação diferentes, a primeira, *HyperText Markup Language* (HTML), será responsável pelo *front-end* da página *web*, ou seja, pela estrutura do *web*site. A segunda, *Hypertext Preprocessor* (PHP), é uma linguagem *server-side*, sendo assim, re-

sponsável pelo *back-end* da página e será usada para aceder à base de dados. A terceira linguagem utilizada será JavaScript, que juntamente com o HTML irá tratar do *front-end* da página, essencialmente executando *scripts* de PHP através de pedidos *Asynchronous Javascript* (AJAX). AJAX, é uma linguagem de programação, *client-side*, que permite que seja enviada e recebida informação para e de um servidor, em *background*, ou seja, sem que a página tenha que ser toda recarregada, melhorando a usabilidade da página *web*.

A base de dados, alojada no Raspberry Pi, irá ser modelada de forma a que tanto a aplicação como a página *web*, caso necessitem, tenham acesso a todos os dados do sistema, como por exemplo, o estado dos dispositivos Z-Wave, os valores reportados por estes, como a potência e a energia consumida, o nome do dispositivo, o tipo de dispositivo, os valores medidos pelos sensores, entre outros. Deste modo, foi escolhido um servidor MySQL para gerir a base de dados do sistema. O MySQL dispõe de uma API em C para que, aplicações desenvolvidas nesta linguagem, consigam interagir com a base de dados. A página *web* poderá aceder à base de dados através de PHP que já inclui funções para aceder a servidores MySQL.

Deste modo, a modelação da base de dados começou por definir os atores do sistema e quais os casos de uso. O sistema em causa irá ter dois atores: o utilizador e a aplicação C/C++. O primeiro, é aquele que interage com o sistema através da página *web*, e a aplicação é a que irá gerir informações sobre os dispositivos Z-Wave e os sensores de luminosidade e de temperatura e irá reagir a comandos do utilizador, enviados da página *web* através da base de dados. Na Tabela 3.1 são apresentados os casos de uso do sistema desenvolvido e a Figura 3.5 apresenta os mesmos casos de uso sobre a forma de um diagrama.

Tabela 3.1: Casos de uso

Ator	Casos de Uso
Utilizador	Obter informações
	Atuar dispositivo
	Adicionar/Remover dispositivo
	Configurar parâmetro e associação em dispositivo
	Criar cenário
	Editar cenário
Aplicação C/C++	Obter informações
	Atualizar valores reportados pelos dispositivos
	Adicionar/Remover dispositivo
	Obter cenários
	Atualizar dados sobre cenários

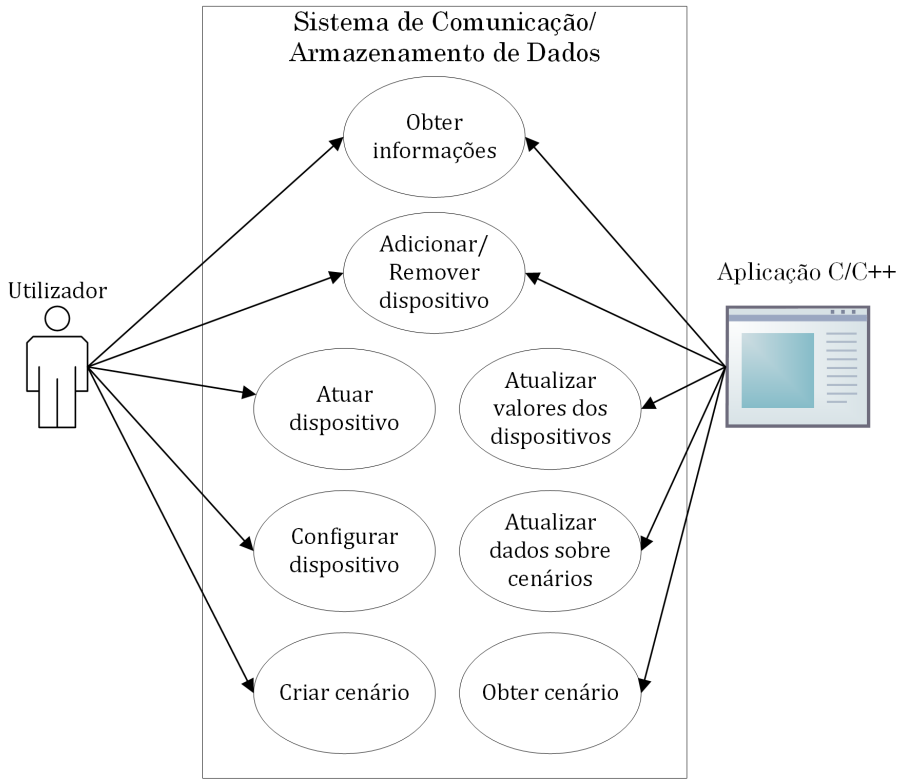


Figura 3.5: Diagrama casos de uso

3.4 Conversor Analógico-Digital

Dado que o Raspberry Pi não tem nenhum pino GPIO capaz de funcionar como entrada analógica, houve a necessidade de usar um ADC externo de forma a adquirir os sinais de saída dos sensores que, em ambos os casos, são analógicos. Deste modo, foi escolhido o circuito integrado MCP3008, que contém 8 canais de entrada de 10 bits de resolução cada, suficiente para o objetivo pretendido, e permite efetuar conversões diferenciais. O resultado da conversão é enviado usando o protocolo de comunicação SPI, que é um dos disponíveis no Raspberry Pi [6]. O circuito de ligação deste dois componentes pode ser visualizado na Figura 3.6.

Como é possível verificar o circuito integrado é alimentado com 3,3 V, e o mesmo valor é usado como tensão de referência para o ADC. Foi usado este valor por duas razões, sendo a primeira o facto de os pinos GPIO do Raspberry terem uma tensão de funcionamento de valor igual e, o uso de um valor superior obrigaria à colocação de um divisor de tensão na linha de envio de dados do MCP3008 para o Raspberry, de forma a reduzir o sinal para os 3,3 V. A segunda razão prende-se com o facto de permitir obter uma resolução melhor em relação a uma tensão de referência de 5 V. Esta resolução será de 3,223 mV dada pela

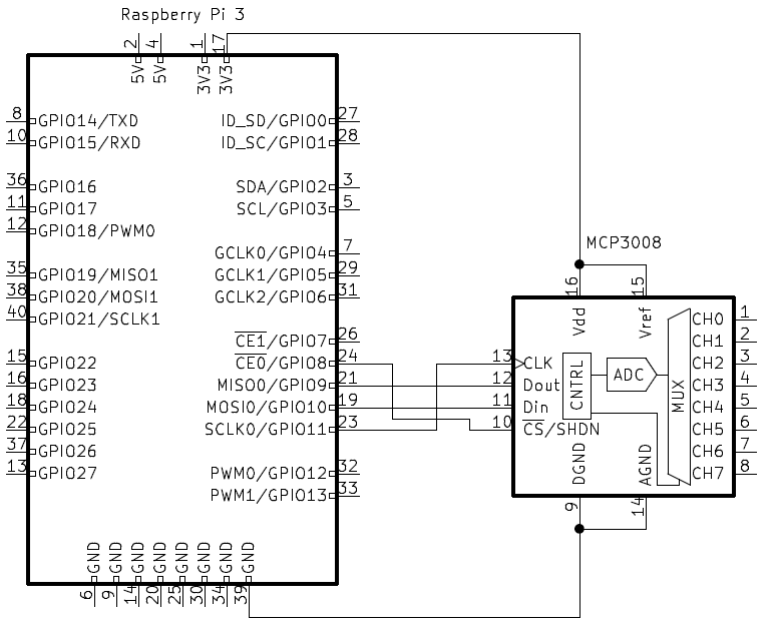


Figura 3.6: Circuito de ligação do MCP3008 com o Raspberry Pi

equação 3.1.

$$V_{res} = \frac{V_{ref}}{2^n} \tag{3.1}$$

Em que $Vres$ é a resolução em Volts, $Vref$ é a tensão de referência do ADC, neste caso 3,3 V, e n é a resolução deste em *bits*. Este ADC permitirá ter $2^{10} = 1024$ valores únicos.

As restantes ligações são referentes à comunicação SPI entre os dois dispositivos. SPI, é um protocolo de comunicação síncrono, ou seja, os dispositivos estão sincronizados através de um linha de relógio, separada da linha de dados, gerada por um destes. Esta linha, indica ao recetor quando este deve amostrar os *bits* na linha de dados. O dispositivo que gera o sinal da linha de relógio é denominado por *Master*, enquanto o outro dispositivo é denominado por *Slave*. A linha de dados, na verdade são duas linhas, uma para as mensagens enviadas do *Master* para o *Slave*, à qual se dá o nome de *Master Out / Slave In* (MOSI) e, a outra linha para as mensagens enviadas do *Slave* para o *Master*, denominada de *Master In / Slave Out* (MISO). Para além destas três linhas, existe ainda mais uma, normalmente denominada de *Slave Select* (SS), usada para "acordar" o *Slave*, para que este comece a enviar, ou receber dados. Também é usada para selecionar o *Slave* pretendido, no caso de haver mais do que um *Slave*. Desta forma, os pinos 23, 19, 21 e 24 do Raspberry vão conetar ao pinos 13, 11, 12 e 10

do MCP3008, respetivamente, formando assim as linhas de relógio, MOSI, MISO e SS, respetivamente [44].

Assim, a comunicação com o MCP3008 inicia quando a linha SS for colocada a nível lógico baixo. Após esta mudança, colocar a linha MOSI a um nível lógico alto, constituirá o *start bit* para o MCP3008. O *bit* seguinte (SGL/DIFF) irá determinar o tipo de conversão a realizar. Caso este *bit* seja 0, então será realizada uma conversão diferencial, caso contrário, será realizada uma conversão singular. Os três *bits* seguintes (D0, D1 e D2) são usados para selecionar o canal pretendido [6]. A Tabela 3.2 apresenta as várias configurações para estes quatro *bits* (SGL/DIFF, D0, D1 e D2).

Tabela 3.2: Configuração de *bits* para o MCP3008 [6]

Seleção de bits de controlo				Configuração	Selecção
Single/Diff	D2	D1	D0	da conversão	do canal
1	0	0	0	singular	CH0
1	0	0	1	singular	CH1
1	0	1	0	singular	CH2
1	0	1	1	singular	CH3
1	1	0	0	singular	CH4
1	1	0	1	singular	CH5
1	1	1	0	singular	CH6
1	1	1	1	singular	CH7
0	0	0	0	diferencial	CH0 = IN+ CH1 = IN-
0	0	0	1	diferencial	CH0 = IN- CH1 = IN+
0	0	1	0	diferencial	CH2 = IN+ CH3 = IN-
0	0	1	1	diferencial	CH2 = IN- CH3 = IN+
0	1	0	0	diferencial	CH4 = IN+ CH5 = IN-
0	1	0	1	diferencial	CH4 = IN- CH5 = IN+
0	1	1	0	diferencial	CH6 = IN+ CH7 = IN-
0	1	1	1	diferencial	CH6 = IN- CH7 = IN+

O MCP3008 começara a amostragem do sinal de entrada no quarto flanco ascendente do sinal de relógio após o *start bit*, e terminará no quinto flanco descendente do sinal de relógio após este mesmo *bit*. A mensagem enviada pelo módulo começará com um NULL *bit* seguido por outros 10 que são o resultado

da conversão. Sendo que estes são enviados do mais significativo para o menos [6]. A Figura 3.7 apresenta estes passos de comunicação entre um *Master* e o MCP3008.

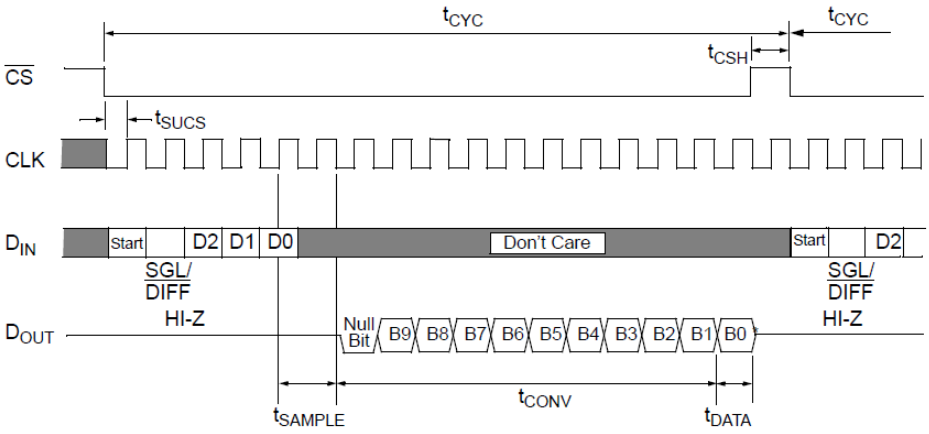


Figura 3.7: Comunicação com o MCP3008 [6]

3.5 Sistema Térmico

Este sistema terá três elementos como seus constituintes: um sensor de temperatura, um transístor e um módulo Z-Wave. A Figura 3.8 apresenta um diagrama com a constituição deste sistema.

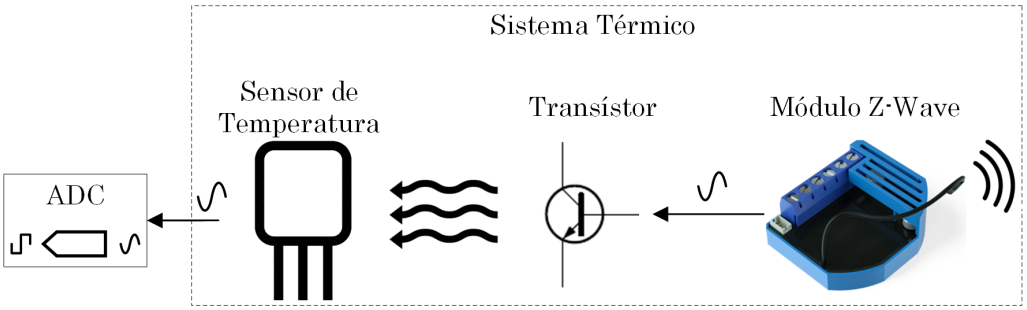


Figura 3.8: Constituição do sistema térmico

O sensor de temperatura, TMP36, permitirá medir esta grandeza em graus Celsius. Este tem um baixo custo, uma precisão razoável, um alcance de -40 °C até +125 °C, que é mais do que suficiente para a aplicação em causa, e não necessita de qualquer tipo de calibração. O circuito de implementação deste sensor está representado na Figura 3.9.

Onde Vcc é alimentação do sensor que será no valor de 3,3 V, fornecidos pelo Raspberry através de um dos seus pinos, 1 ou 17 (Figura 3.6), e $Vout$ é o sinal de saída cujo valor dependerá da temperatura. Este pino irá ser conetado ao

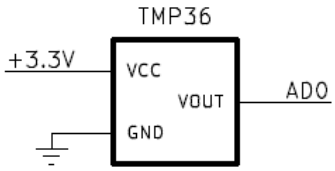


Figura 3.9: Circuito do sensor de temperatura

pino 1 do MCP3008 (Figura 3.6), referente ao AD0. O valor da temperatura em graus Celsius poderá ser calculado através da equação 3.2, baseada no *datasheet* do sensor, sendo que $Vout$ é em mV [45].

$$T(^{\circ}C) = \frac{V_{out} - 500}{10} \tag{3.2}$$

O segundo elemento constituinte do sistema térmico é um transístor que será usado para gerar calor, de forma a aumentar a temperatura do sistema. Neste caso, foi usado o transístor TIP31C, que tem uma potência de dissipação de 2 W, a uma temperatura de 25 °C [46]. Na Figura 3.10 é apresentado o esquema elétrico da implementação do transístor.

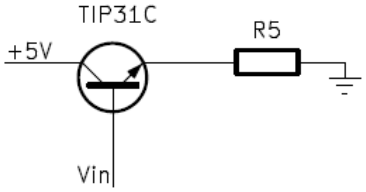


Figura 3.10: Circuito do transístor TIP31C

Onde a tensão no coletor, de 5 V, é fornecida pelo Raspberry através de um dos seus pinos, 2 ou 4 (Figura 3.6). A resistência $R5$, de 5,8 Ω, é utilizada para reduzir a corrente que flui no transístor, de forma a reduzir a velocidade com que este aquece, o que facilitará o controlo da temperatura. Controlo, este, que é realizado através do sinal Vin, aplicado na base, proveniente de um módulo Z-Wave regulável, mais concretamente o Qubino Flush Dimmer 0-10 V, apresentado na Figura 3.11.

Este módulo é alimentado com uma tensão contínua entre 12 e 24 V e é capaz de fornecer uma tensão contínua entre 0 e 10 V. Esta, será colocada na base do transístor, de forma a atuá-lo, com um valor dependente do calculado pelo controlador Proporcional e Integral (PI) (Raspberry Pi), no qual estará o algoritmo de controlo cujo resultado será enviado por Z-Wave para este módulo. Para além disso, este módulo tem uma entrada que permite a ligação de interruptores, ou potenciómetros, para controlar a tensão de saída mas, no caso deste projeto, não será usada.

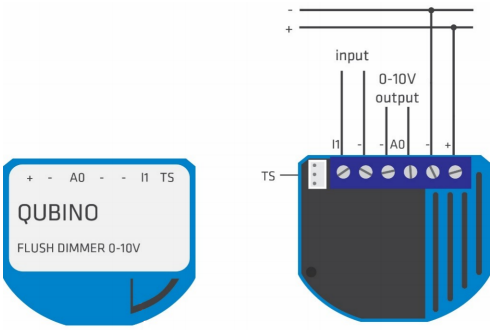


Figura 3.11: Qubino Flush Dimmer 0-10 V

3.5.1 Controlador PID

Um controlador Proporcional, Integral e Derivativo (PID) é a combinação de três ações de controlo, a ação proporcional, a integral e a derivativa, que se relacionam segundo a expressão 3.3. O controlador PID compara o valor medido y, com um valor de referência r (*setpoint*), e a diferença entre estes dois valores, erro, e, é usado para calcular um sinal de controlo, u, que irá ajustar o processo para o valor de referência [7].

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \tag{3.3}$$

Onde K o ganho proporcional, T_i o tempo integral e T_d o tempo derivativo, que têm de ser ajustados para cada sistema. A Figura 3.12 mostra o diagrama de blocos de um sistema de controlo com um controlador PID.

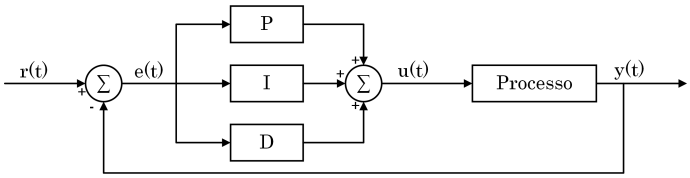


Figura 3.12: Diagrama de blocos de um sistema de controlo com um controlador PID

É de realçar que a equação de um controlador PID pode ser representada de forma diferente à apresentada na equação 3.3, sendo a mais comum a indicada na equação 3.4

$$u(t) = k_p e(t) + k_i\int_0^t e(\tau)d\tau + k_d\frac{de(t)}{dt} \tag{3.4}$$

Onde,

$$k_p = K \tag{3.5}$$

$$k_i = \frac{K}{T_i} \tag{3.6}$$

$$k_d = KT_d \tag{3.7}$$

Estes podem ser denominados de ganho proporcional k_p, ganho integral k_i e ganho derivativo k_d.

Ação Proporcional

Tal como o nome indica a ação proporcional reage de forma proporcional ao erro, e consiste na multiplicação da constante K pelo sinal do erro, $e(t)$. Um controlador proporcional é obtido se $T_i = \infty$ e $T_d = 0$. A Figura 3.13 mostra a resposta de um controlador deste tipo a um degrau unitário, para diferentes valores de K [7].

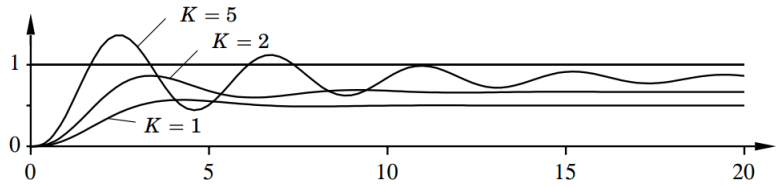


Figura 3.13: Resposta de um controlador proporcional para diferentes valores de K [7]

Constata-se que este tipo de controlador é incapaz de eliminar erros em regime permanente, contudo pode ser reduzido aumentado a constante K o que irá também assegurar reações mais rápidas, no entanto, este aumento provoca mais oscilações que irão aumentar o tempo de estabelecimento. O excessivo aumento de K pode fazer com que o sistema de torne instável [7].

Ação Integral

Esta ação pode ser interpretada como uma ação baseada no passado pois este termo é o somatório dos erros ao longo do tempo. Normalmente é usado em conjunto com o termo proporcional, dando origem a um controlador PI, caso k_d seja igual a zero. A ação integral contribui de forma proporcional ao erro e à duração do erro e permite eliminar o erro em regime permanente [7]. Tal é visível na Figura 3.14 que mostra a resposta de um controlador PI a um degrau unitário, para diferentes valores de T_i.

Para além disso, verifica-se que a influência da ação integral aumenta com a diminuição do tempo integral e vice-versa. Quanto menor for T_i mais rápida será

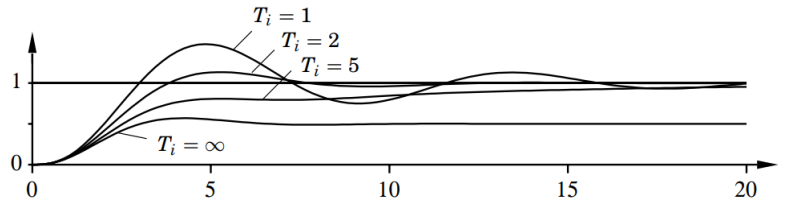


Figura 3.14: Resposta de um controlador PI para diferentes valores de T_i [7]

a reação de resposta, contudo aumentará o tempo de estabelecimento e provocará mais oscilações podendo tornar o sistema instável [7].

Ação Derivativa

A ação derivativa pode ser interpretada como uma ação baseada no futuro, esta reflete a taxa de variação do erro o que permite prever o comportamento do sistema. Assim, antes que o erro se torne demasiado elevado, a ação derivativa aplica uma correção ao sinal de controlo, com base na previsão, que irá permitir ao sistema responder mais rapidamente sem haver *overshoot* [7]. A Figura 3.15 mostra a resposta de um controlador PID a um degrau unitário para diferentes valores de T_d.

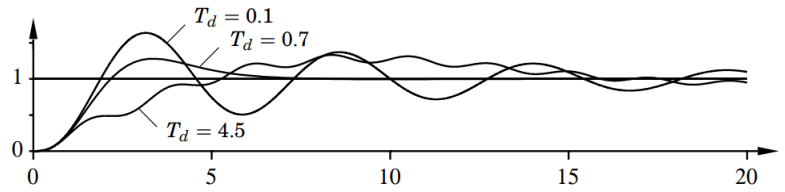


Figura 3.15: Resposta de um controlador PID para diferentes valores de T_d [7]

Tal como referido, para efetuar o controlo da temperatura, foi utilizado um controlador PI que é se trata de um controlador PID, cujo ganho derivativo é nulo. A determinação dos ganhos para o controlador foi realizada experimentalmente e será demonstrada posteriormente.

3.6 Sistema de Luminosidade

Tal como o sistema térmico, o sistema de luminosidade é constituído por 3 elementos: um *Light Dependent Resistor* (LDR), um *Light Emitting Diode* (LED) e um módulo Z-Wave igual ao utilizado no sistema térmico. A Figura 3.16 apresenta um diagrama com a constituição deste sistema.

O LDR será o sensor do sistema, este trata-se de uma resistência cuja resistividade varia de acordo com a luz que incide neste. O seu comportamento em relação à luz é não linear e, num ambiente escuro a sua resistividade aumenta

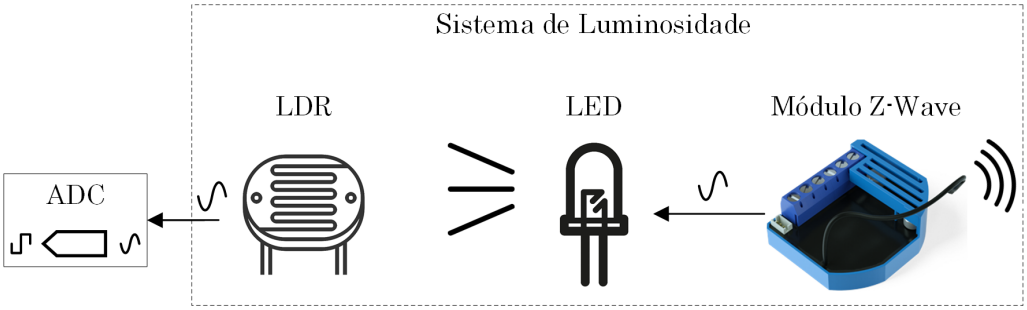


Figura 3.16: Constituição do sistema de luminosidade

e, vice-versa, quanta mais luz incidir no LDR menor será a sua resistência. Na Figura 3.17 é visível o aspeto físico de um LDR.

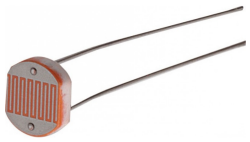


Figura 3.17: LDR

Devido à inexistência de um *datasheet*, a relação entre a luz incidente no LDR e a sua resistividade não é conhecida em termos quantitativos, assim como, cada LDR tem um comportamento diferente. Para implementação deste componente existem duas topologias possíveis, a ponte de Wheatstone ou o divisor de tensão. A primeira, oferece uma maior precisão de aquisição de valores, no entanto, a sua implementação é mais complicada. A segunda topologia, não oferece uma precisão tão grande mas a sua implementação é mais simples. Desta forma, foi utilizado o divisor de tensão, visto que a precisão das medições não é um aspeto crítico para este protótipo. O circuito desta topologia pode ser visualizado na Figura 3.18.

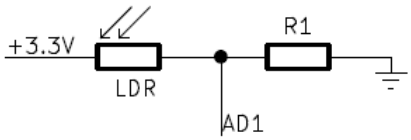


Figura 3.18: Circuito LDR

Neste circuito, o LDR é colocado em série com uma resistência para criar uma tensão proporcional ao valor da alimentação, que é de 3,3 V. Este valor foi escolhido de forma a garantir que o sinal de saída, que será lido pelo ADC, não ultrapassa o valor máximo admitido por este. A equação do divisor de tensão deste circuito, que permitirá calcular a resistência do LDR, será dada pela

seguinte equação:

$$V_{out} = \frac{R1}{R_{LDR} + R1} V_{cc} \tag{3.8}$$

Onde V_{out} é a tensão aos terminais de $R1$, que será amostrada pelo ADC, V_{cc} é a tensão de alimentação do circuito, 3,3 V, R_{LDR} é a resistência do LDR e $R1$ é uma resistência de valor fixo igual a 10 kΩ. Com esta equação é possível verificar que, dado que a resistência do LDR aumenta com a diminuição da luz incidente neste, a tensão de saída do circuito será proporcional à luz incidente no LDR. Manipulando esta equação, obtém-se a equação que permite calcular a resistência do LDR:

$$R_{LDR} = R\Big(\frac{V_{cc}}{V_{out}} - 1\Big) \tag{3.9}$$

No entanto, ainda é necessário quantificar a relação entre a resistência do LDR e a luz incidente. Assim, procedeu-se ao dimensionamento deste.

3.6.1 Dimensionamento do LDR

O dimensionamento deste componente foi realizado através de um método prático, que consiste na medição de duas grandezas: a resistência do LDR e a luminosidade incidente neste utilizando um fotómetro. Esta experiência foi realizada num ambiente escuro, e recorrendo a uma lâmpada de intensidade regulável, foi-se aumentando gradualmente a luminosidade enquanto se efetua as medições das grandezas mencionadas anteriormente. A Tabela 3.3 apresenta os valores adquiridos nesta experiência, com os quais desenhou-se um gráfico da relação entre estas duas grandezas que pode ser observado na Figura 3.19.

Tabela 3.3: Resultados da experiência de dimensionamento do LDR

Luminosidade (Lux)	**Resistência LDR (Ω)**
1	64700
2	27340
20,6	5860
64,5	2500
99,3	1730
206	1020
256	890
303	780
354	720
406	620
512	564
583	540
647	458
712	403
914	350
1274	305
1966	219
2260	200
3230	177

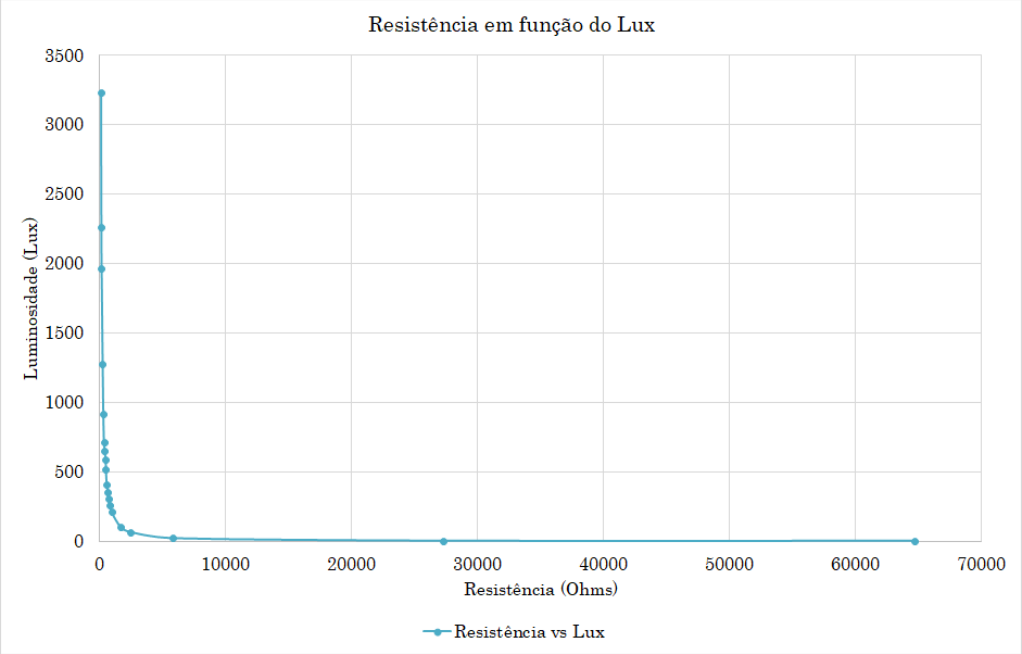


Figura 3.19: Gráfico da resistência do LDR em função da luminosidade

Como ponto de partida, para obter uma relação entre luminosidade e resistência do LDR, admitiu-se a reta com a equação 3.10.

$$y = mx + b \tag{3.10}$$

Contudo, esta reta não fornecerá uma aproximação correta à curva da Figura 3.19. Assim, com o intuito de obter uma resposta mais favorável, foi traçado um novo gráfico, apresentado na Figura 3.20. Neste, aplicou-se o logaritmo a cada variável, criando a relação do logaritmo da resistência pelo logaritmo da luminosidade.

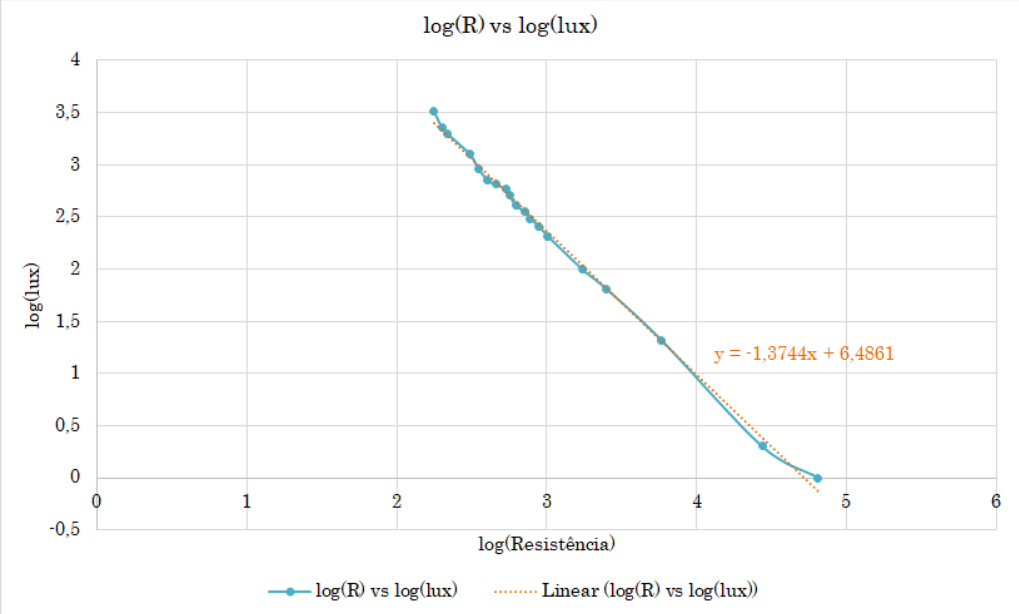


Figura 3.20: Gráfico da relação entre o logaritmo da resistência e o logaritmo da luminosidade

Uma vez que as variáveis foram alteradas, a equação 3.10 também sofre mudanças passando a ser:

$$log_{10}(lux) = m \times log_{10}(R_{LDR}) + b \tag{3.11}$$

Fazendo a substituição pelos valores obtidos na equação da linha de tendência da Figura 3.20, obtém-se a equação 3.12.

$$log_{10}(lux) = -1,3744 \times log_{10}(R_{LDR}) + 6,4861 \tag{3.12}$$

Com várias manipulações matemáticas, obtém-se uma equação que permitirá calcular a luminosidade em função da resistência do LDR (equação 3.15).

$$10^{log_{10}(lux)} = 10^{m \times log_{10}(R_{LDR})+b} \tag{3.13}$$

$$lux = R_{LDR}^{m} \times 10^{b} \tag{3.14}$$

$$lux = R_{LDR}^{-1,3744} \times 3,06196 \times 10^{6} \tag{3.15}$$

Tendo-se assim, todos os dados necessários para calcular a luminosidade medida pelo LDR.

O segundo elemento do sistema de luminosidade é um LED, que será a fonte de luz usada, e é este que será controlado de forma a alterar a luminosidade no sistema. O circuito do LED é muito simples, consistindo numa resistência ligada em série e num divisor de tensão. Este circuito pode ser visualizada na Figura 3.21.

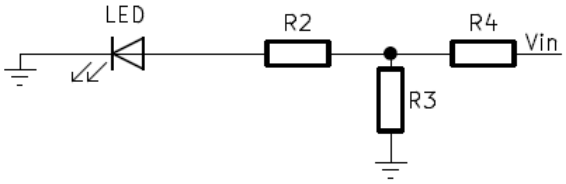


Figura 3.21: Circuito de implementação do LED

O divisor de tensão, resistências $R3$ e $R4$, é usado para diminuir a tensão proveniente do módulo Z-Wave que fornece uma tensão contínua entre 0 e 10 V. Assim, estas duas resistências têm o mesmo valor de 1 kΩ reduzindo a tensão para metade. Este módulo Z-Wave é o terceiro e último elemento do sistema de luminosidade, e trata-se de módulo igual ao do sistema térmico, representado na Figura 3.11, e irá ter um funcionamento igual ao deste sistema.

O esquema de ligações completo de todo o projeto é apresentado na Figura 3.22. De realçar que neste não são representados os módulos Z-Wave.

3.7 Dispositivos Z-Wave

Neste módulo do sistema, consideram-se todos os dispositivos Z-Wave que o utilizador decida emparelhar com este. Estes poderão ser atuadores, isto é, dispositivos que são controlados pelo utilizador, como por exemplo, uma *Smart Plug* que permite que este ligue ou desligue o aparelho ligado a esta, ou por exemplo, dispositivos de regulação luminosa (*dimmers*) com os quais o utilizador

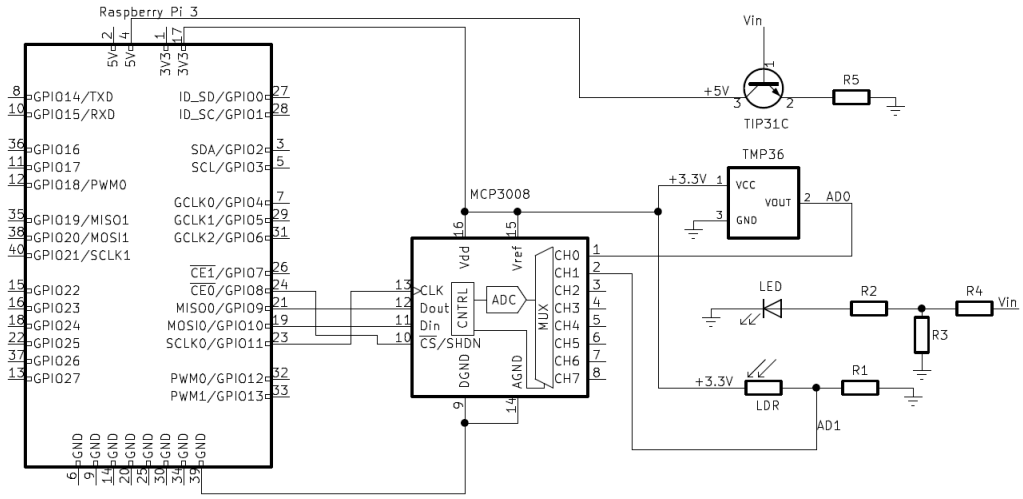


Figura 3.22: Circuito completo do sistema

pode regular a iluminação. Ou então, estes dispositivos, poderão ser sensores que apenas reportam valores, que podem ser de vários tipos, desde medições de grandezas elétricas que podem ser obtidas, por exemplo, com um *Smart Meter*, ou grandezas como temperatura e luminosidade, ou até simples sensores binários.

De realçar, que existem atuadores que também são sensores, ao reportar a potência e energia consumida por estes.

Capítulo 4

Implementação

Neste capítulo é descrita a implementação dos vários módulos apresentados no capítulo anterior de forma a obter-se o bom funcionamento de cada um destes com foco na base de dados, página *web* e aplicação C/C++. Serão estes que irão fazer com que todos os módulos funcionem, sendo igualmente explicado o raciocínio do desenvolvimento do código de cada um. O protótipo desenvolvido pode ser visualizado na Figura 4.1, sendo que é de realçar que, nesta figura não são representados os módulos Z-Wave.

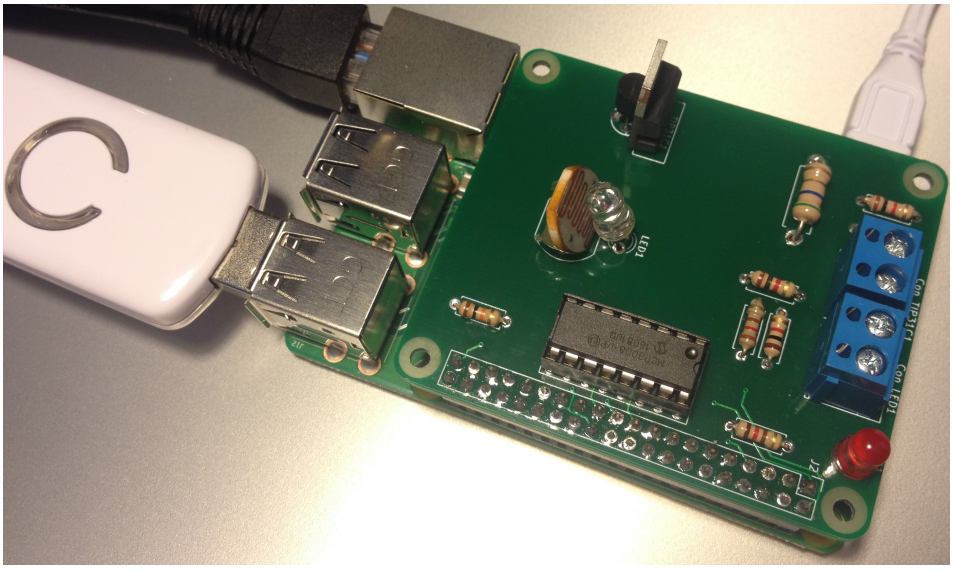


Figura 4.1: Protótipo desenvolvido

Como é possível verificar nesta figura, todos os circuitos foram implementados numa placa de circuito impresso (PCB - *Printed Circuit Board*). O *layout* de ambas as faces podem ser visualizadas nas Figuras A.1 e A.2, respetivamente face inferior e face superior, em Anexo A.

4.1 Base de Dados

A base de dados é constituída por 10 tabelas, de forma a acomodar os casos de uso mencionados no capítulo anterior. Estas tabelas podem ser divididas em três áreas diferentes: as tabelas referentes aos sensores conetados fisicamente ao Raspberry, as tabelas referentes aos dispositivos Z-Wave emparelhados com o sistema e as tabelas referentes aos cenários Z-Wave. Na Figura 4.2 são apresentadas as três tabelas pertencentes à primeira área.

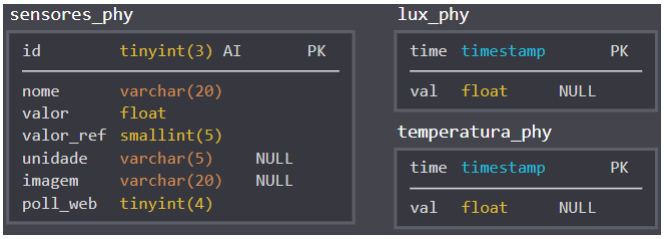


Figura 4.2: Tabelas da base de dados referentes aos sensores conetados fisicamente com o Raspberry Pi

A tabela principal, `sensores_phy`, irá guardar toda a informação sobre os sensores, como o nome, o valor atual medido pelo sensor, o valor de referência (`valor_ref`) que se pretende que o controlador alcance, a unidade da grandeza medida pelo sensor e uma *string* com o nome da imagem que deverá ser colocada na interface gráfica para representar este sensor. Para além disso, tem mais uma coluna, `poll_web`, que indicará à página *web* se o valor medido pelo sensor mudou, para que esta possa atualizar o valor apresentado ao utilizador.

As outras duas tabelas, `lux_phy` e `temperatura_phy`, irão guardar todos os valores da luminosidade e da temperatura, respetivamente, ao longo do tempo de forma a poder apresentar gráficos destas grandezas.

O segundo tipo de tabelas são as tabelas referentes aos dispositivos Z-Wave. Estas podem ser visualizadas na Figura 4.2.

Nestas, a tabela principal é a `dispositivos_zw`, que guardará informação que identifica o dispositivo. O `node_id` é um número que é atribuído a cada dispositivo aquando do emparelhamento e a coluna `nome` guardará o nome do dispositivo atribuído pelo fabricante. As duas colunas seguintes são algo semelhantes, a coluna `generic` terá um número que identificará o tipo genérico de dispositivo enquanto que a coluna `tipo` guardará também o tipo de dispositivo, mas de uma forma legível (uma frase e não número), o que permitirá apresentar ao utilizador de uma forma mais fácil. As duas colunas seguintes servirão para identificar se o dispositivo suporta a *Command Class Sensor Multilevel* e a *Command Class Meter*, estas são necessárias pois vários dispositivos, para além da sua funcionalidade básica, também incluem outras funcionalidades como medição

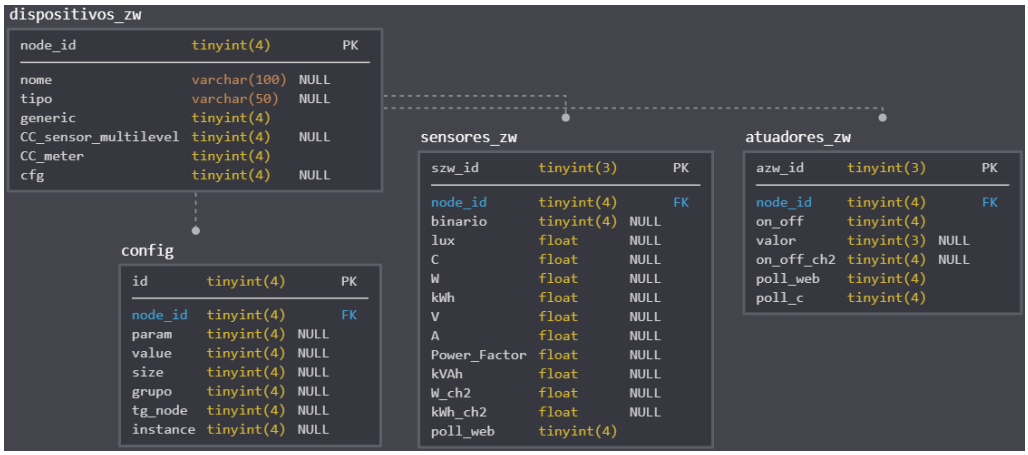


Figura 4.3: Tabelas da base de dados referentes aos dispositivos Z-Wave

da potência e energia que são reportados através da *Command Class Meter*, ou como medição da luminosidade e da temperatura que são reportados através da *Command Class Sensor Multilevel*. Por fim, a última coluna, `cfg`, será usada para indicar à aplicação várias ações que esta deve realizar, como iniciar o processo de inclusão ou remoção de dispositivos, ou para configurar um parâmetro ou associação de um dispositivo.

Na tabela `sensores_zw` serão colocados todos os dispositivos que tenham qualquer tipo de funcionalidade sensorial. Esta tabela terá uma *primary key*, `szw_id`, no entanto, será sempre usada a coluna `node_id` para identificar o dispositivo. Esta coluna é uma *foreign key* da tabela principal, `dispositivos_zw`, apresentando, assim, uma relação de um para muitos. Depois, existem as colunas que contêm as grandezas como: sensor binário, luminosidade, temperatura, potência, energia, tensão, corrente, fator de potência e energia aparente. Para além disso, tem mais uma coluna, `poll_web`, que será usada da mesma forma que a coluna homóloga da tabela `sensores_phy`.

Na tabela `atuadores_zw` serão colocados todos os dispositivos que podem ser atuados. Esta, à semelhança da anterior, irá ter uma coluna `node_id` que é uma *foreign key* da tabela principal. Em relação à atuação de dispositivos existem três colunas, a `on_off`, que poderá ser 0 ou 1, que será o estado do canal 1 do dispositivo Z-Wave, a coluna `valor` que guardará o valor do atuador, de 0 a 99, caso este seja regulável, e por fim, a coluna `on_off_ch2` que é igual à coluna `on_off` mas será referente ao segundo canal do dispositivo, caso este tenha dois canais. De realçar que não existe uma coluna para o valor do canal 2, porque não existem dispositivos Z-Wave reguláveis de dois canais. Para além disso, esta tabela tem mais duas colunas, `poll_web` e `poll_c`, que serão usadas da mesma forma que as colunas `poll_web` descritas anteriormente, contudo a coluna `poll_c` será usada pela aplicação C/C++.

A última tabela, `config`, guardará informações sobre as configurações, de parâmetros ou associações, a fazer aos dispositivos. Esta tabela, para além da *primary key*, contém uma coluna `node_id` que é uma *foreign key* da tabela principal que servirá para identificar o dispositivo a configurar. Para além disso, contém mais seis colunas que podem ser divididas em duas partes iguais. As três primeiras, são referentes à configuração de parâmetros, onde a coluna `param` guardará o número do parâmetro, a coluna `value` guardará ao valor a atribuir a este parâmetro e a coluna `size` conterá o tamanho do valor. As três últimas colunas são referentes à configuração de associações, onde a coluna `grupo` será o número do grupo, a coluna `tg_node` será o ID do nó que se pretende adicionar ao grupo e a coluna `instance` é o identificador da instância.

O terceiro e último tipo de tabelas, referentes aos cenários Z-Wave, podem ser visualizadas na Figura 4.4.

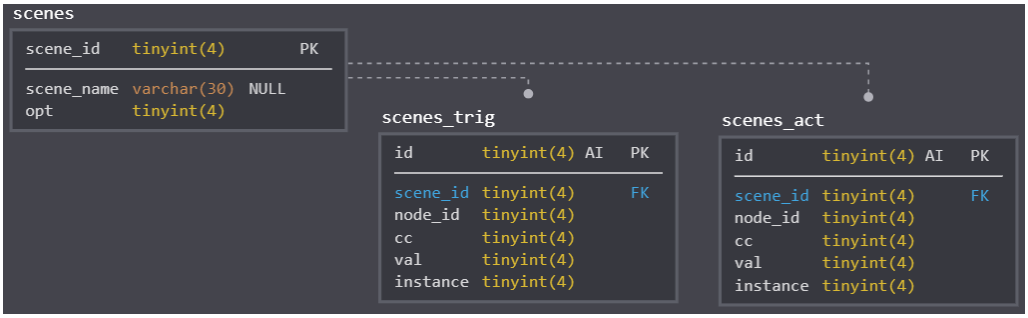


Figura 4.4: Tabelas da base de dados referentes aos cenários Z-Wave

A primeira tabela, `scenes`, guardará informação geral sobre o cenário, como o ID, `scene_id`, e o nome, `scene_name`, que é atribuído pelo utilizador. A coluna `scene_id`, tal como acontece com a coluna `node_id` nas tabelas descritas anteriormente, será usada como *foreign key* em outras tabelas, apresentando também uma relação de um para muitos. A última coluna, `opt`, guardará um número que indicará à aplicação se é para criar, apagar ou editar o cenário. Um cenário é constituído por *triggers* que acionam o cenário e dispositivos que são atuados quando este é acionado. Desta forma, estes dois tipos foram divididos em duas tabelas.

A tabela, `scenes_trig`, armazena os *triggers* que acionam os cenários. Esta, para além de uma *primary key*, contém o ID do cenário, `scene_id`, e mais quatro colunas que identificam o *trigger* que são: o ID do nó, `node_id`, a *command class*, `cc`, o valor, `val` e a instância, `instance`. A tabela `scenes_act` tem exatamente a mesma estrutura, no entanto, será usada de forma diferente por parte da aplicação e da página *web*.

4.2 Aplicação C/C++

Com a estrutura da base de dados definida, passou-se à implementação da aplicação C/C++, que será a principal peça do sistema. Esta, é responsável por todas as funcionalidades Z-Wave, pela leitura dos sensores e pelo controlo das variáveis medidas por estes. Contudo, antes de se proceder à explicação da implementação desta aplicação existem alguns aspetos, relativos à biblioteca OpenZWave, que necessitam de ser esclarecidos para que seja possível compreender a implementação efetuada.

Um aspeto a ter em consideração é algo que a biblioteca denomina de `ValueID`, que é usado para identificar exclusivamente um valor reportado por um dispositivo Z-Wave. Este é construído incluindo várias características num número único de 32 *bits*. Estas características incluem o *home* ID, o ID do nó, o tipo de valor (*bool*, *byte*, *string*, etc), a *command class* do valor reportado, a instância, entre outros. Para além disso, é de realçar que toda a informação sobre os nós fica guardada numa *list* (variável do tipo lista), com o nome `g_nodes`. Esta lista guarda elementos do tipo `NodeInfo`, que é uma estrutura que tem como membros o *home* ID, o ID do nó e uma *list* do tipo `ValueID`, `m_values`, na qual serão guardados os dados mencionados anteriormente. Esta informação, ou qualquer outra acerca da rede Z-Wave (isto exclui os cenários), não tem que ser guardada e restaurada de cada a vez que a aplicação termine e recomece, esta operação é realizada automaticamente pela biblioteca OpenZWave, assim como, também lida com quaisquer alterações na rede que occorram quando a aplicação não está a correr.

Outro aspeto a ter em conta é que qualquer aplicação que use esta biblioteca, deverá ter como ponto central a função `OnNotification()`. Esta, é acionada sempre que um nó, um grupo ou valor muda, no fundo, é nesta função que é reportado toda a informação referente ao estado e às configurações dos dispositivos. De realçar que, toda a aplicação foi desenvolvida a partir do *template*, disponibilizado pela biblioteca OpenZWave, no qual esta função já inclui algumas funcionalidades básicas implementadas. Ao longo desta secção serão explicadas as funcionalidades básicas mais importantes, assim como, aquelas que foram acrescentadas. O fluxograma desta função está representado na Figura 4.5.

Esta função começa por obter um *mutex*, este passo deve ser feito para evitar conflitos com a *thread* principal e é realizado sempre que se acesse a recursos partilhados. Após isto, irá inicializar a conexão com a base de dados mas, dado que esta função é executada várias vezes, esta conexão apenas deve ser inicializada da primeira vez que a função é executada. Para tal, é verificada a variável `g_con` que é um objeto do tipo `MYSQL`, que é inicializada através da função `mysql_init()`, da API da MySQL para C. Esta função aloca, inicializa e retorna um novo objeto, `g_con`. De seguida, será estabelecida a ligação com a base de dados através

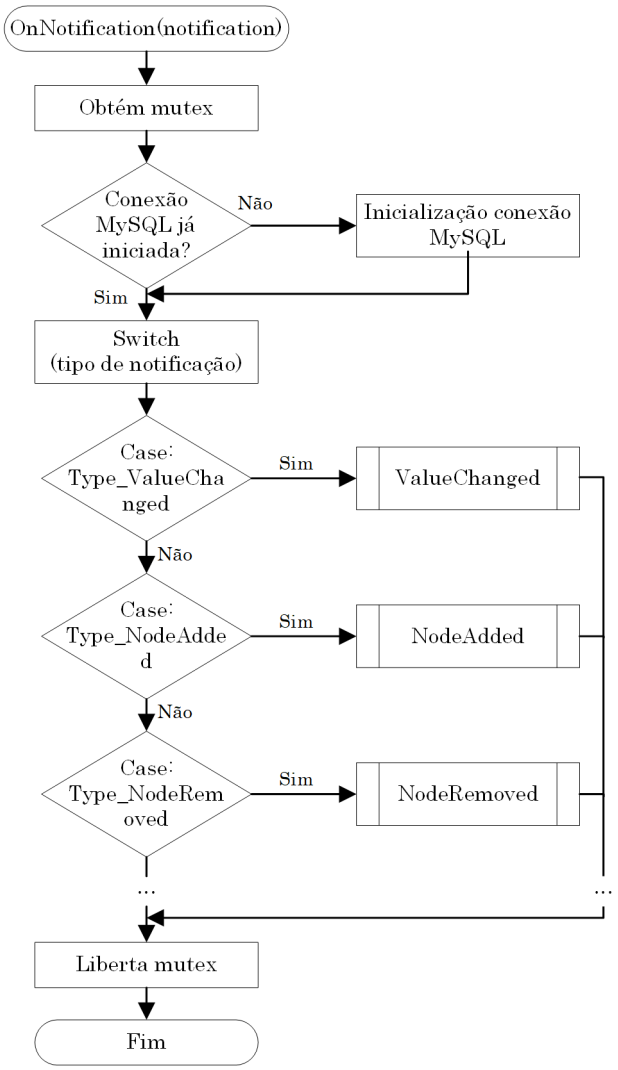


Figura 4.5: Fluxograma da função `OnNotification()`

da função `mysql_real_connect()`, que recebe como parâmetros: a variável `g_con`, o IP do servidor que, neste caso, será o *localhost*, o ID de utilizador do MySQL e a respetiva palavra-passe, o nome da base de dados, a porta da conexão *Transmission Control Protocol/Internet Protocol* (TCP/IP), o socket a usar e o último parâmetro será para especificar algumas configurações caso pretendidas. O código implementado para inicializar a conexão com a base de dados é o seguinte:

```
g_con = mysql_init(NULL);
mysql_real_connect(g_con, "localhost", "root", "123",
"ctrlDB", 0, NULL, 0);
```

Após estabelecida a ligação, ir-se-á verificar o tipo de notificação que acionou a função, sendo que esta função recebe como parâmetro um apontador para um

objeto denominado `notification`, que é do tipo `Notification` que é uma classe da bilbioteca OpenZWave. Assim, é feito um *switch case* do valor retornado pela função `GetType()` deste objeto. As notificações podem ser de muitos tipos, no entanto, no fluxograma apenas estão representados quatro, que são os mais relevantes para aplicação e para os quais foi desenvolvido código, para além do código base do *template*. Apesar de no fluxograma estar representado que em cada *case* do *switch* é executada uma função, estas na verdade não são funções, apenas foi representado como tal para simplificar o fluxograma e, assim, facilitar a sua leitura. O código executado em cada um destes *cases*, em grande parte dos casos, funciona em conjunto com código implementado noutras funções, e por isso, de forma a tornar a sua compreensão mais fácil estas serão explicadas posteriormente, ao mesmo tempo que estas funções.

Com isto, a função principal da aplicação, a função `main()`, poderá ser representada pelo fluxograma da Figura 4.6. Tal como no fluxograma anterior, este foi simplificado de forma a facilitar a sua leitura.

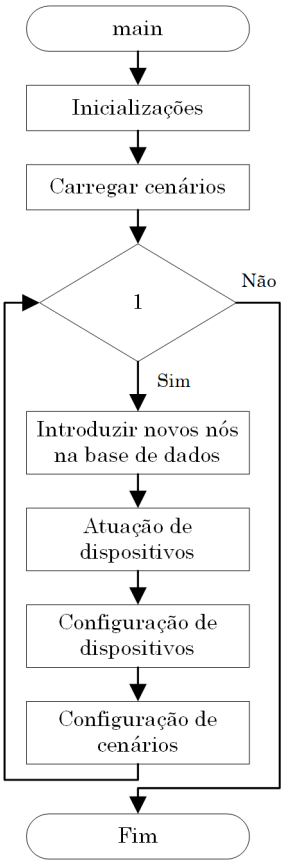


Figura 4.6: Fluxograma da função `main()`

Esta, começa por realizar várias inicializações necessárias, sendo a primeira a criação de conexão com a base de dados, diferente da da função

`OnNotification()`. São usadas duas conexões diferentes, pois estas duas funções correm em *threads* diferentes e poderá acontecer ambas tentarem usar a mesma conexão ao mesmo tempo. Para além disso, efetua múltiplas inicializações referentes à biblioteca OpenZWave, destacando-se a criação de um objeto `Manager` através da função `Create()`. Este objeto fornece a interface pública com o OpenZWave, expondo todas as funcionalidades Z-Wave necessárias. Para além disso, é criado um objeto `Driver` para manipular todas as comunicações com a *pen* Z-Wave. Este objeto é criado com a função `AddDriver()`, enviando-lhe como parâmetro a porta série a que está ligada a *pen*, neste caso `"/dev/ttyACM0"`. Para além destas configurações, também é inicializada a biblioteca WiringPi através da função `WiringPiSetup()`, e é inicializado o canal de comunicação SPI para comunicar com o ADC. Isto é realizado através da função `WiringPiSPISetup()` que recebe como parâmetros o canal SPI a usar, neste caso o canal 0, e a velocidade do *clock* SPI em Hz, neste caso foi escolhido 1 MHz. Por fim, são criadas três *threads*, duas para efetuar o controlo da temperatura e luminosidade, uma *thread* para cada, e a terceira para enviar os valores destas grandezas para a base de dados.

Terminadas as inicializações, serão carregados os cenários Z-Wave guardados na base de dados. Este procedimento é necessário pois, ao contrário de todas as outras configurações Z-Wave, o OpenZWave não guarda as informações referentes ao cenários. O que quer dizer que, caso a aplicação termine, é necessário criar de novo os cenários, através das informações guardadas na base de dados, quando a aplicação reiniciar. Após isto, a aplicação entrará num ciclo infinito no qual irá efetuar *polling* de algumas variáveis que indicarão que tipo de ação deve ser realizada, como por exemplo, introduzir novos nós na base de dados, atuar dispositivos, configurar dispositivos e configurar cenários.

4.2.1 Introdução de Novos Nós na Base de Dados

Esta secção do código, será responsável por verificar se foram adicionados novos nós à rede Z-Wave e, se tal aconteceu, irá obter vários dados sobre este nó e inseri-los na base de dados. O fluxograma desta secção do código pode ser visualizado na Figura B.2.

Sempre que um novo nó é adicionado à rede, é recebida uma notificação do tipo `NodeAdded` e, por isso, o código do fluxograma da Figura B.2, em Anexo B, deveria ser implementado dentro desta notificação. No entanto, após a realização de testes, verificou-se que, quando esta notificação ocorre ainda não está disponível toda a informação sobre o dispositivo. Assim, o processo de recolha de dados tem de ser efetuado mais tarde, daí ser executada na função `main`. Ainda assim, a notificação `NodeAdded` é utilizada para determinar qual o ID do nó que foi adicionado. Desta forma, quando esta notificação é recebida, será guardado

numa *list*, `g_new_nodes`, o ID do nó que deu origem à notificação. O código desta notificação será explicada mais à frente nesta secção.

Desta forma, a condição para executar o código da figura anterior é que a *list* não esteja vazia, o que significa que foi adicionado um novo dispositivo. Contudo, é de realçar que a notificação `NodeAdded` é acionada também quando a aplicação inicia, caso já existissem dispositivos emparelhados. O que quer dizer que serão guardados na lista, ID's de nós que já foram introduzidos na base de dados na iteração anterior da aplicação. Assim, para distinguir quais os nós novos, que possam ter sido adicionados enquanto a aplicação não estava a correr, e os nós que já existiam, é realizada uma *query*, usando a função `mysql_query()`, à tabela `dispositivos_zw` pelo maior `node_id` existente pois, qualquer nó novo que seja introduzido terá sempre um ID sequente a este. Esta *query* será:

```
SELECT MAX(node_id) FROM dispositivos_zw
```

Após isto, serão percorridos todos os elementos da lista, `g_new_nodes`, utilizando um ciclo *for* cuja condição de término é que a variável de inicialização, `i`, seja igual ao *past-end-element* da lista. Este elemento é o que teoricamente se segue ao último elemento da lista. Neste ciclo *for*, começa-se por verificar se `i`, que é igual a um ID de um nó guardado na lista, é maior que o resultado da *query*. Em caso negativo, ir-se-á prosseguir para a próxima iteração do ciclo *for*. Em caso afirmativo, é verificado se o ID é igual a 1, se não o for, é porque não se trata da *pen* USB, e então serão recolhidas informações sobre o dispositivo e será realizada uma *query* à base de dados de forma a inserir estes. A *query* a realizar terá o seguinte formato:

```
INSERT INTO dispositivos_zw (node_id, nome, tipo, generic,
CC_meter, CC_sensor_multilevel) VALUES (i, A, B, C, D, E)
```

Onde, seguindo o seguinte pseudo-código:

```
A = GetNodeManufacturerName( g_homeId, i) +
GetNodeProductName( g_homeId, i)
B = GetNodeType( g_homeId, i)
C = GetNodeGeneric( g_homeId, i)
D = GetNodeClassInformation( g_homeId, i, 0x32)
E = GetNodeClassInformation( g_homeId, i, 0x31)
```

Desta forma, `A` será uma *string* formada pelo nome do fabricante e o nome do produto, `B` será uma *string* que descreve o tipo básico, genérico ou específico, do dispositivo, dependendo do que é reportado por este e `C` irá ser um número inteiro que representa o tipo de produto genérico (Tabela C.1 do Anexo C). Por fim, `D` e `E` irão ser 1 ou 0, que indicará se o dispositivo suporta as *command classes*, *Command_Class_Meter* e *Command_Class_Sensor_Multilevel*, respetivamente. Após efetuar esta *query*, é verificado se o dispositivo é um atuador ou sensor, para que

este seja colocado, ou não, nas tabelas `atuadores_zw` e `sensores_zw`. Para tal, começa-se por verificar se o tipo genérico do dispositivo é igual a 16, ou seja, se é um *binary switch*. Se sim, então é usada a função `check_ch()`, cujo código será explicado mais à frente, que irá retornar 1 se o dispositivo tiver dois canais. Assim, se tiver dois canais, é realizada uma *query* para inserir estes dados na tabela `atuadores_zw`. Esta terá o seguinte formato:

```
INSERT INTO atuadores_zw (azw_id, node_id, on_off,
on_off_ch2, poll_web, poll_c) VALUES (i, i, 0, 0, 0, 0)
```

Desta forma, ao fazer com que a coluna `on_off_ch2` seja diferente de `NULL`, assinala-se que o dispositivo tem um segundo canal e, ao deixar a coluna `valor` igual a `NULL`, assinala-se que o dispositivo não é um *dimmer*. Caso o dispositivo tenha apenas um canal, é realizada uma *query* igual a esta mas sem especificar um valor para `on_off_ch2`.

Caso não se trate de um *binary switch*, é verificado se o tipo genérico é igual a 17, ou seja, se é um *multilevel switch*. Se o for, será realizada uma *query* do mesmo formato que a anterior mas sem especificar um valor para `on_off_ch2` e especificando o valor de zero para a coluna `valor`, assinalando assim que se trata de um *multilevel switch*. Em caso negativo, é porque não se trata de nenhum tipo de atuador e, por isso, não se deve inserir nada na tabela `atuadores_zw`.

Verificadas as capacidades atuadoras do dispositivo, são verificadas as capacidades sensoriais deste. Desta forma, averigua-se se este suporta a *Command_Class_Meter*, ou a *Command_Class_Sensor_Multilevel* ou a *Command_Class_Sensor_Binary*. Se suportar alguma destas, então será realizada uma *query* para inserir dados na tabela `sensores_zw`. Esta *query* terá o seguinte formato:

```
INSERT INTO sensores_zw (szw_id, node_id) VALUES (i, i)
```

As restantes colunas das grandezas são mantidas a `NULL` e apenas quando o dispositivo reportar essa grandeza é que esta mudará o seu valor.

Caso o ID do nó em causa seja 1, este representa sempre a *pen* USB assim, não é necessário verificar se é um atuador, ou sensor, ou quais as *command classes* que suporta, por isso, é feita apenas uma *query* à tabela `dispositivos_zw` indicando o `node_id`, o `nome` e o `tipo`.

Com isto, prossegue-se para a próxima iteração do ciclo *for* repetindo este processo, até que termine. Quando tal acontecer, restam apenas duas ações, que é limpar a lista `g_new_nodes`, utilizando a função `clear()` e libertar o resultado da primeira *query*, terminando assim esta secção do código.

check_ch()

Esta função terá como objetivo determinar se um dispositivo tem dois canais on/off. Esta operação é realizada percorrendo todos os elementos da

lista `g_nodes`, à procura de um com o `node_id` igual ao do pretendido e que tenha um `ValueID` com a *command class* igual a 0x25, ou seja, *Command_Class_Switch_Binary*, e com a instância igual 2, que corresponde ao segundo canal. O fluxograma desta função pode ser visualizado na Figura 4.7.

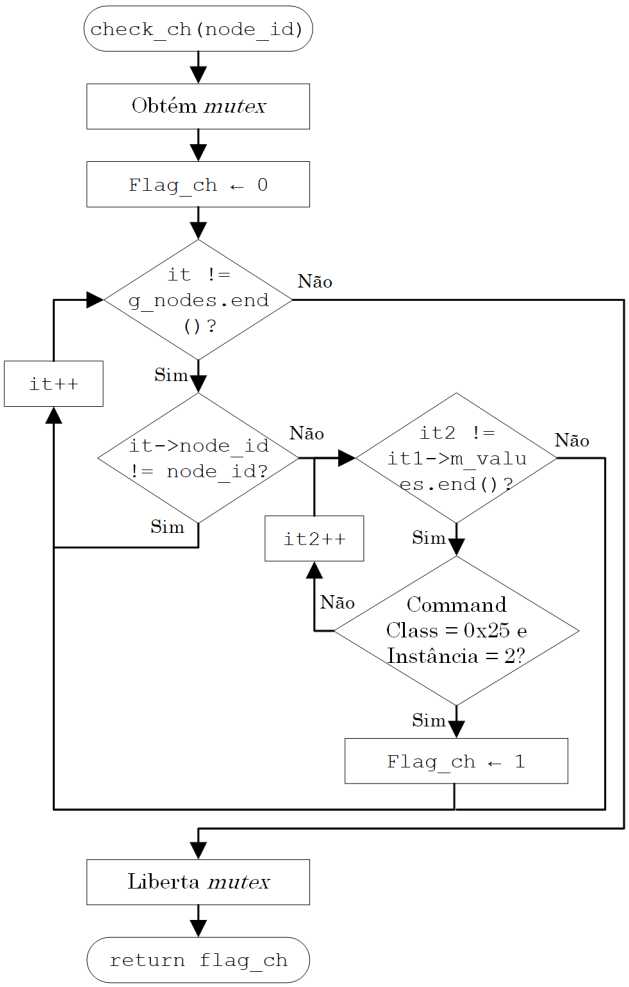


Figura 4.7: Fluxograma da função `check_ch()`

A função começa por obter um *mutex*, o mesmo que foi mencionado na função `OnNotification()`, e por colocar a variável `flag_ch` a zero, é esta que irá assinalar se o dispositivo tem ou não dois canais. De seguida, irá ser percorrida a lista `g_nodes` utilizando um ciclo *for* à semelhança do que foi feita com a lista `g_new_nodes` do código explicado anteriormente. Em cada iteração deste ciclo é verificado se o ID do nó do atual elemento da lista é diferente do ID do nó recebido pela função, `node_id`. Se sim, o código seguinte é ignorado e procede-se para a próxima iteração do ciclo. Se não for, então ir-se-á percorrer a lista `m_values`, também com um ciclo *for*, e dentro deste é verificado se o `ValueID`, guardado na variável `m_values`, tem a *command class* igual a 0x25 e a instância

igual 2. Em caso afirmativo, `flag_ch` é colocada a 1 e é realizado `break` ao ciclo *for*. Em caso negativo, prossegue-se para a iteração seguinte do ciclo *for*. Quando terminado de percorrer as listas, é libertado o *mutex* e é retornado o valor de `flag_ch`.

Este processo de percorrer as listas `g_nodes` e `m_values` é usado em várias funções da aplicação para encontrar um determinado `ValueID` e, por isso, será referenciado várias vezes ao longo desta secção.

4.2.2 Atuação de Dispositivos

Nesta secção de código são verificados quais entradas, da tabela `atuadores_zw`, têm a coluna `poll_c` igual a igual, que assinala que o estado do respetivo dispositivo foi alterado, e irá atuar este de acordo com os dados guardados nesta tabela. O fluxograma desta secção de código está representada na Figura 4.8.

Esta função começa por realizar uma *query* à base de dados, pedindo as colunas `node_id`, `on_off`, `valor` e `on_off_ch2`, das entradas que têm a coluna `poll_c` igual 1. Após isto, o resultado é obtido através da função `mysql_store_result()` que irá guardar este numa estrutura do tipo `MYSQL_RES`. De seguida, o código entrará num ciclo *while* cuja condição de término é que já todas as linhas (*row*) do resultado da *query* já tenham sido lidas. Tal, pode ser determinado ao mesmo tempo que se obtém a próxima linha do resultado da *query*, utilizando a função `mysql_fetch_row()` que irá retornar uma estrutura do tipo `MYSQL_ROW` com a próxima linha do resultado. Caso não hajam mais linhas, esta irá retornar `NULL` e, assim, terminar o ciclo *while*. Dentro deste, começa-se por identificar o tipo de dispositivo, verificando se o dispositivo é *dimmer* através da coluna `valor`. Se esta for diferente a `NULL` é porque se trata de um. Caso contrário, trata-se de um simples on/off e o canal 1 do dispositivo é atuado, com a função `SetValueBool()`, de acordo com o valor da coluna `on_off`. De seguida, é verificado se o dispositivo tem um segundo canal através da coluna `on_off_ch2`. Assim, se esta for diferente de `NULL` o segundo canal do dispositivo será atuado de acordo com esta. Caso o dispositivo se trate de um *dimmer*, será verificada a coluna `on_off` e, se esta for igual a 1, significa que se pretende ligar o dispositivo, o que terá de ser feito com um determinado valor de luminosidade, guardado na coluna `valor`, através da função `SetValueByte()`. Caso `on_off` seja igual a zero, o dispositivo será desligado usando a função `SetValueBool()`.

SetValueBool() e SetValueByte()

Estas duas funções serão responsáveis por atuar os dispositivos. Tal, é realizado através da função, da biblioteca OpenZWave, `SetValue()`, para qual é passado um `ValueID` e o valor a atribuir a este. Deste modo, `SetValueBool()`

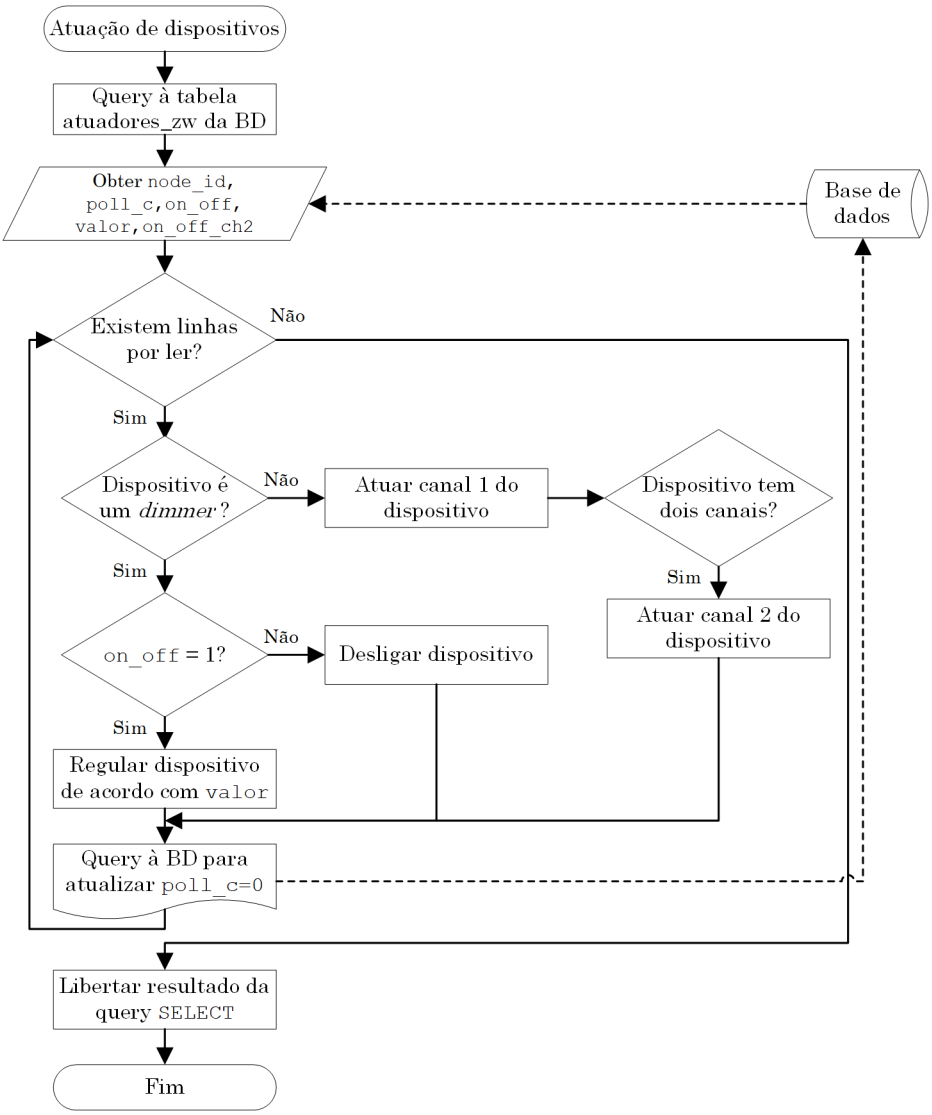


Figura 4.8: Fluxograma do código de atuação de dispositivos

será muito parecida com a função `check_ch()`, no sentido em que se irá pesquisar o `ValueID` com uma determinada *command class*, instância e ID do nó, recebidos como parâmetros. Tendo encontrado o `ValueID` em questão, é evocada a função `SetValue()`, atuando assim o dispositivo. A função `SetValueByte()` é igual à `SetValueBool()` contudo a *command class* que irá procurar será a *Command_Class_Switch_Multilevel* em vez da *Command_Class_Switch_Binary*. Para além disso, esta irá atribuir ao `ValueID` um valor de oito *bits* em vez de um valor booleano, o que obrigará a chamar uma função `SetValue()` diferente, apesar de ter o mesmo nome, dado que lhe é passada uma variável (o valor) de um tipo diferente. O fluxograma da função `SetValueBool()` pode ser visualizado na Figura B.1 em Anexo B.

4.2.3 Interpretação de Mensagens da Rede Z-Wave

Tal como mencionado anteriormente, toda a informação reportada pelos dispositivos Z-Wave passa pela função `OnNotification()` assim, o tratamento destas mensagens é realizada nesta mesma. Neste caso, optou-se por interpretar apenas as que são do tipo `ValueChanged`, estando assim a par de todas as alterações na rede Z-Wave. O fluxograma do código responsável por esta ação está representado na Figura B.3 do Anexo B.

Esta função, começa por usar a função `GetValueID()` com o objeto `notification`, que irá retornar um objeto com os dados do `ValueID` que acionou a notificação. Tendo este, é usada a função `GetCommandClassId()` para obter a *command class* e através de um *switch case* ir-se-á determinar esta e executar o código apropriado.

Desta forma, caso a *command class* seja 0x25, ou seja, *Command_Class_Switch_Binary*, é usada a função `GetValueAsBool()`, dado que nesta *command class* só se receberá valores binários, que corresponderão ao estado, *on* ou *off*, do dispositivo. Para esta função, é passado o `ValueID` e uma variável para guardar o valor. Após isto, procede-se ao envio da informação para a base de dados, contudo, é necessário saber a que tipo de dispositivo se refere este valor e a que canal desse dispositivo pertence. Assim, é obtida a instância deste `ValueID` através da função `GetInstance()` e, caso esta seja igual a 2, então a coluna `on_off_ch2`, da tabela `atuadores_zw` será atualizada com o valor recebido utilizando a seguinte *query*:

```
UPDATE atuadores_zw SET on_off_ch2 = A, poll_web = 1 WHERE
node_id = B
```

Onde `A` será o valor recebido e B será o ID do nó que enviou a mesagem. Este último é obtido com a função `GetNodeId()` do objeto `ValueID`. Caso a instância não seja igual a 2, então é porque o valor é referente ao primeiro canal. Desta forma, torna-se necessário saber se o dispositivo é um *dimmer* pois, caso o seja e o valor recebido seja igual a um, para além de atualizar a coluna `on_off` também tem-se que atualizar a coluna `valor` para igual a 99 (valor máximo de luminosidade). Caso esta última condição não se verifique, então apenas é atualizado a coluna `on_off` de acordo com o valor recebido, ficando memorizado o valor de luminosidade anterior caso se trate de um *dimmer*. Estas *querys*, e todas que serão mencionadas nesta secção, seguem o mesmo formato que a anterior e em todas, a coluna `poll_web` é colocada a um para que a página *web* atualize os dados apresentados nesta, assim como só afetam uma linha, a que corresponde ao nó que enviou a mensagem. Por último, é chamada a função `CheckSceneTrigger()` que irá verificar se este `ValueID` é *trigger* de algum cenário. Esta será descrita posteriormente na secção 4.2.5.

Caso a *command class* seja igual a 0x26, ou seja, *Command_Class_Switch_Multilevel*, é usada a função `GetValueAsByte()` para obter o valor, visto que esta *command class* apenas trabalha com valores de 8 *bits* sem sinal, que será o valor do nível de luminosidade do *dimmer*. De seguida, é verificado se este valor é zero e, caso seja, a coluna `on_off` será atualizada para zero mas o a coluna `valor` não será alterada para que esta guarde o último valor de luminosidade. Caso o valor não seja zero, então a coluna `on_off` será atualizada para um e a coluna `valor` para o valor recebido na notificação. Por último também é evocada a função `CheckSceneTrigger()`.

Quando a *command class* é igual a 0x30, ou seja, *Command_Class_Sensor_Binary*, será usada a função `GetValueAsBool()` de forma a obter o valor reportado, que corresponderá ao estado do sensor binário. Tendo este valor, a coluna `binario` da tabela `sensores_zw`, será atualizada com este valor. E, tal como nos casos anteriores, no final é chamada a função `CheckSceneTrigger()`.

Outra *command class* suportada pelo sistema é a *Command_Class_Sensor_Multilevel* (0x31), pela qual são reportados vários valores medidos por sensores Z-Wave. Quando recebida uma notificação com esta *command class* será usada a função `GetValueUnits()`, para a qual é passada o `ValueID`, que retornará a unidade da grandeza do valor recebido, de forma a identificar este. E, caso a unidade seja igual a "C" ou "lux" então ir-se-á obter o valor usando a função `GetValueAsFloat()`, que é usada para obter valores decimais e com sinal. De seguida, será feita uma *query* de forma a atualizar a coluna da tabela `sensores_zw` com o nome igual à unidade do valor. O nome das colunas, foi atribuído de forma a ser igual às unidades das grandezas, de forma a simplificar o processo de formação da *query*.

Quando a *command class* é igual a 0x32, ou seja, *Command_Class_Meter*, vai-se seguir um processo idêntico à da *command class* anterior, determinando primeiro a unidade da grandeza e depois obtendo o valor. E, caso a unidade da grandeza seja igual a "W", ou "kWh", ou "V", ou "A" ou "kVAh" então ir-se-á usar a função `GetInstance()` para determinar se o valor corresponde ao segundo canal do dispositivo e, caso o seja será realizada uma *query* para atualizar a coluna cujo nome é igual à unidade seguido de "_ch2", caso contrário ir-se-á realizar uma *query* à coluna cujo nome é igual à unidade da grandeza. Se esta não corresponder a nenhuma das mencionadas, então é verificada se é igual a "Power Factor" e, em caso afirmativo, é feita uma *query* para atualizar a coluna `Power_Factor`. Para esta grandeza, não se pôde seguir o mesmo raciocínio que nas grandezas anteriores porque não é possível atribuir o nome da coluna igual ao da unidade devido ao caracter espaço. Desta forma, a aplicação é capaz de interpretar as mensagens mais importantes, do ponto de vista do utilizador, da rede Z-Wave.

4.2.4 Configuração de Dispositivos

Nesta secção será explicado o desenvolvimento do código responsável por efetuar configurações, sendo que por configurações compreende-se a inclusão e exclusão de dispositivos e a configuração de parâmetros e associações. O fluxograma desta secção de código é apresentado na Figura B.4 em Anexo B.

À semelhança de todas as outras secções de código da função `main()`, esta realiza *polling* de uma variável e, dependendo do valor dessa, irá realizar uma determinada ação. Neste caso, essa variável é a coluna `cfg` da tabela `dispositivos_zw`, sendo que, só é usada a primeira linha (referente à *pen* USB). Com isto, existem quatro ações diferentes que podem ser realizadas.

A primeira, é realizada caso `cfg` seja igual a um, o que significa que se pretende entrar em modo de inclusão para adicionar um dispositivo à rede Z-Wave. Desta forma, começa-se por colocar `ad_rm_cnt` igual a zero e `g_flag_add` igual um. A primeira variável será usada para contar tempo, indicando este em segundos, para que, caso o utilizador não efetue o processo de inclusão no dispositivo, o sistema não fique para sempre em modo de inclusão. A segunda variável irá assinalar se o dispositivo foi adicionado ou não. Sempre que um dispositivo é adicionado é recebida uma notificação do tipo `NodeAdded` assim, se tal acontecer, `g_flag_add` será colocada a zero no código desta notificação, que será explicado posteriormente. Com isto, o sistema é colocado em modo de inclusão através da função `AddNode()` para qual é passada o *home* ID. De seguida, o código entra num ciclo que esperará que o utilizador efetue o processo de inclusão do dispositivo. Este ciclo não terminará enquanto `g_flag_add` for igual a 1, `cnt_ad_rm` for menor ou igual a 60 e `cfg` for diferente de 4. A última condição, será falsa, caso o utilizador cancele a operação de inclusão de dispositivos na interface gráfica. Quando tal acontece, a página *web* irá colocar `cfg` igual a 4 para assinalar tal.

Assim, torna-se necessário, no código deste ciclo, verificar se o valor desta coluna mudou, assim como, contar o tempo. Desta forma, este ciclo começa por esperar 1 segundo e, de seguida, incrementa `cnt_ad_rm` em uma unidade para guardar que passou 1 segundo. Após isto, é efetuada uma *query* à tabela `dispositivos_zw` para ler o valor de `cfg` e procede-se à próxima iteração do ciclo. Quando este ciclo terminar, será determinado se algum dispositivo foi adicionado, ou se o ciclo terminou porque o tempo se esgotou ou porque o utilizador cancelou a ação. Para tal, é verificada se `g_flag_add` é igual a zero e, caso o seja, significa que foi adicionado um dispositivo à rede e, por isso, é feita uma *query* para colocar `cfg` igual a zero para assinalar que o processo foi concluído com sucesso, sendo que o resto das operações são realizadas na notificação `NodeAdded` e no código explicado na secção 4.2.1. Caso `g_flag_add` não seja igual a zero, então é verificada se é igual a um, o que significará que

não foi adicionado nenhum dispositivo, pois mantém o valor que lhe foi atribuído no início do ciclo, e que o ciclo terminou por ordem do utilizador ou porque se esgotou o tempo. Quando tal acontece, é feita uma *query* para colocar `cfg` igual a três para assinalar que o processo terminou sem êxito, e é utilizada a função `CancelControllerCommand()` para tirar o sistema do modo de inclusão. Quando é incluído um dispositivo com sucesso, este último passo é realizado na notificação `NodeAdded`.

O segundo caso apresentado no fluxograma, `cfg` igual a dois, não é desenvolvido dado que é igual ao primeiro caso excetuando o facto de se, em vez de colocar o sistema em modo de inclusão, este é colocado em modo de exclusão dado que é o pretendido neste caso. Tal, pode ser alcançado utilizando a função `RemoveNode()` para a qual é passada o *home* ID. E, tal como no caso anterior, esta secção de código também funciona em conjunto com uma notificação, neste caso do tipo `NodeRemoved`.

O terceiro caso, acontece quando `cfg` é igual a 11, o que significa que se pretende configurar um parâmetro de um dado dispositivo. Deste modo, é realizada uma *query* à tabela `config`, onde estão guardados os dados de configuração que o utilizador introduziu na página *web*. Neste caso, serão precisos os valores das colunas `node_id`, `param`, `value`, `size` e `id`. A *query* a realizar será a seguinte:

```
SELECT node_id, param, value, size, id from config WHERE
id=(SELECT MIN(id) FROM config)
```

Onde a condição `WHERE` fará com que apenas se selecione a linha com o menor `id`, ou seja, a mais antiga. Tendo estes valores é usada a função `SetConfigParam()` para configurar o parâmetro. Para esta função é passada o *home* ID, o ID do nó (`node_id`), o número do parâmetro a configurar (`param`), o valor a atribuir a este (`value`) e o tamanho do valor em *bytes* (`size`). Com isto, o parâmetro fica configurado, faltando apenas atualizar a coluna `cfg` para igual a zero, de forma a que na próxima iteração este código não seja executado, e eliminar a linha da tabela que foi lida.

O quarto e último caso, que acontece quando `cfg` é igual a 12, não é desenvolvido no fluxograma dado que é igual ao terceiro caso excetuando as colunas que são lidas da tabela `config` que, neste caso, serão `node_id`, `tg_node`, `instance` e `id`. Para além disso, em vez de ser usada a função `SetConfigParam()`, visto que se pretende fazer uma associação, será usada a função `AddAssociation()`. Para esta será enviada o *home* ID, o ID do nó (`node_id`), o grupo da associação (`grupo`), o ID do nó que se pretende adicionar ao grupo (`tg_node`) e a instância (`instance`).

Com isto, esta secção do código termina com a libertação da memória alocada pelo resultado da *query*.

NodeAdded

Esta parte do código, já mencionada algumas vezes ao longo desta secção, terá como fluxograma o apresentado na Figura B.5 em Anexo B.

Nesta, em primeiro lugar são obtidos os dados sobre o nó que originou a notificação, nomeadamente o *home* ID e o ID do nó, que serão adicionados à lista `g_nodes`. Feito isto, é usada a função `CancelControllerCommand()` para que o sistema saia do modo de inclusão. Até aqui, este código pertence ao *template* fornecido pela biblioteca OpenZWave. De seguida, o valor do ID do nó será guardado na lista `g_new_nodes`. Com isto, é verificado se a variável `g_flag_add` é igual a um, o que significará que a notificação foi gerada por um dispositivo que foi adicionado enquanto o sistema estava em modo de inclusão, e não com o arranque da aplicação, dado que esta variável só toma este valor na secção de código mencionada anteriormente. Assim, quando se verifica esta condição, a variável `g_flag_add` é colocada a zero, para assinalar que foi adicionado um novo dispositivo com sucesso, tal como foi explicado anteriormente.

NodeRemoved

Esta secção de código começa por executar o código base presente no *template* fornecido pela biblioteca OpenZWave. Este consiste na determinação do *home* ID e do ID do nó do dispositivo em causa, e a remoção dos dados referentes a este da lista `g_nodes`. Após isto, é verificada a variável `g_flag_rm`, que corresponde à variável `g_flag_add` mas para a remoção de dispositivos. Caso esta seja igual a um, significa que o sistema encontra-se em modo de exclusão que foi iniciado no código presente na função `main()`. Assim, ir-se-á remover todas as entradas na base de dados que sejam referentes ao nó com o ID igual ao que deu origem a esta notificação, realizando três *querys*, uma para cada tabela, `sensores_zw`, `atuadores_zw` e `dispositivos_zw`. As *querys* terão o seguinte formato, sendo que apenas muda o nome da tabela entre as três:

```
DELETE FROM sensores_zw WHERE node_id = node_id
```

De realçar que, a entrada da tabela `dispositivos_zw` tem de ser eliminada em último lugar, dado que esta contém uma coluna que é *foreign key* nas outras duas. Feito isto, `g_flag_rm` é colocada a zero para assinalar que o dispositivo foi removido com sucesso e é usada a função `CancelControllerCommand()` para retirar o controlador do modo de exclusão. O fluxograma desta parte do código pode ser visualizado na Figura 4.9

4.2.5 Cenários

Apesar de a biblioteca OpenZWave providenciar funções para controlar e gerir cenários Z-Wave, as capacidades destas são bastante rudimentares. Isto faz com que, para implementar cenários, grande parte da implementação fique ao encargo

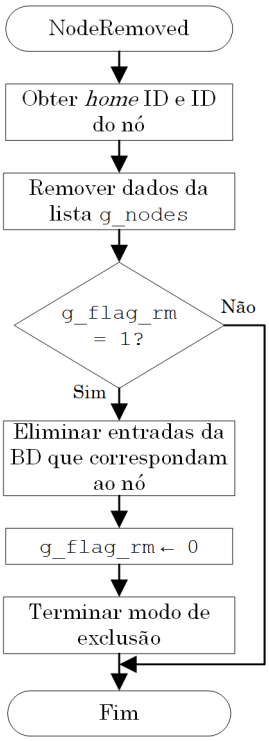


Figura 4.9: Fluxograma do código das notificações do tipo `NodeRemoved`

do desenvolvedor que utilize esta biblioteca. Com isto, as funções disponíveis permitem criar cenários, adicionar `ValueIDs` ao cenário e ativar o cenário, sendo que esta última ação fará com que seja atribuído um valor a todos os `ValueIDs` adicionados, o que por sua vez atuará os dispositivos. O valor a atribuir é definido no momento de adição do `ValueID` ao cenário. Este modo de funcionamento é uma limitação das funções da biblioteca, pois não permite adicionar e identificar os *triggers* dos cenários. Outra limitação deve-se com o facto de, caso a aplicação termine, por exemplo por perda de energia, todos os cenários são perdidos. Desta forma, estas limitações terão de ser contornadas, neste caso, será recorrendo à base de dados. Assim, na Figura 4.10 é apresentado o fluxograma do código responsável por carregar todos os cenários que estejam guardados na base de dados.

Esta secção de código começa por fazer uma pesquisa na tabela `scenes`, da base de dados, por todos os ID's de cenários (`scene_id`). De seguida, entrará num ciclo que, para cada um destes ID's, irá criar um cenário usando a função `CreateScene()`, e irá atualizar o `scene_id`, com o valor retornado por esta que corresponde ao ID do cenário criado, nas três tabelas referentes aos cenários. Este último passo é necessário pois, caso da última vez que aplicação esteve a correr, houve um cenário, que não o último (maior ID), que tenha sido eliminado, todos os cenários com ID maior que este, serão criados com um ID diferente do

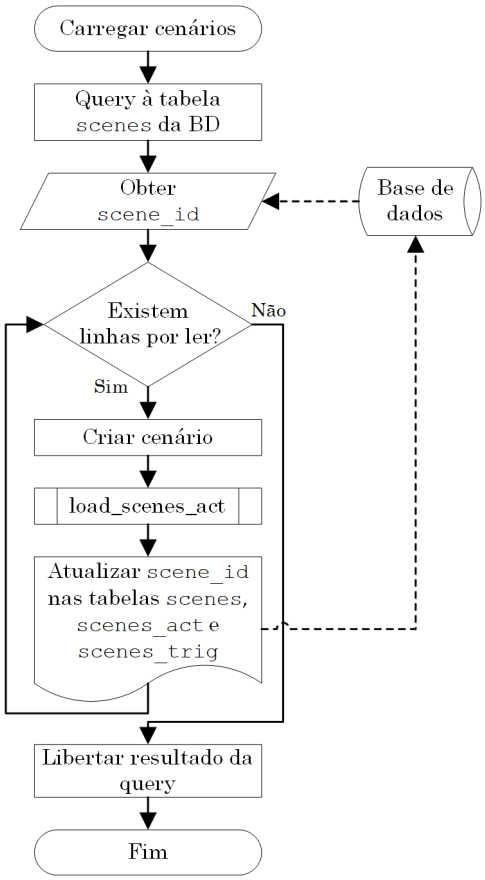


Figura 4.10: Fluxograma do código de carregamento dos cenários

guardado na base de dados. Para além disso, irá carregar todos os `ValueIDs`, e respetivos valores a atribuir a estes quando o cenário é ativado, através da função `load_scenes_act()`.

load_scenes_act()

Esta função será responsável por adicionar a um determinado cenário os `ValuesIDs` guardados na tabela `scenes_act`. Deste modo, esta função receberá como parâmetros o ID do cenário e a variável `con` para comunicar com a base de dados. O fluxograma desta função pode ser visualizado na Figura 4.11.

Neta função, o primeiro passo será obter todas as entradas na tabela `scenes_act`, que tenham a coluna `scene_id` igual ao ID do cenário recebido como parâmetro. De seguida, entrar-se-á num ciclo que irá percorrer todas as entradas obtidas e, por cada uma destas irá identificar se a *command class* (`cc`) é igual a 0x25 ou 0x26 e, dependendo do resultado irá chamar a função `Scene_AddBoolVal()` para o primeiro caso, ou a função `Scene_AddByteVal()` para o segundo. Estas duas funções, têm um princípio de funcionamento igual ao de outras explicadas anteriormente, como a

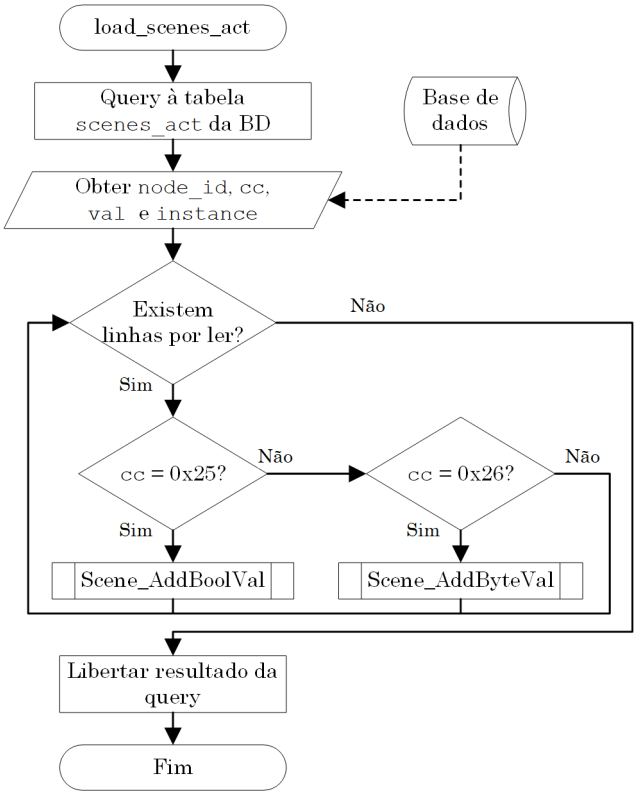


Figura 4.11: Fluxograma da função `load_scenes_act()`

`SetValueBool()`, no sentido em que também irão percorrer a lista `g_nodes` até encontrar um `ValueID` com o ID do nó, a *command class* e a instância pretendidas. Tendo encontrado o `ValueID` irá chamar a função `AddSceneValue()` que adicionará um valor ao cenário. Para esta função é passado o ID do cenário, o `ValueID` e o valor a atribuir a este. Percorridas todas as linhas resultantes da *query*, o ciclo termina e a memória alocada pelo resultado desta é libertada e a função termina.

Configuração de Cenários

Esta secção de código está presente na função `main()` como já apresentado na Figura 4.6. E, esta, tal como todas as outras secções de código na função `main()` irá efetuar *polling* de uma coluna, de uma tabela da base de dados, que indica que tipo de configuração se pretende realizar. De realçar que, por configuração, neste caso, entende-se a criação, edição e eliminação de cenários. O fluxograma desta secção de código pode ser visualizado na Figura B.6 do Anexo B.

Com isto, esta secção de código começa por efetuar uma *query* à tabela `scenes` de forma a obter todas as entradas nesta, que serão uma por cada cenário. De seguida, é usado um ciclo *while* para percorrer cada uma destas entradas, começando por fazer *switch case* da coluna `opt` que pode tomar três

valores diferentes: um, três ou quatro sendo que qualquer outro indica que não se pretende realizar nenhuma ação. Quando igual a um, significa que a entrada em questão corresponde a um cenário novo que tem de ser criado. Assim, ir-se-á criar este através da função `CreateScene()`, de seguida atualiza-se `scene_id` nas três tabelas referentes aos cenários e, por fim, é chamada a função `load_scenes_act()`. Esta parte do código é igual ao explicado anteriormente para o carregamento dos cenários, durante a inicialização da aplicação. Para além disso, a coluna `opt` é atualizada para o valor de zero para que na próxima iteração este código não seja executado.

Caso `opt` seja igual a três, significa que o utilizador introduziu na interface gráfica um novo atuador para ser adicionado no cenário. Assim, é realizada uma *query* à tabela `scenes_act`, de forma a obter a entrada mais recente desta tabela, que será a que tiver a coluna `id` com o maior valor, referente ao cenário em questão. Esta *query* será a seguinte:

```
SELECT node_id, cc, val, instance FROM scenes_act WHERE
id=(SELECT MAX(id) FROM scenes_act WHERE scene_id = scene_id
```

A codição `WHERE` fará com que seja selecionada a linha com o maior `id` das que têm `scene_id` igual ao pretendido. Tendo os dados do novo atuador, é determinado se a *command class* é igual a 0x25 ou 0x26 e, dependendo do resultado é chamada a função `Scene_AddBoolVal()` para o primeiro caso, ou `Scene_AddByteVal()` para o segundo. Esta parte do código é igual à explicada anteriormente na função `load_scenes_act()`. Com o novo atuador adicionado ao cenário, resta atualizar `opt` para igual a zero e libertar a memória alocada pelo resultado da *query*.

Quando `opt` é igual a quatro, quer dizer que o utilizador eliminou o cenário referente a esta linha da tabela. Desta forma, é utilizada a função `RemoveScene()` para a qual é passada o ID do cenário que se pretende eliminar, apagando este. Feito isto, são eliminadas todas as entradas nas tabelas `scenes`, `scenes_act` e `scenes_trig` que tenham a coluna `scene_id` igual ao ID do cenário eliminado. Percorridas todas as linhas do resultado da primeira *query*, o ciclo *while* termina e liberta-se a memória alocada pelo resultado desta *query*.

Check_SceneTrigger()

Como se pode verificar, os *triggers* não usados pela aplicação nem no momento de criação, nem de configuração de um cenário. Estes, serão usados apenas nesta função `Check_SceneTrigger()`, que será usada nas notificações do tipo `ValueChanged`, tal como já foi referido na secção 4.2.3. O fluxograma desta função está representado na Figura 4.12.

Esta função recebe como parâmetros o ID do nó, a *command class*, o valor e a instância, do `ValueID` que originou a notificação. Com estes, irá pesquisar

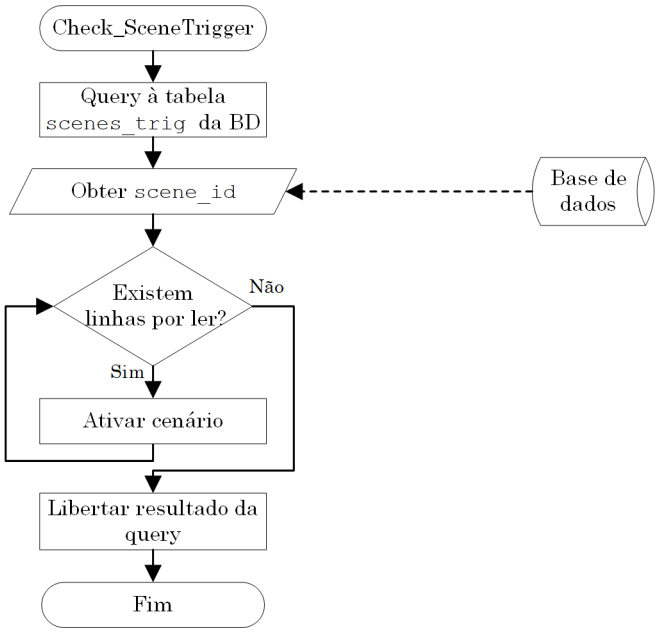


Figura 4.12: Fluxograma da função `Check_SceneTrigger()`

se na tabela `scenes_trig` existe alguma linha com os dados iguais a estes, ou seja, se estes dados são *triggers* de algum cenário. Para tal, é efetuada a seguinte *query*:

```
SELECT scene_id FROM scenes_trig WHERE node_id = A AND cc =
B AND val = C AND instance = D
```

Onde, A, B, C e D correspondem aos valores do ID do nó, da *command class*, do valor e da instância recebidos como parâmetros. Se esta *query* retornar algum resultado significa que existe um cenário que tem um *trigger* com estes dados. Desta forma, será usado um ciclo *while* para percorrer todas as linhas que a *query* retornou, e para cada uma dessas linhas é usado o `scene_id` correspondente para ativar o cenário. Tal, pode ser feito através da função `ActivateScene()` que recebe como parâmetro o ID do cenário. Terminado o ciclo *while*, tal como em todas as outras *querys* do tipo `SELECT` anteriores, é libertada a memória alocada pelo resultado desta.

4.2.6 Sistema Térmico

A implementação do *software* para o sistema térmico consiste num controlador PI para controlar a temperatura. Um dos aspetos mais importantes na implementação de um controlador PI num sistema digital é o tempo de amostragem. Este tem de ser suficientemente baixo para acompanhar a frequência de variação do sistema e quando este é constante obtêm-se melhores resultados. Assim,

para assegurar que o controlador PI é sempre executado com o mesmo tempo de amostragem recorreu-se a uma *thread*.

Esta, irá conter um ciclo infinito no qual será implementado o algoritmo do controlador. Neste ciclo, será usada a função `delay()`, da biblioteca WiringPi, para a qual é passado o número de milissegundos que se pretende esperar, neste caso, o tempo de amostragem. Contudo, esta abordagem não garante que o tempo entre iterações do controlador seja constante, pois não tem em consideração o tempo de processamento do código do controlador. Assim, de forma a melhorar este aspeto, em cada iteração é calculado este tempo de processamento e é descontado no valor passado à função `delay()`. Isto, pode ser alcançado recorrendo à função `millis()` da biblioteca WiringPi. Esta, retorna o tempo em milissegundos que decorreu desde que a aplicação iniciou, assim, memorizando este valor no início de cada iteração do controlador PI, e subtraindo-o ao retornado pela mesma função no final da iteração, obtém-se o tempo de processamento do controlador PI que deve ser subtraído ao tempo de amostragem.

Leitura do ADC

Antes de se proceder à explicação da implementação do controlador PI, é necessário explicar o processo de medição da temperatura, sendo o primeiro passo a leitura do ADC. Assim, foi desenvolvida uma função `lerAD()` que recebe como parâmetro o canal que se pretende amostrar e que irá enviar uma mensagem, via protocolo SPI, para o módulo MCP3008. Para tal, é usada a função `WiringPiSPIDataRW()`, para a qual deve ser passado o canal SPI a usar, a mensagem a enviar e o tamanho desta. A mensagem a enviar, tal como mencionado anteriormente, começará com o *start bit* seguido pelo *bit* SGL/DIFF que neste caso será 1, para fazer uma conversão não diferencial. Estes, serão seguidos pelos 3 *bits* de seleção do canal do ADC, que neste caso serão todos 0, visto que, se pretende o CH0 que é o canal ao qual o sensor de temperatura está conetado. A formação desta mensagem, em pseudocódigo, será:

```
buffer[0] = 1;
buffer[1] = (8 + CanalADC) « 4;
buffer[2] = 0;
```

Onde `buffer[0]` é o *start bit* e `buffer[1]` contém os restantes 4 *bits*, sendo que o "8" corresponde ao *bit* SGL/DIFF, a operação total corresponderá a:

8	→	0	0	0	0	1	0	0	0	
CH0	→	0	0	0	0	0	0	0	0	+
		0	0	0	0	1	0	0	0	
										<<4
buffer[1]	→	1	0	0	0	0	0	0	0	

Os *bits* circundados correspondem aos *bits* SGL/DIFF, D0, D1 e D2, da esquerda para a direita respetivamente. A variável com a mensagem a enviar será usada para guardar a resposta e, esta terá de comprimento três *bytes*, daí a definição de `buffer[2]`, para que a resposta caiba na variável. Deste modo, `buffer[0]` corresponderá ao NULL *bit*, `buffer[1]` conterá, nos seus dois *bits* mais à direita os dois *bits* mais significativos da conversão e `buffer[2]` conterá os restantes 8. Assim, a função `lerAD()` irá retornar a soma de `buffer[1]` shiftado 8 casas para a esquerda com `buffer[2]`, resultando num número de 10 *bits*.

Tendo este valor, é necessário convertê-lo de *bits* para mV utilizando a seguinte equação:

$$V_T = \frac{T_{bits} \times 3300}{1024} \tag{4.1}$$

Onde, V_T corresponde ao sinal de saída do sensor de temperatura, T_{bits} ao valor do resultado da conversão do ADC, em *bits*, 3300 corresponde à tensão de referência do ADC, 3,3 V, em mV e 1024 é a resolução do ADC. Após este cálculo, a temperatura, em graus Celsius, pode ser calculada com a equação 3.2

Controlador PI

Sendo possível obter o valor da temperatura, procede-se à implementação do algoritmo do controlador PI. O fluxograma da *thread* responsável pelo controlo da temperatura, `ThreadTemp` está representado na Figura 4.13.

Esta *thread*, após entrar no ciclo infinito *while*, e após usar a função `millis()` como explicado anteriormente, irá calcular a temperatura através da função `lerAD()` e das equações 4.1 e 3.2. Com este valor irá calcular o erro, desta grandeza, em relação ao valor de referência que se definiu como 40 °C. O erro por sua vez é usado no algoritmo do controlador PI. Este, foi implementado usando a equação 3.4, mas sem a ação derivativa. No entanto, visto que se está a trabalhar num sistema digital, é necessário discretizar o termo integral:

$$I(t_{a+1}) = I(t_a) + k_i \times h \times e(t_a) \tag{4.2}$$

Onde t_a é instante de amostragem e h o tempo de amostragem. O sinal de controlo será calculado segundo a equação 4.3.

$$u(t_a) = k_p \times e(t_a) + I(t_{a+1}) \tag{4.3}$$

Após o cálculo do sinal de controlo é necessário aplicar uma saturação a este de forma a atuar o módulo Z-Wave com um valor entre 0 e 99, que é o intervalo admitido por estes módulos. Para tal, verifica-se o valor calculado pelo

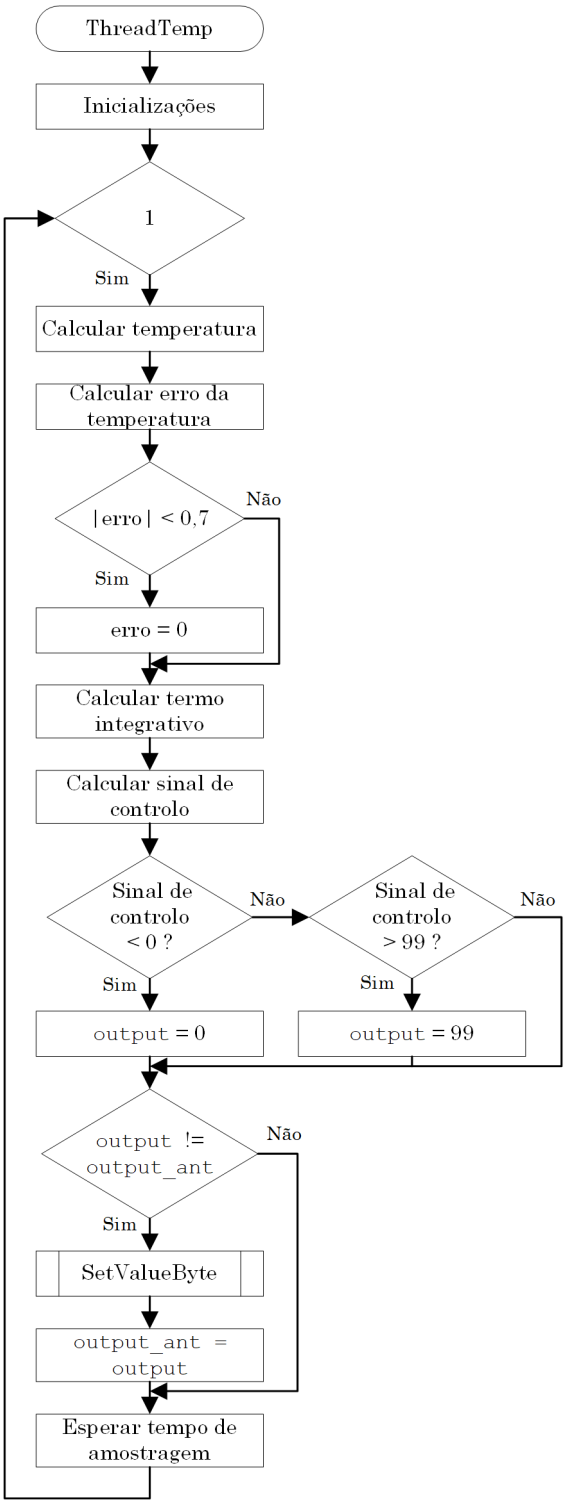


Figura 4.13: Fluxograma da *thread* do controlador PI para a temperatura

controlador, caso este seja maior que o limite de saturação superior, ou seja, 99, então a este valor será atribuído um valor igual a este limite. O mesmo raciocínio é aplicado para o caso de o valor do controlador ser menor que o limite de saturação inferior. Quando o valor calculado pelo controlador está entre os limites de saturação nenhuma ação é necessária.

Para reduzir a quantidade de mensagens Z-Wave enviados aos módulos, introduziu-se uma verificação ao valor do sinal de controlo para que, caso este seja igual ao calculado na iteração anterior do controlador, este não seja enviado para o módulo. Caso contrário, o módulo Z-Wave será atuado usando a função `SetValueByte()`, e o valor enviado será guardado para a próxima iteração. Feito isto, é usada a função `delay()` para esperar o tempo de amostragem pretendido, tal como explicado anteriormente.

De realçar que, a determinação do valor para os ganhos k_p e k_i, assim como, do tempo de amostragem, foi feita através da realização de várias experiências até se chegar a valores que proporcionassem uma boa resposta por parte do sistema. Para além disso, de realçar que ao erro foi adicionada uma *deadband* de forma a ajudar o sistema estabilizar mais facilmente o que, embora torne a estabilização mais lenta, irá reduzir o número de mensagens Z-Wave na rede.

4.2.7 Sistema de Luminosidade

A implementação do *software* para o sistema de luminosidade é bastante similar com o do sistema térmico. As diferenças estarão na medição da grandeza e no tipo de controlador PI, que será um mais simples.

Para obter a grandeza, é usada na mesma a função `lerAD()` mas, desta vez, enviando-lhe como parâmetro o número 1, que corresponderá ao canal do ADC que se pretende amostrar. De seguida, este valor terá de ser convertido de *bits* para Volts através da seguinte equação:

$$V_{Lux} = \frac{L_{bits} \times 3.3}{1024} \tag{4.4}$$

Onde, V_{Lux} corresponde ao sinal de saída do circuito do LDR, L_{bits} ao valor do resultado da conversão do ADC, em *bits*, 3,3 corresponde à tensão de referência do ADC, 3,3 V, e 1024 é a resolução do ADC. Tendo este valor, é usada a equação 3.9 para calcular a resistência do LDR, R_{LDR}, que permitirá calcular a luminosidade em Lux através da equação 3.15.

Para além disso, é aplicada uma *deadband* ao erro. Esta, é implementada verificando o erro, antes do cálculo do controlador, e, caso este esteja dentro dos limites da banda, que se definiu como sendo $\pm$ 10 Lux, ao erro será atribuído o valor de 0. A *deadband*, é aplicada devido ao facto de a luminosidade ser uma grandeza muito sensível, o que significa que o seu valor oscila. Consequentemente,

o sinal de controlo irá igualmente oscilar para tentar corrigir a luminosidade. Contudo, isto implica que sejam enviadas bastantes mensagens pela rede Z-Wave, de forma a tentar obter um valor preciso que não farão diferença num caso prático. Ao introduzir a *deadband* o sinal de controlo irá estabilizar num valor que manterá a luminosidade dentro de uma gama de valores.

O controlador PI implementado para este sistema foi exatamente igual ao do sistema de temperatura. Tal como no sistema anterior, após os cálculos do controlador, é necessário aplicar uma saturação ao sinal de controlo, também com os limites inferior e superior de 0 e 99, respetivamente. E, também para este sistema, a determinação dos ganhos e do tempo de amostragem foi realizada de forma prática.

4.2.8 Envio da Temperatura e Luminosidade para a Base de Dados

Para colocar os valores da luminosidade e de temperatura na base de dados, é utilizada mais uma *thread* para não sobrecarregar os controladores PI, nem atrasar as ações da função `main()`. Assim, esta *thread*, `ThreadSQLWrite_phy()`, começa por inicializar a conexão com a base de dados e entra num ciclo infinito *while* no qual irá enviar os dados. O fluxograma desta função pode ser visualizada na Figura 4.14.

O ciclo *while* desta *thread* terá uma temporização de um segundo, ou seja, o código dentro deste é executado apenas a cada segundo. Com isto, neste ciclo, começa-se por verificar se a diferença entre o valor atual da luminosidade, e o último que foi enviado para a base de dados, é maior que 15 Lux e, apenas caso o seja, é que o valor será colocado na base de dados. É realizado este processo para que apenas mudanças significativas no sistema sejam registadas na base de dados, não enchendo esta com valores praticamente iguais, assim como, para que o valor apresentado não esteja sempre a mudar. O novo valor será colocado na coluna `valor`, da tabela `sensores_phy`, da linha referente ao sensor de luminosidade. Para a temperatura, o processo é o mesmo, mas a diferença entre a temperatura atual e a última colocada na base de dados, apenas tem de ser maior que 0,8 °C.

4.3 Página Web

A página *web* é o que servirá como interface entre o utilizador e o sistema, permitindo ao primeiro atuar dispositivos e observar informações acerca destes.

4.3.1 Acesso à Base de Dados

Grande parte das funcionalidades da página *web* recorrem à base de dados, visto que, é através desta que são enviadas informações para a aplicação, que é o elemento que tem capacidade para controlar todo o sistema. Assim, foi

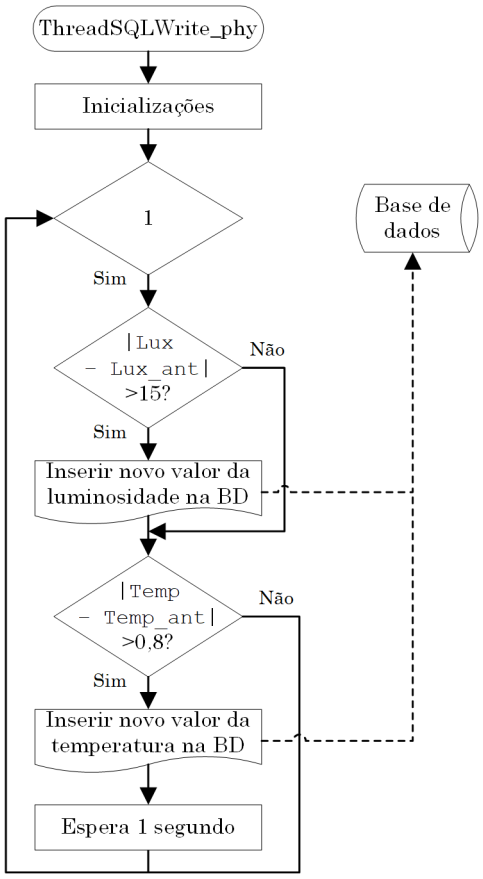


Figura 4.14: Fluxograma da *thread* `ThreadSQLWrite_phy()`

desenvolvido um ficheiro PHP, `AcessoDB.php`, que irá iniciar a conexão com a base de dados. Para tal, é usada a função `mysqli_connect()` que tem como parâmetros de entrada o nome do servidor, nome de utilizador, palavra-passe e o nome da base de dados. Este ficheiro, irá ser incluído em todos os outros que tenham a necessidade de aceder à base de dados.

4.3.2 Construção da Interface Gráfica

No desenvolvimento da interface, recorreu-se à biblioteca de *Cascading Style Sheets* (CSS) Bootstrap, que permite o desenvolvimento de páginas *web* com uma interface mais apelativa na perspetiva do utilizador. A página principal do *web*site pode ser visualizada na Figura 4.15.

Como visível na figura, esta é constituída por uma barra de navegação no seu topo com três opções: "Adicionar dispositivo", "Remover dispositivo" e "Cenários". E, tal como o nome indica, as duas primeiras opções irão permitir adicionar e remover dispositivos, respetivamente, apresentando ao utilizador uma janela sobre o conteúdo da página, denominada de modal, que mostrará ao uti-

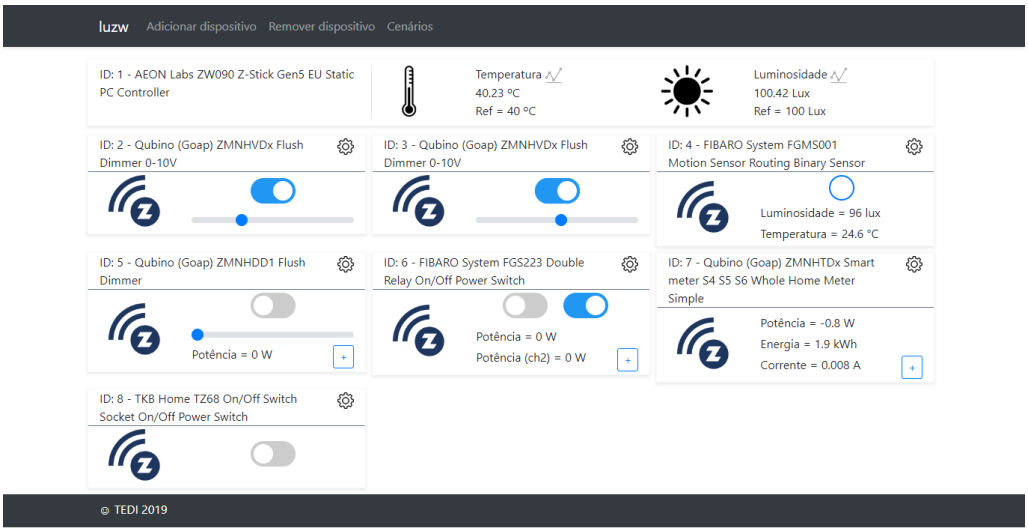


Figura 4.15: Página principal do *website*

lizador dados sobre o estado do processo. A terceira opção irá redirecionar para uma página diferente, na qual será possível configurar cenários.

Abaixo desta barra de navegação, é apresentada uma primeira linha que, apesar de ser construída com os dados presentes na base de dados, será fixa. Esta, está dividida em três colunas. A primeira, apresenta o ID do nó da *pen* USB e o nome desta. As duas outras colunas apresentam os dados referentes aos sensores conetados fisicamente ao Raspberry. Em cada uma destas colunas também é possível observar um pequeno símbolo ao lado do nome das grandezas, que quando clicado irá abrir um modal com um gráfico da respetiva grandeza ao longo do tempo. A colocação desta linha, e todos os dados nesta, é feita incluíndo um ficheiro PHP denominado de `colocar_phy.php`, que irá pesquisar na base de dados todos os dados e colocá-los na página principal.

Na zona inferior a esta, é visível a área onde são colocados todos os dispositivos Z-Wave que estão emparelhados com o sistema ou, do ponto de vista da página *web*, os dispositivos que estão na tabela `dispositivos_zw`. Esta área é uma grelha de 3 colunas, em que cada elemento desta grelha corresponde a um dispositivo. Em cada um destes é apresentado, na parte superior, o ID do nó (dispositivo), o respetivo nome e ainda uma imagem de uma roda dentada, que quando clicada irá abrir um modal que permitirá configurar parâmetros e associações do respetivo dispositivo. Na parte inferior da coluna, são colocados todos os elementos necessários para interagir com o dispositivo, como por exemplo, botões *toggle*, barras de *range* para definir o valor de um dispositivo regulável, texto para apresentar informações de sensores, como potência, energia, temperatura, entre outros. A formação desta grelha com os dispositivos Z-Wave é realizada, incluíndo um ficheiro PHP denominado de `colocar_dev` que irá pesquisar na

base de dados toda a informação e colocá-los na página principal.

colocar_phy

Este ficheiro, começa por realizar uma *query* usando a função `mysql_query()`, que recebe como parâmetros a variável da conexão à base de dados, criada no ficheiro de acesso à base de dados, e uma *string* com a *query* a realizar. Esta última, será feita à tabela `dispositivos_zw`, de forma a obter a entrada que tiver a coluna `node_id` igual a 1, que corresponderá à *pen* USB. De seguida, utiliza a função `mysqli_fetch_assoc()` que irá retornar uma linha do resultado da *query* sobre a forma de um vetor. Tendo os dados acerca da *pen* USB, estes são colocados na página *web* fazendo o *output*, através da função `echo`, do código HTML que deveria ser colocado no ficheiro da página principal, realizando concatenações com as variáveis que guardam os dados da *pen* USB sempre que se pretender fazer o *output* destes dados. Em PHP, a concatenação de dois argumentos é realizada através do operador ".". Este processo de fazer o *output* de código HTML, concatenado com variáveis, é a forma de apresentar dados de um ficheiro PHP para uma página *web*, e será usado várias vezes ao longo deste implementação.

Com isto, falta colocar os dados acerca dos sensores de temperatura e luminosidade na página *web*. Para tal, é realizada uma segunda *query*, desta vez à tabela `sensores_phy`, de forma a obter todos os dados nesta tabela. Tendo estes, segue-se o mesmo processo que anteriormente, realizando o *output* do código HTML de forma a apresentar os dados obtidos. O fluxograma deste ficheiro pode ser visualizado na Figura 4.16.

colocar_dev

Este ficheiro, começa por realizar uma *query* à tabela `dispositivos_zw`, de forma a obter os dados relativos aos dispositivos Z-Wave emparelhados com o sistema. Após isto, será usado um ciclo *while*, que irá executar um número de vezes igual ao número de linhas do resultado da *query*, sendo a condição de término o valor retornado pela função `mysqli_fetch_assoc()`. Quando esta retorna NULL, em vez de uma linha do resultado da *query*, significa que já não há mais linhas por ler. Neste ciclo, começa-se por identificar o dispositivo que, tal como foi referido anteriormente, é realizado através da coluna `generic`. Desta forma, é realizado um *switch case* desta coluna e, caso esta seja igual a 16, significa que o dispositivo se trata de um *Binary Switch*. No entanto, este pode ser de dois, ou de apenas um canal, assim, é realizada uma *query* à tabela `atuadores_zw` de forma a obter os valores de `on_off` e `on_off_ch2`. Caso esta última seja diferente de NULL, significa que o dispositivo tem dois canais, assim, resta determinar o estado das saídas (se estão ligadas ou desligadas), verificando os valores destas duas colunas, e fazer o *output* dos botões *toggle*, de acordo com o estado das saídas. Caso `on_off_ch2` seja igual a NULL, o dispositivo só tem um canal.

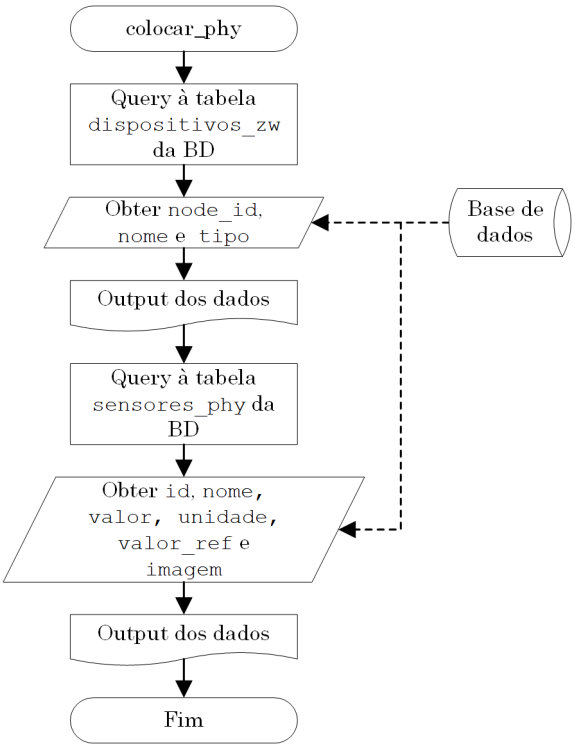


Figura 4.16: Fluxograma do ficheiro `colocar_phy.php`

Assim, é verificada a coluna `on_off` para determinar o estado da saída e é feito o *output* do botão para ligar e desligar esta saída. Os botões *toggle* correspondem a *checkboxes* modificadas com CSS para parecerem-se com tal. Assim, o estado destes botões é definido através do atributo `checked`.

Quando a coluna `generic` é igual a 17, significa que se trata de um atuador regulável, ou seja, de um *Multilevel Switch*. Assim, é realizada uma *query* à tabela `atuadores_zw`, de forma a obter as colunas `on_off` e `valor`. Com estes, ir-se-á determinar o estado do dispositivo para que se coloque um botão *toggle* e uma barra de *range*, que permitirá regular a saída do dispositivo, de acordo com o estado deste. O valor da barra de *range* é definido através do atributo `value`. Para além disso, são usados os atributos `max` e `min` nestas barras, para que estas apenas variem o seu valor apenas entre 0 e 99.

Caso a coluna `generic` seja igual a 32, então o dispositivo é um sensor binário. Deste modo, é feita uma *query* à tabela `sensores_zw` para obter o valor da coluna `binario` correspondente ao dispositivo. O valor desta coluna servirá para determinar o estado do dispositivo e, caso seja igual a 1, irá ser desenhado na página um círculo azul. Caso contrário, será desenhado um círculo com o seu perímetro a azul e o seu interior a branco. Este círculo é desenhado recorrendo a *Scalable Vector Graphics* (SVG). Com isto, termina o *switch case* e

procede-se à verificação se o dispositivo tem mais algumas funcionalidades para além da sua principal, como medição de temperatura, medição de potência, entre outros.

Para tal, é verificado se a coluna `CC_sensor_multilevel` é igual a um, o que significaria que o dispositivo suportaria esta *command class* e, portanto, este mede e reporta algum tipo de grandeza. Neste caso, apenas são suportados a medição de temperatura e de luminosidade. Então, caso o dispositivo suporte esta *command class*, começa-se por efetuar uma *query* à tabela `sensores_zw` pelas colunas `C` e `lux`. De seguida, é verificada a variável, `elem_cnt`, esta conta o número de elementos (botões, barras de *range*, etc), referentes ao dispositivo em causa, que já foram colocadas na interface gráfica, para que, caso já existam três elementos os restantes sejam colocados noutra parte para não sobrecarregar a apresentação dos dados. Assim, caso esta variável tenha um valor menor que três e o valor da coluna `lux` seja diferente de NULL, então será realizado o *output* dos dados acerca da luminosidade e a variável `elem_cnt` será incrementada em uma unidade. E, caso esta variável continue menor que três e o valor da coluna `C` seja diferente de NULL, então será realizado o *output* dos dados desta grandeza. Quando a variável `elem_cnt` for maior ou igual a três, e ainda houver dados por colocar na interface, é colocado um botão que permitirá abrir e fechar uma nova área (Figura 4.17) na qual serão colocados os dados restantes.

Figura 4.17: Elemento referente a um *Smart Meter* não expandido, à esquerda, e elemento referente a um *Smart Meter* expandido

Para além da *Command_Class_Sensor_Multilevel*, também é verificada a *Command_Class_Meter*. E, caso o dispositivo suporte esta *command class*, é feita uma *query* à tabela `sensores_zw` de forma a obter os valores das colunas `W`, `kWh`, `V`, `A`, `Power_Factor` ,`kVAh`, `W_ch2` e `kWh_ch2`. Após esta *query*, é utilizado um ciclo *for* para colocar os dados relativos a todas estas colunas na página *web*. Neste ciclo, começa-se por verificar se o valor da coluna em questão é igual a NULL, pois caso o seja, não se pretende fazer o *output* deste procedendo-se para a próxima iteração do ciclo. Se for diferente de NULL, então começa-se por ver-

ificar se `elem_cnt` é menor que três e, em caso afirmativo, é feito o *output* dos dados e esta variável é incrementada. Em caso negativo, então será verificada a variável `btn_flag` e, caso esta seja diferente de um, significa que ainda não foi colocado o botão que permite abrir e fechar a nova secção com os restantes dados e, portanto, será feito o *output* deste e a variável `btn_flag` será colocada a um para assinalar tal, para a próxima iteração. Independentemente do valor desta variável, é realizado o *output* do valor da coluna em questão, na nova área expansível.

De realçar que a todos os campos colocados na interface, isto é, botões *toggle*, barras de *range*, campos de texto, etc, é atribuído um `id` de acordo com o campo em questão. Este permite identificar, que tipo de campo se trata e o ID do nó a que corresponde, para que mais tarde se possa interagir e identificar os elementos. Por exemplo, o botão *toggle* do dispositivo com o ID quatro terá como `id`: "`tog_bt_4`". No caso do elemento que apresenta a potência consumida pelo dispositivo com ID cinco terá como `id`: "`W1_5`".

Todas as alterações a realizar com estes campos, quer seja atualizar com dados novos vindos da aplicação, quer seja enviar dados para a aplicação, irão recorrer a AJAX para comunicar com o servidor. E, dado que a realização de pedidos ao servidor através de AJAX será repetido várias vezes, foi realizada uma função para efetuar tais pedidos, denominada de `ajax()`.

ajax()

Esta função recebe como parâmetros uma *string*, que corresponderá ao url ao qual deve ser realizado o pedido, e uma função de *callback* que será chamada para interpretar a resposta do servidor quando esta for recebida. A função `ajax()` começa por criar um objeto XMLHttpRequest e, com este, será definida uma função para ser executada quando for recebida uma resposta. Tal, pode ser alcançado utilizando a propriedade `onreadystatechange`. Dentro desta função, é verificado se a propriedade `readystate` é igual a quatro, o que significa que o pedido terminou e a resposta está pronta, e se a propriedade `status` é igual a 200, o que significa que não ocorreram quaisquer erros. Se estas duas condições se verificarem, então é executada a função de *callback*, recebida como parâmetro, passando-lhe a resposta recebida do servidor. Após a definição da função, é efetuado o pedido ao servidor. Primeiro é usado o método `open()`, do objeto XMLHttpRequest, que irá definir o tipo de pedido. Para este método é passado como parâmetro o tipo de pedido, neste caso GET, o url para a localização do ficheiro no servidor e, como terceiro parâmetro, é enviado `true` para especificar que o pedido é assíncrono. Por fim, é usado o método `send()` para enviar o pedido.

Por vezes, também são realizados pedidos ao servidor, através de AJAX, apenas de escrita, ou seja, não sendo necessário qualquer resposta por parte do

servidor. Assim, foi criada uma outra função `submit_ajax()`, que recebe um url como parâmetro, cujo código é igual ao da anterior mas sem a parte referente à resposta do servidor, Isto é, apenas cria o objeto XMLHttpRequest, define o tipo de pedido e envia este.

4.3.3 Adicionar e Remover Dispositivos

Tal como mencionado anteriormente, a inclusão e remoção de dispositivos pode ser efetuada através das respetivas opções na barra de navegação que, quando clicadas, irão fazer surgir um modal, como representado na Figura 4.18

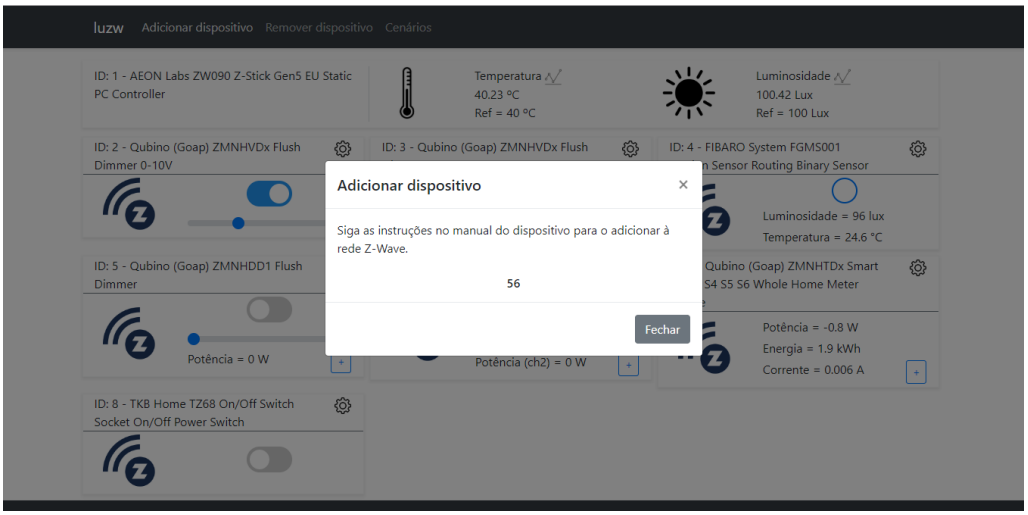


Figura 4.18: Janela de inclusão de dispositivos

Para além de abrirem este modal, as opções "Adicionar dispositivo" e "Remover dispositivo", têm um evento de JavaScript do tipo `onclick` que irá executar a função `modal()`, enviando como parâmetro um 1 no caso da primeira opção, e um 2 no caso da segunda opção, para identificar qual opção acionou a função.

Esta função começa por identificar o valor recebido como parâmetro e, de acordo com este, irá colocar o texto no modal. Isto é, o modal utilizado é o mesmo tanto para adicionar dispositivos como para remover, assim, quando esta função é executada, é necessário colocar o texto adequado de acordo com a operação. Para tal, é atribuída à propriedade `innerHTML`, do método `getElementById` do objeto `document`, o texto a colocar. Por exemplo, para definir o título do modal(a parte a negrito na Figura 4.18):

```
document.getElementById("ModalTitle").innerHTML = "Adicionar
dispositivo";
```

Onde `ModalTitle` é o `id` atribuído à *tag* correspondente à zona de colocação do título. Feito isto, é realizado um pedido ao servidor através da função `submit_ajax()` de forma a atualizar a coluna `cfg`, na tabela `dispositivos_zw`, na linha correspondente à *pen* USB. Para tal, é enviado o seguinte url para a função:

```
"update.php?tbl=dispositivos_zw&col=cfg&val=" + id_ad_rm +
"&node_id=1"
```

Assim, o pedido irá executar o ficheiro `update.php` no qual é efetuada uma *query* `UPDATE`, à base de dados, de forma a atualizar um único elemento desta, de acordo com os parâmetros recebidos através do url. Estes são identificados no url, sendo que `tbl` indica a tabela, `col` a coluna, `val` o valor a atribuir e `node_id` servirá para identificar a linha da tabela. Sendo que `id_ad_rm` é o valor recebido como parâmetro pela função `modal()`, ou seja, será 1 ou 2. Tendo colocado a coluna `cfg` igual a 1, ou a 2, a aplicação colocará o sistema em modo de inclusão, ou de exclusão, respetivamente.

Desta forma, de seguida é usada a função de JavaScript `SetInterval()`, que executa uma dada função a um intervalo de tempo especificado. Assim, com esta, colocar-se-á a função `ajax()` a ser executada a cada 500 ms, enviando como parâmetro o url para o ficheiro `AJAX_ad_rm_end.php` e enviando como função de *callback* a função `ad_rm_end()`. Deste modo, a cada 500 ms será efetuado um pedido AJAX ao ficheiro mencionado anteriormente, que irá fazer uma *query* à base de dados de forma a obter a coluna `cfg` e irá enviar o valor desta através da resposta do pedido. Quando esta estiver pronta, é executada a função `ad_rm_end()` que irá verificar se o valor é igual a zero, o que significará que o processo de inclusão, ou de remoção, terminou com sucesso. Caso esta condição não se verifique, a função termina, e passados 500 ms será executada de novo para verificar novamente. Quando esta condição for verdadeira, onde é apresentada a contagem decrescente do tempo que o utilizador tem para realizar o processo de inclusão/exclusão, passará a ser apresentada uma mensagem a informar o utilizador de que o processo foi concluído com sucesso. Mais uma vez, esta operação é realizada através da propriedade `innerHTML`. Para além disso, é usada a função `ClearInterval()` que irá fazer com que esta função deixe de ser executada.

Ao mesmo tempo que se coloca esta função a ser executada a cada 500 ms, é colocada uma outra que irá ser executada a cada segundo, que terá como objetivo fazer a contagem decrescente do tempo. Nesta função, começa-se por colocar no modal o valor guardado na variável `cnt`, que na primeira iteração irá ser 60. De seguida, esta variável é decrementada em uma unidade e é verificada se é igual zero. Em caso negativo, a função termina, sendo executada novamente passado 1 segundo. Quando `cnt` chegar a zero, é apresentada uma mensagem a indicar

que o tempo expirou e é usada a função `ClearInterval()` para fazer com que esta função deixe de ser executada a cada segundo.

Evento de Fecho de Modal

Para além de todas as funções explicadas anteriormente, é necessário mais uma para detetar e indicar à aplicação que o utilizador fechou o modal, para que esta saia do modo de inclusão, ou exclusão, de dispositivos. Para tal, é usado um evento JavaScript que a classe modal da biblioteca Bootstrap dispõe. Neste caso, foi usado o evento `hide.bs.modal` que é acionado quando o modal é fechado por qualquer uma das vias possíveis. Neste evento, é usada a função `clearInterval()` para que as duas funções iniciadas com a função `SetInterval()` deixem de ser executadas, à variável `cnt` é atribuído o valor de 60, para que da próxima vez que se realizar uma contagem decrescente esta comece a partir do 60. Para além disso, é usada a função `submit_ajax()` para colocar a coluna `cfg` igual a 4, utilizando o ficheiro `update.php` à semelhança de como foi explicado previamente.

4.3.4 Atuação de Dispositivos

A atuação de dispositivos pode ser realizada de duas formas, ou através de um botão *toggle* ou através de uma barra de *range*. Desta forma, aos elementos deste tipo, foram atribuídos eventos de JavaScript do tipo `onclick`. Sendo que, para o caso dos botões, este evento irá chamar a função `but_click()` e, para o caso das barras de *range*, irá chamar a função `range()`. Em ambos os casos, é enviado o `id` do elemento (botão ou barra) para a respetiva função.

but_click()

Esta função é executada de cada vez que um botão *toggle* é pressionado, o que implica que o seu estado tenha mudado, e como parâmetro é recebido o `id` do botão em causa. Este id, pode ter dois formatos que são: `tog_bt_x` ou `tog_bt2_x`, sendo que `x` será um número inteiro que corresponderá ao ID do dispositivo Z-Wave. Como é possível verificar, a diferença entre estas duas *strings* é o número 2 antes do ID do dispositivo. Este serve para assinalar que o botão em causa é referente ao segundo canal do dispositivo. O fluxograma desta função está representado na Figura 4.19.

Assim, quando esta função inicia, o primeiro passo é identificar se o `id` recebido é referente ao segundo canal de um dispositivo. Para tal, é utilizado o método `substr()` que permite extrair partes de *strings*, indicando a posição do primeiro caracter e o comprimento da *string* pretendida. Desta forma, ir-se-á usar esta função de forma a obter a *string* que começa no sétimo caracter e que tem comprimento 1, ou seja, irá extrair apenas o sétimo caracter. Este, irá corresponder ao número 2, no caso da segunda *string*, ou ao "_" no caso da primeira *string*. Assim, será verificado se este caracter é igual a 2, e caso o seja, ir-se-á usar

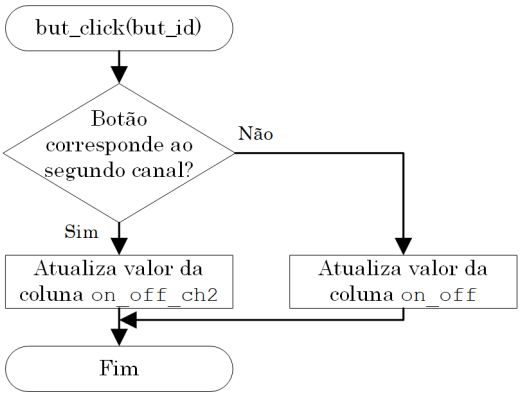


Figura 4.19: Fluxograma da função `but_click()`

a função `submit_ajax()` de forma a atualizar a coluna `on_off_ch2` da tabela `atuadores_zw`, do dispositivo em causa, através do ficheiro `update.php`, como explicado anteriormente. O valor a atribuir a esta coluna será o mesmo que for retornado pelo atributo `checked` do botão com o `id` igual ao recebido como parâmetro. A linha a modificar será identificada através do `node_id` que é determinado através da *string* do id, usando a função `substr()` para extrair o nono e sequentes caracteres. Caso o sétimo caracter da *string* não seja igual a 2, então procedimento a seguir é o mesmo, mas atualizando o valor da coluna `on_off` e, para determinar o ID do dispositivo Z-Wave, é extraído o oitavo e sequentes caracteres.

range()

Esta função quando executada, receberá como parâmetro o `id` da barra de *range* em causa, que terá o seguinte formato: `range_x`, sendo que `x` será um número inteiro que corresponderá ao ID do dispositivo Z-Wave. Utilizando a função `substr()` extrair-se-á o sétimo caracter e os seguintes que corresponderão a este número. O fluxograma desta função é apresentado na Figura 4.20.

Nesta função, o primeiro passo será obter o valor da barra através do atributo `value`. De seguida, é verificado se este é maior que zero e, em caso afirmativo, a coluna `valor`, da tabela `atuadores_zw`, será atualizada para um valor igual ao da barra de *range*. Para além disso, a coluna `on_off` da mesma tabela também é atualizada, neste caso, para o valor de 1. Caso o valor na barra de *range* não seja maior que zero, então seguir-se-á o mesmo procedimento mas atualizando a coluna `on_off` para o valor de zero.

De realçar que, no ficheiro `update.php`, caso a tabela à qual se efetua a *query* seja a `atuadores_zw`, a variável `poll_c` é colocada a 1 para que a aplicação reaja às alterações realizadas.

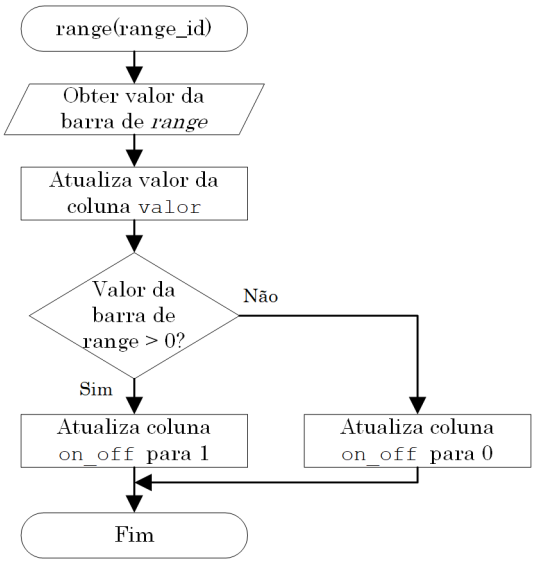


Figura 4.20: Fluxograma da função `range()`

4.3.5 Atualização do Estado dos Dispositivos Z-Wave

Nesta secção irá ser explicado como é feita a atualização das informações apresentadas na página *web*, acerca do estado dos dispositivos Z-Wave. Sendo que, por informações compreende-se os valores medidos por estes, como a potência consumida, a temperatura, etc, assim como, se o dispositivo está ligado ou desligado e qual o valor do dispositivo, no caso de ser regulável. Estes dois últimos podem ser alterados através de interruptores ligados fisicamente ao dispositivo. A implementação desta funcionalidade foi dividida em duas partes, uma para a atualização dos dados referentes aos atuadores, e outra para os dados referentes aos sensores. No entanto, ambas seguem uma lógica semelhante.

Atualização dos Atuadores

Para o caso dos atuadores, quando a página *web* carrega, é usada a função `SetInterval()` para colocar a função `ajax()` a executar a cada 500 ms. Para esta função são passados como parâmetro o nome do ficheiro ao qual o pedido deverá ser feito, neste caso `AJAX_poll_azw.php`, e a função a executar para interpretar a resposta, a função `poll_azw()`.

No ficheiro `AJAX_poll_azw.php`, começa-se por fazer uma *query* pedindo os valores das colunas `node_id`, `on_off`, `on_off_ch2` e `valor`, da tabela `atuadores_zw`, das linhas que tiverem `poll_web` igual a 1, ou seja, apenas as linhas que sofreram alterações. De seguida, será usado um ciclo *while* para percorrer todas as linhas do resultado da *query*. Neste ciclo, é formada uma *string* que conterá os dados recolhidos da base de dados, seguindo o seguinte formato: ” `node_id on_off valor on_off_ch2 /`”. Em que `node_id`, `on_off`,

`valor` e `on_off_ch2` são os valores guardados nestas colunas. Depois, a coluna `poll_web` será atualizada para zero, e procede-se para a próxima iteração do ciclo *while*. De realçar que, estas *strings* formadas, uma por cada iteração do ciclo, são todas concatenadas umas com as outras formando apenas uma *string* com todos os dados que será enviada como resposta. Também de realçar que estas estão separadas por uma barra ("/"), de forma a se poder separá-las mais à frente, da mesma forma que os valores estão todos separados por um espaço.

Desta forma, a função `poll_azw()`, cujo fluxograma está representado na Figura 4.21, irá receber esta *string* e terá como objetivo identificar cada valor e atribuí-los ao respetivo elemento da página *web*. Assim, esta *string* será separada usando o método `split()` que realiza esta operação, retornando um vetor de *substrings*. Para este método, é passado o caracter pelo qual deverá ser feita a separação, neste caso "/", ficando assim com uma *string* para cada nó. De seguida, é usado um ciclo *for* no qual, em cada iteração ir-se-á interpretar uma das *strings* anteriores. Assim, neste ciclo, começa-se por fazer usar o método `split()`, na *string* cujo *index* é igual à variável contadora do ciclo *for*, delimitando-a pelo caracter espaço. Desta forma, ir-se-á obter um vetor com as *substrings* que terão os valores lidos da base de dados no ficheiro `poll_azw.php`. Com este vetor, começará-se por atribuir o valor guardado no segundo elemento (*index* igual a 1) ao botão *toggle*, referente ao canal um, dispositivo. Qualquer que seja o tipo de atuador, este terá sempre pelo menos um botão *toggle*.

De seguida, verifica-se existe algum elemento com o id: "`range_x`", onde `x` será o ID do dispositivo, que está no primeiro elemento (*index* igual a 0) vetor de *strings*. Isto é feito através do valor retornado pelo método `getElementById()` para o `id` mencionado. Caso exista, e se o valor da coluna `on_off`, ou seja, o segundo elemento do vetor de *strings* for igual a um (dispositivo ligado) então, ao atributo `value` da barra de *range* será atribuído o valor do terceiro elemento do vetor (coluna `valor`). Caso o segundo elemento do vetor de *strings* não seja igual a um, então o dispositivo está desligado e ao atributo `value` será atribuído o valor de zero. Nesta última situação, não é usado o valor recolhido da base de dados pois, embora o dispositivo esteja desligado, o último valor que foi atribuído ao dispositivo fica memorizado na base de dados, para que quando se ligar através do botão *toggle*, seja colocado este valor. Depois, é verificado se existe algum elemento com o id: "`tog_bt2_x`", onde `x` será o ID do dispositivo. Se esta condição for verdadeira, ao atributo `checked` deste botão será atribuído o valor guardado no quarto elemento do vetor das *strings* de dados, que contém o valor da coluna `on_off_ch2`. Com isto, procede-se para a próxima iteração do ciclo *for*. E, quando este terminar, todos as mudanças nos atuadores do sistema já foram refletidas na página *web* e a função `poll_azw()` termina. Esta, tal como referido, irá executar novamente passados 500 ms para verificar se houve alguma mudança no sistema.

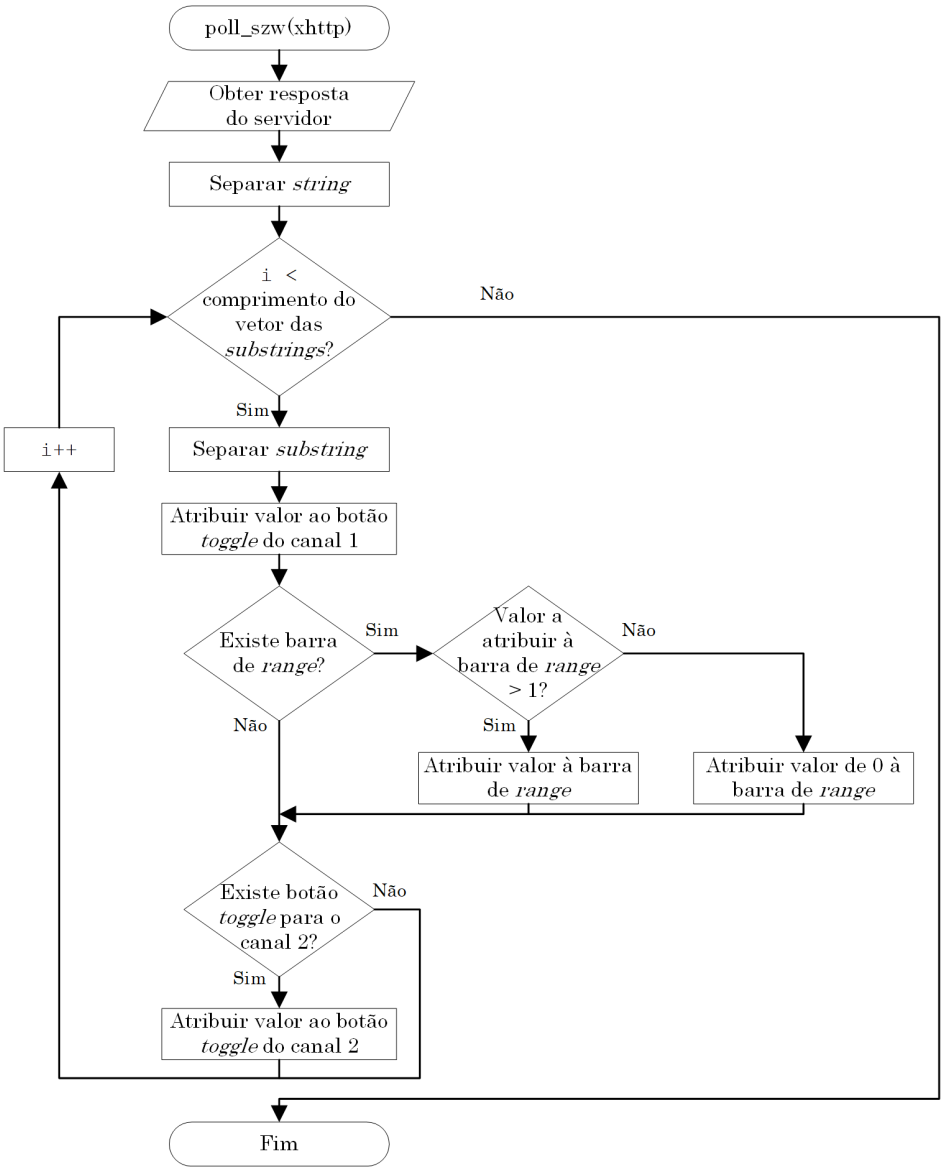


Figura 4.21: Fluxograma da função `poll_azw()`

Atualização dos Sensores

Tal como referido anteriormente, o algoritmo para atualização dos dados acerca dos sensores Z-Wave segue a mesma lógica que o algoritmo para os atuadores. Assim, através da função `SetInterval()` é colocada a função `ajax()` a ser executada a cada 500 ms. Para esta última será enviado como parâmetros o url para o ficheiro a realizar o pedido, `AJAX_poll_szw.php`, e a função que irá interpretar a resposta, `poll_szw()`.

No ficheiro `AJAX_poll_szw.php` começa-se por fazer uma *query* pedindo os valores de todas as colunas, exceto a `poll_web`, da tabela `sensores_zw`, das

linhas que tiverem `poll_web` igual a 1. De seguida, é usado um ciclo *while* para percorrer todas as linhas do resultado da *query*, dentro do qual se irá formar um *string*. Esta, irá seguir o mesmo formato que a *string* formada para os atuadores, incluindo os valores de todas as colunas separadas por um espaço, em que o primeiro valor será o ID do nó e terminará com uma barra ("/"). Esta *string*, será concatenada com as outras referentes aos outros dispositivos, formando uma única que será enviada como resultado do pedido. Ainda dentro do ciclo *while*, a coluna `poll_web` é colocada a zero. Após ler todas as linhas do resultado da *query*, a *string* final é enviada.O fluxograma deste ficheiro é apresentado na Figura 4.22.

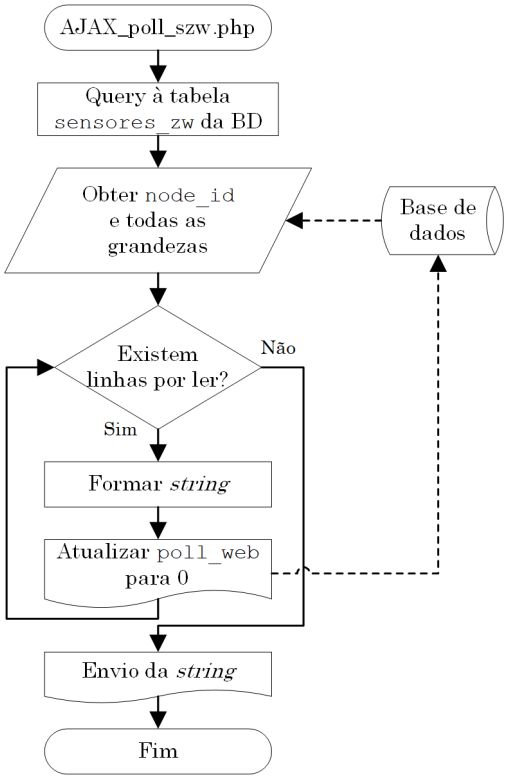


Figura 4.22: Fluxograma do ficheiro `AJAX_poll_szw.php`

O resultado do pedido, será interpretado na função `poll_szw()`, que começa por separar este em *substrings* delimitadas pelo caracter "/", e entra num ciclo *for* no qual ir-se-á percorrer todas estas *substrings*. Neste ciclo, a *substring* é dividida em *substrings* menores delimitadas pelo caracter espaço. Estas segundas *substrings* estarão guardados num vetor em que cada elemento corresponde a um valor da coluna da base de dados para um dado dispositivo. Tendo os valores, é verificado se existe na interface gráfica algum elemento com `id` igual a "`bin_x`", onde `x` corresponde ao número do ID do dispositivo Z-Wave, que é o `id` dos elementos que assinalam o estado de sensores binários. Caso exista, é verificado

o valor do segundo elemento do vetor de dados, e se este for igual a 1, o circulo que assinala o estado dos sensores binários será colocado a azul, atribuindo à propriedade `fill`, do atributo `style`, o código hexadecimal da cor azul (#007bff). Caso o segundo elemento do vetor de dados não seja igual a um, então o circulo será colocado a branco. As restantes grandezas serão atualizadas num ciclo *for* que irá percorrer todos os elementos do vetor de dados. Neste ciclo, começa-se por verificar se existe na interface um elemento que apresenta a grandeza em questão e, caso exista, o valor para esta grandeza, guardado no vetor de dados, será atribuído a este elemento através da propriedade `innerHTML`. De seguida, é verificado se existe um elemento que apresenta esta grandeza na secção expansível da interface (Figura 4.17) referente a este dispositivo. E, caso exista, o valor da grandeza será colocado neste elemento. Percorridas todas as grandezas, procede-se para a próxima iteração do primeiro ciclo *for* que terminando, irá concluir a função `poll_szw()`. O fluxograma desta função pode ser visualizado na Figura B.7 em Anexo B.

Ambas as funções, `poll_szw()` e `poll_azw()`, são colocadas a correr quando a página carrega através do evento `onload` do objeto `window`. Este evento irá executar uma função de cada vez que a página carrega, na qual irá ser usada a função `SetInterval()` para colocar `poll_szw()` e `poll_azw()` com o tempo de intervalo especificado.

4.3.6 Configuração dos Dispositivos

Ao lado do nome de todos os dispositivos Z-Wave, existe um ícone de uma engrenagem que, quando clicada, irá abrir um modal que permitirá introduzir dados para configurar parâmetros e associações do dispositivo em questão. Este modal tem duas abas, uma para cada tipo de configuração, sendo que em cada aba existe um formulário para submeter os dados.

No caso dos parâmetros, o formulário é constituído por três campos de `input` do tipo `number`, que servirão para o utilizador introduzir o número do parâmetro a configurar, o valor a atribuir a este e o tamanho deste em *bytes*. Para além disso, existe um campo escondido que conterá o ID do dispositivo Z-Wave. A determinação e atribuição do valor a este campo será explicado mais à frente. E, por fim, existe o botão `Submit` que irá submeter o formulário, através do método POST, para o ficheiro `param.php`. Este ficheiro, recebe todos os dados inseridos no formulário e, através de uma *query* do tipo `INSERT`, irá colocar estes na tabela `config`. Após isso, irá colocar a coluna `cfg`, da linha referente à *pen* Z-Wave, da tabela `dispositivos_zw`, igual a 11, para assinalar que se pretende configurar um parâmetro. Por fim, este ficheiro irá redirecionar para a página principal.

A aba das associações é igual ao dos parâmetros, incluindo também três campos de `input` do tipo `number` mas que, neste caso, servirão para o utilizador

especificar o grupo, o ID do nó a adicionar ao grupo e a instância. Também inclui um campo escondido que conterá o ID do dispositivo Z-Wave. A submissão do formulário será realizado para o ficheiro `assoc.php`. Este, à semelhança do `param.php`, irá colocar os dados na tabela `config`, no entanto, as colunas alvo serão outras. Estas serão as colunas `node_id`, `grupo` e `tg_node`. Outra diferença é que a coluna `cfg` será atualizada com o valor 12 para assinalar que se pretende configurar uma associação. Por fim, este ficheiro irá redirecionar o utilizador para a página principal.

Tal como referido, em ambos os formulários existe um campo escondido que servirá para enviar para o servidor o ID do dispositivo Z-Wave. Assim, de forma a atribuir um valor, igual ao ID do nó, a este campo, foi adicionado ao elemento com o ícone da engrenagem um evento JavaScript do tipo `onclick`, que irá executar a função `config_id()`. Esta, recebe como parâmetro o ID do dispositivo Z-Wave e, através do atributo `value`, irá atribuir este valor aos campos escondidos. Sendo que os `id` destes campos são `config_param_id`, no caso do formulário dos parâmetros, e `config_assoc_id` no caso do formulário das associações.

4.3.7 Cenários

Tal como mencionado no início desta secção, os cenários têm uma página dedicada apenas para a configuração destes. Esta página pode ser visualizada na Figura 4.23.

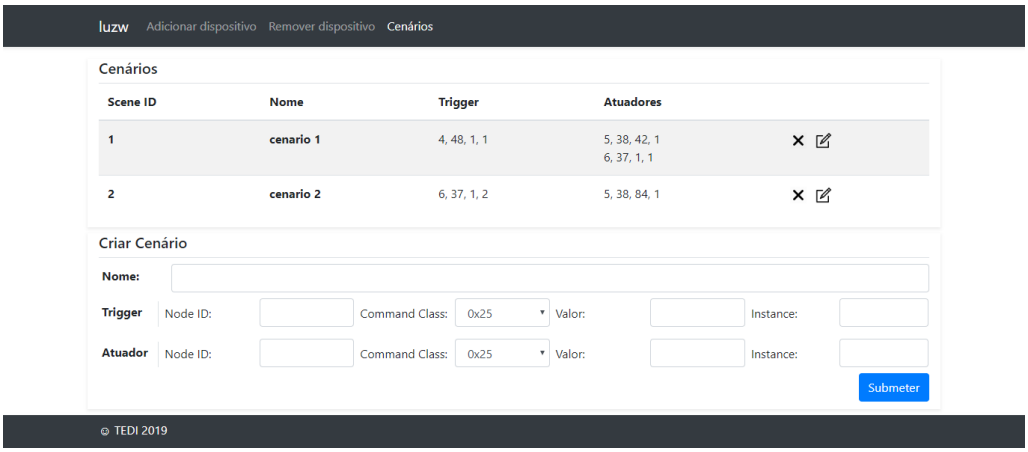


Figura 4.23: Página de configuração dos cenários

Esta página, para além da barra de navegação, apresenta duas áreas distintas. Na primeira, são apresentados os cenários existentes no sistema sobre a forma de uma tabela, especificando o ID do cenário, o nome deste, os seus *triggers* e os seus atuadores. Para além disso, nesta tabela, para cada cenário são disponibilizados dois ícones que permitem eliminar e editar o cenário. Na segunda área, é disponi-

bilizado um formulário que permite ao utilizador introduzir os dados necessários para criar um cenário novo.

Apresentação de Cenários

A colocação da tabela, que apresenta os dados acerca dos cenários, na página *web* é realizada através de um ficheiro PHP denominado de `get_scenes.php`. Este ficheiro, começa por realizar uma *query* à tabela `scenes` pedindo as colunas `scene_id` e `scene_name`. De seguida, é usado um ciclo *while* para percorrer cada uma das linhas do resultado da *query*. Para cada uma destas linhas, é realizada uma segunda *query*, desta vez à tabela `scenes_trig`, pelas linhas que tenham `scene_id` igual ao da primeira *query*, obtendo-se assim todos os *triggers* para este dado cenário. Deste modo, será usado um ciclo *while* para percorrer todas as linhas com os *triggers* do cenário. E, para cada uma destas linhas, os dados desta serão adicionados a uma variável formando uma *string* com os dados de cada *trigger* separados por vírgulas. Quando este ciclo terminar, será realizada uma *query* à tabela `scenes_act` pelas linhas que tenham `scene_id` igual ao obtido na primeira *query*. De seguida, é formada uma *string* da mesma forma que foi feito para os *triggers*. Terminada a pesquisa dos dados acerca dos *triggers* e atuadores do cenário, é feito o *output* de uma linha da tabela, utilizando as variáveis das *strings* com os dados destes. O fluxograma deste ficheiro é representado na Figura B.8 em Anexo B.

Criar Cenário

Na área mais inferior da página cenários, é possível preencher um formulário para criar um cenário. Este, contém um primeiro campo de texto onde o utilizador pode colocar o nome que pretende para identificar o cenário. Para além disso, contém duas linhas de campos iguais, na primeira deverão ser colocados os dados referentes ao *trigger* e na segunda os dados referentes ao atuador. Cada uma destas linhas é constituída por 4 campos, três deles do tipo `number` para especificar o ID do dispositivo Z-Wave, o valor e a instância. O quarto campo, é do tipo `select`, no qual o utilizador pode escolher entre três *command classes*: *Switch Binary* (0x25), *Switch Multilevel* (0x26) e *Binary Sensor* (0x30). Este formulário será submetido, através do método POST, para o ficheiro `criar_scene.php`. Neste, os dados serão recebidos e, recorrendo a três *querys*, serão inseridos nas três diferentes tabelas `scenes`, `scenes_trig` e `scenes_act`. Na *query* à tabela `scenes`, para além da inserção do nome, também é atribuído o valor zero à coluna `scene_id`. Este valor é atribuído a todos os cenários novos, dado que, o verdadeiro ID do cenário só será determinado quando a aplicação o criar. Nesta mesma *query*, também é de realçar que a coluna `opt` é colocada a um para assinalar que se pretende criar um novo cenário.

Eliminar Cenário

A eliminação de cenários é possível através da cruz disponível em cada linha da tabela que apresenta os cenários existentes. Esta, tem associado um evento de JavaScript do tipo `onclick` que fará com que, quando o utilizador carregar na cruz, seja executada uma função, mais especificamente a função `rm_scene()`. Esta, recebe como parâmetro o `id` do elemento que contém o ícone da cruz, que terá o seguinte formato: "`rm_scene_x`", onde `x` será o ID do cenário. Usando o método `substr()`, é possível retirar-se este ID. Para tal, é-lhe passado o número nove de forma a obter o décimo caracter e sequentes. Sabendo o ID do cenário que se pretende eliminar, é realizado um pedido ao servidor usando a função `submit_ajax()`, para atualizar a coluna `opt`, da tabela `scenes`, na linha referente ao cenário em questão, para o valor de 4 para assinalar que se pretende eliminar o cenário. Para a função `submit_ajax()` é enviado como parâmetro o url para o ficheiro, `scene_rm.php`, que irá atualizar a coluna `opt`. No url também é especificado o ID do cenário, que será recebido pelo ficheiro.

Editar Cenário

A edição de cenários é realizada num modal, visível na Figura 4.24, que abre quando o ícone de edição é pressionado. Neste modal é disponibilizado um formulário com um primeiro campo do tipo `select` onde o utilizador pode escolher se pretende adicionar um *trigger* ou um atuador. Também neste formulário, existem mais 4 campos, três deles do tipo `number` para especificar o ID do dispositivo Z-Wave, o valor e a instância. O quarto campo, é do tipo `select`, no qual o utilizador pode escolher entre as três *command classes*. Para além disso, existe um campo escondido que conterá o ID do cenário, cuja determinação e atribuição do valor a este campo segue o mesmo raciocínio usado no campo escondido do formulário de configuração dos parâmetros e associações.

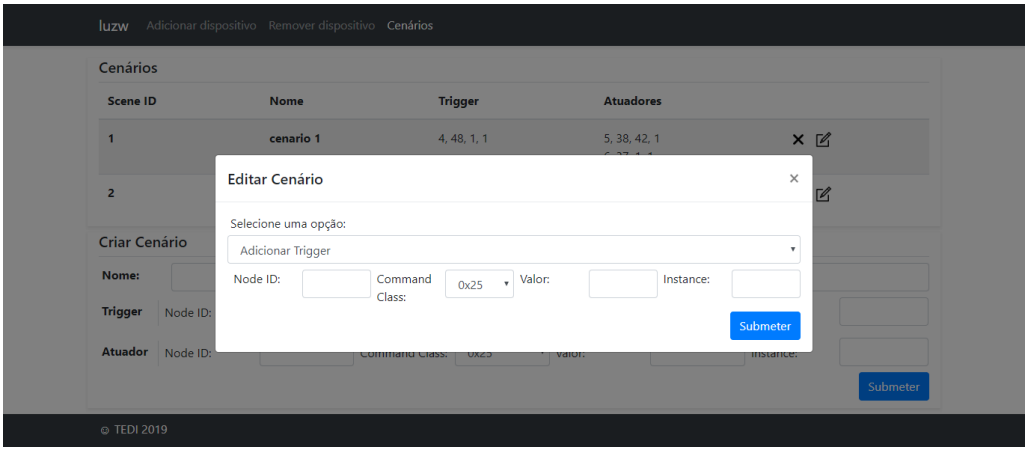


Figura 4.24: Modal de edição de cenários

Este formulário será submetido, através do método POST, para o ficheiro `edit_scene.php`. Este ficheiro, após receber todos os dados do formulário, irá identificar se é para adicionar um *trigger* ou um atuador. Tal, é realizado verificando o valor do campo do formulário destinada para esta seleção. Caso o valor deste campo seja igual a 2, então o utilizador pretende adicionar um *trigger* e, assim, será realizada uma *query* para inserir os dados na tabela `scenes_trig`. Caso contrário, será realizada uma *query* para inserir os mesmos dados na tabela `scenes_act`, e uma outra *query* para atualizar a coluna `opt`, na tabela `scenes`, referente ao cenário em questão, para o valor de três para se assinalar que se pretende adicionar um novo atuador. Por fim, este ficheiro PHP irá redirecionar para a página dos cenários. O fluxograma deste ficheiro é apresentado na Figura 4.25.

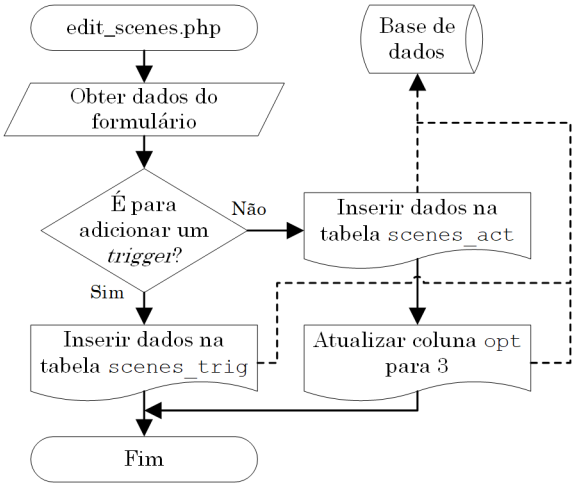


Figura 4.25: Fluxograma do ficheiro `edit_scene.php`

4.3.8 Atualização dos Valores da Temperatura e da Luminosidade

O procedimento para atualização dos valores da temperatura e de luminosidade, medidos pelos sensores conetados fisicamente ao Raspberry, é semelhante à atualização do estado dos dispositivos. Sendo que, também é executada a função `ajax()`, com um determinado intervalo de tempo, para efetuar um pedido ao servidor, de forma a obter os valores guardados na base de dados. Deste modo, através da função `SetInterval()`, ir-se-á colocar a função `ajax()` a executar a cada 500 ms. Para esta, serão passados como parâmetros o url para o ficheiro ao qual se vai efetuar o pedido, `AJAX_poll_phy.php`, e a função *callback* que irá interpretar a resposta do servidor, `poll_phy()`.

O ficheiro `AJAX_poll_phy.php`, cujo fluxograma está representado na Figura 4.26, começa por realizar uma *query* à tabela `sensores_phy`, de forma a obter apenas as linhas que tenham a coluna `poll_web` igual a 1. Após a *query*,

é usado um ciclo *while* para percorrer todas as linhas do resultado desta, que, no máximo, serão duas, uma para a luminosidade e outra para a temperatura. Assim, em cada iteração deste ciclo, será formada uma *string* com o seguinte formato: "`id valor` /", onde `id` e `valor` correspondem aos valores guardados nestas colunas. De seguida, é realizada uma *query* para atualizar o valor da coluna `poll_web` para zero. Após esta *query*, o próximo passo é colocar o valor da grandeza na respetiva tabela que guarda todos os valores desta para desenhar gráficos da grandeza em relação ao tempo. Deste modo, é verificado o valor da coluna `id` e, caso este seja igual a 1, então o valor será inserido na tabela `temperatura_phy`, caso contrário será colocado na tabela `lux_phy`. Tendo percorrido todas as linhas resultantes da primeira *query*, será retornada a *string* formada com os dados retirados da tabela.

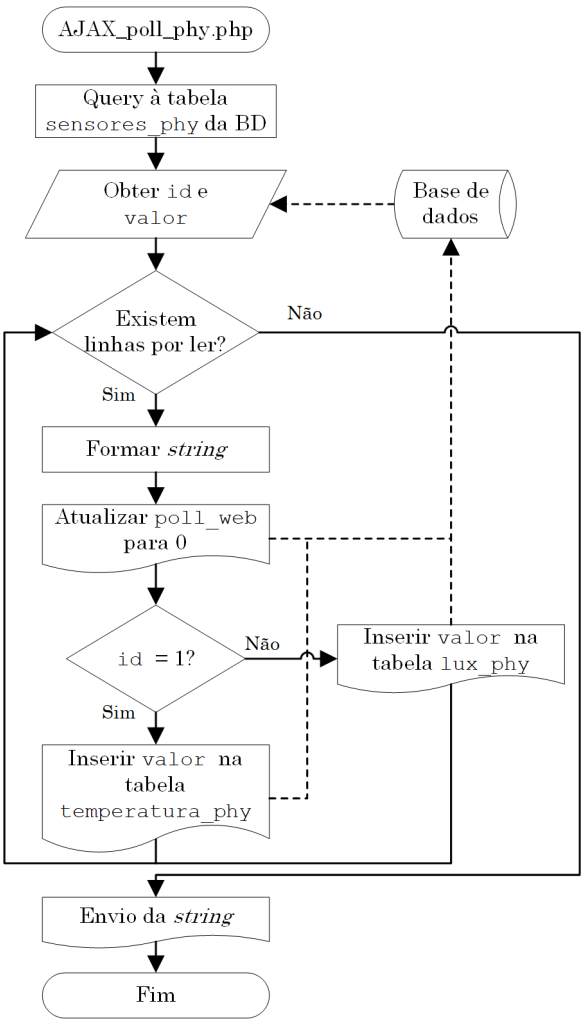


Figura 4.26: Fluxograma do ficheiro `AJAX_poll_phy.php`

Esta *string*, será interpretada do lado do cliente pela função `poll_phy()`.

Nesta, começa-se por usar o método `splt()` para separar a *string* através do caracter ("/"). De seguida, é usado um ciclo *for* para percorrer as *strings* obtidos com esta separação, que no máximo serão duas. Em cada iteração deste ciclo, será analisada uma destas *strings*, começando por separar esta pelo caracter espaço, obtendo-se duas *substrings*, uma com o valor da coluna `id` e outra com o valor da coluna `valor`. Com estes, e através da propriedade `innerHTML`, o valor medido pelo sensor será colocado na interface no elemento que tiver o id: "`phyx`", onde `x` é o valor da coluna `id`. Após este passo, procede-se para a próxima iteração do ciclo *for* e, quando este terminar, conluíra a função e, por isso a atualização dos dados acerca dos sensores. O fluxograma desta função pode ser visualizado na Figura 4.27.

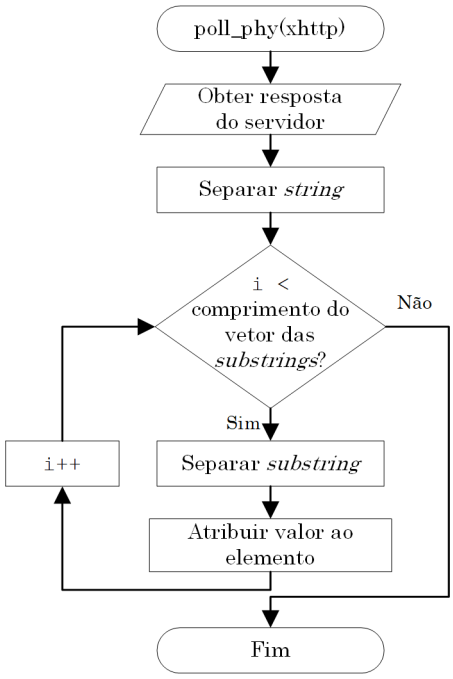


Figura 4.27: Fluxograma da função `poll_phy.php`

4.3.9 Gráficos da Temperatura e Luminosidade

Para desenhar gráficos das grandezas em relação ao tempo, recorreu-se à biblioteca Chart.js. Esta, é um biblioteca em JavaScript que permite criar gráficos, em elementos HTML do tipo `canvas`, através de um objeto do tipo `Chart`. Neste objeto são definidos não só os valores a colocar o gráfico, assim como, todas as configurações, como o tipo de gráfico (linha, barras, radar, etc), a cor dos elementos, o tamanho, entre outras. Os gráficos disponíveis são apresentados num modal, visível na Figura 4.28, que surge após o utilizador clicar no ícone apresentado à direita do nome da grandeza. A este ícone está associado

um evento JavaScript do tipo `onclick`, de forma a que seja executada a função `modal_graf()`.

Figura 4.28: Gráfico da luminosidade em relação ao tempo

Esta função, cujo fluxograma é apresentado na Figura 4.29, começa por identificar qual dos dois ícones (temperatura ou luminosidade) foi clicado, verificando a variável recebida como parâmetro que corresponde ao `id` do ícone. Assim, caso este seja igual a "`graf_2`", significa que foi pressionado o ícone correspondente à luminosidade. Nesse caso, ir-se-á guardar numa variável uma legenda adequada a colocar no gráfico, que mais tarde será usada nas configurações do gráfico. De seguida, através da propriedade `innerHTML`, é definido o título do modal para algo que se adeque à grandeza e, por fim, será executa a função `ajax()` para efetuar o pedido ao servidor pelos valores da luminosidade. Desta forma, para esta função será passada o url para o ficheiro `AJAX_graf.php` especificando também o nome da tabela que se pretende retirar os dados, neste caso, `lux_phy`. Para a função `ajax()` também é passado como parâmetro a função que irá interpretar a resposta do servidor. Caso, o `id` do ícone não seja igual "`graf_2`", então este corresponderá ao ícone da temperatura. E, nesse caso, segue-se a mesma lógica, alterando apenas o nome a atribuir à variável que guarda a legenda do gráfico, o título do e o nome da tabela da qual o ficheiro `AJAX_graf.php` deverá retirar os valores.

Este ficheiro irá realizar uma *query* à tabela, cujo nome é recebido, de forma a retirar todos os dados nesta, que serão as colunas `val` e `time`. De seguida, através de um ciclo *while* que irá percorrer todas as linhas do resultado da *query*, serão formadas duas *strings*, uma com os valores da coluna `val`, separados por vírgulas, e outra para os valores da coluna `time`, também separados por vírgulas. Quando o ciclo terminar, estas serão enviadas sobre a forma de uma única *string*

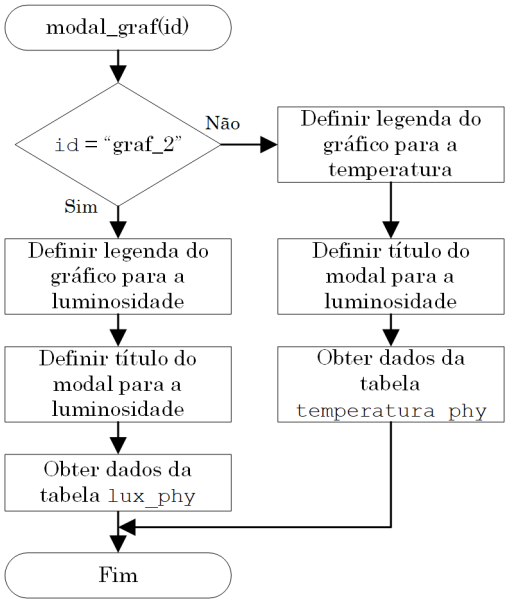


Figura 4.29: Fluxograma da função `modal_graf()`

separadas pelo caracter "/". O fluxograma deste ficheiro pode ser visualizado na Figura 4.30.

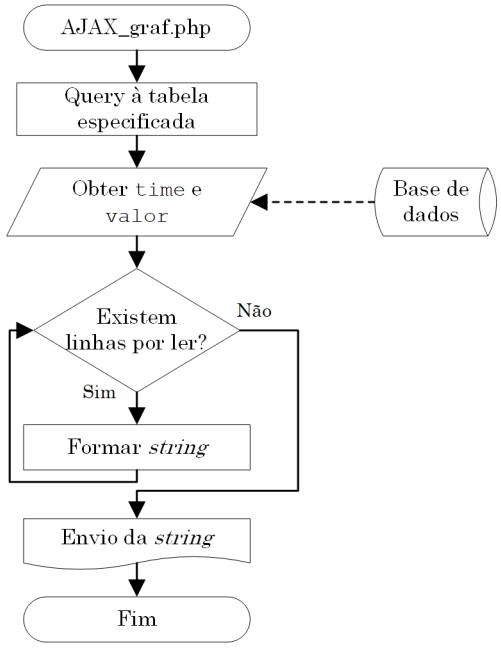


Figura 4.30: Fluxograma do ficheiro `AJAX_graf.php`

Esta *string* será interpretada na função `graf_res()`, cujo primeiro passo será separar a *string*, através do método `split()`, pelo caracter delimitador "/". Obtendo assim uma *string* com os valores da grandeza, e outra com os

valores do tempo. De seguida, ambas estas *strings* será dividida sendo o caracter delimitador a vírgula. Desta forma, tem-se dois vetores em que cada elemento corresponde ao valor da grandeza ou ao tempo, dependendo do vetor. Estes, por sua vez, serão enviados como parâmetros para a função `renderChart()`, que irá configurar o objeto `Chart` que deverá ser colocado. Esta função começa por usar o método `getContext()`, que irá retornar um objeto que proporcionará métodos e propriedades para desenhar no `canvas`. De seguida, é definido o objeto `Chart` onde são realizadas as configurações do gráfico como a legenda, referida anteriormente, e o valores a colocar nos eixos dos xx e dos yy.

Implementando todas estas funcionalidades descritas neste capítulo, obtém-se o sistema desenvolvido que combina um controlador de domótica residencial Z-Wave, que suporta os principais dispositivos utilizados numa casa, com sensores conetados fisicamente ao controlador, cujas grandezas são controladas através de Z-Wave.

Capítulo 5

Resultados

Neste capítulo serão apresentados os resultados obtidos das várias experiências realizadas para verificação do correto funcionamento do sistema desenvolvido. Numa primeira parte são apresentados os resultados relativos às funcionalidades Z-Wave do sistema e, numa segunda parte, são apresentados os resultados relativos ao controlo da temperatura e luminosidade.

5.1 Rede Z-Wave

Como forma de verificar o correto funcionamento das funcionalidades Z-Wave, foi usado um equipamento disponibilizado pela Z-Wave Alliance denominado de *Certified Installer Toolkit* (CIT). Este, pode ser adicionado a uma rede Z-Wave e, através de uma página *web*, apresenta várias informações acerca da rede, assim como, algumas ferramentas de análise de rede. Neste caso, foi usado uma ferramenta chamada Zniffer que apresenta todas as mensagens Z-Wave que são transmitidas na rede. Para além desta, também foi retirado o mapa da rede Z-Wave criada pelo sistema desenvolvido. Este mapa pode ser visualizado na Figura 5.1, que deverá ser comparado com a Figura 4.15 que apresenta os dispositivos emparelhados com o controlador.

Nesta figura é possível ver a representação de todos os dispositivos, através do respetivo ID, emparelhados com o controlador desenvolvido, coincidindo com o apresentado na Figura 4.15, à exceção do dispositivo com o ID 9, que corresponde ao CIT, que não é apresentado nesta figura mencionada em último lugar. De realçar que estas experiências foram realizadas com todos os dispositivos relativamente próximos uns dos outros, daí todos terem conexão direta com todos, assim como, com o controlador (ID 1).

A ferramenta Zniffer foi usada de forma a verificar que mensagens são transmitidas na rede Z-Wave, quando se atua dispositivos através da página *web* desenvolvida e quando sensores Z-Wave reportam valores.

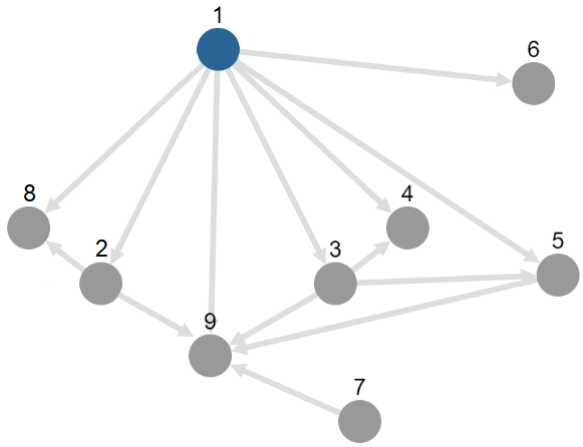


Figura 5.1: Mapa da rede Z-Wave formada pelo sistema desenvolvido

5.1.1 Atuadores

Deste modo, após atuar o módulo Z-Wave *dimmer* (ID 5), através da barra de *range*, foram visualizadas as mensagens captadas pelo CIT. Estas podem ser visualizadas na Figura 5.2.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:44:20	5	1	-	-68 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 28, 00, 00)
→	13.05.2019	19:44:14	5	1	-	-66 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 2A, 00, 00)
→	13.05.2019	19:44:13	5	1	-	-68 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 21, 00, 00)
→	13.05.2019	19:44:11	5	1	-	-68 dBm	-	-	Switch Multilevel Report (53)
→	13.05.2019	19:44:11	5	1	-	-66 dBm	-	-	Switch Multilevel Report (00)
→	13.05.2019	19:44:10	1	5	-	too high	-	-	Switch Multilevel Get
→	13.05.2019	19:44:10	1	5	-	too high	-	-	Switch Multilevel Set (53)

Figura 5.2: Mensagens captadas pelo CIT quando atuado o *dimmer* com o ID 5

Através desta figura verifica-se que, após a atuação na página *web*, é enviada uma mensagem do controlador, dispositivo com o ID 1 (*pen USB*), para o *dimmer* com a *command class Switch Multilevel*, com o comando *Set* e o valor 0x53 (83 em decimal), que é o valor para o qual o *dimmer* se deve regular. Sequentemente a esta mensagem, é enviada uma segunda do controlador para o *dimmer* a pedir o valor atual deste, utilizando a mesma *command class* que na mensagem anterior mas com o comando *Get*. O envio desta segunda mensagem é sempre realizado, pela biblioteca OpenZWave, após a atuação de um dispositivo. Após estas mensagens, o *dimmer* reporta inicialmente que o seu valor é zero, em resposta ao pedido *Get* efetuado pelo controlador, mas logo de seguida, após regular a sua saída de acordo com a atuação efetuada, reporta o valor de 0x53. De seguida, o *dimmer* reporta por três vezes o valor da potência consumida por este. através da *command class Meter*. Este valor é reportado várias vezes devido ao facto de a regulação da saída do *dimmer* ser feita de forma gradual, demorando três

segundos a aumentar a luminosidade desde o zero até ao valor especificado. Por isso, durante esta transição, a potência é reportado várias vezes à medida que muda. Os valores apresentados entre parênteses são valores incluídos na trama como o tipo de medição, a escala, entre outros. Na Figura 5.3 é possível ver o *output* efetuado pela aplicação C/C++ desenvolvida que coincide com as mensagens captadas pelo CIT.

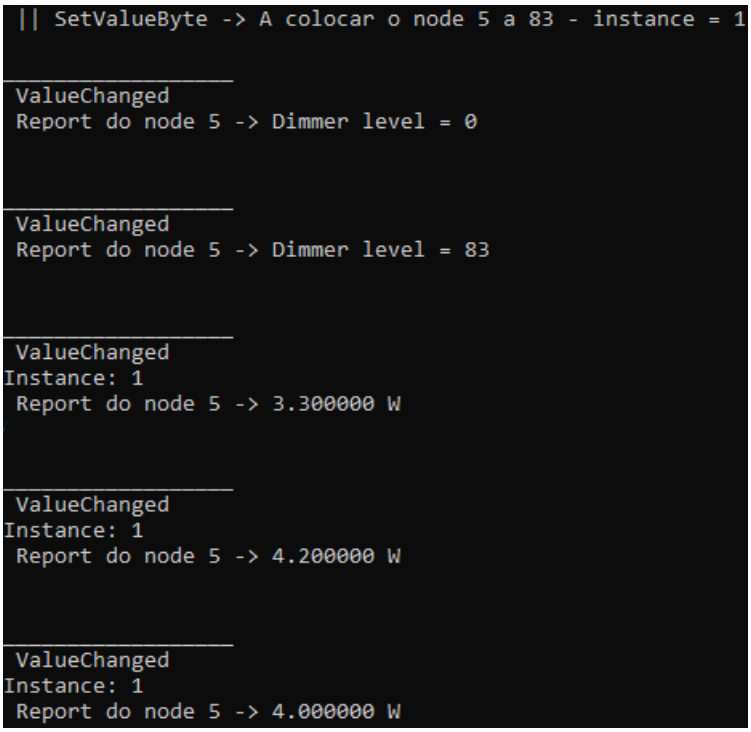


Figura 5.3: *Output* da aplicação desenvolvida quando atuado o *dimmer* com o ID 5

Como é possível verificar, após a atuação do *dimmer*, através da função `SetValueByte`, são recebidos dois *reports* acerca do nível do *dimmer*. seguidos por três *reports* acerca da potência deste. A atuação de *dimmers* será sempre igual ao processo que foi descrito, exceto quando for desligado através do botão *toggle*, dado que esta foi implementada de forma a ser realizada através da *command class Switch Binary*. As mensagens captadas pelo CIT aquando da realização desta ação podem ser visualizadas na Figura 5.4.

Mais uma vez, o controlador (ID 1) envia duas mensagens, uma para atuar (*Set*) e outra a pedir o estado do dispositivo (*Get*) mas, neste caso, através da *command class Switch Binary*. O *dimmer* irá reportar o seu estado que, neste caso, já está atualizado e, de seguida irá reportar a potência consumida. A informação interpretada pela aplicação desenvolvida é apresentada na Figura 5.5 e coincide com a informação captada pelo CIT.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:46:22	5	1	-	-65 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 00, 00, 00)
→	13.05.2019	19:46:21	5	1	-	-68 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 01, 00, 00)
→	13.05.2019	19:46:19	5	1	-	-68 dBm	-	-	Switch Binary Report (00)
→	13.05.2019	19:46:19	1	5	-	too high	-	-	Switch Binary Get
→	13.05.2019	19:46:19	1	5	-	too high	-	-	Switch Binary Set (00)

Figura 5.4: Mensagens captadas pelo CIT quando desligado o *dimmer* com o ID 5

```
|| SetValueBool -> node 5 - instance = 1 - valor = 0


____________________
 ValueChanged
Report do node 5 -> Bool value = 0 - Instance = 1


____________________
 ValueChanged
Instance: 1
 Report do node 5 -> 0.100000 W


____________________
 ValueChanged
Instance: 1
 Report do node 5 -> 0.000000 W
```

Figura 5.5: *Output* da aplicação desenvolvida quando desligado o *dimmer* com o ID 5

A *command class Switch Binary* é usada para transmitir um estado binário, por isso, é usada para atuar dispositivos on/off simples. Na Figura 5.6 são apresentadas as mensagens captadas pelo CIT ao ligar o primeiro canal de um dispositivo on/off de dois canais. De realçar que, tal como foi explicado no capítulo anterior, nos dispositivos multicanal, independentemente do canal cujo estado foi alterado, ambos os canais são atuados quando pelo menos um destes é atuado.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:50:04	6:2	1:1	-	-71 dBm	-	I	Switch Binary Report (00)
→	13.05.2019	19:50:01	6:1	1:1	-	-72 dBm	-	I	Switch Binary Report (FF)
→	13.05.2019	19:50:01	1:1	6:2	-	too high	-	I	Switch Binary Get
→	13.05.2019	19:50:01	1:1	6:2	-	too high	-	I	Switch Binary Set (00)
→	13.05.2019	19:50:01	6:1	1:1	-	-71 dBm	-	I	Switch Binary Report (00)
→	13.05.2019	19:50:01	1:1	6:1	-	too high	-	I	Switch Binary Get
→	13.05.2019	19:50:01	1:1	6:1	-	too high	-	I	Switch Binary Set (FF)

Figura 5.6: Mensagens captadas pelo CIT quando ligado o canal 1 do dispositivo com o ID 6

Nesta figura é de realçar que, em todas as mensagens, é discriminado qual a instância do dispositivo a que a mensagem se destina. As mensagens captadas, são de resto similares às apresentadas anteriormente, com o controlador a enviar duas mensagens para atuar o dispositivo, uma para cada canal, seguida pelo pedido *Get*

para obter o estado do dispositivo. Tal como mencionado, a experiência realizada consistiu na atuação do canal um para ligado, daí este reportar inicialmente que está desligado e de seguida que está ligado, enquanto que o canal dois, cujo estado não se alterou apenas reporta o seu estado uma vez. Para além disso, também são reportados os valores das potências consumidas por cada canal, no entanto, por uma questão de simplificar a apresentação da figura, estas mensagens não são apresentadas. A informação captada destas mensagens, pela aplicação desenvolvida, é apresentada na Figura 5.7 e coincide com a informação captada pelo CIT.

```
|| SetValueBool -> node 6 - instance = 1 - valor = 1

|| SetValueBool -> node 6 - instance = 2 - valor = 0


_____________________
 ValueChanged
Report do node 6 -> Bool value = 0 - Instance = 1


_____________________
 ValueChanged
Report do node 6 -> Bool value = 1 - Instance = 1


_____________________
 ValueChanged
Report do node 6 -> Bool value = 0 - Instance = 2
```

Figura 5.7: *Output* da aplicação desenvolvida quando ligado o canal 1 do dispositivo com o ID 6

De realçar que, caso um dispositivo seja atuado através de interruptores ligados fisicamente a este, os valores da potência e do estado do dispositivo são reportados da mesma forma que quando atuados pela página *web*.

5.1.2 Sensores

Em termos de sensores foram testados dois dispositivos diferentes, um *Smart Meter*, utilizado para medir grandezas elétricas, e um sensor de movimento que, para além desta funcionalidade, também mede temperatura e luminosidade. Quando este último dispositivo deteta movimento, transmite as mensagens representadas na Figura 5.8 que foram captadas pelo CIT.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:21:49	4	1	-	-54 dBm	-	-	Basic Set (FF)
→	13.05.2019	19:21:49	4	1	-	-54 dBm	-	-	Sensor Binary Report (FF)

Figura 5.8: Mensagens captadas pelo CIT quando o dispositivo com o ID 4 deteta movimento

Como é possível verificar, este envia uma *report* através da *command class Sensor Binary* com o valor de 255 (FF em hexadecimal), que indica que o sensor

detetou mmovimento. Para além disso, envia uma segunda mensagem através da *command class Basic*, com o comando *Set* que é usado para definir um valor num dispositivo, mas que neste caso é ignorado pelo controlador. A aplicação desenvolvida irá interpretar o *report* recebido e retirar a informação apresentada na Figura 5.9.

Figura 5.9: *Output* da aplicação desenvolvida quando o dispositivo com o ID 4 deteta movimento

Este sensor, ao fim de 30 segundos sem detetar movimento envia de novo estas mensagens mas com o valor de zero em vez de 255, como representado na Figura 5.10.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:12:33	4	1	-	-57 dBm	-	-	Basic Set (00)
→	13.05.2019	19:12:33	4	1	-	-57 dBm	-	-	Sensor Binary Report (00)

Figura 5.10: Mensagens captadas pelo CIT 30 segundos após o dispositivo com o ID 4 detetar movimento

Tal como referido, este sensor de movimento também é capaz de medir temperatura e luminosidade. O *report* destes valores é realizado dependendo da configuração do dispositivo, sendo que por omissão é realizado quando é detetada uma alteração significativa na grandeza. Na Figura 5.11 estão representadas as mensagens captadas pelo CIT, aquando do *report* destas grandezas por parte do dispositivo.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	20:39:53	4	1	-	-58 dBm	-	-	Multilevel Sensor Report (01, 22, 01, 24)
→	13.05.2019	20:39:53	4	1	-	-57 dBm	-	-	Multilevel Sensor Report (03, 0A, 00, 9C)

Figura 5.11: Mensagens captadas pelo CIT quando o dispositivo com o ID 4 reporta os valores da temperatura e luminosidade edidos

Como é possível verificar, ambas as grandezas são reportadas através da *command class Multilevel Sensor*, sendo que a primeira mensagem (a inferior) se refere à luminosidade e a segunda à temperatura. Na Figura 5.12 é apresentada a informação recolhida destas mensagens pela aplicação desenvolvida.

Quanto ao *Smart Meter*, o *report* dos valores medidos por este também depende da configuração que foi realizada. O *Smart Meter* usado é capaz de medir várias grandezas como potência, energia consumida, corrente, tensão, fator de potência, entre outros. Na Figura 5.13 estão representadas as mensagens captadas pelo CIT quando o *Smart Meter* reportou os valores de potência, tensão e corrente medidos.

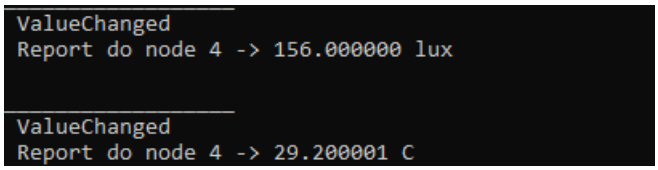


Figura 5.12: *Output* da aplicação desenvolvida quando o dispositivo com o ID 4 reporta os valores de temperatura e luminosidade

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:15:08	7	1	-	-43 dBm	-	-	Meter Report (A1, 6C, 00, 00, 00, 3B, 00, 00, 00, 00, 00, 00)
→	13.05.2019	19:15:08	7	1	-	-43 dBm	-	-	Meter Report (A1, 24, 00, 00, 09, 34, 00, 00, 00, 00, 00, 00)
→	13.05.2019	19:15:08	7	1	-	-43 dBm	-	-	Meter Report (21, 34, 00, 00, 00, 6B, 00, 00, 00, 00, 00, 00)

Figura 5.13: Mensagens captadas pelo CIT quando o dispositivo com o ID 7 reporta os valores de potência, tensão e corrente medidos

Como é possível verificar a mensagem é enviada através da *command class Meter* com o comando *Report* sendo que, tal como referido anteriormente, os valores apresentados entre parênteses são valores incluídos na trama, que podem ser descodificados através das especificações das *command classes* disponíveis em [8]. No caso do comando *Report* da *command class Meter*, a estrutura será a apresentada na Figura 5.14.

7	6	5	4	3	2	1	0
Command Class = COMMAND_CLASS_METER							
Command = METER_REPORT (0x02)							
Scale (2)		Rate Type		Meter Type			
Precision			Scale (1:0)		Size		
Meter Value 1							
...							
Meter Value N							
Delta Time 1							
Delta Time 2							
Previous Meter Value 1 (optional)							
...							
Previous Meter Value N (optional)							
Scale 2							

Figura 5.14: Formato da trama do comando *report* da *command class Meter* [8]

Tomando como exemplo a primeira mensagem (a inferior) da Figura 5.13, o primeiro valor nesta mensagem é 0x21 e corresponderá à terceira linha da tabela da Figura 5.14, ou seja, indicará o terceiro *bit* da unidade do valor reportado (*Scale*), o *rate type* e o tipo de medição. Sendo que 0x21 em binário corresponde

a 0b00100001, através dos primeiros cinco *bits* (00001) conclui-se que o tipo de medição é igual a um que, como especificado em [8], corresponde à medição de uma grandeza elétrica. Para além disso, retirando que o terceiro *bit* da unidade da grandeza é zero e, juntamente com o valor seguinte apresentado pelo CIT, 0x34, 0b00110100 em binário, do qual retira-se o *bit* número três e quatro, 1 e 0, a escala da grandeza será 0b10 que, segundo [8] corresponde a Watts. Concluindo que, esta mensagem se refere ao *report* da potência medida pelo *Smart Meter*. Este raciocínio pode ser aplicado para descodificar os seguintes parâmetros e mensagens recebidas. A aplicação desenvolvida recolheu a informação apresentada na Figura 5.15 que coincide com o apresentado pelo CIT e com o raciocínio apresentado.

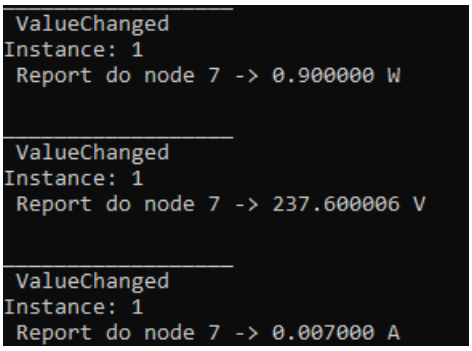


Figura 5.15: *Output* da aplicação desenvolvida quando o dispositivo com o ID 7 reporta os valores de potência, tensão e corrente medidos

5.1.3 Cenários

Para validar a funcionalidade dos cenários foi igualmente utilizado o Zniffer do CIT. Como cenário de teste foi usado o cenário 1, apresentado na Figura 4.23, o qual define que quando o sensor de movimento (ID 4) for acionado, o *dimmer* (ID 5) deverá ser regulado para o valor 42 (0x2A) e o canal um do dispositivo on/off de dois canais (ID 6) deverá ser ligado. Assim, na Figura 5.16 são apresentadas as mensagens captadas pelo CIT quando o sensor de movimento foi acionado.

Tal como apresentado na figura, assim que o controlador recebe um *report* do sensor de movimento a indicar que este foi acionado, o controlador envia mensagens de forma a atuar o *dimmer* e o canal um do módulo on/off, sendo estas seguidas por *reports* por parte destes dois dispositivos a indicar o seu estado.

5.1.4 Parâmetros

Para validar a implementação da configuração de parâmetros, seguiu-se a mesma lógica que para os casos anteriores. Desta forma, foi efetuada a configuração do parâmetro 65 do *dimmer* (ID 5), de forma a definir este com o valor

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	13.05.2019	19:57:01	5	1	-	-54 dBm	-	-	Switch Multilevel Report (2A)
→	13.05.2019	19:57:00	6	1	-	-77 dBm	-	-	Switch Binary Report (FF)
→	13.05.2019	19:57:00	6:1	1:1	-	-75 dBm	-	I	Switch Binary Report (00)
→	13.05.2019	19:57:00	1:1	6:1	-	too high	-	I	Switch Binary Get
→	13.05.2019	19:57:00	1:1	6:1	-	too high	-	I	Switch Binary Set (FF)
→	13.05.2019	19:57:00	5	1	-	-54 dBm	-	-	Switch Multilevel Report (00)
→	13.05.2019	19:57:00	1	5	-	too high	-	-	Switch Multilevel Get
→	13.05.2019	19:57:00	1	5	-	too high	-	-	Switch Multilevel Set (2A)
→	13.05.2019	19:57:00	4	1	-	-82 dBm	-	-	Basic Set (FF)
→	13.05.2019	19:57:00	4	1	-	-82 dBm	-	-	Sensor Binary Report (FF)

Figura 5.16: Mensagens captadas pelo CIT quando o dispositivo com o ID 4 deteta movimento e é acionado o cenário 1

de 50, e visualizadas as mensagens trocadas na rede. Este parâmetro define o tempo de transição entre o valor mínimo e máximo de luminosidade quando o dispositivo é atuado através de um interruptor, ligado fisicamente a este, com um único *click*. O valor por omissão deste parâmetro é de 100 que corresponde a um segundo. As mensagens captadas pelo CIT, aquando da configuração podem ser visualizadas na Figura 5.17.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	08.07.2019	15:20:07	5	1	-	-68 dBm	-	-	Configuration Report (41, 02, 00, 32)
→	08.07.2019	15:20:01	1	5	-	too high	-	-	Configuration Get (41)
→	08.07.2019	15:20:01	1	5	-	too high	-	-	Configuration Set (41, 02, 00, 32)

Figura 5.17: Mensagens captadas pelo CIT quando configurado o parâmetro 65 do dispositivo com o ID 5

Como é possível verificar, o controlador começa por enviar uma mensagem para o *dimmer* através da *command class Configuration*, com o comando *Set*, na qual especifica o parâmetro 65 (0x41 em hexadecimal) e o valor a atribuir a este 50 (0x32 em hexadecimal). De seguida, o controlador envia uma outra mensagem, desta vez, a pedir a configuração do parâmetro 65, através do comando *Get*, à qual o *dimmer* responde através do comando *Report* com os dados coincidentes com os definidos pelo controlador na primeira mensagem.

5.1.5 Associações

Para validar a configuração de associações implementada, foi definida uma associação entre o sensor de movimento (ID 4) e o *dimmer*, na qual este último foi adicionado ao primeiro grupo do sensor de movimento. Este grupo envia uma *trama Basic Set*, aos dispositivos associados a este grupo, quando o sensor de movimento é acionado. As mensagens captadas pelo CIT estão representadas na Figura 5.18

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	08.07.2019	15:25:51	4	5	-	-41 dBm	-	-	Basic Set (FF)
→	08.07.2019	15:25:51	4	1	-	-66 dBm	-	-	Basic Set (FF)
→	08.07.2019	15:25:51	4	1	-	-68 dBm	-	-	Sensor Binary Report (FF)

Figura 5.18: Mensagens captadas pelo CIT quando o dispositivo com o ID 4 deteta movimento e quando está configurada uma associação com o dispositivo com o ID 5

Nesta figura é visível que, inicialmente, o sensor de movimento envia duas mensagens para o controlador iguais às apresentadas anteriormente na Figura 5.8 mas, para além disso, envia também uma mensagem a através da *command class Basic*, com o comando *Set* e o valor de 255 (0xFF em hexadecimal), para o *dimmer* (ID 5), fazendo com que este se ligue. Tal como explicado anteriormente, ao fim de 30 segundos sem detetar movimento, este sensor envia de novo estas mensagens mas com o valor de zero em vez de 255. Quando configurada esta associação, também é enviada uma para o *dimmer*, fazendo com que este se desligue. A Figura 5.19 apresenta estas mensagens.

	Date	Time	SRC	DEST	Speed	RSSI	Hops	Encaps	Application
→	08.07.2019	15:26:21	4	5	-	-41 dBm	-	-	Basic Set (00)
→	08.07.2019	15:26:21	4	1	-	-66 dBm	-	-	Basic Set (00)
→	08.07.2019	15:26:21	4	1	-	-68 dBm	-	-	Sensor Binary Report (00)

Figura 5.19: Mensagens captadas pelo CIT 30 segundos após o dispositivo com o ID 4 detetar movimento e quando está configurada uma associação com o dispositivo com o ID 5

Com a realização destes testes, foi possível comprovar o correto funcionamento de todas as funcionalidade Z-Wave implementadas no sistema.

5.2 Sistema de Temperatura

Para validar o controlo da temperatura foram realizadas várias experiências, nas quais foi feita uma análise ao controlador PI implementado. Assim, foi analisada a resposta do sistema para vários ganhos, k_p e k_i, para o mesmo *setpoint*, 40 °C, sendo a temperatura inicial a temperatura ambiente, cerca de 25 °C. O tempo de amostragem utilizado foi de 150 ms, sendo que este valor foi escolhido tendo em conta que, para um valor inferior a 100 ms, antes de o módulo Z-Wave 0-10 V atualizar o seu valor para o recebido, a iteração seguinte do controlador PI já estava a enviar novos valores, tornando a primeira atuação inútil e desnecessária. Para além disso, foi tido em conta que o módulo Z-Wave ao mudar o seu valor, fá-lo de forma gradual e não instantânea, demorando um segundo a realizar este processo. Este valor é o valor mínimo e foi programado através de um parâmetro.

Deste modo, foi analisado primeiro apenas a ação proporcional do controlador, colocando k_i igual a 0, efetuando dois testes para dois valores de k_p diferentes, $k_p = 4$ e $k_p = 8$. O resultado obtido é apresentado na Figura 5.20.

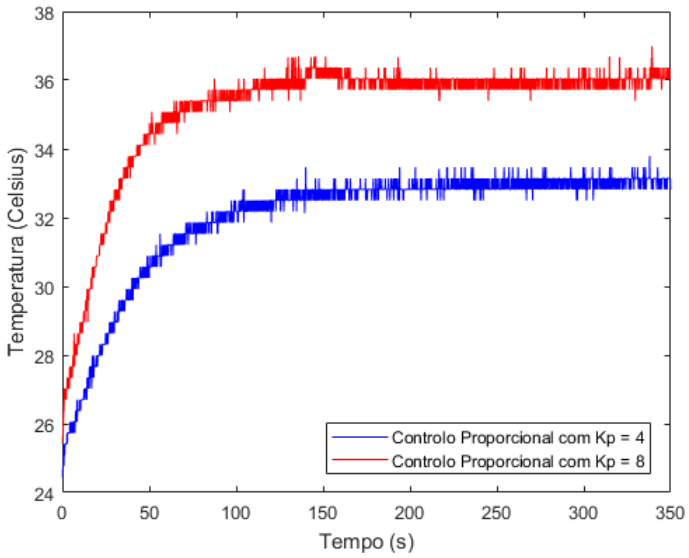


Figura 5.20: Resposta do sistema de temperatura com controlador proporcional para $k_p = 4$ (a azul) e $k_p = 8$ (a vermelho)

Como é possível verificar na figura, a ação proporcional funciona tal como seria de esperar, com o aumento do ganho a refletir-se numa resposta mais rápida e numa redução do erro em regime permanente.

Para validação da ação integral, foram realizadas experiências similares às realizadas para ação proporcional. Desta forma, foram observadas as respostas do sistema, utilizando $k_p = 8$ em ambas, e $k_i = 0,1$ numa e $k_i = 0,2$ noutra. Estas respostas podem ser visualizadas na Figura 5.21.

Tal como na ação proporcional, é possível constatar o correto funcionamento da ação integral, verificando-se que esta elimina o erro em regmie permanente e que, quanto maior for o ganho integrativo, mais rápida será a resposta. De todas as experiências realizadas, a melhor resposta foi obtida com $k_p = 8$ e $k_i = 0,1$ tendo sido estes os valores definidos para o controlador PI.

Todas as respostas apresentadas anteriormente, não utilizam a *deadband* no erro, mencionada na secção 4.2.6. Desta forma, na Figura 5.22 é apresentada a resposta do sistema com *deadband* para os melhores ganhos.

Como é possível verificar, a introdução da *deadband* faz com que a resposta apresente mais oscilações e um maior tempo de estabelecimento. Para além disso, irá fazer com que o sistema apresente um erro em regime permanente, no entanto, num caso prático este erro não será percetível e, como vantagem, a *deadband* irá

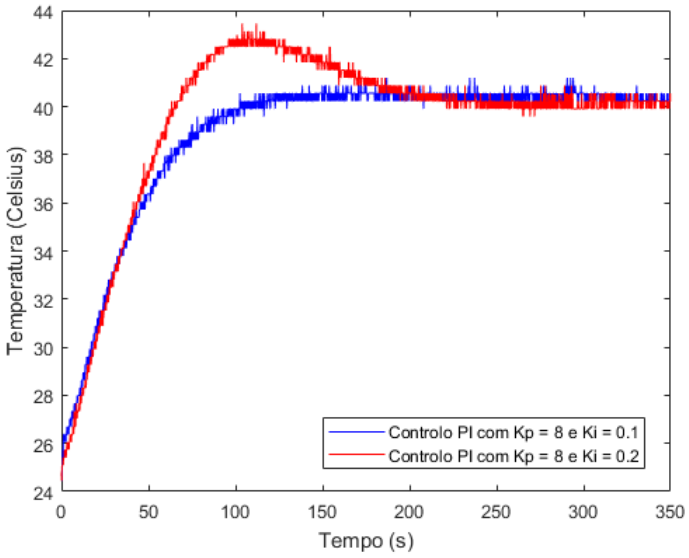


Figura 5.21: Resposta do sistema de temperatura com controlador PI para $k_p = 8$ e $k_i = 0,1$ (a azul) e $k_i = 0,2$ (a vermelho)

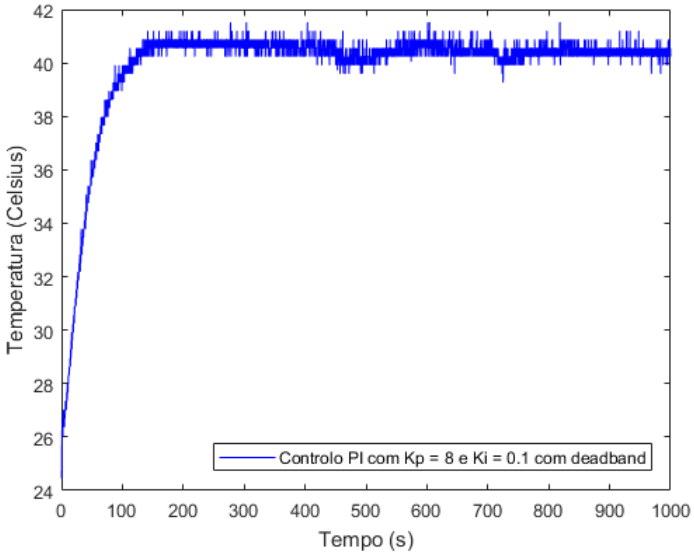


Figura 5.22: Resposta do sistema de temperatura com controlador PI para $k_p = 8$ e $k_i = 0,1$ com *deadband*

reduzir o número de mensagens enviadas para o módulo Z-Wave, não sobrecarregando esta.

5.3 Sistema de Luminosidade

O processo seguido para validação do controlador do sistema de luminosidade, foi o mesmo que o utilizado para o sistema de temperatura, observado-se a resposta do sistema para diferentes ganhos. O valor de *setpoint* escolhido foram os 100 Lux, sendo que o valor medido pelo LDR quando o módulo Z-Wave está no seu valor máximo é cerca de 200 Lux. De realçar que, este valor depende do posicionamento do LED face ao LDR e, para além disso, de realçar que os testes foram realizados com estes dois componentes tapados de forma a que no LDR não incida luz para além da emitida pelo LED.

Com isto, de forma a analisar a ação proporcional, foram obtidas as respostas do sistema para $k_p = 0,7$ e $k_p = 1$, mantendo-se k_i igual a zero em ambos os casos. As respostas obtidas podem ser visualizadas na Figura 5.23.

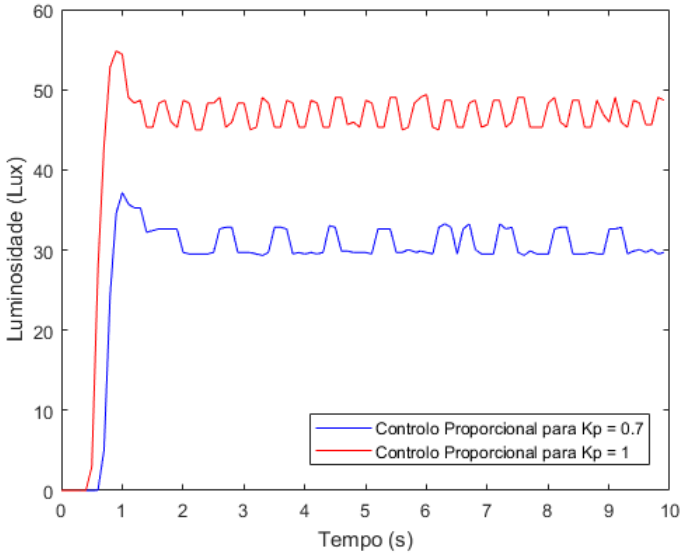


Figura 5.23: Resposta do sistema de luminosidade com controlador proporcional para $k_p = 0,7$ (a azul) e $k_p = 1$ (a vermelho)

Com esta figura, constata-se que o correto funcionamento da ação proporcional, ao verificar-se respostas mais rápidas e menor erro em regime permanente com o aumento do ganho proporcional.

De seguida, foram efetuados dois testes de forma a obter-se respostas que permitam analisar a ação integral. Estas duas respostas foram obtidas utilizando $k_i = 0,4$ numa e $k_i = 0,7$ noutra, mantendo k_p igual a 1 em ambos os testes. Os resultados obtidos são apresentados na Figura 5.24.

Também para a ação integral verifica-se o seu correto funcionamento ao visualizar as respostas apresentadas na figura anterior. Nesta é possível constatar

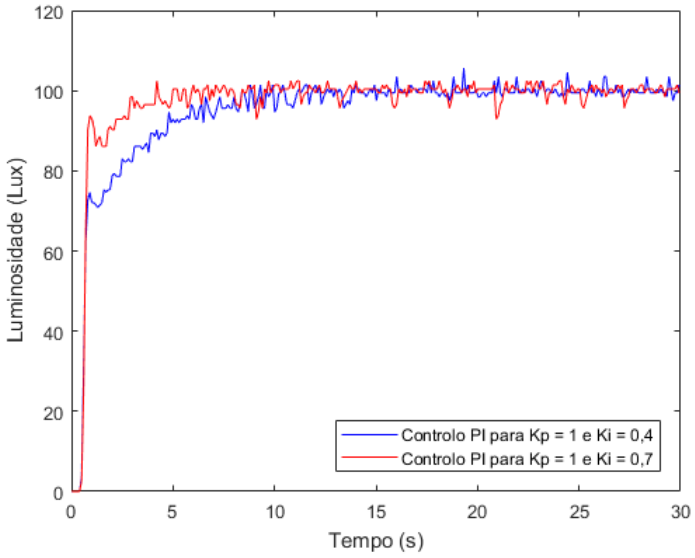


Figura 5.24: Resposta do sistema de luminosidade com controlador PI para $k_p = 1$ e $k_i = 0,4$ (a azul) e $k_i = 0,7$ (a vermelho)

a eliminação do erro em regime permanente e que o aumento do ganho integral refelete-se numa resposta mais rápida. A melhor resposta foi obtida utilizando os ganhos $k_p = 1$ e $k_i = 0,7$ tendo ficado estes valores definidos para controlo da luminosidade.

Tal como no sistema da temperatura, as respostas apresentadas não incluem a *deadband* no erro, mencionado anteriormente. Desta modo, para analisar a influência desta, foi obtida a resposta do sistema para os melhores ganhos com *deadband* implementada. A resposta obtida está representada na Figura 5.25, juntamente com uma resposta com os mesmos ganhos mas sem *deadband*.

Como é possível verificar, a influência da *deadband* neste sistema é a mesma que no sistema de temperatura, ao introduzir algumas oscilações, a aumentar o tempo de estabelecimento e a fazer com que haja um erro em regime permanente. Contudo, este erro, num caso prático nunca será percetível e, em contrapartida, a *deadband* permitirá obter uma luminosidade mais estável e reduzir as mensagens na rede Z-Wave.

Com isto, conclui-se igualmente o correto funcionamento dos controladores PI implementados em ambos os sistemas.

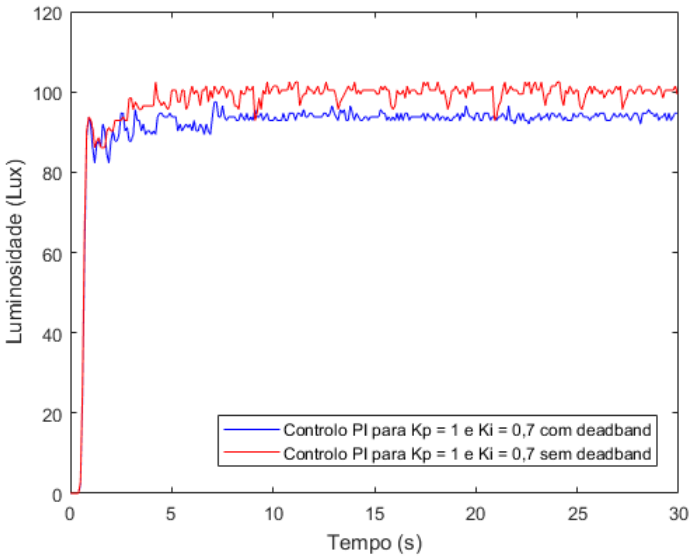


Figura 5.25: Resposta do sistema de luminosidade com controlador PI para $k_p = 1$ e $k_i = 0,7$ com *deadband* (a azul) e sem *deadband* (a vermelho)

Capítulo 6

Conclusão

Nesta última secção é realizada uma síntese das principais conclusões do trabalho realizado e aspetos a melhorar em desenvolvimentos futuros.

Inicialmente, este projeto partiu de um estudo acerca da domótica e do protocolo Z-Wave, que permitiu adquirir os conhecimentos necessários para o desenvolvimento deste, quer a nível de análise de requisitos, quer a nível de implementação. E os resultados apresentados ao longo deste relatório sustentam o bom funcionamento do sistema desenvolvido. Este permite realizar as funcionalidades básicas de um controlador de domótica, como configurar e atuar dispositivos e obter informações recolhidas por estes e definir cenários de forma a automatizar atividades. Para além disso, as grandezas medidas pelos sensores incorporados são controladas de forma eficaz, recorrendo às capacidades Z-Wave, e é possível observar a variação destas grandezas ao longo do tempo através de gráficos.

Este projeto, para além de ter possibilitado aplicar, e aprofundar, várias áreas como a eletrónica, o controlo de sistemas e a programação, permitiu adquirir conhecimentos novos acerca da domótica, nomeadamente acerca do protocolo Z-Wave. Este apresenta-se como o protocolo mais relevante e a melhor solução na área da domótica, principalmente para edifícios já construídos e sem estrutura dedicada para domótica. Isto, deve-se não só à qualidade do protocolo, quer em termos de desempenho, quer em termos de interoperabilidade entre fabricantes, mas também ao facto de este protocolo se atualizar para se tornar ainda melhor, como por exemplo, tornar-se mais seguro. Contudo, com o surgimento do protocolo Thread será interessante ver a evolução destes dois protocolos e, acima de tudo, da domótica.

Um dos melhoramentos a realizar no sistema seria tornar a interface dos cenários mais *user-friendly*, dado que a atual obriga o utilizador a ter alguns conhecimentos mais aprofundados acerca do protocolo Z-Wave, como por exemplo, qual o número que representa cada *command class*. Para além disso, não é possível criar condições do tipo maior, ou menor, por exemplo, quando a tem-

peratura medida por um determinado dispositivo for superior a um dado valor, realizar uma dada ação. Outro melhoramento a realizar seria aumentar o tipo de dispositivo suportados, como por exemplo, módulos de estores, e o tipo de mensagens, recebidas dos dispositivos, interpretadas pelo sistema.

Contudo, este projeto apresenta, acima de tudo, uma forma de realizar sistemas com capacidade de comunicação Z-Wave a um nível baixo, o que permite uma maior versatilidade e, consequentemente, permite desenvolver soluções para problemas mais específicos, para os quais não existe solução no mercado.

Bibliografia

[1] P. Shorty, "System design specification - z-wave protocol overview," *Zensys*, Fevereiro 2005. [cited on p. xi, 9, 10, 12, 13, 14, 15]

[2] Silicon Labs, "Ug373: Zgm130s zen gecko wireless starter kit user's guide," Novembro 2018. [cited on p. xi, 19]

[3] Silicon Labs, "Zgm130s z-wave 700 sip module datasheet," Agosto 2018. [cited on p. xi, 19]

[4] Vera, "Smart home controllers - vera." [cited on p. xi, 21]

[5] Samsung, "Smartthings - smartthings hub." [cited on p. xi, 21, 22]

[6] Microchip, "Mcp3004/3008 - 2.7v 4-channel/8-channel 10-bit a/d converters with spi serial interface," 2008. [cited on p. xi, xv, 30, 32, 33]

[7] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design and Tuning.* Instrument Society of America, 1995. [cited on p. xi, 35, 36, 37]

[8] Silicon Labs, "Software design specification - z-wave application command class specification," *Silicon Labs*, Abril 2019. [cited on p. xiii, 14, 15, 103, 104]

[9] B. Bogovac, D. Mijić, D. Milinkov, and M. Sági, "Smart home automation," *IEEE 20th Telecommunications forum TELFOR 2012*, 2012. [cited on p. 5, 6]

[10] R. Y. M. Li, H. C. Y. Li, C. K. Mak, and T. B. Tang, "Sustainable smart home and home automation: Big data analytics approach," *International Journal of Smart Home*, 2016. [cited on p. 5, 6]

[11] A. Roy, S. Das, and S. Dey, "Home automation using internet of thing," *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2016. [cited on p. 5, 6]

[12] A. Colon and E. Griffith, "The best smart home devices for 2019," *PCMAG*, Janeiro 2019. [cited on p. 5]

[13] I. Unwala, J. Lu, and Z. Taqvi, "Thread: An iot protocol," *2018 IEEE Green Technologies Conference*, 2018. [cited on p. 6, 7, 8, 9]

[14] K. Wieland, "Iot experts fret over fragmentation," Fevereiro 2016. [cited on p. 7]

[15] M. Wallace, "Fragmentation is the enemy of the internet of things," Abril 2016. [cited on p. 7]

[16] H. Bauer, J. Veira, and M. Patel, "Internet of things: Opportunities and challenges for semiconductor companies," Outubro 2015. [cited on p. 7]

[17] C. Withanage, C. Yuen, K. Otto, and R. Ashok, "A comparison of the popular home automation technologies," *2014 IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA)*, 2014. [cited on p. 7, 8, 9]

[18] KNX Association, "Knx - introdução," Outubro 2016. [cited on p. 7]

[19] KNX Association, "Knx tecnologia - introdução," Outubro 2016. [cited on p. 7]

[20] KNX Association, "Knx - meios de comunicação," Outubro 2016. [cited on p. 7]

[21] Z-Wave, "Z-wave: The basics," Janeiro 2017. [cited on p. 8, 10]

[22] O. Hersent, D. Boswarthick, and O. Elloumi, *The internet of things: key applications and protocols*. John Wiley & Sons Ltd, 2012. [cited on p. 9, 10, 12, 13, 14, 15, 16]

[23] Silicon Labs, "Acquisition of sigma designs z-wave products," Abril 2018. [cited on p. 9]

[24] Silicon Labs, "Instruction - z-wave node type overview and network installation guide," *Silicon Labs*, Março 2018. [cited on p. 10, 11, 12]

[25] I. Unwala, Z. Taqvi, and J. Lu, "Iot security: Zwave and thread - ieee conference publication," *2018 IEEE Green Technologies Conference*, Junho 2018. [cited on p. 16, 17, 18]

[26] B. Fouladi and S. Ghanoun, "Security evaluation of the z-wave wireless protocol," 2013. [cited on p. 16]

[27] J. D. Fuller and B. W. Ramsey, "Rogue z-wave controllers: A persistent attack channel," *40th Annual IEEE Conference on Local Computer Networks*, 2015. [cited on p. 16, 17]

[28] i-SCOOP, "Z-wave alliance announces new s2 framework for better smart home security," Janeiro 2018. [cited on p. 17]

[29] S. Marksteiner, V. J. E. Jiménez, H. Vallant, and H. Zeiner, "An overview of wireless iot protocol security in the smart home domain," *IEEE - Internet of Things Business Models, Users, and Networks*, 2017. [cited on p. 18]

[30] C. Pätz, "How to develop z-wave devices," Outubro 2012. [cited on p. 18, 28]

[31] Z-Wave Alliance, "Z-wave public specification - z-wave alliance," Agosto 2016. [cited on p. 18]

[32] E. Brown, "Z-wave opens up with new public sdk and developer site," Outubro 2018. [cited on p. 18]

[33] Silicon Labs, "Getting started with z-wave 700," Março 2019. [cited on p. 18]

[34] Z-Wave Alliance, "Z-wave certification: The key to interoperability," Abril 2014. [cited on p. 20]

[35] Z-Wave Alliance, "Z-wave plus™ certification - z-wave alliance," Abril 2014. [cited on p. 20]

[36] Silicon Labs, "Z-wave specification," *Silicon Labs*, Março 2019. [cited on p. 20]

[37] Martin, "Top 4 z-wave controllers for 2018," Novembro 2018. [cited on p. 21]

[38] H. Techie, "5 best z-wave hubs (smart controllers) of 2019 – smart home controllers," May 2019. [cited on p. 21]

[39] Christy, "Open source home automation software that's actually worth using," Setembro 2018. [cited on p. 23]

[40] J. Baker, "6 open source home automation tools," Dezembro 2017. [cited on p. 23]

[41] Aeotec, "Z-stick: Z-wave usb antenna - aeotec by aeon labs," Fevereiro 2018. [cited on p. 26, 27]

[42] Raspberry Pi Foundation, "Raspberry pi 3 model b+," Março 2018. [cited on p. 27]

[43] OpenZWave, "Introduction to openzwave," 2016. [cited on p. 28]

[44] M. Grusin, "Serial peripheral interface (spi)," *Sparkfun*, Fevereiro 2013. [cited on p. 32]

[45] Analog Devices, "Low voltage temperature sensors - tmp35/tmp36/tmp37," 2015. [cited on p. 34]

[46] Fairchild Semiconductor Corporation, "Tip31a/tip31c - npn epitaxial silicon transistor," Novembro 2014. [cited on p. 34]

Anexo A

Layout da PCB Desenvolvida

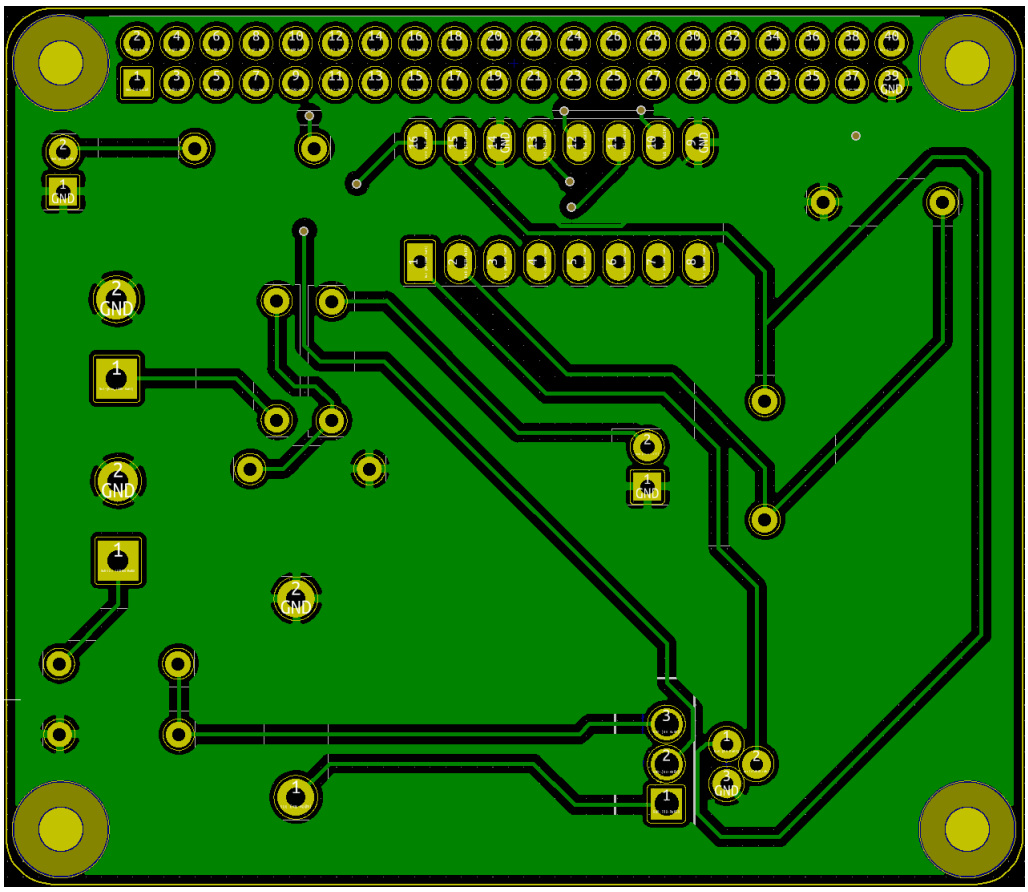


Figura A.1: Face inferior da PCB desenvolvida

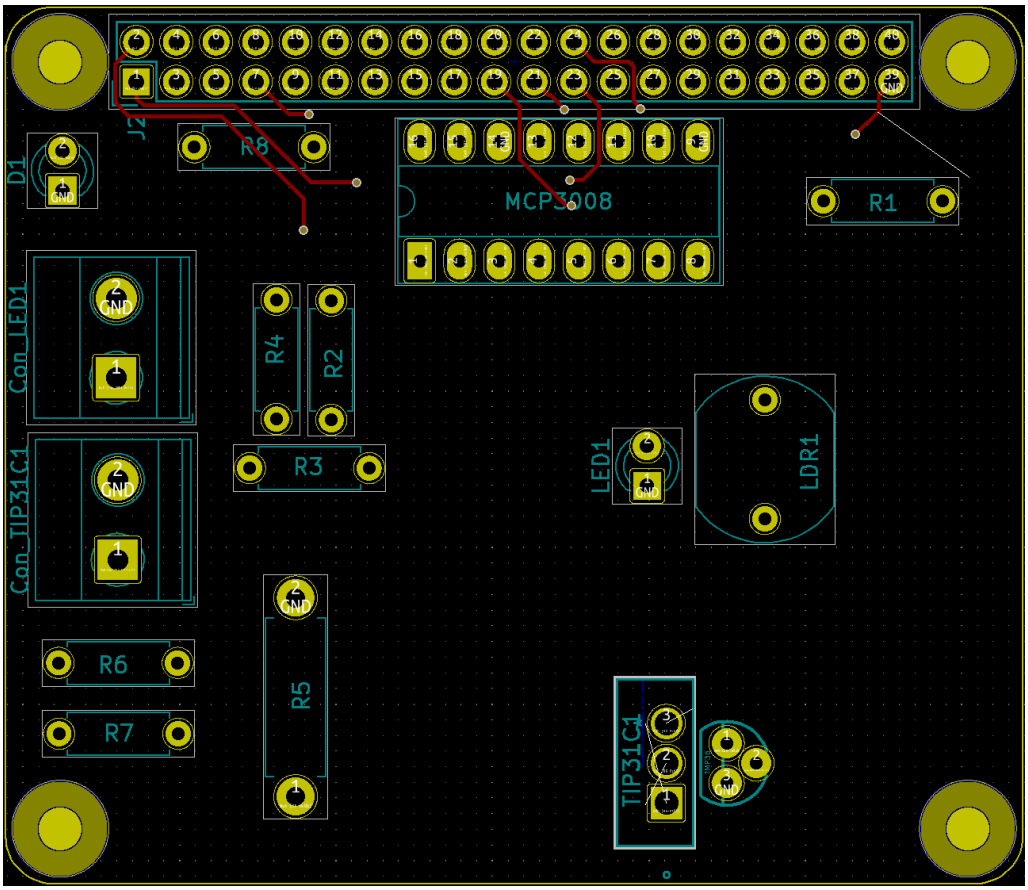


Figura A.2: Face superior da PCB desenvolvida

Anexo B

Fluxogramas

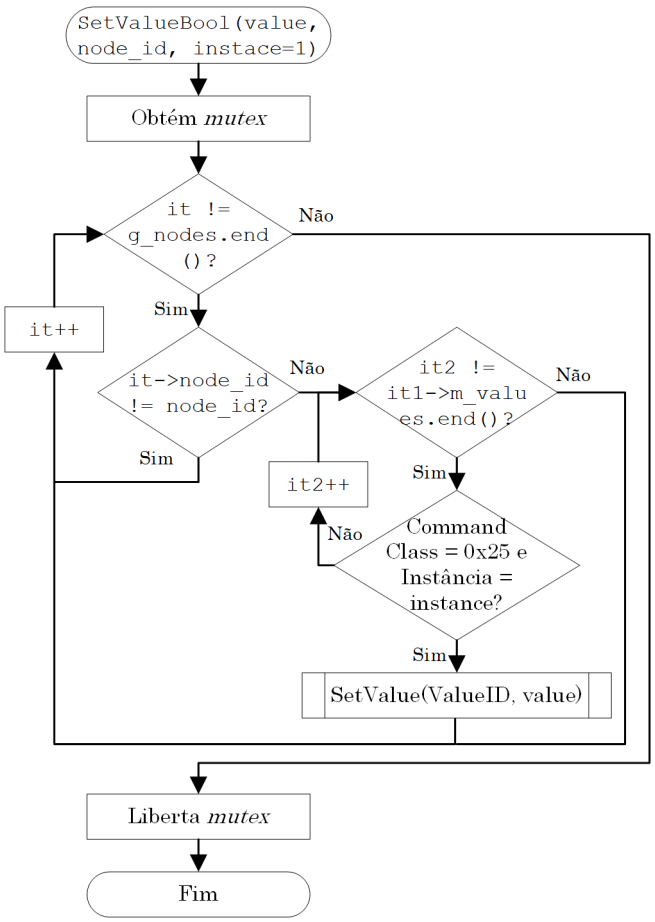


Figura B.1: Fluxograma da função `SetValueBool()`

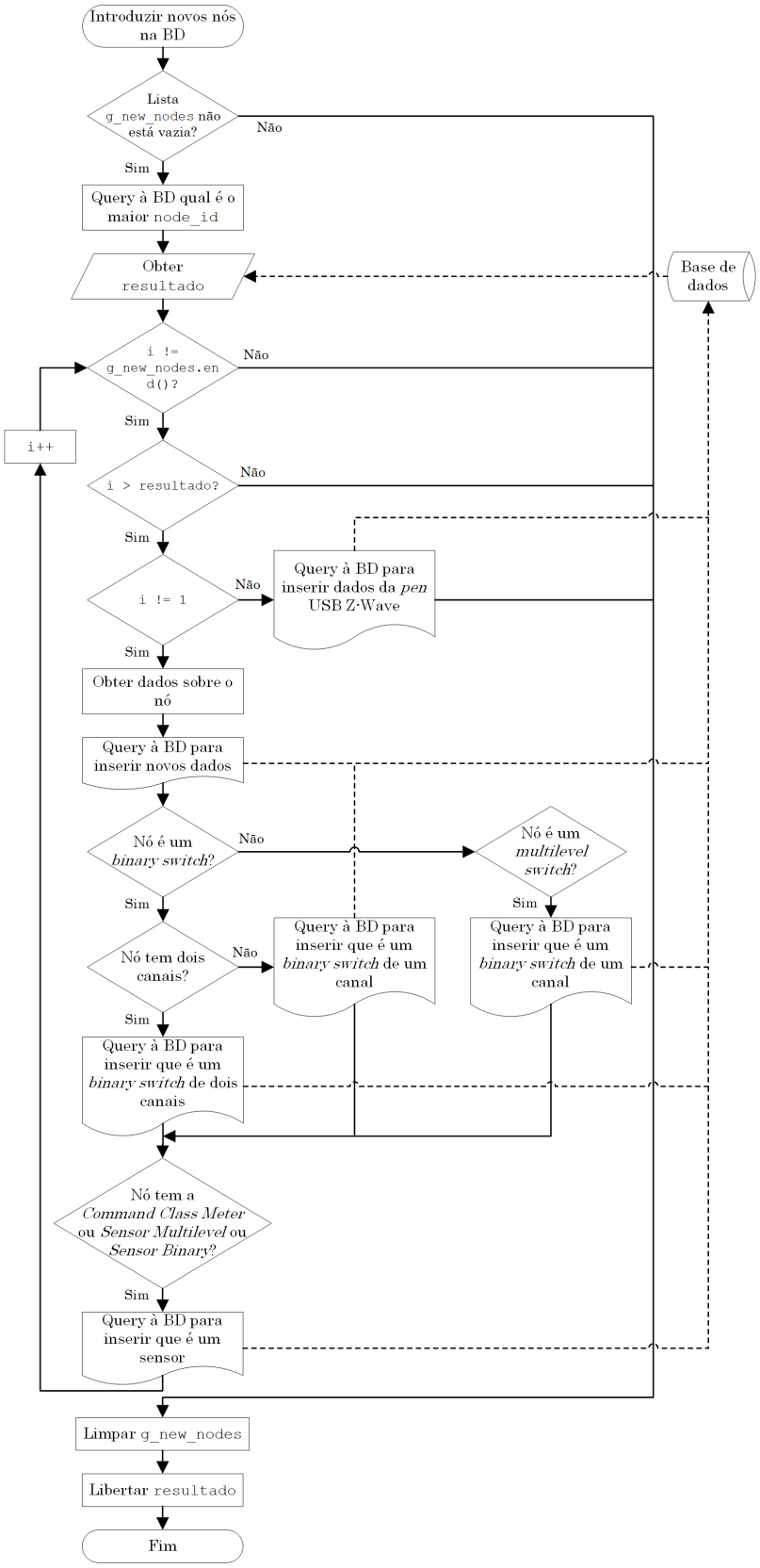


Figura B.2: Fluxograma do código de introdução de nós na base de dados

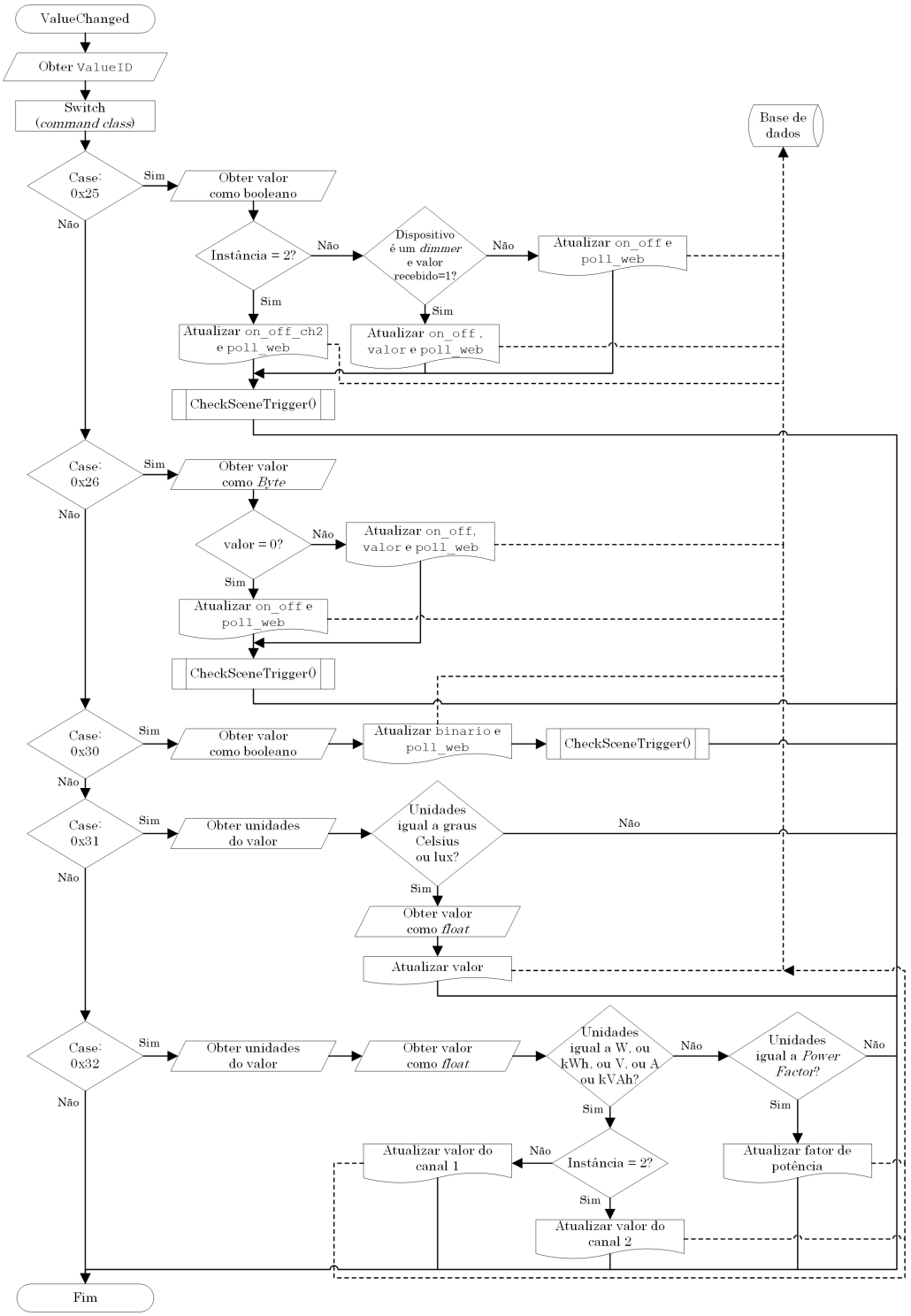


Figura B.3: Fluxograma do código de tratamento de notificações do tipo `ValueChanged`

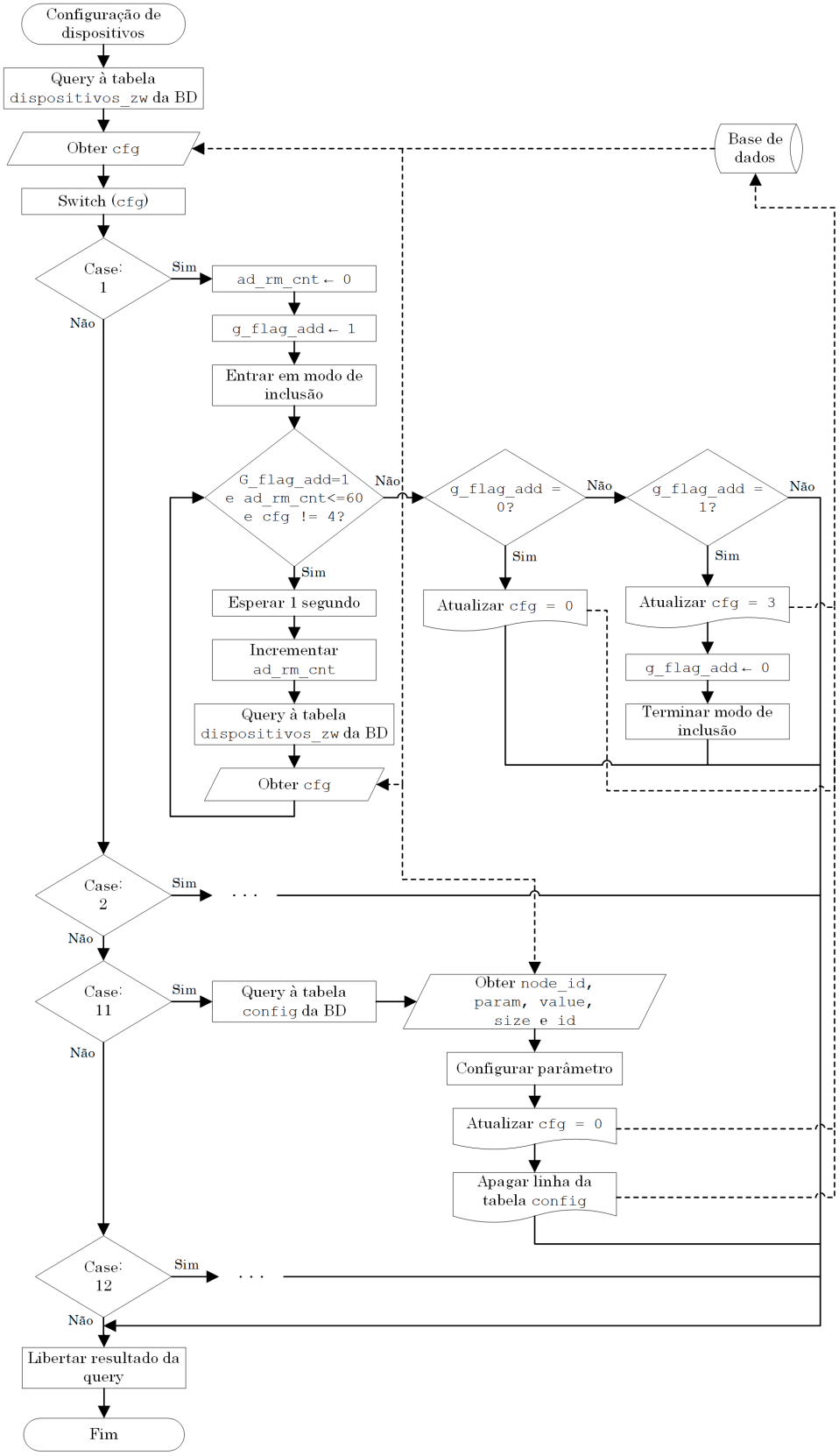


Figura B.4: Fluxograma da secção de código de configuração de dispositivos

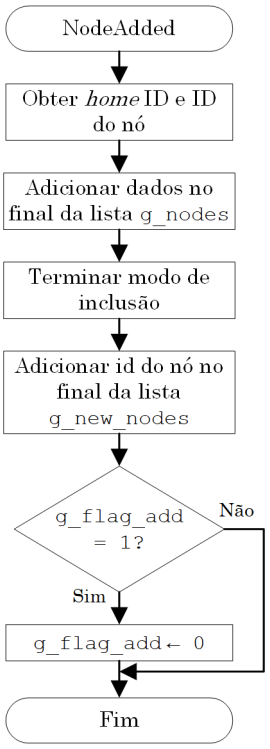


Figura B.5: Fluxograma do código das notificações do tipo `NodeAdded`

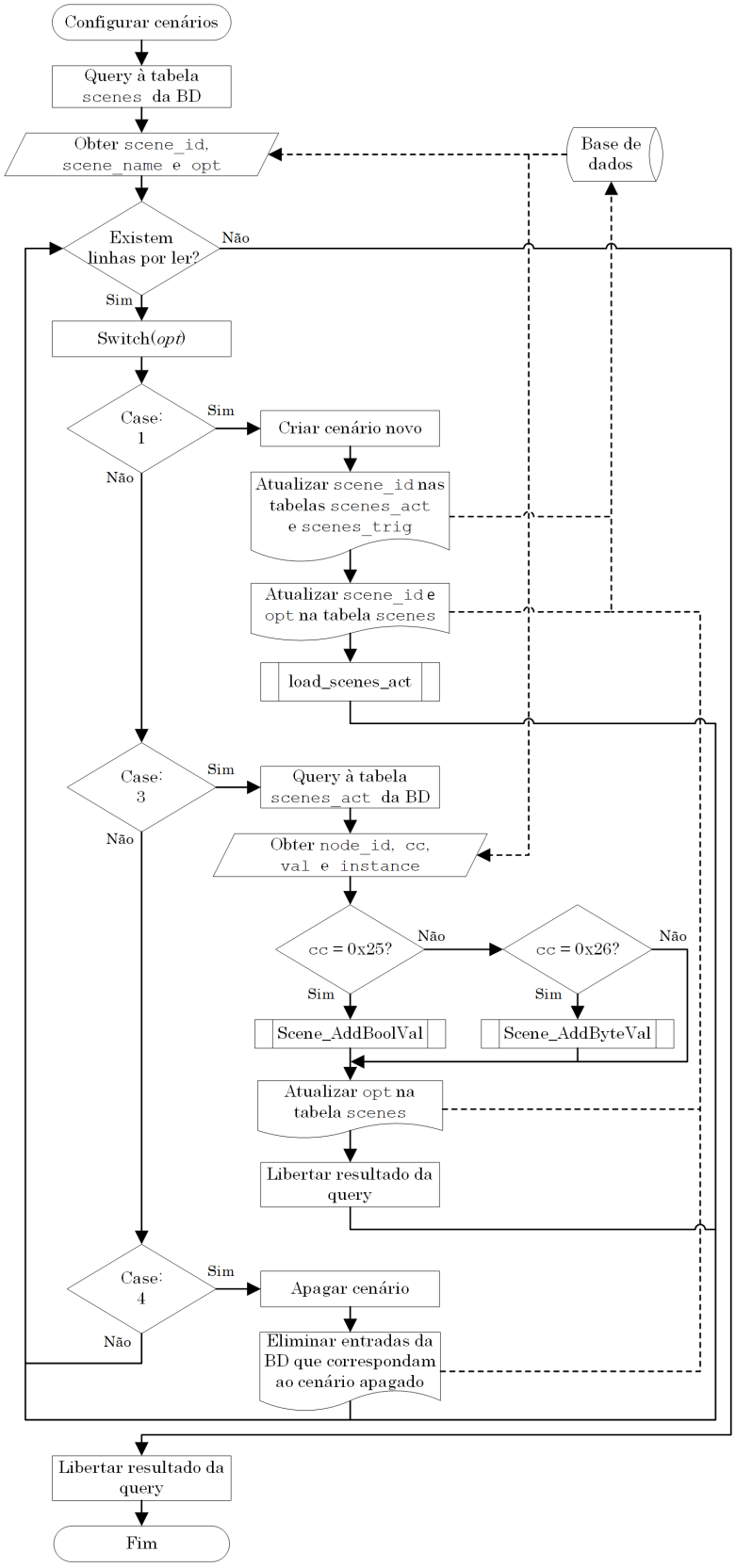


Figura B.6: Fluxograma da secção de código de configuração de cenários

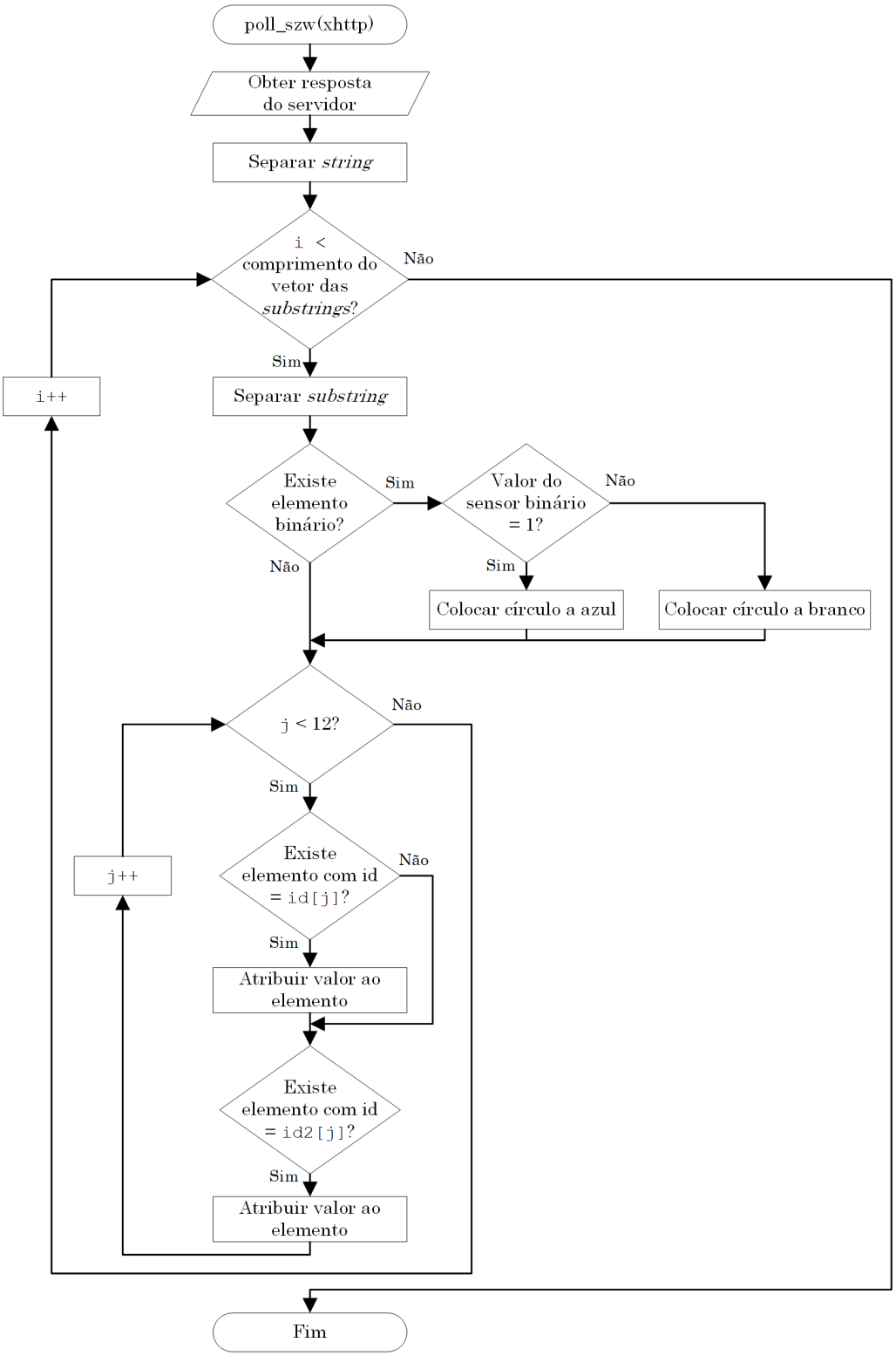


Figura B.7: Fluxograma da função `poll_szw()`

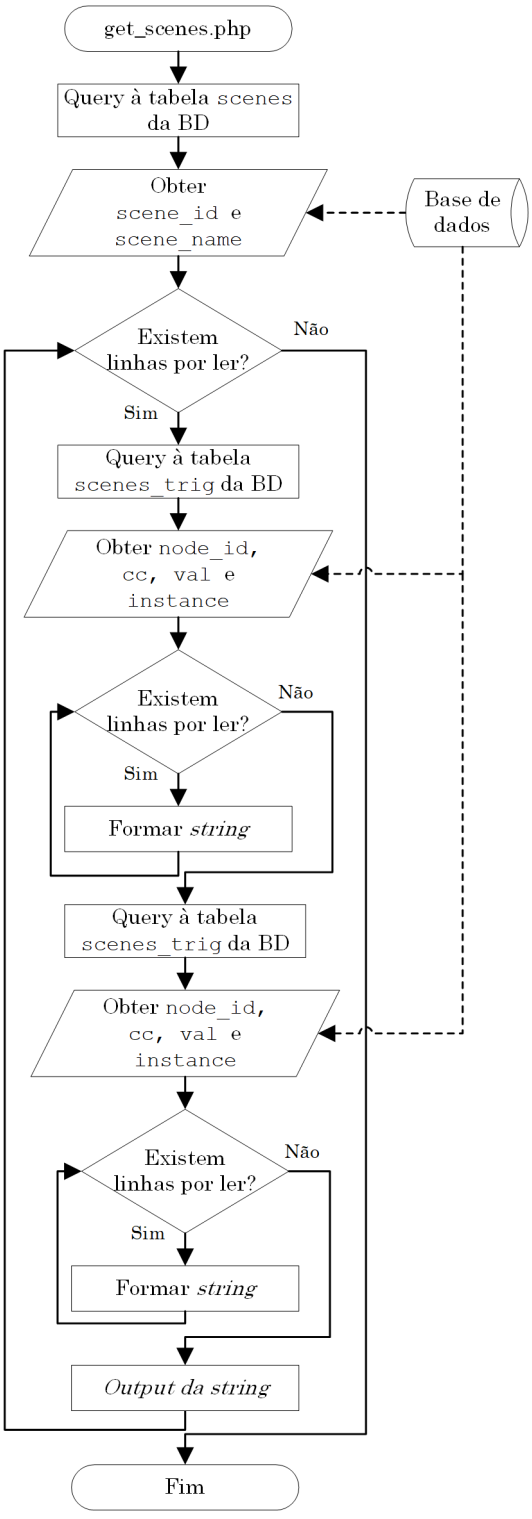


Figura B.8: Fluxograma do ficheiro `get_scenes.php`

Anexo C

Tabela Tipos Genéricos de Dispositivos Z-Wave

Na tabela seguinte, são apresentados alguns dos principais tipos genéricos de dispositivos Z-Wave e o correspondente número (*key*) que o identifica, em hexadecimal e decimal, no OpenZWave.

Tabela C.1: Tipos genéricos de dispositivos Z-Wave

Key (HEX)	*Key* (DEC)	Tipo genérico de dispositivo
0x01	1	*Remote Controller*
0x02	2	*Static Controller*
0x09	9	*Window Covering*
0x10	16	*Binary Switch*
0x11	17	*Multilevel Switch*
0x20	32	*Binary Sensor*
0x21	33	*Multilevel Sensor*
0x31	49	*Meter*