FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Monitorização Em Tempo Real

Ricardo Gonçalves

VERSÃO DE TRABALHO

Mestrado Integrado em Engenharia Eletrotécnica e Computadores

Orientador: André Restivo

Co-Orientador: Miguel Sousa

27 de Julho de 2018

Resumo

Ao longo dos anos, tem-se assistido a uma tendência de sofisticação tecnológica acentuada nos mais variados setores. A justificação assenta na necessidade de diferenciação dos produtos num mercado tecnológico cada vez mais competitivo e complexo. Consequentemente surge a necessidade de desenvolver ferramentas auxiliares que permitam a monitorização do bom funcionamento destas tecnologias após serem disponibilizadas no mercado. A AddVolt, uma *start-up* fundada em 2014, opera no cruzamento de dois grandes mercados: o de transporte e o da energia. A solução desenvolvida pela empresa consiste num sistema de armazenamento de energia que pode ser instalado, de forma não invasiva, em qualquer camião, permitindo assim alimentar eletricamente os sistemas auxiliares a bordo (tais como plataforma elevatória ou câmara frigorífica). A AddVolt é ainda detentora de uma tecnologia patenteada que permite a recuperação de energia durante as travagens nos veículos pesados, aumentando assim a autonomia em modo elétrico dos sistemas auxiliares.

Sendo os sistemas da AddVolt constituídos por diversos módulos, a identificação de eventuais problemas é uma tarefa de elevada complexidade, justificando a necessidade de desenvolvimento de uma ferramenta que permita a monitorização e registo de operações, em tempo real. Nesta ferramenta foi ainda implementada a funcionalidade de atualização de *firmware* usando *drag and drop*. A junção de todas estas funcionalidades possibilitou um aumento bastante significativo na rapidez do processo de monitorização, quando comparativamente com o conjunto de ferramentas anteriormente utilizadas pela empresa.

De modo a cumprir com as necessidades e requisitos desta aplicação, desenvolveu-se um servidor local de processamento e aquisição de dados e também uma interface gráfica. O servidor local foi desenvolvido em *Python 2.7*, sendo responsável pela aquisição dos dados provenientes da linha CAN implementada previamente nos sistemas da empresa e, posteriormente, responsável pelo processamento desses mesmos dados. A interface gráfica desenvolvida consiste numa *dashboard* implementada em *React* (*framework* de *JavaScript*) que cumpre com os objetivos de dispor de maneira intuitiva e inteligente a informação recebida e também de proporcionar aos utilizadores um meio simples de atualizar o *firmware* do sistema e consultar leituras efetuadas.

Após a implementação, os resultados obtidos da solução proposta comparativamente com a solução atual são: um terço das ferramentas necessárias para uma operação completa, tamanho da aplicação é aproximadamente quatro vezes maior, é mais rápido cerca de 40% e tem um acréscimo de 42% dos requisitos cumpridos.

Abstract

Over the years, a trend of technological sophistication has emerged on many different sectors. This is justified by the need to differentiate products in an extremely competitive and complex technological market. Consequently, the need to develop auxiliary tools to validate such technologies during the lifetime of the system arises. In this context, AddVolt launched the challenge of developing a validation method for monitoring the systems it produces.

AddVolt, a start-up company founded in 2014, operates at the union of two major markets: transport and energy. The developed solution consists of an energy storage system that can be installed non-invasively on any truck, enabling the auxiliary systems to be electrically supplied on board (such as lift platform or refrigeration chamber). AddVolt is also the owner of a patented technology that allows energy recovery when heavy vehicles brake, increasing the electrical autonomy of the auxiliary systems.

Since AddVolt systems are composed by several modules, problem identification is a highly complex task, justifying the need to develop a tool that allows the monitoring and registration of operations in real time. This tool should also be able to update the firmware module. The union of all these functionalities allowed a significant speed increase of the monitoring process, when compared to the set of tools previously used by the company.

In order to follow along with the needs and requirements of this application, it was developed a local server for processing and acquisition of data and also a graphical user interface. The local server is developed in *Python 2.7* and it is responsible for the data acquisition of the Controller Area Network previously implemented in the company's systems, and later responsible for the processing of all this data. The developed graphical user interface consists of a dashboard implemented in React (Framework of JavaSript) that fulfills the objectives of providing the received information in a intuitively and intelligently way, providing to users a simple way to update the system and consult older logs.

These were the obtained results when compared with the current solution: one-third of the tools necessary for a complete operation, size of the application is approximately four times greater, is faster about 40% and has a42% increase of the requirements fulfilled.

Agradecimentos

A realização e entrega desta dissertação é o resultado de todo o meu percurso académico, simbolizando o fim de uma etapa. Por essa razão quero agradecer a quem me ajudou a ultrapassar todos os desafios de uma forma direta ou indireta.

Ao meu **Orientador e Professor André Restivo**, pela disponibilidade e auxílio prestado sempre que precisei e também por me despertar interesse em aprender novas coisas com algumas conversas paralelas que foram surgindo durante estes meses.

Ao meu **Orientador da AddVolt Miguel Sousa**, por ser um excelente colega do dia-a-dia, por ajudar-me a melhorar o meu trabalho de forma metódica e ajudar-me a crescer enquanto trabalhador.

A **todos os trabalhadores da AddVolt** por me terem recebido da melhor forma possível e pelas boas práticas que me incutem de forma involuntária todos os dias apenas com o seu o profissionalismo e boa disposição com que trabalham. Em especial quero agradecer à equipa de engenharia, mais concretamente ao Leonel Araújo, José Pedro e Ricardo Soares que se mostraram sempre disponíveis para me ensinar novos conceitos e ajudar com dificuldades. Certamente este agradecimento estende-se a todos os outros departamentos.

A um nível mais pessoal quero agradecer à minha **família, namorada, amigos e inimigos** por estarem sempre presentes comigo ao longo destes anos quer eu esteja a celebrar ou a lamentar.

Por último quero agradecer à **FEUP** por ter sido uma ótima segunda casa durante os últimos anos. A toda a gente que referi, um muito obrigado!

Ricardo Abílio Leite Falcão Correia Gonçalves

"I have not failed. I've just found 10,000 ways that won't work."

Thomas A. Edison

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

CAN	Controller Area Network
MLU	Management Logging Unit
API	Application Programming Interface
GUI	Graphical User Interface
OSI	Open System Interconnection
AJAX	Asynchronous JavaScript And XML
UDP	User Datagram Protocol
OBD	On-Board Diagnostic
AES	Advanced Encryption Standard
WWW	*World Wide Web*

Capítulo 1

Introdução

Neste capítulo introdutório é efetuada uma contextualização sobre o tema da dissertação assim como uma apresentação da empresa que o propôs. Pretende-se ainda definir objetivos, definir uma metodologia e expor a estrutura que o documento irá seguir.

1.1 Contexto

O constante desenvolvimento tecnológico atingiu um nível em que é possível monitorizar toda a operação de um sistema em tempo real, com recurso à operação conjunta de *hardware* e *software* especializados. Nos mais variados setores estes tipo de sistemas tem vindo a proliferar-se [22]: mecânica, eletrónica, agricultura, vendas, etc.

A complexidade destes sistemas é variável em função das restrições temporais e das consequências que advêm de uma possível falha. De um lado do espetro de complexidade de sistemas *real-time* temos micro-controladores embutidos em eletrodomésticos e do outro sistemas militares ou sistemas de controlo de tráfego aéreo [17]. Estes sistemas podem ainda ser analisados do ponto de vista de restrições temporais, onde sistemas de monitorização e controlo poderão ter intervalos de atuação mais curtos, exigindo assim elevada performance.

Este tipo de sistema surge com a necessidade de validar tecnologias com recurso monitorização ou até mesmo do ponto de vista de diferenciação tecnológica num mercado competitivo. Foi então neste contexto que a empresa AddVolt avançou a proposta do tema de dissertação "*Real-Time Data Monitor*".

A AddVolt, uma *start-up* fundada em 2014, opera no cruzamento de dois grandes mercados: o de transporte e o da energia. A solução desenvolvida consiste num sistema de armazenamento de energia que pode ser instalado de forma não invasiva em qualquer camião, permitindo assim alimentar eletricamente os sistemas auxiliares a bordo (tais como plataforma elevatória ou câmara frigorífica). A AddVolt é ainda detentora de uma tecnologia patenteada que permite a recuperação de energia durante as travagens nos veículos pesados, aumentando assim autonomia em modo elétrico dos sistemas auxiliares. Consequentemente surge a necessidade de desenvolver ferramentas auxiliares que permitam a monitorização do bom funcionamento destas tecnologias após serem

disponibilizadas no mercado. Nesse sentido, a AddVolt lançou o desafio de desenvolver um método de validação e registo de monitorização em tempo real dos sistemas que produz.

1.2 Motivação

A competitividade é uma das caraterísticas do mercado tecnológico, o que obriga as empresas a investirem cada mais na diferenciação. Em muitos casos essa mesma diferenciação consiste apenas na forma como as empresas realizam o acompanhamento dos seus produtos e serviços e, para tal, por vezes surge a necessidade de se desenvolver ferramentas customizadas que permitam assegurar a qualidade do produto ao longo do respetivo ciclo de vida.

A empresa AddVolt, enquanto empresa a operar no setor tecnológico, apresenta precisamente essa necessidade, nomeadamente, de uma ferramenta que permita a monitorização e *debugging* dos seus sistemas. É nesse sentido que surge então o desafio de criar uma "*Real-time Data monitor tool*"para os seus produtos.

Atualmente, existem dois principais objetivos que justificam a necessidade dessa ferramenta:

- Controlo de qualidade de produção: De acordo com o referencial normativo ISO9001 implementado na empresa, todo e qualquer produto em fim de linha deverá ser inspecionado, e deverão ser utilizadas ferramentas de validação que permitam fornecer uma evidência da correta operação do produto antes da entrada no mercado;
- Manutenção: Uma vez no mercado, o apoio ao cliente será assegurado pelas entidades parceiras da AddVolt, e como tal, a empresa terá de lhes fornecer as ferramentas necessárias para possam realizar diagnósticos de forma rápida e precisa e consequentemente prestar um serviço de maior qualidade.

1.3 Objetivos

Após algumas notas introdutórias sobre o tema e sobre a motivação para a elaboração deste projeto, seguem os objetivos propostos:

- Revisão bibliográfica no âmbito de sistemas em tempo real, que engloba todo o percurso de dados desde a linha de comunicação até à representação da informação;
- Projeção do sistema computacional *real-time* a implementar;
- Implementação do sistema computacional *real-time*;
- Validação por meio de testes da solução implementada;
- Escrita do documento que descreve todo o procedimento utilizado na implementação, assim como uma análise dos resultados.

Pretende-se que no final da dissertação todos os objetivos sejam cumpridos com sucesso.

1.4 Metodologia

A abordagem para o desenvolvimento deste projeto irá ser dividida em várias fases sequenciais, tal como se observa na Figura 1.1:

- **Levantamento de requisitos:** Com base na descrição do problema, uma lista de requisitos foi evidenciada. Esta lista irá ser o ponto de partida tanto para uma proposta de solução como para a sua validação;

- **Proposta de solução:** A solução proposta deverá cumprir com todos os requisitos funcionais e estruturais e deverá ser uma solução genérica. Com isto entende-se que a solução seja independente das linguagens escolhidas e que funcione de maneira semelhante para diferentes escolhas;

- **Validação de Conceito:** O passo da validação de conceito é importante para garantir que com as tecnologias especificadas, é possível alcançar as funcionalidades básicas necessárias à escalabilidade da implementação;

- **Implementação:** A implementação irá ser dividida em duas etapas: Versão beta e versão final. Para a primeira versão irão apenas ser definidos alguns *use cases* a ser cumpridos. Com esta versão pretende-se mostrar o esqueleto da ferramenta e o funcionamento básico. Espera-se também *feedback* dos clientes para ajustar o rumo da ferramenta conforme os requisitos que poderão surgir. A versão final irá conter todos os *use cases* estipulados no início do projeto;

- **Validação de proposta:** Após o lançamento da versão final, é necessário validar a solução através de testes que mostrem que a ferramenta veio resolver/melhorar o problema existente. Para isso, irão se estabelecer vários testes que terão em vista a validação desta proposta;

- **Análise de resultados:** Após a execução dos testes de validação irá proceder-se a uma análise de resultados, tanto de uma perspetiva de resultados dos testes, como do cumprimento dos requisitos.

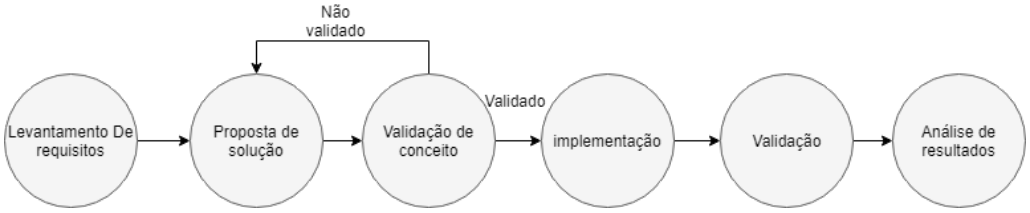

Figura 1.1: Esquema da metodologia adotada

1.5 Estrutura do Documento

Este documento divide-se em 7 capítulos. Cada um destes capítulos aborda o tema da dissertação em diferentes fases da sequência lógica adotada para o desenvolvimento do tema. Apresenta-se de seguida, de forma sintetizada, o conteúdo de cada um desses capítulos.

No Capítulo 1 foi efetuada uma contextualização sobre o tema da dissertação assim como uma apresentação da empresa que o propôs. Foram ainda definidos os objetivos a cumprir e exposta a metodologia adotada.

No Capítulo 2 foi efetuado um levantamento bibliográfico sobre tecnologias e técnicas relacionadas com o tema da dissertação.

No Capítulo 3 foi realizada uma contextualização do problema proposto pela AddVolt e uma descrição das condições de uma possível solução, sendo posteriormente listados os requisitos que deverão ser contemplados na proposta de solução.

No Capítulo 4 foi apresentada uma visão geral da solução proposta e validação do conceito descrito.

No Capítulo 5 foi demonstrado detalhadamente todo o processo de implementação da solução proposta.

No Capítulo 6 foram efetuados testes de verificação de validação da solução do ponto do vista de resolução do problema proposto, sendo posteriormente efetuada uma análise dos resultados.

No Capítulo 7 abordaram-se as conclusões retiradas no decorrer da dissertação e indicaram-se algumas sugestões de trabalho futuro.

Capítulo 2

Estado da Arte

Sendo o objetivo deste projeto o de desenvolver uma ferramenta de monitorização de variáveis em tempo real, várias áreas tecnológicas irão ser abordadas nas fases de aquisição, processamento e representação de dados. Este capítulo irá servir então para conhecer e entender as tecnologias e técnicas utilizadas em cada uma das fases mencionadas. Desta maneira, poupa-se tempo de investigação e desenvolvimento e adquire-se também conhecimento atual sobre o tema. O capítulo irá dividir-se em:

- **Sistemas *Real-Time*** – onde se irá abordar a definição e distinção de diferentes tipos de sistemas *real-time*. Sendo a base de desenvolvimento deste projeto o sistema da AddVolt que fornece informações em tempo real sobre o seu estado, este tema apresenta-se bastante relevante para a tese;
- ***Controller Area Network* (CAN)** – onde se irá recolher informações acerca da relevância e utilidade deste protocolo, especificações nos diferentes níveis de comunicação, métodos de aceder à linha CAN e ainda diferenças entre *standards* CAN utilizados em níveis mais superiores da comunicação. Este tema é alvo de pesquisa porque o sistema de AddVolt está equipado com uma rede CAN, onde circulam mensagens alusivas a todos os módulos do sistema. Fazendo com que o estudo aprofundado deste protocolo seja uma mais valia para o decorrer do projeto;
- ***Dashboards*** – onde se irá abordar o tema, investigando os tipos e propósitos de *dashboards* existentes, assim como uma comparação ao nível de software existentes para a construção dos mesmos. Como um dos objetivos deste projeto é de desenvolvimento de uma interface gráfica, é importante entender os mecanismos de construção e a relação entre eventuais requisitos e respetivas funcionalidades;
- **Aplicações Existentes** – Esta última secção analisa uma arquitetura de processamento de dados que poderá ser um bom ponto de partida para arquitetura da aplicação a desenvolver, assim como um modelo de desenvolvimento de *software*. Desta maneira é pertinente abordar este tema com relativo detalhe, porque permite a análise geral de todo o processo de desenvolvimento de uma ferramenta deste género.

2.1 Sistemas *Real-Time*

2.1.1 Visão geral

Define-se por sistema computacional em tempo real, qualquer sistema em que o comportamento esperado dependa não só de resultados lógicos de operações, mas também do instante em que os resultados são produzidos. Um sistema computacional deste tipo é apenas uma parcela daquilo a que se chama *real-time system* ou até mesmo *cyber-physical system* [22]. O estado destes sistemas varia em função de tempo, ou seja, mesmo depois do sistema computacional ser desligado. A Figura 2.1 decompõe em três componentes um sistema em tempo real:

- **Componente física** - *Hardware* a ser controlada/monitorizada;
- **Componente operacional** - Sistema computacional em tempo real;
- **Componente humana** - Interação humana.

A interação entre a componente física e a componente operacional é denominada interface de instrumentação, onde são feitas as conversões analógicas-digitais (e vice-versa) dos valores através de sensores e atuadores. A interação entre a componente humana e a componente operacional é denominada interface homem-máquina, onde interação é feita através de teclados ou *displays*.

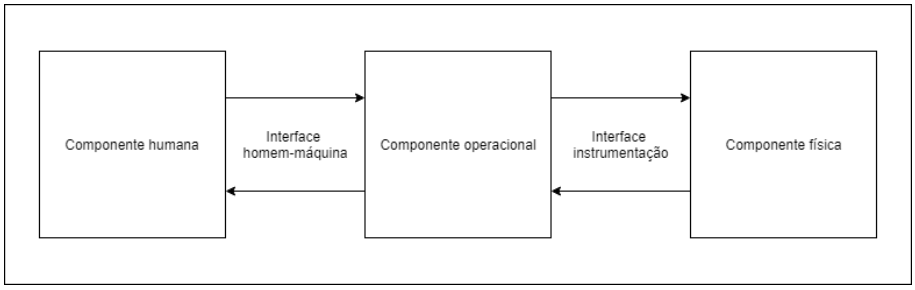

Figura 2.1: Relações entre componentes de um sistema em tempo-real

2.1.2 Classificação de sistemas *Real-time*

Um sistema computacional *real-time* deve reagir em função de mudanças de estímulos provenientes da componente física ou da componente humana. O instante em que o sistema deve reagir é denominado *deadline*. A classificação de um sistema em tempo-real pode ser classificada segundo as seguintes condições [22][17]:

- Caso o resultado tenha utilidade mesmo após o tempo de reação ter ultrapassado a *deadline*, pode classificar-se em ***soft real-time system***;
- Caso o resultado não tenha utilidade após tempo de reação ter ultrapassado a *deadline*, pode classificar-se em ***firm real-time system***;

- Caso o tempo de reação ultrapasse a *deadline*, e daí resultarem consequências indesejadas, pode classificar-se em ***hard real-time system***.

A classificação acima descrita é feita segundo a perspetiva das características da aplicação e as suas necessidades, tendo em conta a validade dos resultados e as consequências após o tempo de reação exceder a *deadline*.

É importante que a classificação de um sistema seja feita antes da implementação e do *design* do mesmo, pois a implementação pode ser direcionada de maneira diferente consoante as necessidades e aplicabilidade. Além das características que foram tidas e conta na classificação anterior, há mais aspetos a equacionar:

- Disponibilidade do sistema - Pode ou não ser requerido que o sistema apresente uma resposta em caso de sobrecarregamento ou falha. Podendo assim o sistema ser classificado em sistemas de resposta garantida ou de reposta sempre que possível;
- Recursos disponíveis - Sistemas em que há sempre reposta, seguem o principio que os recursos disponíveis são os mais adequados. No entanto, pode assumir-se desde o inicio que os recursos a utilizar não são os mais adequados, admitindo-se assim que pode não haver resposta em todas a situações e que a qualidade da mesma pode ter limitações;
- Desencadeamento de resposta - É expectável que o sistema responda em função de certos eventos ocorridos ou então e função de certos instantes. Deve ter-se em conta os *triggers* do sistema aquando o planeamento do desenvolvimento.

Alguns sistemas que despoletam repostas através de eventos específicos servem-se de protocolos de comunicação como CAN, *ethernet* ou *User Datagram Protocol* (UDP) [17]. O foco deste levantamento bibliográfico é neste tipo de sistemas condicionados por eventos, mais concretamente nos que fazem uso do protocolo CAN.

2.2 *Controller Area Network*

2.2.1 Visão Geral

CAN é um protocolo de comunicação que foi criado para a indústria automóvel em 1986 por *Robert Bosch*, sendo utilizado hoje em dia em diferentes indústrias [36][37]. O principal objetivo deste protocolo era o de simplificar as complexas cablagens que permitiam a comunicação entre os diversos micro-controladores (nós) existentes nos componentes dos veículos, permitindo uma monitorização e controlo dos mesmos. Ao longo dos anos tem vindo a verificar-se uma crescente utilização destes micro-controladores na indústria automóvel, sendo o protocolo CAN amplamente utilizado e assunto de *standards* internacionais aprovados pela ISO (*International Standard Organization*)[37].

Embora o protocolo CAN já tenha surgido há mais de 20 anos, a utilização do mesmo está cada vez mais presente nas mais variadas indústrias e com standards cada vez mais evoluídos e

específicos. As características deste protocolo que se seguem foram preponderantes na afirmação e proliferação do CAN nas indústrias [6][37][24]:

- **Rede de custo reduzido**: Redes CAN apresentam uma diminuição na cablagem que resulta numa diminuição no custo do fabrico e no peso dos carros, pois cada nó tem apenas dois fios conectados (independentemente do número de nós), eliminando ligações analógicas e digitais entre cada um dos nós;
- **Comunicação *Broadcast***: Como cada nó é inteligente (possuindo um micro-controlador) e todas as mensagens são enviadas para todos os nós (*Broadcast*), cada nó tem liberdade para ignorar ou recolher cada uma das mensagens da rede;
- **Prioridade**: Cada mensagem tem um grau de prioridade, evitando assim que dois nós transmitam ao mesmo tempo e que se perca informação;
- **Capacidade de gestão de erros**: O protocolo CAN possui campos específicos em cada mensagem para controlo de erros (tal como o *CRC*). Desta maneira, é possível desconectar um certo nó da rede se este gerar muitos erros.

A especificação deste protocolo tem em vista tornar compatíveis implementações de diferentes nós CAN, permitindo a comunicação entre os mesmos. A compatibilidade depende da parte elétrica (*Hardware*) e da forma como os dados são interpretados. Para alcançar a uniformização, foram então definidas 3 distintas secções numa implementação CAN: *Object Layer*, *Transfer Layer* e *Physical Layer* [6].

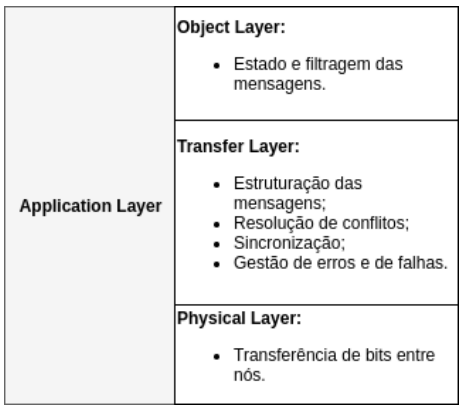

Figura 2.2: CAN *layers*

(CAN) *Object Layer* é a secção responsável por selecionar as mensagens a transmitir e por selecionar do *Transfer Layer* as mensagens a ser utilizadas [6]. A seleção das mensagens a transmitir e a receber varia conforme as necessidades de cada nó.

(CAN) *Transfer Layer* é a secção responsável pela estruturação das mensagens, resolução de conflitos, sincronização, deteção e sinalização de erros, confirmação de ações e deteção de falhas em nós da rede [6].

Physical Layer é a secção responsável pela transferência de bits entre os diferentes nós [6]. A única condição existente nesta secção é a que o meio físico seja o mesmo em toda a rede de nós.

A especificação do protocolo CAN foca-se essencialmente no protocolo de transporte (*Transfer Layer*) e nas consequências deste nas secções adjacentes [23]. A Figura 2.2 (baseado em [23]) comprime as funções das 3 secções numa tabela.

2.2.2 Mensagens CAN

No protocolo de comunicação CAN existem dois formatos distintos usados para a estrutura das mensagens: **Standard CAN (*version* 2.0A)** e ***Extended* CAN (*version* 2.0B)** [6]. Estes dois apenas diferem num dos campos da estrutura das tramas: o identificador (11 bits no Standard CAN e 29 bits no *Extended* CAN) [24]. Independentemente do formato usado, apenas existem 4 tipos de mensagens diferentes: *Data Frame*, *Remote Frame*, *Error Frame* e *Overload Frame*. Como cada uma destas mensagens tem uma estrutura única, sendo necessária uma análise detalhada de cada campo.

2.2.2.1 *Data Frame*

A Figura 2.3 contém o esquemático de uma *Data frame*. Este tipo de mensagens é usado de transmissores para recetores [37] [6]. Relativamente aos campos desta mensagem:

- ***Start frame* (1 bit)**:É um campo comum às *Data Frames* e às *Remote Frames*, sendo útil na sincronização do sistema. Este campo consiste simplesmente num bit recessivo (Conceito de bit recessivo e bit dominante abordados mais à frente);
- ***Arbitration* (12 bits ou 30 bits)**: este campo é subdividido em dois distintos campos *Identifier* e RTR BIT. Caso seja uma *Data frame*, o campo RTR BIT tem o valor de recessivo. Caso seja uma *Remote frame* assume o valor de dominante. (Nota: O nº de bits do campo *Identifier* difere consoante *standard* ou *extended* for usado);
- ***Control Field* (6 bits)**: Este campo engloba o campo *Data length code* de 4 bits onde é representado o nº de bytes do campo Data. Os restantes dois bits são reservados para futura expansão;
- **Data (até 64 bits, podendo ter 0 bits)**: Campo destinado para alocar a informação a transmitir;
- **CRC (15 bits)**: Campo utilizado para controlo de erros. É subdividido em CRC *Sequence* (14 bits) e *Delimiter* (1 bit). O Primeiro é calculado com base nos coeficientes dos bits da mensagem e o segundo é um bit 0 que assinala o fim do campo CRC;
- **ACK (2 bits)**: Campo subdividido em *ACK Slot* e *ACK Delimiter*. Se a mensagem tiver origem num transmissor, estes dois campos assumem o valor de dominante. Caso tenha origem no recetor, o último bit é recessivo;

- ***End Frame* (7 bits)**: Assinala o fim de uma mensagem e cada um dos 7 bits tem o valor dominante;

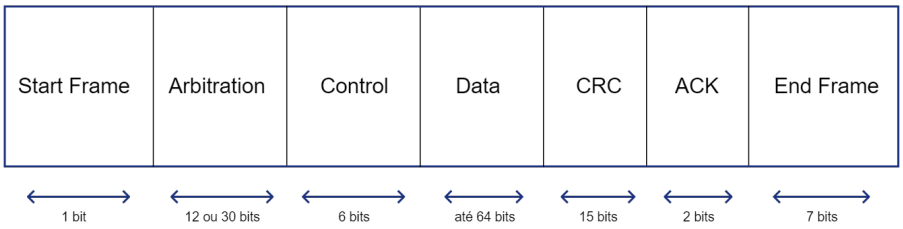

Figura 2.3: *Data Frame*

2.2.2.2 *Remote Frame*

A Figura 2.4 contém o esquemático de uma ***Remote frame***. Este tipo de mensagens é utilizado para solicitar informação de algum dos nós da rede. Sendo normalmente este tipo de mensagens precedido por uma *Data frame* [37] [6]. Apenas existem 3 diferenças neste tipo de mensagem, relativamente a uma *Data frame*:

- O bit RTR do campo *Arbitration* assume agora o valor de recessivo;
- Não existe campo *Data*;
- O valor do campo *Data length code* assume o valor correspondente ao número de *bytes* do campo *Data* da *Data frame* associada.

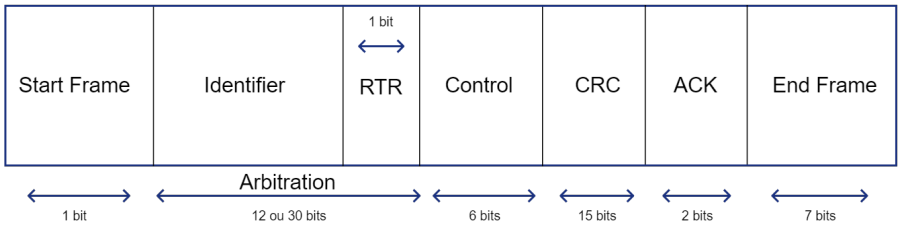

Figura 2.4: *Remote Frame*

2.2.2.3 *Interframe space*

As mensagens do tipo ***Data Frames*** e ***Remote Frames*** são separadas por uma ***Interframe Space*** [37][6], como se verifica na Figura 2.5. Esta mensagem pode conter até 3 campos:

- ***Intermission* (3 bits)**: Consiste em 3 bits dominantes e durante este campo, nenhum nó pode começar a transmitir. Apenas *Overload frames* podem ser geradas;

- ***Bus Idle***: o número de bits deste campo é arbitrário e indica que os nós podem começar a transmitir. Quando surge um bit recessivo neste campo, este é visto como uma *start frame*;
- ***Suspend Transmission***: Após ser enviado um bit recessivo durante o campo *bus idle*, a estação que estava a transmitir passa a ser vista como um recetor e recebe a mensagem gerada por um dos nós transmissores.

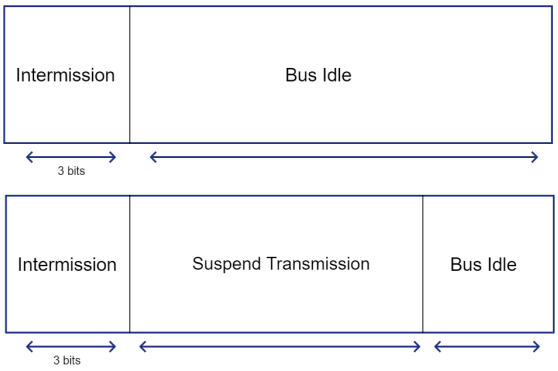

Figura 2.5: *Interframe Space*

2.2.2.4 *Error Frame*

A Figura 2.6 contém o esquemático de uma **Error frame**. Estas mensagens são utilizadas quando são detetadas falhas no sistema e contém apenas dois campos [24][6]:

- **Error Flag (6 - 12 bits)**: Resulta da sobreposição de Error Flags provenientes de todos os nós e possuem entre 6 e 12 bits;
  - **Active Error Flag**: 6 bits com valor de recessivo;
  - **Passive Error Flag**: 6 bits com valor de dominante ou mais de bits (até 6) provenientes de outros nós.
- **Error Delimiter (8 bits)**: 8 bits dominantes consecutivos, assinalando o fim da mensagem.

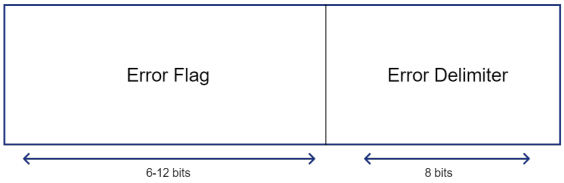

Figura 2.6: Error Frame

2.2.2.5 Overload Frame

A figura 2.7 contém o esquemático de uma **Overload frame**. Este tipo de mensagens ocorre em duas situações: Quando um recetor necessita de um delay entre uma *Data frame* e *Remote frame* (Começando no primeiro *bit* do campo *intermission* e podem ser enviadas até duas *Overload frames* consecutivas) e quando é detetado um *bit* dominante durante o campo *Intermission* de uma mensagem *Interframe* (Começando no *bit* após o *bit* 1 do campo intermission)[6]. Este tipo de mensagens contém dois campos:

- **Overload Flag**: 6 bits recessivos;
- **Overload Delimiter (8 bits)**: Semelhante a uma trame Error Delimiter com 8 bits dominantes.

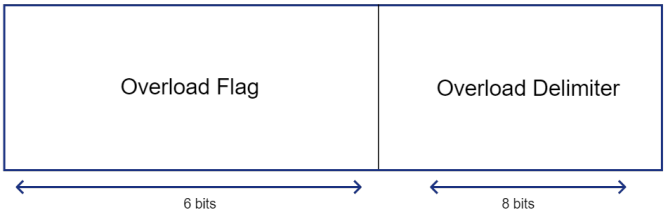

Figura 2.7: Overload Frame

2.2.3 CAN Standards

A implementação do protocolo CAN engloba ainda processos de codificação (*Bit Stuffing*), sincronização e mecanismos complexos de deteção e controlo de erros. Por essa razão surgiram *standards* do protocolo CAN que se focam em níveis superiores do protocolo, não precisando o programador de lidar com este tipo de tarefas executadas num nível inferior do modelo *Open System Interconnection* [1]. A Figura 2.8 representa os níveis existentes no modelo OSI. Alguns exemplos de standards CAN direcionados para a indústria automóvel são: CANopen, CANfd, Device Net, FTT-CAN, entre outros. Contudo, cada um destes protocolos poderá ser estendido e tornando-se específico de um fabricante, como é o caso do *standard* CAN GMLAN, específico do fabricante General Motors [1].

2.2.3.1 CANopen

Como foi referido anteriormente, este *standard* foi desenvolvido em níveis superiores da camada OSI. Portanto, mecanismos como o de *bit stuffing* ficam fora da responsabilidade do desenvolvedor. O CANopen contém ainda duas funcionalidades de elevada utilidade que são as de permitir ligações ponto-a-ponto e, através de objectos de comunicação (COB), é possível definir o comportamento de cada nó na rede.

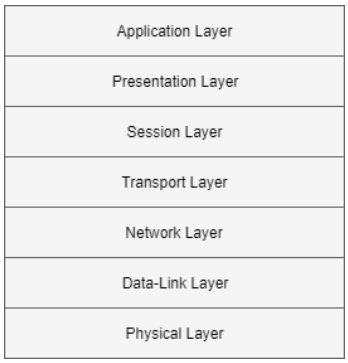

Figura 2.8: OSI Model

O CANopen pode ser dividido em três partes[20]:

- **CANopen *Protocol*:** responsável pela comunicação através da rede CAN. Engloba protocolos como SDO, PDO, NMT e protocolos de controlo de erros;
- ***Application Software*:** Responsável pelo controlo de funcionalidades, sendo uma interfaces para as interfaces de hardware;
- **CANopen *object dictionary*:** Contém índices para todos os tipos de dados usados, guardando todos os parâmetros da aplicação e comunicação.

2.2.3.2 CANfd

Este *standard* é um dos mais recentes e foi desenvolvido com ambição de satisfazer os requisitos de largura de banda cada vez mais exigentes no ramo automóvel [5]. A ideia deste *standard* assenta na métrica de que quando apenas um nós está a transmitir na rede, a *bitrate* pode ser aumentada, pois não há necessidade de sincronismos com outros nós. Portanto, como se observa na Figura 2.9, na fase de *arbitration* e ACK, a *bitrate* está limitada a 1Mb/s por causa da necessidade de sincronismo. Sendo que na fase de transmissão de dados, a *bitrate* está limitada apenas às características físicas da rede.

Figura 2.9: *Bit-Rate* em CANfd

2.2.3.3 FTT-CAN

A particularidade deste *standard* CAN é a de permitir a existência de tráfego síncrono e tráfego assíncrono na mesma rede CAN [2]. Contém ainda mecanismos de rotas alternativas e mecanismos de comunicação, usando sempre o mesmo CANbus. No entanto, tem a desvantagem de aumentar o número de *frames* na rede, devido ao uso mensagens de controlo, limitando a *bitrate* máxima e degradando a performance.

2.2.4 *Physical Layer*

Na especificação ISO11898 (Especificação CAN Bosch), onde são definidas especificações dos object and transfer layer, não há especificações para o physical layer [6]. Desta maneira, os fabricantes poderiam optimizar o protocolo, adaptando o physical layer às necessidades da aplicação. No entanto, com a crescente popularidade deste protocolo foram criados stantards também para esta secção, facilitando a implementação deste protocolo [23].

Independentemente *standard* utilizado, a comunicação CAN é feita através de dois fios (CAN-H e CAN-L) que são conectados a todos os dispositivos. Os sinais têm a mesma sequência de dados, mas amplitude oposta. Desta maneira o circuito fica mais imune a interferências eletromagnéticas e a probabilidade de se corromper dados é menor [37]. Como a Figura 2.10 representa, se um pulso for enviado pelo fio CAN-H, este apresentará uma voltagem entre 2.5V e 3.75 V. Caso seja enviado pelo CAN-L, este apresenta uma amplitude entre 2.5V e 1.25V. Para efeitos de nomenclatura, um bit com tensão igual 2.5V, é um bit 0 (*dominant* bit) e um bit com tensão igual a 0 é bit 1 (*recessive* bit) [37].

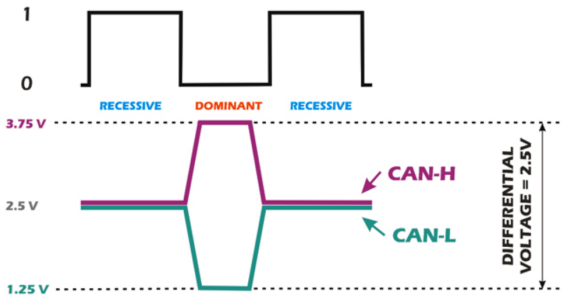

Figura 2.10: Can-High e Can-Low

Relativamente à velocidade e cablagem existem quatro standards [37], sendo os mais utilizados os seguintes:

- **High Speed CAN(ISO 11898-2):** Permite baud rates compreendidas entre 40 Kbit/s e 1 Mbit/s, dependendo do comprimento do fio [23]. Cada uma das extremidades da rede possuí uma resistência de 120 *ohm*. É o standard mais utilizado devido à facilidade de cablagem. Esta topologia está representada na Figura 2.11;

- **Low Speed CAN(ISO 11898-1):** Permite baud rates compreendidas entre 40 Kbit/s e 125 Kbit/s, dependendo do comprimento do fio [23]. Neste standard, a topologia da rede permite continuar a comunicação no caso de um fio apresentar problemas. Cada um dos dispositivos tem a sua própria resistência. Esta topologia está representada na Figura 2.12.

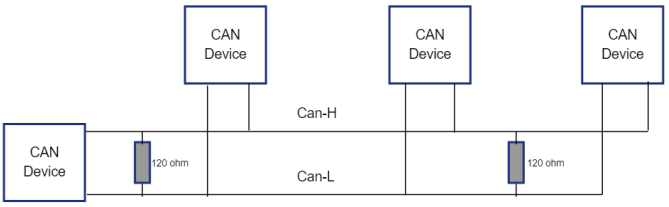

Figura 2.11: High Speed CAN

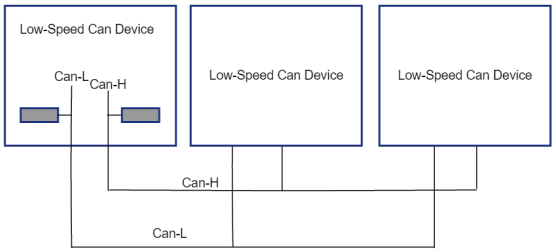

Figura 2.12: Low Speed CAN

De modo a existir interoperabilidade, cada nó tem uma estrutura semelhante. Por essa razão, um micro-controlador e um *transceiver* são componentes obrigatórios. O *transceiver* é conectado à rede enviando os dados para o micro-controlador (e vice-versa), através de um componente do do controlador). A Figura 2.13 mostra o esquemático de um nó.

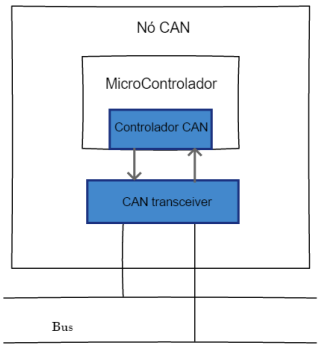

Figura 2.13: Esquemático de um nó

A Tabela 2.1 especifica sintetiza ainda a relação da *baud rate* máxima para um certo comprimento de fio [23].

Can Bus Maximum Length

Bus Speed	Bus Length (L)	Cable Stub Length (l)	Node Distance (d)
1 Mbit/Sec	40 meters (131 feet)	0.3 meters (1 foot)	40 meters (131.2 feet)
500 kbits/Sec	100 meters (328 feet)	0.3 meters (1 foot)	100 meters (328 feet)
100 kbits/Sec	500 meters (1640 feet)	0.3 meters (1 foot)	500 meters (1640 feet)
50 kbits/Sec	1000 meters (3280 feet)	0.3 meters (1 foot)	1000 meters (3280 feet)

Tabela 2.1: *Bus Speed* Vs Comprimento do fio

2.2.4.1 Acesso à rede CAN

A vantagem das redes CAN é a de poder conectar-se à rede, atualizar e monitorizar a informação respetiva de cada nó. Para aceder à linha CAN, é necessário um *On-Board Diagnostics* (OBD), que é um sistema que permite o acesso direto à rede CAN através de conector J1962 de 16 pinos [1]. Este pode ser encontrado na maior parte dos veículos, normalmente no porta-luvas. Um esquemática de um OBD é o da Figura 2.14.

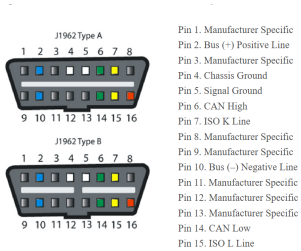

Figura 2.14: Esquemático do *pinout* de um OBD

No entanto, para se conectar um dispositivo (p.e. computador) a estes conectores, é necessário um adaptador (DB-9) que permita converter o sinal para uma porta USB [1]. A Figura 2.15 representa a arquitetura completa de um sistema com uma rede CAN implementada.

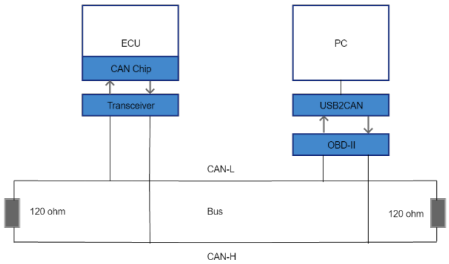

Figura 2.15: Arquitetura completa

2.3 *Dashboard*

2.3.1 Visão Geral

Com o avanço tecnológico e estratégico, a quantidade de dados recolhidos e processados tem vindo a aumentar significativamente [35]. Estes dados podem assumir diferentes formatos e tamanhos, tornando complicado o seu processamento. Por essa razão desenvolvem-se ferramentas que facilitam a manipulação e representação destes dados, traduzindo-os em informação valiosa por meios de *dashboards*. Um *dashboard* é um meio de visualizar os dados recolhidos de um sistema/organização fornecendo ao utilizador informação que lhe permite tomar decisões com base no que se observa ou então monitorizar a atividade [30].

Os *dashboards* são utilizados em diversas áreas como saúde, educação e indústrias tecnológicas [35]. Embora o propósito, tipo de representação e funcionalidade dos *dashboards* não sejam os mesmos, estes são úteis por aumentarem a capacidade de decisão do utilizador. Por exemplo, na área da Educação, são usados *dashboards* para medição da performance dos estudantes nas diferentes atividades, facultando informações sobre os seus resultados podendo estas ter impacto nos seus objetivos e na sua atitude [15]. Já num contexto da indústria automóvel, os *dashboards* tanto são utilizados em contexto de manutenção como de utilização de um veículo [25]. Neste último, o condutor necessita de dados sobre a autonomia e velocidade atuais do veículo, de modo a adaptar a sua condução. Já do ponto de vista da manutenção, o foco é em dados provenientes dos componentes existentes num veículo que fornecem informações sobre o estado dos mesmos [25].

Portanto, o design de um *dashboard* é concebido depois de se definir o tipo de informação que se pretende observar, ou seja, depois de se definir o propósito. Este processo é mais complicado do que aparenta, pois o tipo de informação expectável num *dashboard* varia de utilizador para utilizador. É então possível identificar cinco diferentes propósitos [35]:

- **Consistência:** Representação da performance de uma forma estandardizada, contendo procedimentos, medidas e valores;
- **Monitorização:** Avaliação da performance, com base em valores pré-definidos, seja performance individual ou coletiva;
- **Planeamento:** Planeamento futuro de alguma tarefa ou plano;
- **Análise:** Análise da performance, tendo em vista uma posterior tomada de decisão;
- **Comunicação:** Partilha de informação entre utilizadores diferentes.

Conclui-se então que, do ponto de vista do utilizador, o propósito de um *dashboard* varia consoante a necessidade de manipulação dos dados, que pode ser classificada em três níveis [25]:

- **Dashboard estratégico**: Este tipo de *dashboards* serve para monitorizar a implementação de objetivos estratégicos e para rever os índices de performance;

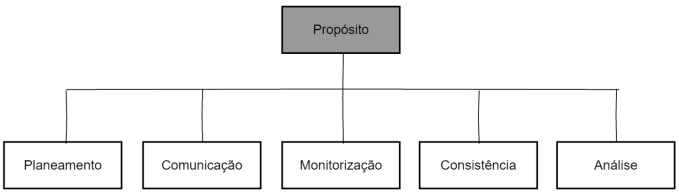

Figura 2.16: Propósitos de um *dashboard*

- **Dashboard tático:** Comparado com o Dashboard estratégico, confere informação mais detalhada. Sendo utilizado para monitorizar a performance de processos específicos;
- **Dashboard operacional:** Utilizado pelos utilizadores que lidam com processos operacionais. São usados para o controlo e monitorização em tempo real deste tipo de processos.

A classificação dos *dashboards* condiciona também as características dos mesmos. Estas características podem ser divididas em [15][25]:

- **Características funcionais:** Estas características englobam o tipo de gráfico, interações possíveis e a forma de análise dos gráficos;
- **Características visuais:** Estas características englobam o espaço ocupado pelo *dashboard*, linhas auxiliares e cores escolhidas.

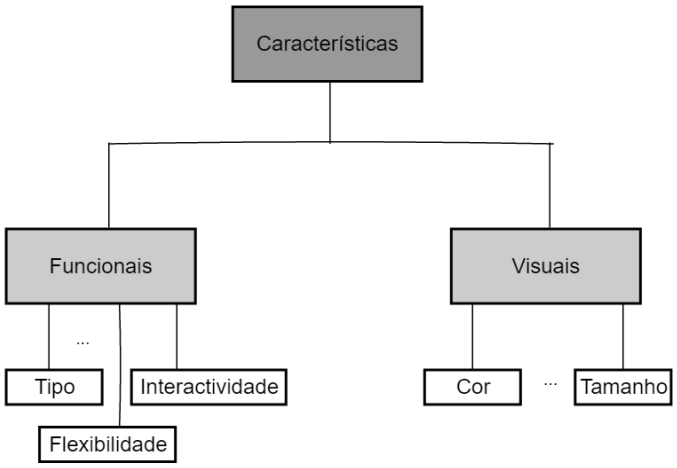

Figura 2.17: Características de um *dashboard*

2.3.2 Requisitos

Os requisitos e a finalidade de um *dashboard* têm de ser definidos antes da escolha da biblioteca a utilizar. Desta maneira garante-se que as necessidades da aplicação são satisfeitas com a

escolha de determinadas bibliotecas. Alguns requisitos podem assentar nas seguintes premissas [30][15]:

- **Tipo de gráfico:** Existem vários tipos de gráficos possíveis (barras, circulares, linhas, mapas, etc). Desta forma, é necessária uma análise às varáveis a representar e identificar os diferentes tipos de gráficos necessários à representação de todas as variáveis;

- **Linguagem a utilizar:** Com as inúmeras linguagens de programação existentes, é necessário selecionar previamente um leque de linguagens a que estamos predispostos (ou mesmo imposto) a utilizar para o desenvolvimento da aplicação;

- **Dinâmica do gráfico:** A dinâmica do gráfico traduz-se na forma com que o *dashboard* interage com utilizador. Este pode representar os dados de forma estática, dinâmica (*real-time*) ou interativa. Neste último, o utilizador pode obter mais informações interagindo com o gráfico, passando com o cursor por certas zonas ou pontos do *dashboard*.

2.3.3 Bibliotecas

Como existe um elevado número de bibliotecas direcionadas para a construção e representação de *dashboards*, a pesquisa das bibliotecas foi restringida com base na linguagem de programação utilizada pelas mesmas (*Java*, *JavaScript*, *C*, *PHP* e *Python*). Com esta pesquisa pretende-se apurar as funcionalidades de cada biblioteca e identificar o que as distingue umas das outras. A informação recolhida neste subcapítulo irá ser um dos fatores a ter em conta na escolha da linguagem para o desenvolvimento da aplicação. Segue-se então uma visão generalista de algumas bibliotecas para a elaboração de *dashboards*.

Chart.js [9] é uma biblioteca *JavaScript* simples e flexível que pode ser utilizada tanto por designers como desenvolvedores de *software*. As principais vantagens desta biblioteca traduzem-se em 4 aspetos: (1) É diversificada, pois contém mais de 8 tipos de gráficos, (2) É *open-source*, podendo adaptar cada exemplo à necessidade do utilizador, (3) Tem grande compatibilidade com todos os *browsers* a partir de *IE9* e (4) Adapta-se ao tamanho da janela, encaixando com qualquer tipo de dimensão da janela. Esta biblioteca possuí ainda um vasto leque de interações com os *dashboards*. Na Figura 2.18, pode observar-se um exemplo de um *dashboard* que utiliza um gráfico de barras e um gráfico de linhas em simultâneo, onde se pode recolher informações específicas sobre determinados pontos apenas com a passagem do cursor por cima dos mesmos.

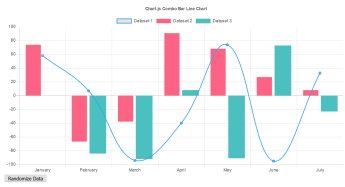

Figura 2.18: Exemplo de Dashboard Chart.js

D3.js [14] é uma biblioteca *JavaScript* de representação de dados poderosa que utiliza HTML, SVG e CSS. É uma biblioteca extremamente poderosa no que toca a representação de dados. Tem a vantagem de gerar representações como a da Figura 2.19 a partir de documentos SVG, tornando simples este processo. Porém, os exemplos disponíveis poderão ser demasiado específicos, o que faz com que não seja uma biblioteca apropriada para um simples gráfico de linhas em tempo real (por exemplo). Outro tipo de representação interessante é o da Figura 2.20, onde é possível selecionar a amostra temporal desejada para a representação de algumas variáveis, sendo este processo altamente responsivo e sem necessidade de *refresh* na página.

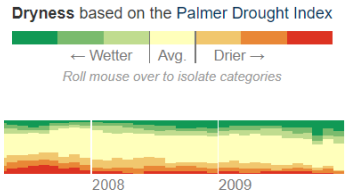

Figura 2.19: Exemplo (1) de Dashboard D3.js

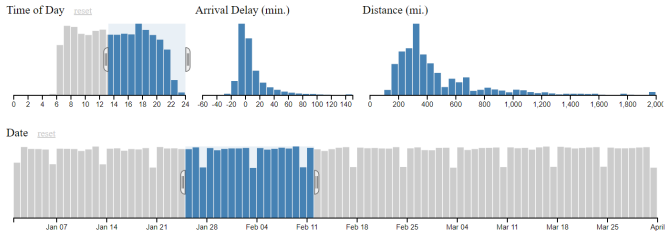

Figura 2.20: Exemplo (2) de Dashboard D3.js

Flot [11] é uma biblioteca *JavaScript* que utiliza *JQuery* e Ajax para dar interatividade e dinâmica aos seus exemplos. É uma biblioteca *open-source*, dando liberdade ao utilizador de juntar funcionalidades de diferentes exemplos num mesmo gráfico. Um exemplo de um gráfico *real-time* desta biblioteca é o da Figura 2.21. É ainda compatível com os seguintes *browsers*: *Internet Explorer 6+*, *Chrome*, *Firefox 2+*, *Safari 3+* e *Opera 9.5+*.

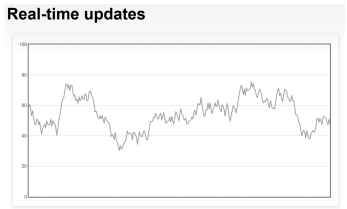

Figura 2.21: Exemplo de *dashboard Flot*

Charts 4 php [10] é uma biblioteca desenvolvida em *php* bastante versátil. É possível utilizar os *dashboards* desta biblioteca a partir de um *array* PHP, da base de dados ou mesmo de um

ficheiro *CSV*. Contém exemplos estáticos, interativos e dinâmicos prontos a utilizar, bastando apenas adaptar a fonte de dados. É compatível com *Chrome, IE 7, IE 8, Firefox, Safari e Opera*. Esta biblioteca contém ainda uma plataforma de assistência ao utilizador em caso de dificuldades na utilização da biblioteca. A Figura 2.22 representa um tipo de gráficos que podemos utilizar com esta biblioteca.

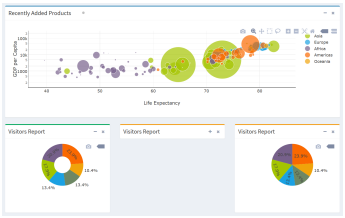

Figura 2.22: Exemplo de *dashboard* Charts 4 php

Anychart [4] é uma das bibliotecas *JavaScript* mais famosas, pois é utilizada por inúmeras empresas como *Bosch*, *BP*, *City Bank*, *Ford*, *Ericsson*, *Nokia*, *McDonadls*, etc. O facto de ser utilizada por várias empresas, deve-se à grande variedade de *dashboards* disponíveis: Barras, circular, mapas, *Gantt*, gráficos da bolsa (etc). Outra vantagem é que é extremamente rápido, sendo possível a representação de 10000 pontos por segundo (teste disponível em [4]). A imagem da figura 2.23 representa um exemplo de dashboard desta biblioteca.

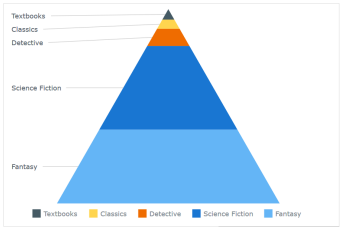

Figura 2.23: Exemplo de Dashboard AnyChart

Highcharts [20], ao par da biblioteca *AnyChart*, é também bastante famosa no mundo das empresas, sendo utilizado por 75 das 100 maiores empresas do mundo [20]. Tem uma vasta gama de exemplos de *dashboards*, incluindo *dashboards* virados para a indústria automóvel, como é o caso do *dashboard* da Figura 2.24. É altamente compatível com os *browsers*, é *open-source*, tem uma quantidade de exemplos aliciante e a linguagem de programação é *JavaScript*.

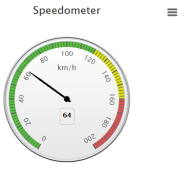

Figura 2.24: Exemplo de Dashboard HighChart

CanvasJS [8], pertence ao leque de bibliotecas famosas no mundo das empresas. Os desenvolvedores desta biblioteca afirmam [8] que *CanvasJS* é capaz de representar 100 000 pontos em apenas algumas centenas de milissegundos, o que faz que esta biblioteca tenho um ponto muito positivo a registar. Para além da *performance*, contém ainda alta compatibilidade com os *browsers* e mais de 30 exemplos de *dashboards*. A Figura 2.25 representa um exemplo de um gráfico representado com 100000 pontos. A linguagem de programação é *JavaScript*.

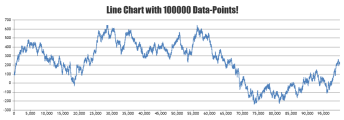

Figura 2.25: Exemplo de *dashboard* CanvasJS

PLotly [29] é a biblioteca mais versátil no que toca a linguagens de programação. Contém versões para *Python*, *JavaScript* e *R*, com imensos exemplos para quase todos os casos. Alguns pontos fortes desta biblioteca são o design moderno, a qualidade de documentação de apoio e a simplicidade da API para lidar com os *dashboards*. A figura 2.26 representa um *dashboards* baseado num para gerado por esta biblioteca.

Figura 2.26: Exemplo de *dashboard* PLotly

Google charts [19] é uma biblioteca *JavaScript* quase que com garantias de qualidade, pois os *dashboards* disponíveis, são os mesmos que a própria *Google* utiliza. A grande vantagem desta biblioteca é que tem funcionalidades de controlo e disposição dos *dashboards*, ou seja, é possível controlar e associar facilmente os diferentes tipos de *dashboards*. Tal como as outras bibliotecas possui uma grande variedade *dashboards* interativos e versáteis. A figura 2.27 representa alguns gráficos partilhados pelo mesmo espaço e gerados por esta ferramenta.

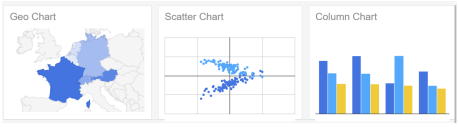

Figura 2.27: Exemplo de *dasboard* Google Chart

A pesquisa destas bibliotecas foi feita no sentido de entender que tipo de funcionalidades embebidas é que existem. Assim, na altura de desenvolvimento, pode-se ter em conta as capacidades que as bibliotecas *open-source* existentes possuem. A escolha de uma biblioteca irá depender essencialmente da linguagem de programação a utilizar e das características que se pretende. A especificação dos requisitos mencionados no subcapítulo anterior facilita bastante uma eventual escolha de uma das bibliotecas pesquisadas, filtrando todas as que não cumprem determinados requisitos, deixando apenas como principal fator de escolha o gosto pessoal do *layout* de cada biblioteca. Com base nos requisitos mencionados e na pesquisa efetuada, elaborou-se a Tabela 2.2.

	Dinâmica			Tipos de Gráfico				Linguagem			
	Real-Time	Interactivos	Estáticos	Circulares	Linhas	Mapas/imagens	Indústria automóvel	Python	JavaScript	php	C
Charts		x	x	x	x				x		
D3		x	x	x		x			x		
Flot	x	x	x	x	x				x		
Charts 4 php		x	x	x	x					x	
AnyChart		x	x	x	x	x			x		
HighChart	x	x	x	x	x	x	x		x		
CanvasJs	x	x	x	x	x	x			x		
Plotly	x	x	x	x	x			x	x		
GoogleCharts		x	x	x	x	x	x		x		

Tabela 2.2: Tabela comparativa das bibliotecas analisadas

2.4 Aplicações

2.4.1 Modelo de desenvolvimento

Com o avanço da tecnologia, a indústria automóvel tem presenciado o aparecimento de novos *softwares* destinados para os mais variados fins [26]. Estes *softwares* irão ser testados por especialistas e que irão comparar o comportamento observado com o comportamento expectável do *software*, garantido assim que não existem anomalias no mesmo [41].

De forma a uniformizar os testes e a melhorar a estrutura e organização de um projeto de desenvolvimento de *software*, o uso do modelo em *V* representado na Figura 2.28 é o mais recorrente. Consiste num *design* visual e matemático da metodologia a adotar durante o desenvolvimento de um *software* complexo [41].

Do lado esquerdo da Figura 2.29, estão representadas as fases do desenvolvedor de *software* e do lado direito as fases da entidade que irá proceder à sua verificação. Analisando o modelo, é de fácil entendimento a relação direta que existe entre as fases de desenvolvimento e as fases de teste. Portanto, o ciclo de um *software* deverá ser separado em 4 partes: Análise de Requisitos, Desenvolvimento de *software*, Verificação de funcionalidades e Manutenção.

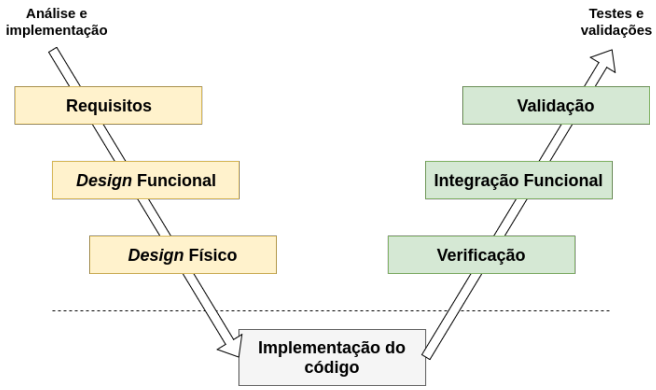

Figura 2.28: Modelo em V

2.4.2 Arquitetura

Um tipo de arquitetura recorrente no tipo de aplicações que temos vindo a dissertar e arquitetura "*Fast data*". É a definição atribuída a sistemas que analisem continuamente um elevado volume de dados proveniente de várias fontes de dados ou com diferentes graus de prioridade [28]. A arquitetura representada na Figura 2.29 sugere um modelo onde os dados são transportados através de vários módulos:

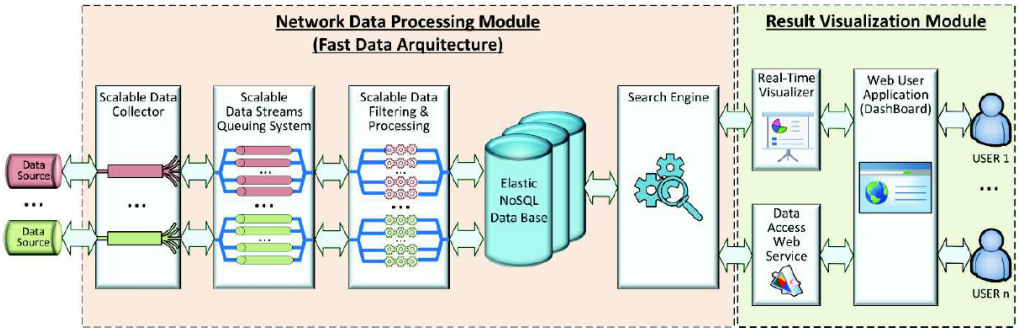

Figura 2.29: Fast Data

- **Recolha de dados:** Responsável pelo reencaminho dos dados adquiridos para uma fila de espera;
- **Fila de espera:** Garante que os dados sejam acessíveis e processados na ordem correta;
- **Processamento dos dados:** Responsável pelo tratamento de informação e por pequenos processos de correlação dos dados, e posteriormente, pelo envio dos mesmos para na base de dados;
- ***Elastic NoSql*:** Base de dados direcionada para grandes volumes de informação;
- **Sistema de pesquisa:** Responsável por fazer pedidos contínuos à base de dados para a extração dos dados;

- **Sistema de visualização dos dados:** *Dashboards* que convertem os dados extraídos e processados em informação percetível ao utilizador.

2.5 Sumário

O objetivo deste projeto é o da elaboração de um protótipo inovador, desta forma, este capítulo serviu para:

- Entender o mecanismo do protocolo CAN, assim como a eficiência dos standards consoante a aplicabilidade;
- Entender como funciona o processo de escolha de software para a construção de *dashboards*;
- Entender o propósito de uma *dashboard* e adquirir conceitos que permitam a exploração de todo o potencial que uma Dashboard pode atingir;
- Entender os passos por detrás do desenvolvimento de um *software Real-Time*;
- Analisar algumas arquiteturas de representação de dados CAN por meios de *dashboards real-time*.

Pretende-se utilizar o conhecimento adquirido na pesquisa e elaboração deste capítulo no desenvolvimento do projeto, garantindo que as tecnologias e técnicas utilizadas são as mais atuais e eficazes para o nosso problema. Concluímos ainda que o tema monitorização em tempo real é atual e pertinente.

Capítulo 3

Problema

Neste capítulo é realizada uma contextualização do problema proposto pela AddVolt e uma descrição das condições de uma possível solução, sendo posteriormente listados os requisitos que deverão ser contemplados na proposta de solução.

3.1 Contextualização

A utilização de veículos pesados refrigerados é essencial em diversas atividades que requerem um controlo de temperatura, como por exemplo no transporte de alimentos e medicamentos. Este tipo de veículos diferencia-se de todos os outros pelo facto de conter dois motores diesel, um dedicado à tração do próprio veículo e um outro, exclusivo para a alimentação do sistema de refrigeração, quando o veículo se encontra em operação. Por norma, os sistemas de refrigeração têm a possibilidade de operar igualmente em modo elétrico, contudo para tal é necessário que o veículo se encontre imobilizado e que seja possível a ligação deste motor a um ponto mono/trifásico.

De modo a tirar um maior partido desta característica dos sistemas de frio, a AddVolt avançou com o desenvolvimento de uma *"Power Bank"* para veículos pesados. Este sistema permite emular a bordo do veículo uma rede elétrica mono/trifásica, e dessa forma alimentar o sistema de refrigeração em modo elétrico, durante a operação do veículo. Consequentemente, ao deixar o motor diesel em segundo plano, as vantagens são evidentes: redução drástica do consumo de diesel, das emissões CO2 e do ruído.

A solução da AddVolt, além de oferecer o armazenamento de energia elétrica que permite o uso do modo elétrico do sistema auxiliar, permite também (opcionalmente) a incorporação de um sistema de travagem regenerativa (*Motor Controller*), aumentando assim o número de horas em modo elétrico entre carregamentos da bateria do sistema AddVolt. A Figura 3.1 representa o sistema da AddVolt antes da instalação.

O sistema AddVolt é um sistema complexo que, de forma macroscópica, pode ser dividido em:

- **Baterias**: Consiste num conjunto de módulos assemblados, onde o número de módulos pode ser ajustado conforme a necessidade de armazenamento de energia;

- ***Load Controller***: Consiste num conversor DC-DC e DC-AC;
- ***Braking Controller***: Consiste num mecanismo de travagem regenerativa onde se aproveita a energia dissipada na travagem para o carregamento das baterias;
- ***Management Logging Unit* (MLU)**: Este módulo é responsável pela comunicação permanente através de GSM;
- **Filter Box (FIBO)**: Contém elementos passivos necessários para uma filtragem adequada;
- **Junction Box (JUBO)**: Contém um sistema de gestão de utilização das baterias (BMS).

Figura 3.1: Sistema AddVolt antes de ser instalado a bordo do camião

Cada um destes módulos é constituído por dezenas de componentes de reduzidas dimensões, fazendo com que o sistema, na sua totalidade, tenha um elevado número de componentes que podem necessitar de monitorização. Este facto faz com que o processo de identificar a origem de uma eventual falha no produto da AddVolt seja demorado e de elevada dificuldade. Outro aspeto que dificulta a identificação de um problema é a necessidade de utilização de um osciloscópio e de um medidor de corrente.

3.2 Condições de uma possível solução

De maneira a contornar esta situação, o sistema foi equipado de raiz com um barramento CAN onde todos os componentes enviam informações sobre o seu estado atual em tempo-real. Assim, para extrair esta informação é necessária uma plataforma de aquisição, processamento e demonstração de dados. Com esta eventual ferramenta, os processos de atualização de *firmware* de componente e de verificação do estado e comportamento de componentes específicos seria facilitado.

No desenvolvimento de uma ferramenta de análise de dados em tempo real para o uso em oficinas, há essencialmente três aspetos a ter em conta: A aquisição de dados, a representação de dados e os requisitos gerais de uma ferramenta com um público alvo específico.

A solução atual atualmente utilizada na empresa é a operação conjunta de várias ferramentas disponibilizadas pelos fabricantes das baterias e do transceiver CAN utilizado. No entanto, por razões de confidencialidade, esta solução não irá ser descrita com afinco neste documento.

3.2.1 Aquisição de dados

Atualmente, no produto da AddVolt, a aquisição de dados é feita através de um *transceiver* CAN com o programa de base que é fornecido pelos fabricantes. Com este software é possível ler as tramas e representar as mesmas sem qualquer processamento. A mensagem vem sob a estrutura de uma *Data Frame* do protocolo CAN, sendo composta por um identificador e por um conjunto de dados separados por grupos de *bytes* representados em hexadecimal.

De modo a converter estes dados em informação útil, é necessário processar os dados de acordo com o protocolo específico de cada componente. Sendo que para cada mensagem, é possível receber vários tipos de informação proveniente de vários nós de um componente. Os vários nós do sistema fornecem mensagens com uma cadência diferente uma das outras, podendo a taxa de mensagens atingir os 500Kb/s. É necessário então um sistema robusto de aquisição que seja capaz de ler e guardar todas as mensagens para posterior tratamento. É preferível armazenar todas as mensagens ao invés de descartar mensagens para representa-las mais rapidamente. A Figura 3.2 mostra um exemplo muito genérico da relação de algumas variáveis com uma mensagem com um certo identificador, onde se observa que, dependendo do tipo de informação, os dados relativos às variáveis podem ser representados sob a forma de *bits*, *byte* ou múltiplos *bytes*.

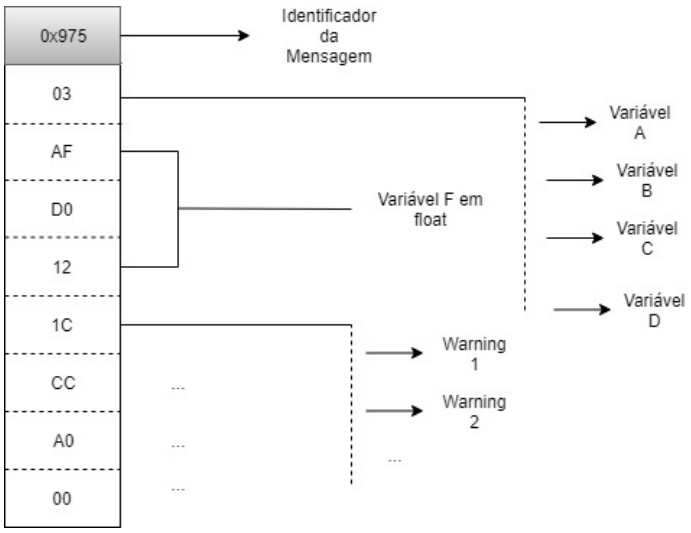

Figura 3.2: Exemplo de uma relação entre variáveis e uma trama de dados

Nesta perspetiva pode afirmar-se que os principais problemas enumerados são:

- Não perder nenhuma mensagem proveniente do sistema;
- Processar os dados das mensagens utilizando o protocolo adequado;

- Minimizar o atraso entre o instante de chegada das mensagens e o instante da representação.

3.2.2 Representação de dados

É necessária a criação de uma interface gráfica que permita aceder a toda a informação processada de forma rápida e flexível. De modo a facilitar a avaliação de uma eventual falha, os sistemas da AddVolt estão preparados para enviar mensagens para o barramento CAN contendo informações sobre o componente e o tipo de falha. No entanto, nem sempre as falhas são determinadas com base nestes mecanismos de aviso. Por vezes é necessário avaliar diferentes variáveis e analisar o seu comportamento em diferentes fases de operação. É necessário que esta ferramenta evidencie o mecanismo de avisos, que permita observar temporalmente variáveis através de representações gráficas e numéricas e que permita analisar dados gerais sobre o sistema tais como versões, modelo e *firmware* de componentes. Além da necessidade de criar uma interface intuitiva e preenchida maioritariamente por informação, numa perspetiva visual, a plataforma deverá seguir as mesmas linhas do *template* de uma outra plataforma de monitorização da empresa.

Nesta perspetiva pode afirmar-se que os principais problemas são:

- Dispor de maneira intuitiva toda a informação;
- Rapidez na representação dos dados, de maneira a não haver representações que já não traduzem o estado atual do sistema.

3.2.3 Requisitos funcionais

A criação de uma ferramenta de análise irá facilitar a análise do estado atual do sistema e, com base em comportamentos padrão irá permitir a utilizadores e técnicos de manutenção identificar comportamentos anómalos. O facto desta ferramenta ser um software faz que com que a portabilidade se torne um requisito. É requerido que esta ferramenta tenha a capacidade de operar em diferentes plataformas com o mesmo grau de desempenho.

Como esta ferramenta será maioritariamente utilizada em oficinas, é requerido que esta opere sem necessidade de uso de Internet. Consequentemente, esta ferramenta deve ter o mínimo de dependências na instalação. As falhas que ocorrem no sistema podem ser resolvidas por uma alteração do comportamento de um nó de um componente. Isto pode ser feito através de uma alteração física ou de uma atualização de *firmware*. Com isto, surge então o requisito de ter a capacidade de instalar um ficheiro *hex* através de um protocolo de escritas e leituras de mensagens CAN.

Nesta perspetiva pode afirmar-se que os principais problemas são:

- Portabilidade do *software*;
- Operar sem necessidade de ligação à *Internet*;
- Capacidade de fazer *update* de *firmware* de um componente específico;

.

3.3 Requisitos

De seguida encontram-se listados os requisitos definidos juntamente com a equipa de engenharia da AddVolt. É expectável que uma possível solução satisfaça todos estes itens, de modo a cumprir todas as especificidades do problema. Os requisitos de uma possível solução são:

- Não deve necessitar de *internet* para nenhuma funcionalidade;
- Deve ser multi-plataforma. Tendo a capacidade de ser executada em diferentes sistemas operativos;
- Deve ser livre de dependências. Entende-se por isto que não seja necessário instalação e que não seja necessário algum *software* instalado previamente;
- Deve possibilitar a atualização de *firmware* de um módulo;
- Deve processar todas as mensagens disponíveis, sem perda de informação (Até ao momento *Load Controller* e Baterias);
- Deve possibilitar a representação gráfica de variáveis periódicas (p.e. correntes alternadas) com elevada resolução;
- Dever possibilitar uma visualização intuitiva de variáveis binárias tais como avisos e falhas;
- Deve registar a monitorização efetuada através da criação de um ficheiro de *Log* em cada sessão;
- Deve conter um sistema de *troubleshooting* que compare valores atuais das variáveis com valores padrões, identificando eventuais falhas que não constem nas mensagens recebidas.
- Deve restringir dados a *login*: *Login* Administrador com acesso a todas as variáveis e utilizadores genéricos, com acesso a um conjunto restrito de dados;
- Deve ter proteção de código-fonte, tendo em vista o resguardo de protocolos utilizados e expostos na implementação;
- Deve executar todas as funcionalidades numa única ferramenta.

.

3.4 Sumário

Neste capítulo realizou-se uma análise das necessidades e das limitações encontradas, sendo possível elaborar uma lista de requisitos. Estes requisitos foram divididos em requisitos de aquisição de dados, requisitos de representação de dados e requisitos gerais típicos de uma ferramenta com um público alvo específico. No próximo capítulo iremos explorar duas possíveis soluções, dando ênfase à solução escolhida para a resolução do problema.

Capítulo 4

Solução Proposta

Uma vez apresentado o problema e levantados os requisitos técnicos e funcionais, neste capítulo apresenta-se uma visão geral da solução proposta, validação de conceito e um sumário do capítulo.

4.1 Visão Geral

Após a análise dos requisitos e de ambientação com as tecnologias e procedimentos a utilizar, entende-se que o sistema em tempo real a implementar classifica-se como *soft real-time system* e espera-se reação por parte do mesmo após receção de mensagens provenientes da linha CAN. Apuraram-se então duas possíveis soluções que visam a resolver o problema exposto. O principio das duas soluções abaixo descritas é o mesmo, no entanto há diferenças em alguns aspetos. A principal diferença é na ligação ao sistema AddVolt: Ligação por fios (Hipótese A) e ligação sem fios (Hipótese B). De um modo geral a solução que se propõe, permite a monitorização e recolha de dados em tempo real dos sistemas da AddVolt. A solução proposta conta ainda com um *layer* de processamento, que aliada a uma interface minimalista, permite ao utilizador obter informações adicionais num curto espaço de tempo.

Esta aplicação deverá consistir numa *Dashboard* de monitorização e manutenção de todos os componentes, com recurso a uma interface intuitiva. A solução irá cobrir ainda todo o processamento de dados e comunicações inerentes. Com o intuito de delimitar tarefas a solução irá ser desenvolvida sob uma arquitetura cliente-servidor. Desta maneira, entende-se por cliente uma interface gráfica e por servidor a unidade responsável pelo processamento de aquisição de dados. Assim, os focos de desenvolvimento assentam em:

- **Interface gráfica:** Nesta interface o utilizador poderá optar pelos módulos do sistema que pretende analisar e registar toda leitura efetuada num ficheiro *csv*. Pretende-se que a interface seja desenvolvida no sentido de facilitar comparações de variáveis e de detetar facilmente a presença de erros. Deve ainda disponibilizar uma interface que permita a atualização de *firmware* de um certo módulo;

- **Servidor:** Neste servidor, o algoritmo implementado deve permitir o processamento e aquisição de dados de forma eficiente. As comunicações com as bases de dados (local e externa) e com a interface de comunicação com o sistema da AddVolt serão também asseguradas pelo servidor;

- **Encapsulamento da solução:** A solução deve ser disponibilizada sob o formato de um ficheiro executável que deva correr em computadores com sistema operativo *Windows*/*Linux* sem qualquer tipo de dependência com *drivers* ou compiladores.

As duas hipóteses a serem descritas seguem os princípios referidos na lista anterior. No entanto diferem essencialmente no encapsulamento da aplicação e também no modo de acessibilidade numa perspetiva de utilizador.

4.1.1 Hipótese A

Nesta hipótese a interface gráfica e o servidor estariam localizados num ficheiro executável que poderia ser corrido no computador do utilizador. A comunicação com o Sistema AddVolt seria efetuada através da porta USB onde seria conectado um *transceiver* CAN, tendo a máquina do utilizador de estar fisicamente ligada ao sistema. Num cenário de operação de um sistema num veículo, a ligação seria feita na MLU (localizada no porta-luvas do veículo) através de um *transceiver* com recurso a um adaptador DB9. A Figura 4.1 representa um esquemático da localização onde por norma são instalados a unidade de comunicação MLU e a unidade de armazenamento de energia da AddVolt. Observa-se ainda a máquina do utilizador e as relações existentes entre as unidades a desenvolver.

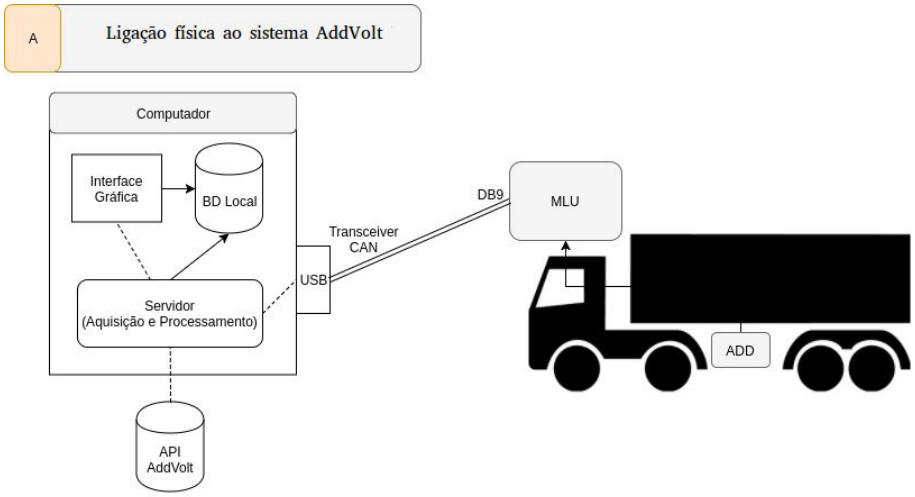

Figura 4.1: Hipótese A - Ligação com fios

4.1.2 Hipótese B

Nesta hipótese o utilizador poderia monitorizar o sistema sem a necessidade de estar fisicamente conectado ao sistema. Com a integração de um computador *singleboard* (ex. *Raspberry Pi*) com um módulo *wireless* na MLU, seria possível fazer a ligação na MLU do *transceiver* com o computador *singleboard* e, através de um *hotspot* de rede local gerado pelo módulo de *wireless*, disponibilizar o acesso ao servidor de processamento de dados remotamente. Desta maneira, o utilizador para utilizar a ferramenta, necessitaria apenas do executável contendo a interface para se ligar à rede criada pelo *singleboard*. A Figura 4.2 representa o esquemático explicativo desta solução. Esta hipótese cobre todos os passos de implementação da hipótese anterior mais os desenvolvimentos adicionais da integração do *singleboard* na MLU e as configurações de ligação sem fios de forma segura, robusta e rápida.

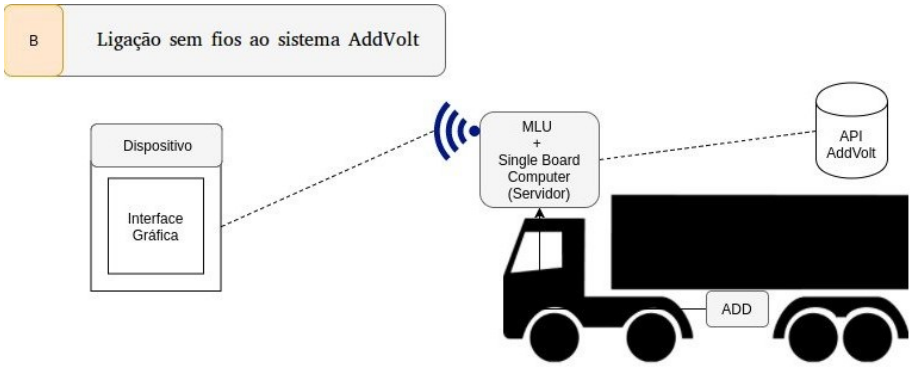

Figura 4.2: Hipótese B - Ligação sem fios

4.1.3 Solução escolhida

Embora a hipótese B apresente a vantagem na forma como se conecta com o sistema, como o princípio das duas soluções é muito semelhante, optou-se por avançar primeiramente com a hipótese A, garantindo-se assim um base sólida de suporte à futura implementação da hipótese B. Outra razão que justifica a escolha da hipótese A é que, de momento, não existe espaço para integrar um computador *singleboard* na MLU. Irá então implementar-se a hipótese A, com vista a escalar para a hipótese B caso esta seja validada com sucesso.

4.2 Validação de conceito

4.2.1 Objetivo

Neste subcapítulo pretende-se selecionar as linguagens de programação a utilizar e provar que a solução proposta é válida segundo as tecnologias especificadas. De modo a validar todo o percurso da informação desde os componentes até à sua representação em tempo real, estabeleceu-se a tarefa inicial de representar em simultâneo e graficamente três *bytes* provenientes de três

mensagens com um *id* distinto. A informação contida nos *bytes* das mensagens com este *id* é conhecida, assim como o processamento necessário dos *bytes*. Com esta tarefa é possível fazer as três validações essenciais:

- Validação da aquisição de mensagens CAN recorrendo ao *transceiver* CAN e respetiva API;
- Validação do processamento de mensagens CAN, selecionado de todas as mensagens recebidas as pretendidas, precedendo-se o respetivo processamento dos *bytes*;
- Validação da representação em tempo real de variáveis provenientes de um componente.

Pretende-se portanto provar a viabilidade da solução proposta, testando as funcionalidades básicas que permitam a escalabilidade para a solução final.

4.2.2 Escolha de linguagens

4.2.2.1 Servidor

A aquisição de dados do sistema AddVolt é feita através da leitura da rede CAN implementada com recurso a um *transceiver*, sendo que todos os nós foram programados para disponibilizar a informação para a MLU. A primeira decisão em termos de tecnologias usadas surge nesta fase. O *transceiver* utilizado tem várias API's alusivas a diferentes linguagens de programação: *C#*, *C++*, *Java*, *Delphi*, *Python* e *Visual Basic*. Portanto, a escolha da linguagem para a aquisição de dados é restringida pelas bibliotecas existentes para a API do *transceiver*. Não havendo nenhum requisito por parte da empresa que recaísse sobre API a utilizar, a linguagem escolhida foi a de *Python* devido à familiarização com a linguagem. Além das API's disponibilizadas pelo fabricante do *transceiver*, existem também dois tipos de drivers associadas à instalação do *transceiver*:

- *Drivers* para *Linux*;
- *Drivers* para *Windows*.

Identifica-se de imediato a limitação da não compatibilidade deste *transceiver* com outros sistemas operativos. Como foi descrito no sub-secção anterior, o processamento e a aquisição de dados irão ser implementados no servidor. Por essa razão, optou-se por desenvolver o processamento do dados também em *Python*.

4.2.2.2 Interface Gráfica

Dos requisitos mencionados no Capítulo 3, os que são relativos ao desenvolvimento da interface gráfica são:

- Plataforma que siga as mesmas linhas do *template* da *dashboard* da empresa;
- Apresentar-se sob a forma de uma aplicação *desktop*;

- Disposição de maneira intuitiva e modular de toda a informação disponível.

Dos requisitos mencionados, os dois primeiros tiveram mais peso na decisão da linguagem. A *dashboard* da empresa foi desenvolvida em *React* e, por haver um conhecimento sólido nesta área por parte da empresa, optou-se por desenvolver a GUI também em *React*. Além desta razão, aproveita-se a situação para aprender a desenvolver nesta *framework JavaScript* que tem vindo a popularizar-se nos últimos tempos [40]. Apesar desta *framework* ser direcionada para interfaces *web* existem componentes capazes de incubar *websites* em aplicações desktop tais como *electron* (*React*) [16], *webview* (*Python*) [34] ou *CEFpython* [13]. Algo que irá ser abordado com detalhe apenas no Capítulo 5.

4.2.3 Implementação do conceito

Para efeitos de validação, o objetivo seria implementar no servidor funcionalidades de leitura e processamento de todas as mensagens com um *id* específico. Para facilitar a leitura de dados num ambiente de desenvolvimento fisicamente distante do sistema, foi possível emular na MLU uma sequência conhecida de mensagens. Desta forma, fornecendo energia elétrica apenas à MLU, é possível simular um ambiente de operação com informação conhecida, sem a necessidade de estar conectado ao sistema durante o desenvolvimento de algumas operações. Tal como se observa na figura 4.3, para a simulação de um ambiente de operação, é apenas necessária a conexão da MLU a uma fonte de tensão e a um *transceiver* CAN. A solução desenvolvida irá comunicar diretamente com o *transceiver* CAN através da ligação USB existente entre o *transceiver* e o computador.

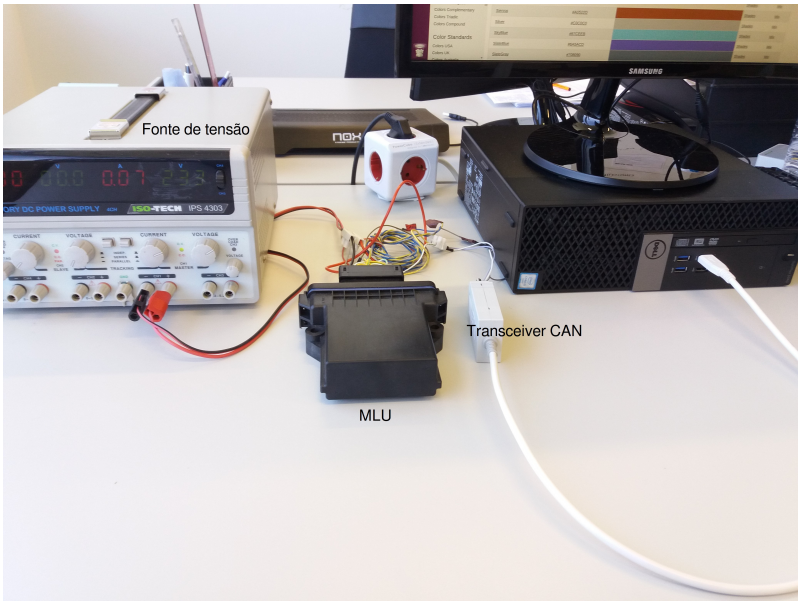

Figura 4.3: Esquema de montagem

Na parte de representação de dados espera-se que a interface gráfica seja capaz de representar a variação temporal dos 3 *bytes* através de leituras à base de dados.

4.2.3.1 Primeira versão do algoritmo

O esquema da Figura 4.4 representa a arquitetura do protótipo de validação. Representa todo o processo de extração e representação dos dados de uma mensagem (Neste caso específico o *id* 4f2). Com este protótipo é esperada a validação dos pontos mencionados no Capítulo 3 e espera-se também o registo de conclusões pertinentes que possam ajudar na otimização da ferramenta ao longo do seu desenvolvimento.

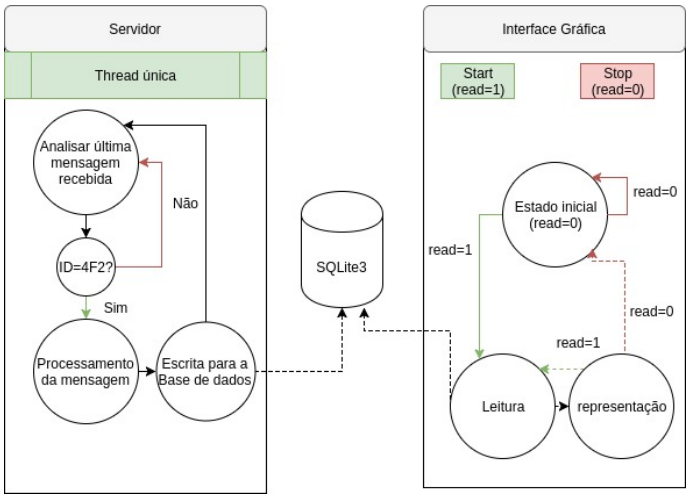

Figura 4.4: Algoritmo adotado para a validação de conceito

4.2.3.2 Servidor

Seguindo o algoritmo descrito na Figura 4.4, irá implementar-se um *script* em *Python* que seja capaz analisar e processar mensagens, e que faça escritas numa base dados. Pretende-se então que o *script*:

- Comunique com API do *tansceiver*;
- Filtre as mensagens com os identificadores pretendidos;
- Converta o primeiro *byte* dessas mensagens de hexadecimal para decimal;
- Escreva para a base de dados o valor obtido tendo em conta o *id* da mensagem de onde foi extraída a informação.

Começou-se por trabalhar num *script* que utilizasse a API disponibilizada para comunicar com o *transceiver*. Estabeleceram-se funções de inicialização de parâmetros, inicio e fim de leitura. Após este passo, seria necessário criar uma função de filtragem e de processamento que permitisse a filtragem de mensagens com os *ids* pretendidos e convertessem de hexadecimal para decimal o primeiro byte. Após a validação da aquisição e processamento das mensagens, seria necessária a criação de uma base de dados que, como se verifica na Figura 4.4, seria a interface entre o servidor

e a interface gráfica. Neste ponto do desenvolvimento, a base de dados irá ajudar a entender se está a ser feita a aquisição e processamento de todas as mensagens pretendidas. Isto é possível porque sabe-se de antemão que o primeiro byte da mensagem '4f2' corresponde a uma rampa que varia entre 0 e 255.

Para a escolha da base de dados foi efetuada uma pesquisa no sentido de entender qual seria a mais apropriada. Um tipo de base de dados que se encaixa nas condições do projeto a desenvolver é *SQLite3*. É uma base de dados que consiste num ficheiro e aconselhada para uso local. Com base na sua documentação disponibilizada pelo site oficial [38], há algumas coisas a ter em conta no uso desta base de dados:

- *SQLite3* não é aconselhado para quando há várias escritas ao mesmo tempo. Embora seja possível um número ilimitado de leituras ao mesmo tempo, as escritas bloqueiam a base de dados para outras escritas e libertam-na após finalizarem a sua ação;
- O tamanho da base dados é limitado a 140 *terabytes*;
- A documentação oficial desaconselha o uso desta base de dados se existirem acessos através da Internet, se existir várias escritas ao mesmo que tempo que levem a condições de bloqueio da base de dados e se a base de dados for utilizada num contexto de *Big Data*.

Optou-se então pela base de dados *SQLite3* pelo facto desta se enquadrar com os requisitos do projeto.

4.2.3.3 Interface gráfica

Como até ao momento nunca se tinha trabalhado com esta nova tecnologia, foi necessário aprender a sintaxe e o funcionamento da *framework JavaScript - React*. Dessa maneira investigou-se e praticou-se com exemplos minimalistas antes de se começar a implementação.

Após uma familiarização com a tecnologia iniciou-se a implementação com a tarefa de efetuar leituras à base de dados e representar numericamente numa etiqueta o o último valor escrito pelo servidor. Estendeu-se o processo até às outras duas variáveis com sucesso. Neste momento poderíamos observar o crescimento do primeiro byte de '4f2' entre 0 e 255. De forma a completar a validação, seria necessário apenas necessário representar graficamente os valores lidos. Inicialmente começou-se por ler apenas o último valor escrito de cada *id* e em seguida incrementa-lo num *array* que faria o armazenamento de todos os valores recebidos. Posteriormente esse *array* seria representado por num gráfico temporal.

Em *React* existem inúmeros componentes que disponibilizam API's para representação de dados. A escolha destes componentes irá ser abordada com mais detalhe no Capítulo 5. Por agora, escolheu-se um componente básico baseado em D3 (*Data-Driven represententation*) que fosse capaz efetuar a representação de dados. Embora tenha sido possível fazer a representação dos valores lidos, a forma de onda não era a esperada. Uma possível causa poderia estar relacionado com o facto da escrita de valores por parte do servidor para a BD ter uma cadência de escritas

menor do que que a cadência de leituras por parte da BD. Posto isto, procedeu-se ao aumento significativo da cadência de leituras, e o problema agora encontrado foi o de efetuar repetidas leituras do mesmo valor alusivas à mesma instância temporal. As Figuras 4.5 e 4.6 mostram a forma de onda que se obteve modificando a cadência de leituras.

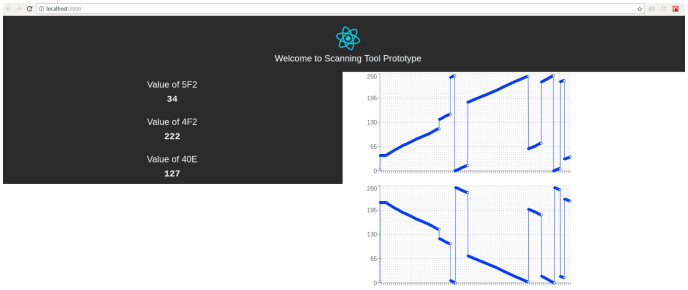

Figura 4.5: Forma de onda para cadência de leituras inicialmente estabelecida

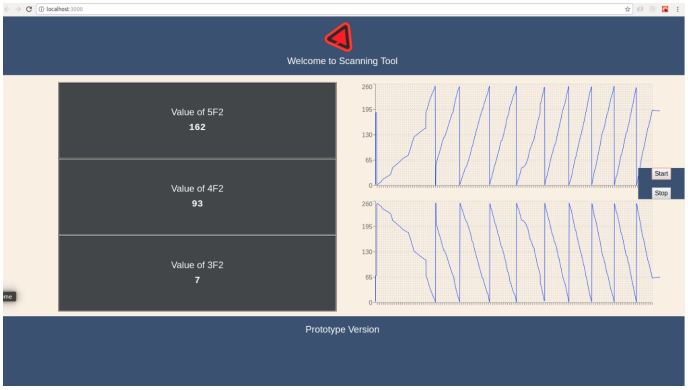

Figura 4.6: Forma de onda após aumento da cadência de leituras inicialmente estabelecida

4.2.4 Análise de resultados

Concluiu-se com sucesso a validação de conceito, já que validou-se a aquisição processamento e representação de algumas mensagens. No entanto, é necessário resolver os problemas que não permitam a representação da forma de onda. O problema de sincronismo de escritas e leituras que não permitem uma representação gráfica correta dos valores não é o único problema. Analisou-se a tabela da base de dados e verificou-se existiam diversos valores em falta, tendo permitido concluir que o servidor não se encontrava a fazer a leitura da totalidade das mensagens. Sintetizando, os principais problemas detetados foram:

- Incapacidade da aquisição de todas as mensagens pretendidas;
- Sincronismo de leituras por parte da interface com novas escritas por parte do servidor;
- Volume de dados não foi suficiente para testar robustez dos componentes utilizados.

4.3 Sumário

Neste capítulo foi exposta a solução a utilizar assim como uma outra solução equacionada. Posteriormente demonstrou-se a metodologia a adotar no decorrer do projeto e procedeu-se a uma pré-implementação que é vista como uma validação de conceito, valida a ideia transmitida na exposição da solução e justifica a escolha de linguagens a utilizar. Este sub-capítulo da validação irá servir como ponto de partida para o capítulo 5, sendo que todo o desenvolvimento irá assentar na resolução dos problemas identificados na análise de resultados deste capítulo.

Capítulo 5

Implementação da Solução

5.1 Introdução

Este capítulo tem como principal objetivo demonstrar detalhadamente todo o processo de implementação da solução proposta. Na sequência do Capítulo 4, foram identificados alguns problemas que surgiram na validação de conceito. A resolução destes problemas é determinante para o decorrer do projeto, pois será necessária uma base sólida que sustente a implementação dos requisitos definidos. Portanto, de modo a abordar todas as etapas necessárias, encontra-se dividido em: Introdução, Visão geral, Resolução de Problemas no modelo atual, Funcionalidades implementadas, Encapsulamento da aplicação e Sumário.

5.2 Visão geral

No Capítulo 4 foi efetuada uma exposição da solução proposta e respetiva validação de conceito. Desenvolveu-se um modelo de comunicação primordial que é constituído por um servidor local desenvolvido em *Python 2.7* e uma interface gráfica desenvolvida em *React* que comunicam através de uma base de dados local *SQLite3*.

Deste modelo desenvolvido foram identificados vários pontos de melhoria que ao serem resolvidos, permitiram um aumento significativo da performance assim como a implementação de novas funcionalidades. Os pontos de melhoria identificados estão essencialmente relacionados com:

- O algoritmo que gere a aquisição e processamento de dados no servidor local;
- A forma de comunicação entre o servidor local e a interface gráfica.

Como também foi referido no Capítulo 4, emularam-se um conjunto de mensagens com identificadores reais e dados conhecidos na MLU para suprimir a necessidade de estar permanentemente ligado a um sistema real.

Os próximos sub-capítulos contêm a descrição da solução implementada para resolver os problemas e também das funcionalidades implementadas.

5.3 Resolução de Problemas identificados na validação

5.3.1 Aquisição de dados

Conforme se constatou, um dos problemas detetados na validação de conceito estaria relacionado com a aquisição de dados. Era conhecido que uma das mensagens continha uma rampa que variava entre 0 e 255 em alguns segundos. No entanto, graficamente, isso não se verificava. Analisando os valores dessa variável na base de dados verificou-se também que havia falta de alguns valores compreendidos entre 0 e 255. Portanto, a origem do problema estaria à partida na aquisição de dados. De modo a verificar se o problema residia na aquisição de dados, gerou-se um gráfico através dos valores registados na tabela da base de dados como o que se observa na Figura 5.1 . E, de facto, concluiu-se que nem todos os valores estariam a ser recebidos.

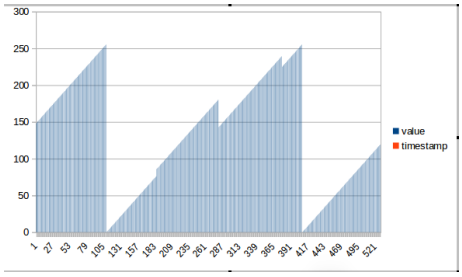

Figura 5.1: Forma de onda incorreta gerada a partir de uma das variáveis na base de dados.

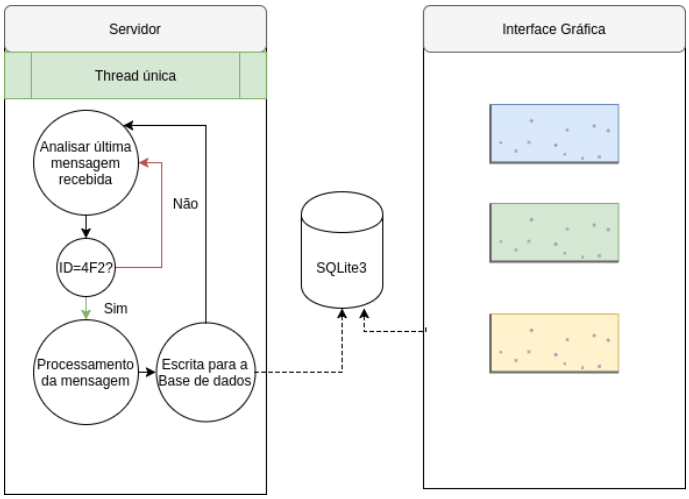

Figura 5.2: Algoritmo do servidor local

Na Figura 5.2 evidencia-se o algoritmo do servidor, onde se pode observar que após a receção de uma mensagem, o servidor procede ao processamento da mesma. Com esta prática, enquanto o servidor processa uma mensagem, várias mensagens vindas do *transceiver* são perdidas. Para resolver este problema de perda de mensagens, é necessário então que as tarefas de aquisição e processamento de mensagens sejam executadas em simultâneo.

De modo a tornar possível a aquisição e processamento de todas as mensagens, é necessário proceder a duas alterações:

- Criação de uma fila de espera FIFO (*first in first out*) que irá ser o elo de ligação entre a aquisição e o processamento. Optou-se por uma fila de espera deste tipo para garantir a mesma sequência de mensagens ao longo de todas as fases;
- Após o início de uma sessão de monitorização, criar duas *threads* que operem em paralelo. Uma *thread* fazendo a aquisição das mensagens para a fila, enquanto a outra *thread* assegura o processamento da totalidade das mensagens, enviando-as posteriormente para a base de dados.

A Figura 5.3 ilustra a nova versão do algoritmo que contempla a existência das duas *threads* referidas no servidor. Após a implementação das alterações referidas, representaram-se novamente os valores guardados na base de dados, tendo-se verificado que as alterações realizadas resolveram o problema da perda de valores durante o processamento dos dados.

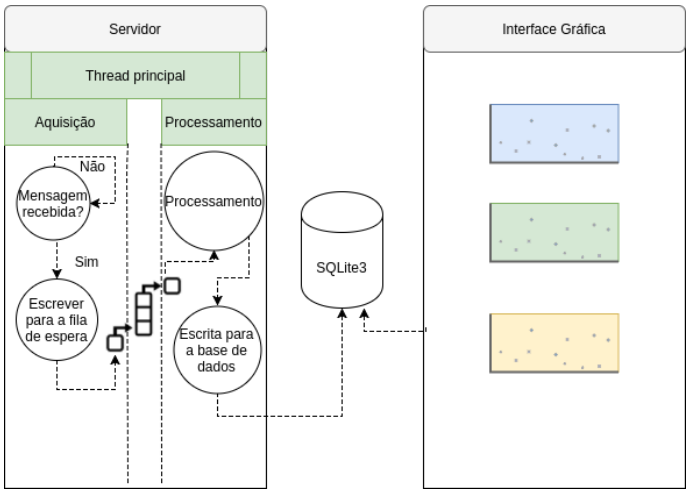

Figura 5.3: Algoritmo do servidor local após implementação de modificações inerentes à aquisição e processamento

Apesar da aquisição ter sido realizada com sucesso, existiram ainda alguns problemas ao nível da representação gráfica, nomeadamente na forma e no *delay* com que esta era realizada. A forma de onda observada na interface é a representada na Figura 5.4. Apesar de se garantir a aquisição de todos os dados, a representação não é válida, reforçando assim a ideia que uma remodelação na comunicação entre o servidor e a interface gráfica seria necessária.

5.3.2 Comunicação

No modelo atual, a comunicação entre o servidor e a interface gráfica é feita através de uma base de dados local *SQLite3*. O servidor realiza a escrita de todos os valores processados para a base de dados enquanto que a interface gráfica faz as leituras do último valor registado. Este

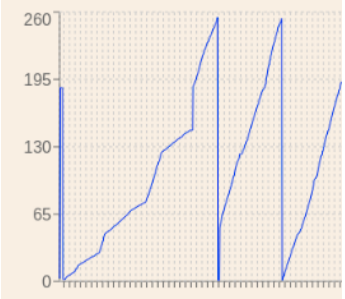

Figura 5.4: Representação errada de uma das mensagens disponibilizada na linha CAN.

método é inconsistente porque a cadência de leituras à base dados por parte da interface gráfica é fixa, podendo assim haver repetidas leituras do mesmo valor ou então perda de valores entre duas leituras consecutivas consoante a cadência estipulada. Outro problema reside no facto das mensagens de cada identificador não terem a mesma cadência. Conclui-se então que o mecanismo de comunicação entre o servidor e a interface deverá ser assíncrono, ou seja, quando o servidor tiver um valor pronto a ser enviado, envia diretamente para a interface.

No sentido de encontrar o melhor mecanismo de comunicação entre servidor e interface, a pesquisa efetuada conduziu ao conceito de *WebFrameworks* de *Python* [33]. Uma *WebFramework* é um conjunto de módulos que permitem o desenvolvimento de aplicações *Web* sem a necessidade de lidar com matérias de baixo nível como *sockets*, *threads* ou protocolos.

Existem várias *frameworks Python* com performances diferentes consoante as aplicações. Em [21] são apresentadas e comparadas diversas *benchmarks* que comparam as diferentes funcionalidades:

- Codificar um objeto JSON e retornar para um cliente;
- Obter um objeto de um servidor remoto e retornar para um cliente;
- Obter um objeto de uma base de dados e retorna para um cliente.

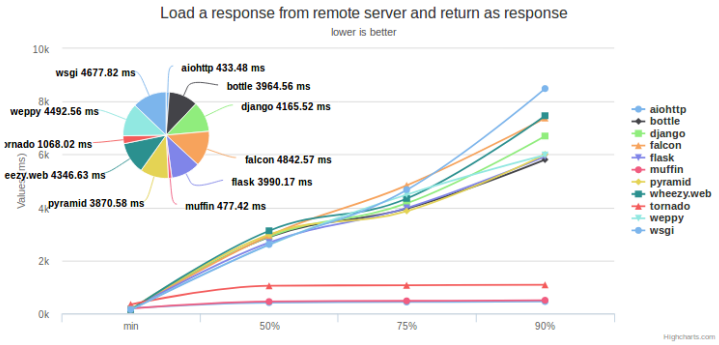

Figura 5.5: Comparação de *frameworks* a nível de codificação de um objeto e envio para cliente [21]

Para efeitos de escolha da nossa *framework* apenas nos interessa a primeira comparação que pode ser observado na Figura 5.5;

Analisando o gráfico e os tempos de codificação e envio de resposta, chega-se à conclusão que qualquer uma das *frameworks* poderá satisfazer as necessidades da aplicação a nível de performance. Das *frameworks* em questão as consideradas de mais fácil implementação foram *Flask* [18], *Bottle* [7] e *Weepy* [42]. Acabou por optar-se pela *framework flask* devido à quantidade de documentação de qualidade e bem estruturada [18]. A *framework Flask* funciona à base de rotas como mostra a figura 5.6.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Figura 5.6: Definição de uma rota em *Flask*

A estruturação do servidor à base de rotas irá permitir escalar o número de funcionalidades que aplicação poderá ter. Podendo haver no futuro rotas únicas para *Log In*, *Log Out*, Início de Leitura, Fim de leitura e análise de *logs*. A base de funcionamento desta *framework* assenta no principio de que o servidor está constantemente à escuta numa determinada porta (Ex: 5000) sempre que o servidor recebe um pedido, criando uma nova *thread* mantendo paralelamente a *thread* principal á escuta de novos pedidos. Estes pedidos funcionam através de chamadas Ajax que são chamadas síncronas. Ou seja, a interface faz um pedido ao servidor através de *http://localhost:5000* e este envia uma resposta.

Tal como mencionado anteriormente, a necessidade de criar um mecanismo de comunicação mais eficiente poderia ser resolvido com um mecanismo assíncrono em que o servidor enviava diretamente para a interface os dados processados. Contudo isto não é possível com chamadas Ajax por ser um mecanismo síncrono. Alternativamente, estudou-se o conceito de *WebScockets* e a biblioteca *SocketIO* [39] adequa-se por ser detentora de *packages* para *Python* e para *React*. Esta biblioteca permite que o servidor envie os dados diretamente para a interface gráfica através de um *socket*, e também uma comunicação de cariz mais imediato face à solução com a base de dados. Apesar de, nesta configuração, a base de dados não ser utilizada para a configuração, esta irá permanecer na arquitetura da aplicação, sendo agora usada efeitos de registo de *log*.

Tendo em conta as alterações feitas até ao momento, o algoritmo de implementação desta aplicação pode agora ser representado segundo Figura 5.7. A *thread* principal mantém-se à espera de pedidos Ajax provenientes da interface e, após a chegada de um pedido, por consequência do funcionamento de um servidor *Flask*, uma nova *thread* é criada para responder ao pedido, enquanto que a *thread* principal permanece à escuta de novos pedidos. Após o início de uma leitura, o servidor e a interface comunicam através de *sockets* por ser um mecanismo assíncrono, permitindo um aumento significativo da performance da comunicação, graças á respetiva caraterística assíncrona.

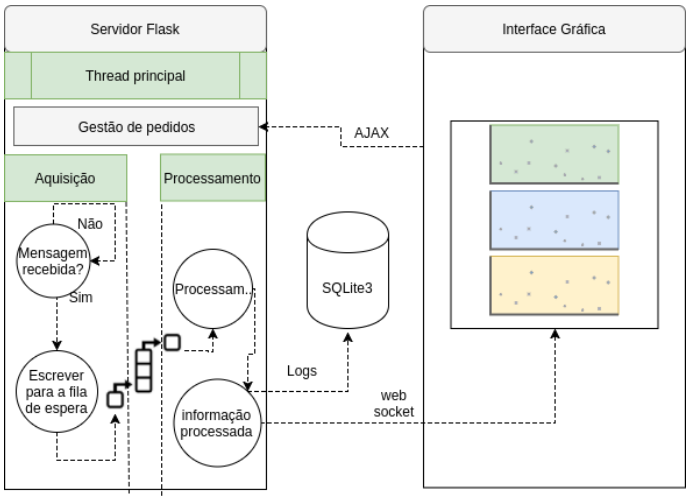

Figura 5.7: Algoritmo atualizado de acordo com alterações especificadas

5.3.3 Resultados

Uma vez identificados os problemas e implementadas as estratégias de resolução dos mesmo, procedeu-se novamente ao teste da cadeia e monitorização das variáveis conhecidas. Desta vez, a forma de onde já era a esperada, validando assim totalmente a aquisição, processamento e representação de um byte de uma mensagem. A Figura 5.8 mostra a representação da forma de onda.

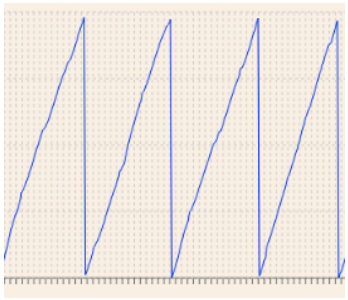

Figura 5.8: Forma de onda esperada de acordo com atualização de algoritmo

5.4 Funcionalidades

Os problemas identificados ao longo do processo de validação, e respetivas soluções implementadas permitiram que a aplicação fosse escalável, sendo facilmente possível adicionar e remover funcionalidades. Para a AddVolt observar como vai o avanço da aplicação optou-se por disponibilizar uma versão beta antes da versão final com vista a recolher *feedback* sobre o que fora implementado e sobre o que seria necessário implementar. De modo a descrever o processo, este subcapítulo divide-se em : Funcionalidades da versão beta e funcionalidades da versão final.

5.4.1 Funcionalidades da Versão beta

As funcionalidades que esta primeira versão contém são as listadas a seguir:

- Decisão sobre início e fim de leitura;
- Monitorização de *Load Controller*;
- Registo de *Logs*.

5.4.1.1 Leituras

A primeira funcionalidade implementada foi a de permitir ao utilizador decidir quando iniciava e terminava a leitura. Até ao momento, após o inicio de atividade do servidor, este realizava leituras contínuas ao barramento CAN através do *Transceiver*. Com a implementação da *framework Flask*, é possível iniciar o servidor sem iniciar a leitura. Para iniciar e terminar a leitura, existem dois botões na interface que fazem uma chamada Ajax para as respetivas rotas do servidor. Na figura 5.9 pode observar-se o código que permite fazer uma chamada Ajax para a rota de início de leitura. Esta chamada irá despoletar do lado do servidor a inicialização do leitura, estabelecendo ligação com o *transceiver* e criando as *threads* de aquisição e processamento.

```
let api_url = `http://localhost:5000/initialize/0`;

fetch(api_url, {
  method: 'get',
  mode: 'cors'
})
.then(function(response) {
  return response.text()
})
.catch(err => {
  console.log("err:", err);
})
```

Figura 5.9: Chamada Ajax para inicio de leitura

5.4.1.2 Protocolo *Load Controller*

O primeiro passo em direção à monitorização de variáveis reais do sistema foi dado nesta fase. Após todo o sistema de base estar implementado, uma vez ligados diretamente módulo *Load Controller*, o número de mensagens recebidas é superior face ao número registado quando apenas ligado à MLU. As mensagens recebidas são provenientes de todos os módulos que constituem o sistema.

Para uma primeira abordagem à monitorização das variáveis, decidiu-se apenas processar e enviar para a interface gráfica, a informação relativa ao *Load Controller*. Para a conversão dos *bytes* das mensagens para informação é necessário seguir o protocolo criado pela AddVolt. Embora o protocolo proprietário seja confidencial, podem-se descrever como se encontram estruturas as tramas de informação:

- Um único *byte* pode corresponder ao valor de um único componente;
- Um *byte* pode corresponder a vários componentes. Neste caso é necessária a conversão de *bytes* para *bits* e respetivo agrupamento e conversão de bits para decimal/*ASCII*;
- Para variáveis que requerem valores mais próximos da realidade (Digamos *floats*), o valor destas mensagens poderá vir codificado sob a norma IEEE754, sendo necessário a descodificação;
- O valor das variáveis poderá ainda ser representado sob a forma de complemento para 2, sendo necessária a conversão de *bytes*, para *bits* e depois para decimal.

Foi necessário então adaptar o parte do processamento do lado do servidor de acordo com o protocolo e ajustar a interface gráfica de maneira a haver liberdade para selecionar o que visualizar. Na figura 5.10 pode observar-se de que forma se podem selecionar as variáveis a observar.

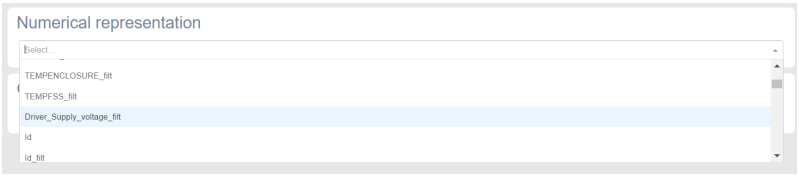

Figura 5.10: Variáveis do *Load Controller*

5.4.1.3 Representação gráfica

Embora seja possível representar todas as variáveis graficamente, por vezes a cadência/resolução dos valores das variáveis provenientes nas mensagens padrão recebidas da linha CAN, não permitem a observação correta da forma de onda. Isto acontece não só pela amostragem definida, mas também pela resolução dos valores enviados que poderá não ter casas decimais suficientes para traduzir a realidade. Por essa razão, o sistema AddVolt foi programado de maneira a que quando seja necessário observar graficamente uma variável, é necessário enviar uma mensagem direcionada a esse nó para este aumentar a taxa de amostragem até um máximo de 250 pontos e aumentar a resolução de casas decimais. Assim torna-se possível observar um período inteiro de cada variável.

É possível observar até oito variáveis com elevada resolução, sendo a amostragem de valores dessas variáveis feita no mesmo intervalo temporal. Podendo assim observar formas de onda e desfasamento entre diferentes variáveis. Graficamente pode-se observar corretamente a forma de onda correta durante a amostragem de 250 pontos (aproximadamente um período temporal), no entanto não se pode representar o que realmente acontece entre dois conjuntos de 250 pontos recebidos. Por essa razão, a janela da representação gráfica está sempre limitada a uma representação de 250 que pertencem ao mesmo intervalo de amostragem. A Figura 5.11 representa a amostragem para quatro diferentes variáveis.

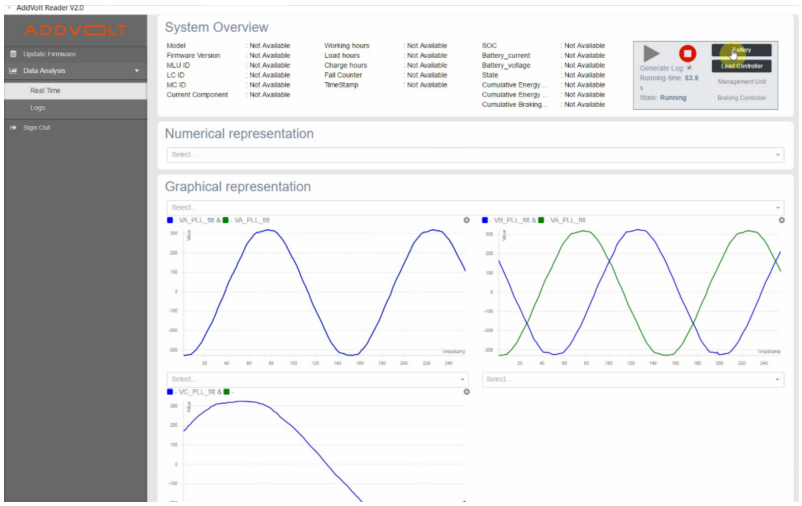

Figura 5.11: Variáveis do *Load Controller*

5.4.1.4 Registo de Logs

Na tentativa de registar os valores das variáveis durante uma leitura, utilizou-se a base de dados *SQLite3*. Após o fim de uma leitura, realizava-se escritas de todos os valores para a base de dados e, posteriormente, convertia-se a tabela para um ficheiro de extensão *csv*. Com este método, o registo dos *logs* era efetivamente possível. No entanto a performance era demasiado baixa porque, para uma monitorização de 10 segundos, o registo dos *logs* demorava mais de 1 minuto, devido à quantidade de escritas para a base de dados que eram necessárias. Face às limitações da base de dados a todos os níveis, decidiu-se avançar com a implementação do registo de *logs* apenas na versão final da aplicação. Esta funcionalidade é um dos requisitos mais importantes enumerados pela empresa, porque foi também um dos requisitos necessários à certificação (*Conformity Of Production*) do produto AddVolt.

5.4.2 Funcionalidades da Versão final

Uma vez recolhido o *feedback* dos utilizadores da versão beta da aplicação, procedeu-se à atualização da mesma, maioritariamente ao nível da estrutura de apresentação da informação e a nível de grafismo. Este *feedback* foi refletido no aspeto gráfico e estruturação da informação por componentes. Apesar de terem sido elaborados *mockups* que seriam a meta a nível de *template*, este *feedback* veio mudar o rumo do *template*, tende este sido desenvolvido numa fase final do desenvolvimento. As funcionalidades implementadas nesta versão foram:

- Decisão sobre início e fim de leitura;
- Monitorização de *Load Controller*;
- Monitorização das Baterias;
- Registo de *Logs*;

- Visualização de *Logs*;
- *Update Firmware*;
- *Login*;
- *Template* apropriado.

Para esta versão final foi necessário proceder à implementação das funcionalidades estipuladas, sendo que a nível funcional tem-se a monitorização das baterias, a visualização dos *logs* registados, a atualização do *firmware* de um componente e o *login*.

Para a monitorização das baterias, foi necessário executar um processo semelhante ao da monitorização do *Load Controller*. No entanto o protocolo utilizado é completamente diferente, porque as baterias não são produzidas na empresa. Desta maneira, caso haja uma mudança de baterias, o protocolo de processamento das mensagens das baterias tem de ser totalmente modificado. A nível de representação de dados, optou-se por um método diferente, de acordo com as necessidades de consulta por parte do utilizador. A Figura 5.12 e 5.13 mostram de que forma se pode consultar duas das secções de informação disponíveis relativamente às baterias.

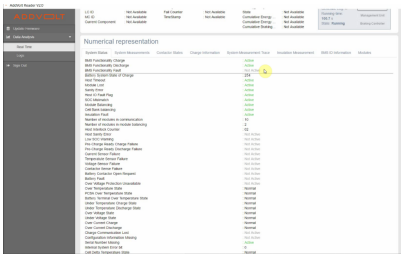

Figura 5.12: Variáveis das Baterias

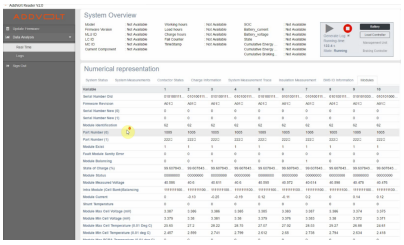

Figura 5.13: Variáveis das Baterias

5.4.2.1 Análise de *Logs*

É expectável que a aplicação tenha a capacidade de fazer leituras dos *logs* registados previamente. Para cumprir com este requisito, foi necessário utilizar uma biblioteca que fosse capaz de tratar individualmente colunas e linhas de um ficheiro de extensão *csv*. Utilizou-se então a biblioteca *Pandas* [27] que é bastante reconhecida na comunidade *Python*. Desta maneira, foi possível

extrair as tabelas no ficheiro deste tipo como uma matriz, e processar individualmente as colunas de cada variável. Como se pode ver na Figura 5.14, é possível observar o comportamento das variáveis ao longo de uma leitura a partir de de um ficheiro *csv*, sendo igualmente possível fazer *zoom* em determinada zona temporal do gráficos.

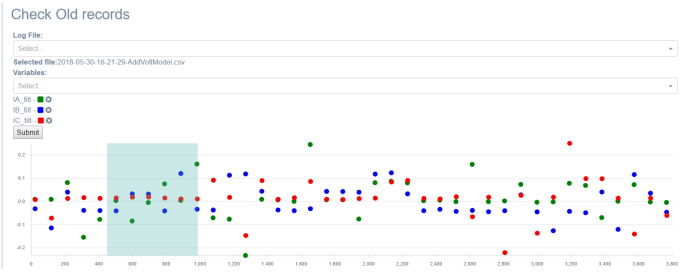

Figura 5.14: Análise de *logs*

5.4.2.2 Atualização de *firmware*

Atualização de *firmware* de um componente específico consiste num dos requisitos mais importantes. Para fazer *update* do *firmware* é necessário executar um conjunto de escritas e leituras conforme o protocolo proprietário desenvolvido pela empresa. Do lado da interface apenas é necessário fazer upload de um ficheiro de extensão *hex* usando *Drag n' Drop*. O ficheiro *hex* é gerado após compilar o código destinado ao respetivo componente. Do ponto de vista da servidor é necessário apenas interpretar os vários *bytes* do ficheiro enviar conforme uma sequência de escritas e leituras definidas pelo protocolo. Na Figura 5.15, mostra este use case do ponto de vista da interface.

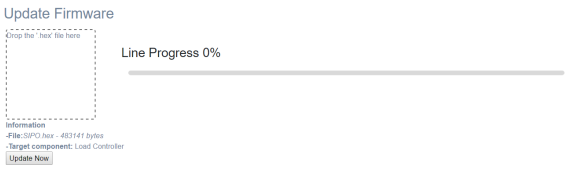

Figura 5.15: Upload de ficheiro ao estilo *Drag n' Drop*

5.4.2.3 Login

Um mecanismo de *Login* é um dos requisitos propostos pela empresa, limitando assim o acesso a determinadas secções da aplicação, para determinados utilizadores.

Outro aspeto relacionado com o *login*, é que este não deve recorrer ao uso da Internet, pois a aplicação irá ser utilizada em ambientes que cujo o acesso à Internet pode não existir. Por essa razão definiu-se um mecanismo de *Login* local.

Contudo, devido ao facto dos mecanismos de *login* locais estudados não apresentarem um nível de segurança desejado, optou-se pelo *login* com recurso à Internet, estabelecendo-se assim

ligação à API localizada nos servidores externos da AddVolt. No entanto, por requisito da equipa da AddVolt, seria preferível ter um *login* local com pouca segurança do que ter um mecanismo de *login* com recurso à internet. Isto, porque o *login* com este método limitava o local de utilização do uso da aplicação ou imposição de uso de um *hotspot* de internet móvel. Por esta razão, o cumprimento do requisito do *login* local com segurança assegurada irá ficar registado no Capítulo 7 na secção que irá abordar o Trabalho Futuro, assim como a limitação do conteúdo através de vários níveis de *login*. Na Figura 5.16 pode observar-se a página de *login*.

Figura 5.16: Página de login

5.4.2.4 Template

Após a implementação de todas as funcionalidades propostas, optou-se por assemelhar esta aplicação à *dashboard* existente da empresa, de maneira a manter a o aspeto coerente e consistente em todos os produtos produzidos na AddVolt. Após várias iterações de remodelação de aspeto visual que foi feito ao longo do desenvolvimento e também de inspiração em *mockups* realizados no inicio, a aplicação apresenta-se com o aspeto que pode ser observado nas figuras representadas nesta secção.

5.5 Encapsulamento da aplicação

Este subcapítulo é importante para garantir satisfação de três requisitos apresentados no Capítulo 3 (Problema): Aplicação Multi-plataforma, menor número de dependências possível, aplicação *desktop*. Em ambiente de desenvolvimento, a presença das seguintes tecnologias é imprescindível:

- ***Python 2.7*** - De maneira a compilar e executar *scripts python*, a instalação desta ferramenta é necessária. No entanto, de forma a facilitar este processo em *Windows*, instalou-se a ferramenta *Anaconda* [3] que disponibiliza uma consola onde é possível executar em qualquer diretoria os *scripts python*. Sem esta ferramenta, seria necessário guardar o nosso projeto na diretoria de instalação */Python27* e executá-lo a partir daí todos os *scripts*. Outra solução seria a de criar variáveis de ambiente que apontassem para os comandos *python* desejados (*python* e *pip*) e, desta maneira, seria então possível utilizar estes comandos em qualquer diretoria;

- ***Node.js*** - Uma vez que a interface gráfica foi desenvolvida numa *framework JavaScript*, foi necessária a instalação de um compilador *JavaScript*. É ainda possível neste servidor converter um projeto em desenvolvimento para um projeto de produção. Nesta conversão, é gerada uma pasta que contém uma versão otimizada do projeto e apresenta-se com uma estrutura que permite o alojamento em servidores externos;

- ***Driver do transceiver*** - De maneira a ser possível utilizar o *transceiver*, é necessária a presença dos respectivos drivers do *transceiver*. Em ambiente Windows apenas é necessária a inclusão no projeto do ficheiro de extensão *dll* que é responsável pela comunicação entre a API do *transceiver* e o próprio transceiver. Já em ambiente Linux é necessária a instalação manual da driver no sistema.

O encapsulamento da aplicação foi conduzido no sentido de facilitar o uso da ferramenta por parte do utilizador. Evitando que este tenha de proceder à instalação de várias ferramentas e à execução de vários passos adicionais para a execução da ferramenta.

Houve uma evolução no encapsulamento da aplicação ao longo do projeto. Para a primeira versão foi disponibilizada uma versão pouco automatizada deste instalador. A razão reside no facto da aplicação não estar finalizada e o objetivo seria o de recolher *feedback* importante sobre a aplicação. Por isso, só na versão final é que se teve em conta os requisitos do número de dependências e o aspeto de aplicação *desktop*.

5.5.1 Versão inicial do instalador

Nesta versão a solução desenvolvida consistiu na recriação do ambiente de desenvolvimento, sendo para isto necessária a instalação de todas as dependências necessárias durante o desenvolvimento. A pasta de instalação continha:

- Projeto do servidor - Todos os *scripts* e pastas necessárias para a execução do servidor;

- Projeto de produção da interface gráfica - Projeto em modo de produção da interface gráfica;

- Instaladores das dependências - A pasta continha executáveis de instalação das seguintes tecnologias: *Python27* e *NodeJs*. A primeira tecnologia a executar o servidor e a segunda para disponibilizar o projeto *build* numa porta à escolha do *localhost*. Para aceder ao projeto através de um browser, basta, por exemplo, considerar o seguinte endereço: 'http://localhost:3000';

- *Script* de execução - Foi criado um *script* que executa os comandos necessários à execução das seguintes tarefas: Instalação das dependências (tanto a *nivel* de tecnologias, como de *packages* requeridos pelo servidor *python*), *disponibilzação* da interface gráfica numa porta à escolha e execução do servidor local disponível na porta *localhost*:5000;

- Readme.txt - Um ficheiro de texto que contém instruções necessárias à utilização da aplicação.

5.5.2 Versão Final

Nesta versão final da ferramenta pretende-se automatizar todo o processo de instalação, suprimir as dependências do *NodeJS* e *Python* e converter a interface gráfica numa aplicação *desktop*. Nos próximos subcapítulos mostram-se as soluções encontradas para cumprir com os requisitos.

5.5.2.1 Alternativa ao *NodeJS*

O *NodeJS* não pode deixar de ser uma dependência para a fase de desenvolvimento, pois é o responsável pela conversão do projeto de desenvolvimento para o projeto de produção. No entanto, embora tenha sido também usado para servir a interface gráfica, disponibilizando-a numa porta à escolha (ex: *localhost*:3000), esta tarefa está incluída no vasto leque de funcionalidades de um servidor *Flask* através da função *sendfromdirectory()*. Devido a este facto, foi necessário redefinir a arquitetura da aplicação, tendo-se procedido à criação de uma nova rota no servidor *Flask*, disponibilizando assim a interface gráfica através desta rota. Desta maneira, centralizaram-se ainda mais as funções da aplicação no servidor *Flask*, fazendo agora a interface parte do servidor.

5.5.2.2 Aplicação *Desktop*

Na situação atual apenas é possível aceder à interface gráfica através da rota definida no servidor *Flask* utilizando um *Web Browser*. O objetivo passa por ser apenas necessária a execução do servidor *flask* e, automaticamente, uma janela *Desktop* se abrir com interface disponibilizada na rota criada para o efeito. Desta maneira, garante-se que a interface é apresentada da mesma maneira em diferentes computadores, forçando assim o uso do mesmo *browser* (*browser* embebido). As duas primeiras medidas a tomar seriam a de encapsular a interface e, posteriormente, incluir esse código no servidor *Flask*.

Como, aparentemente e funcionalmente, a interface gráfica assemelha-se a um *Web Site*, procuraram-se soluções para encapsular um *website*. Encontraram-se três *packages* de *python* que poderiam resolver esta situação: *PyQt5* [32], *WebView* [34] ou *CEFPython* [13]. *PyQt* é um *package* utilizado para produzir interfaces gráficas de raiz, enquanto que o *WebView* e *CEFPython* são *browsers* embebidos que podem ser personalizáveis de acordo com as nossas necessidades e que, utilizam *PyQt* como base do seu desenvolvimento. Optou-se então por testar as duas últimas soluções encontrada por implicarem menos linhas de código e tempo de investigação.

Enquanto que *Webiew* é baseado no browser *Internet Explorer*, *CEFPython* (*Chromium Embebbed Framework Python*) é baseado no *browser Chromium*, uma versão leve e portátil do *google Chrome*. Optou-se então pelo uso da biblioteca *CEFPython* pelo motivo que todo o processo de desenvolvimento foi efetuado no *browser Google Chrome*. A inclusão desta funcionalidade da aplicação é importante, pois garante que não há problemas de compatibilidade com o *browser* escolhido pelo utilizador para correr a interface gráfica, pois desta maneira a aplicação disponibiliza também o meio indicado de se aceder à interface gráfica.

Inicialmente, a ideia consistia em incluir o código da interface no servidor *Flask*. No entanto, de modo a manter a separação de tarefas, optou-se por manter em *scripts* separados e procedeu-se à

criação de um *script* que inicializava dois processos independentes (*CEFPython* e Servidor *Flask*) e terminava. Procedeu-se ainda à criação de uma rota no servidor *Flask* responsável pelo término do mesmo. Assim, o *script* do *CEFPython*, quando detetasse que o utilizador tinha fechado a interface gráfica, automaticamente seria enviado um comando para o servidor dando indicação ao mesmo de que pode terminar.

5.5.2.3 *PyInstaller*

Após suprimir o uso do *NodeJS* e do *browser*, decidiu-se encontrar um meio de não ser necessária a instalação do *Python* na máquina dos utilizadores para poder utilizar a ferramenta que, neste momento, se resume ao servidor. Após uma pesquisa efetuado nesse sentido, foram encontrados vários *packages* de *Python* que fazem a conversão de *scripts python* para executáveis livres de qualquer tipo de dependências externas. Alguns dos *packages* encontrados são o *cx-freeze* [12] e o *Pyinstaller* [31].

Procurou-se críticas e modo de funcionamento de cada uma destas bibliotecas, tendo-se concluído que a mais adequada seria o *PyInstaller*. Após a leitura da documentação oficial deste *package*, teoricamente, apenas seria necessário correr o comando "*pyinstaller server.py*"sendo detetados e instalados automaticamente no mesmo executável todos os *packages* utilizados. No entanto, foram necessários alguns comandos adicionais de modo a incluir também a driver do *transceiver* CAN e também de algumas bibliotecas que não foram detetas automaticamente como é o caso de bibliotecas como o *pandas* e *CEFPython*.

No sentido de proteger o código do servidor, utilizou-se uma opção de ofuscação de código disponível no *Pyinstaller*. Com recurso ao algoritmo de criptografia AES-256 e com base numa chave de 16 bits, o código do servidor foi ofuscado. A principal razão da necessidade de proteção reside no facto de que é necessário esconder os protocolos internos e externos utilizados.

5.5.3 Notas finais

Após definir as soluções que suprimiam as dependências, alcançou-se então o conteúdo final da aplicação. Posto isto, a pasta de instalação é constituída por:

- *addvolt-reader.exe* - Executável responsável por iniciar o servidor *Flask* e *CEFPython*.
- Projeto *build* da interface gráfica - optou-se por não incluir esta parte no executável de maneira a facilitar a atualização da interface gráfica sem necessidade de fazer um instalador sempre que há uma atualização.

Desta maneira os três requisitos responsáveis pelo encapsulamento da aplicação foram cumpridos com sucesso.

5.6 Sumário

Neste capítulo foi feita a descrição detalhada da implementação da solução. As fases de implementação assentaram sob a resolução dos problemas identificados no capítulo 4, o que permitiu ter um modelo de comunicação escalável para a implementação de todas as funcionalidades previstas.

Neste capítulo de implementação seguiu-se ainda a metodologia de lançar uma versão inicial com vista a recolher *feedback* valioso para implementar para a solução final. Por fim abordou-se o encapsulamento total da aplicação com vista a eliminar qualquer tipo de dependência a tornar a aplicação web numa aplicação *desktop* multi-plataforma.

Capítulo 6

Validação da Solução

Neste capítulo são efetuados testes que validam a solução do ponto do vista de resolução do problema proposto. Foi também feita uma análise dos resultados e uma verificação de satisfação dos requisitos da empresa.

6.1 Testes de validação

De modo a conseguir validar a solução implementada e a quantificar a *performance* em relação à solução existente atualmente, definiu-se uma conjunto de tarefas que permitem a comparação direta entre as duas soluções.

Segue-se então a enumeração de tarefas propostas para validação da solução:

(a) **Instalação da Solução**;

(b) **Verificação do desvio de fase das variáveis Va, Vb e Vc**;

(c) **Número de avisos ativos nas baterias**;

(d) **Temperatura máxima da bateria**;

(e) **Módulo da bateria com a tensão mais elevada**;

(f) **Módulo da bateria com a temperatura mais elevada**;

(g) **Análise gráfica de uma variável através de um *log***;

(h) **Atualização de *firmware* de um componente**;

As tarefas estipuladas foram definidas tendo em conta os requisitos que foram expostos no Capítulo 3. Com base em alguns parâmetros avaliados durante a execução das tarefas, foram medidos os seguintes índices de *performance* de maior relevância: **Número de ferramentas necessárias para executar as tarefas**, **Tamanho total da aplicação**, **Tempo total para completar a totalidade das tarefas** e **Percentagem de sucesso na satisfação dos requisitos**.

6.2 Resultados

Para uma exposição clara dos resultados, esta secção foi divida em três partes: Verificação de satisfação de requisitos, Índices de *performance*, e Observações.

6.2.1 Verificação de satisfação requisitos

Esta secção destina-se à verificação do cumprimento dos requisitos propostos. Para comparar esta verificação em ambas as soluções, todas as tarefas mencionadas na secção anterior foram executadas em ambas as soluções de forma sequencial, no mesmo computador e por um membro da empresa sem contacto prévio com a solução proposta. Para os resultados obtidos serem válidos e mais objetivos, as tarefas foram efetuadas na mesma máquina sem qualquer registo prévio de instalação de alguma solução. As características da máquina em questão são descritas na Figura 6.1.

Figura 6.1: Características do computador utilizado

Com base nos resultados obtidos na solução proposta foi elaboraram-se duas tabelas: (1) Tabela 6.1 que relaciona os requisitos, tarefas descriminadas na secção anterior e verificação de requisitos da solução proposta e (2) Tabela 6.2 que relaciona a verificação de requisitos entre a atual solução e a solução proposta. Ambas as tabelas foram elaboradas com base na seguinte enumeração de requisitos:

1. Não deve necessitar de *internet* para nenhuma funcionalidade;
2. Deve ser multi-plataforma. Tendo a capacidade de ser executada em diferentes sistemas operativos;
3. Deve ser livre de dependências. Entende-se por isto que não seja necessário instalação e que não seja necessário algum *software* instalado previamente;
4. Deve possibilitar a atualização de *firmware* de um módulo;
5. Deve processar todas as mensagens disponíveis, sem perda de informação (Até ao momento *Load Controller* e Baterias);
6. Deve possibilitar a representação gráfica de variáveis periódicas (p.e. correntes alternadas) com elevada resolução;

7. Dever possibilitar uma visualização intuitiva de variáveis binárias tais como avisos e falhas;
8. Deve registar a monitorização efetuada através da criação de um ficheiro de *Log* em cada sessão;
9. Deve conter um sistema de *troubleshooting* que compare valores atuais das variáveis com valores padrões, identificando eventuais falhas que não constem nas mensagens recebidas.
10. Deve restringir dados a *login*: *Login* Administrador com acesso a todas as variáveis e utilizadores genéricos, com acesso a um conjunto restrito de dados;
11. Deve ter proteção de código-fonte, tendo em vista o resguardo de protocolos utilizados e expostos na implementação.
12. Deve executar todas as funcionalidades numa única ferramenta.

Verificação de requisitos da solução proposta										
Testes / Requisitos	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	Contexto da totalidade de tarefas	Não validado
1									✔	
2										✖
3	✔									
4								✔		
5		✔	✔	✔	✔	✔				
6		✔								
7			✔							
8							✔			
9										✖
10										✖
11									✔	
12									✔	

Tabela 6.1: Verificação de requisitos da solução proposta

6.2.2 Índices de *performance*

Esta secção destina-se à exposição dos valores atingidos por ambas as soluções nos índices de *performance* estipulados. Cada índice foi medido durante a execução das tarefas da seguinte maneira:

- **Número de ferramentas:** Contabilizou-se o número de *softwares* necessários em cada solução para efetuar a totalidade de tarefas discriminadas;
- **Tamanho da solução:** Contabilizou-se a soma do tamanho em *megabytes* dos *softwares* utilizados em cada solução para a realização dos testes;
- **Tempo de execução das tarefas:** Contabilizou-se o tempo necessário em minutos e segundos por parte de cada solução desde o início da primeira tarefa até ao fim da última tarefa;

Comparação de verificação de requisitos		
Requisitos \ Soluções	Atual	Proposta
1	✓	✓
2	✗	✗
3	✗	✓
4	✓	✓
5	✓	✓
6	✓	✓
7	✗	✓
8	✗	✓
9	✗	✗
10	✗	✗
11	✗	✓
12	✗	✓

Tabela 6.2: Comparação de verificação de requisitos entre soluções

- **Satisfação dos requisitos:** com base na Tabela 6.2 determinou-se a percentagem de satisfação de requisitos, relacionando a totalidade de requisitos com os requisitos efetivamente cumpridos por cada solução.

Na Figura 6.2 pode observar-se os valores assumidos na solução atual do lado esquerdo e na solução proposta do lado direito.

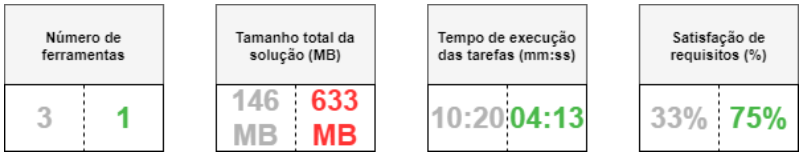

Figura 6.2: Índices de *performance* das soluções

6.2.3 Observações

Analisando os resultados obtidos nos testes de validação e verificação, chegou-se a conclusões comparativas das duas soluções a nível quantitativo.

Observando-se a Figura 6.2, facilmente se entende que a solução atual tem uma *performance* mais elevada em 3 dos índices estipulados: Número de ferramentas necessário para executar as tarefas, Tempo de execução das tarefas e percentagem de verificação de requisitos.

Do ponto de vista de satisfação da empresa, o índice de *performance* mais importante é a percentagem de verificação de requisitos. A Figura 6.2 compara as duas soluções, sendo a percentagem da solução proposta de 75% de satisfação dos resultados enquanto que a solução atual assume o valor de 33%. Conclui-se então que na solução proposta em relação à solução atual são necessárias **um terço das ferramentas**, o tamanho da aplicação é aproximadamente **quatro vezes maior**, é **mais rápido cerca de 40%** e tem um **acréscimo de 42% dos requisitos cumpridos**.

A Figura 6.1 faz-se uma relação entre as tarefas executadas e os requisito que satisfez. Os requisitos que não foram verificados foram:

- **Deve ser multi-plataforma**;
- **Deve conter um sistema de *TroubleShooting***;
- **Restringir dados a Login**.

Não foram efetuados testes para satisfação dos mesmos, pois sabe-se que os requisitos não estão cumpridos e seria necessário mais tempo de desenvolvimento para os cumprir.

Entende-se que os requisitos não foram cumpridos na totalidade, mas houve um aumento significativo da *performance* na solução proposta em relação à solução atual.

Termina-se então este subcapítulo de análise de resultados com as Figuras 6.3 e 6.4 que mostram o *template* de uma das três ferramentas utilizadas na solução atual e o *template* utilizado na solução proposta.

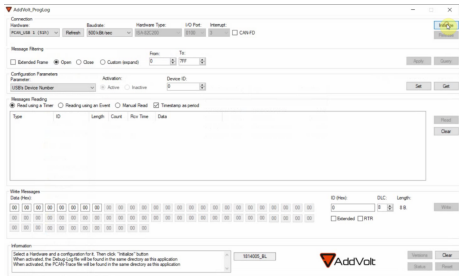

Figura 6.3: *Template* da solução atual de uma das três ferramentas

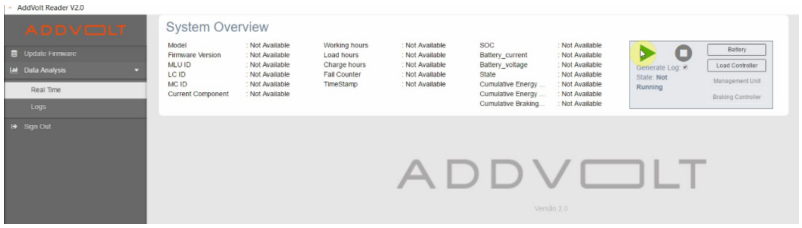

Figura 6.4: *Template* da solução proposta

6.3 Sumário

Neste capítulo foi efetuada a validação da solução por meio da execução de um conjunto de tarefas. A *performance* da solução proposta e da solução atual foi comparada através de alguns *benchmarks* definidos. Desta maneira foi possível comparar as duas soluções de maneira quantitativa, tendo em conta o sucesso de satisfação de requisitos.

Capítulo 7

Conclusões e Trabalho Futuro

Neste último capítulo efetua-se um sumário do desenvolvimento deste projeto, expõe-se as conclusões retiradas no decorrer da dissertação e também se indica pontos de melhoria do projeto que podem ser abordados em trabalho futuro. O caminho percorrido desde o levantamento de requisitos até à validação de solução implementada proporcionou um enorme desafio onde foi necessário dispor de centenas de horas de pesquisa e desenvolvimento para a conclusão do projeto.

7.1 Sumário

Este projeto iniciou-se com o levantamento teórico de conteúdos vitais para desenvolvimento do mesmo. No capítulo introdutório foram abordados os temas: Sistemas *Real-Time, Controller Area Network, Dashboards* e Aplicações existentes. Todos estes temas tiveram um papel importante, pois permitiram um primeiro contacto com diferentes fases de implementação do projeto, permitindo assim antecipar eventuais acontecimentos com medidas a nível da arquitetura e tecnologias a utilizar.

Após a pesquisa do Estado da arte, seguiu-se para uma contextualização do problema proposto pela AddVolt e para o levantamento de requisitos que definiu a fasquia de exigência do projeto. Uma vez apresentado o problema e levantados os requisitos técnicos e funcionais, apresentaram-se duas possíveis soluções e razões para a escolha da solução selecionada. Nesta fase foi apenas definida uma arquitetura genérica desprovida de tecnologias associadas. Sendo que a solução proposta consistiria num servidor com um *layer* de aquisição e processamento, que aliada a uma interface minimalista, permitiria ao utilizador obter informações sobre todos os módulos do sistema AddVolt num curto espaço de tempo. Numa segunda etapa de validação deste conceito, partiu-se para a decisão de escolha de tecnologias.

Após a exposição da solução selecionada seguiu-se para a fase de implementação onde o objetivo foi o de demonstrar detalhadamente todo o processo de implementação da solução proposta. Na sequência do Capítulo 4 tinham sido identificados alguns problemas que surgiram na validação de conceito efetuada nesse mesmo capítulo. A resolução destes problemas foi o ponto de partida para a implementação, abrindo assim caminho para inclusão de novas funcionalidades.

Após a implementação seguiu-se para a fase de validação da solução onde foram efetuados testes que validaram a solução do ponto do vista de resolução do problema proposto. Foi também feita uma análise dos resultados e uma verificação de cumprimento dos requisitos da empresa.

7.2 Conclusões da Dissertação

Durante o desenvolvimento do projeto e após a execução dos testes de verificação foram apuradas conclusões relativas à evolução da arquitetura e aos resultados obtidos na comparação de soluções e verificação de requisitos.

A arquitetura inicialmente proposta na exposição da solução no Capítulo 4 foi variando no decorrer do desenvolvimento. As conclusões que foram retiradas relativas a mecanismos e tecnologias utilizadas:

- **Mecanismo de transmissão de dados:** A utilização de uma base de dados, nomeadamente *SQLite3*, mostrou-se ineficiente como mecanismo de transmissão de dados entre o servidor e a interface gráfica. Posteriormente, foram utilizadas chamadas AJAX para o efeito, mostrando-se igualmente ineficientes. O recurso a *websockets* para transmissão de dados apresentou-se como o mecanismo mais eficiente a nível de latência de comunicação.

- ***Threads*:** A criação de *threads*, que não foram previstas inicialmente, para as etapas de aquisição de dados, processamento de dados e gestão de pedidos da interface gráfica foi um mecanismo que foi introduzido posteriormente no algoritmo utilizado no servidor, que se mostrou essencial para os resultados obtidos.

Conclui-se ainda que a solução implementada resolve 75% dos requisitos propostos na exposição do problema e, conforme os valores dos índices de *performance*, apresenta uma melhoria face à solução previamente existente.

7.3 Trabalho Futuro

Após a conclusão do projeto identificam-se vários pontos de melhoria focados em 2 dos 4 índices de *performance* e na alternativa de solução proposta exposta no Capítulo 4:

- **Tamanho da solução:** O tamanho da solução proposta aumentou face à solução atual, justificando-se pelo facto de esta solução não necessitar de dependências. No entanto, sabe-se que é possível diminuir o tamanho da solução, alterando o modo de encapsulamento dos vários componentes da arquitetura;

- **Percentagem de satisfação de requisitos:** Embora este índice tenha melhorado significativamente, os requisitos não foram cumpridos na totalidade. Portanto, considera-se que este seja um dos pontos de maior investimento em termos de desenvolvimento. Os requisitos que ficaram por cumprir são:

  - Restringir dados a login;
  - Sistema de *Troubleshooting*;
  - Solução multiplataforma.

- **Alternativa de solução proposta:** Todas as tecnologias e mecanismos utilizados na solução proposta, denominada em 4 como Hipótese A, seriam utilizados na denominada Hipótese B. Sendo que esta última apresenta vantagens a nível de mobilidade de monitorização ao sistema, permitindo uma ligação sem fios.

- **Manutenção e melhoramentos:** Com a utilização mais assídua da solução, é natural que surjam sugestões de melhoria a nível de representação de dados, *template* e algoritmos de processamento

Referências

[1] S. Abbott-McCune e L. A. Shay. Intrusion prevention system of automotive network can bus. Em *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, páginas 1–8, Oct 2016.

[2] L. Almeida, P. Pedreiras, e J. A. G. Fonseca. The ftt-can protocol: why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, Dec 2002.

[3] Anaconda. Where packages, notebooks, projects and environments are shared. Disponível em https://anaconda.org/. [Online; acedido 12-Maio-2018].

[4] AnyChart. Visualize and analyze financial or any timeline-based data. Disponível em https://www.anychart.com/. [Online; acedido 2-Janeiro-2018].

[5] Bosch. Can fd - flexible data rate. Disponível em https://www.can-cia.org/can-knowledge/can/can-fd/. [Online; acedido 20-Dezembro-2017].

[6] Robert Bosch et al. Can specification version 2.0. *Rober Bousch GmbH, Postfach*, 300240:72, 1991.

[7] Bottle. Bottle: Python web framework. Disponível em https://bottlepy.org/docs/dev/. [Online; acedido 26-Maio-2018].

[8] CanvasJS. Appropriate uses for sqlite. Disponível em https://canvasjs.com/. [Online; acedido 3-Janeiro-2018].

[9] ChartJS. Simple yet flexible javascript charting for designers and developers. Disponível em http://www.chartjs.org/. [Online; acedido 17-Janeiro-2018].

[10] ChartPHP. Easy html5 responsive dashboards in php. Disponível em http://www.chartphp.com/. [Online; acedido 6-Janeiro-2018].

[11] Flot Charts. Attractive javascript plotting for jquery. Disponível em http://www.flotcharts.org/. [Online; acedido 4-Janeiro-2018].

[12] CxFreeze. create standalone executables from python scripts. Disponível em https://pypi.org/project/cx_Freeze/. [Online; acedido 2-Maio-2018].

[13] Cztomczak. Chrome browser control, a html 5 based python gui framework. Disponível em https://github.com/cztomczak/cefpython. [Online; acedido 15-Março-2018].

[14] D3. Data-driven documents. Disponível em https://d3js.org/. [Online; acedido 5-Janeiro-2018].

[15] L. Einhardt, T. A. Tavares, e C. Cechinel. Moodle analytics dashboard: A learning analytics tool to visualize users interactions in moodle. Em *2016 XI Latin American Conference on Learning Objects and Technology (LACLO)*, páginas 1–6, Oct 2016.

[16] Electron. Electron - build cross platform desktop apps with javascript, html, and css. Disponível em https://electronjs.org/. [Online; acedido 20-Maio-2018].

[17] Jean-Marie Farines, Joni da Silva Fraga, e RS de Oliveira. Sistemas de tempo real. *Escola de Computação*, 2000:201, 2000.

[18] pocoo Flask. Flask web development, one drop at a time. Disponível em http://flask.pocoo.org/. [Online; acedido 15-Fevereiro-2018].

[19] Google. Display live data on your site. Disponível em https://developers.google.com/chart/. [Online; acedido 2-Janeiro-2018].

[20] HighCharts. Make your data come alive. Disponível em https://www.highcharts.com/. [Online; acedido 2-Janeiro-2018].

[21] Klen. Python's web framework benchmarks. Disponível em https://klen.github.io/py-frameworks-bench/. [Online; acedido 14-Fevereiro-2018].

[22] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[23] Wolfhard Lawrenz. Can system engineering. *From theory to practical applications, New York*, 1997.

[24] M. Marchetti e D. Stabili. Anomaly detection of can bus messages through analysis of id sequences. Em *2017 IEEE Intelligent Vehicles Symposium (IV)*, páginas 1577–1583, June 2017.

[25] L. Marques, V. Vasconcelos, P. Pedreiras, e L. Almeida. A flexible dashboard panel for a small electric vehicle. Em *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, páginas 1–4, June 2011.

[26] I. Nikolic, M. Ilic, N. Popovic, e M. Milosevic. Application environment for browser-based in-vehicle. Em *2017 IEEE International Conference on Consumer Electronics (ICCE)*, páginas 269–270, Jan 2017.

[27] Pandas. Python data analysis library. Disponível em PythonDataAnalysisLibrary. [Online; acedido 2-Junho-2018].

[28] M. A. L. Peña, C. A. Rua, e S. S. Lozoya. A *Fast Data* architecture: Dashboard for anomalous traffic analysis in data networks. Em *2016 Eleventh International Conference on Digital Information Management (ICDIM)*, páginas 37–42, Sept 2016.

[29] PlotLy. Modern visualization for the data era. Disponível em https://plot.ly. [Online; acedido 2-Janeiro-2018].

[30] P. Pokorný e K. Stokláska. Graphics visualization of specific dashboards in transport technologies. Em *2016 Third International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, páginas 203–206, Aug 2016.

[31] PyInstaller. A program that freezes (packages) python programs into stand-alone executables. Disponível em https://www.pyinstaller.org/. [Online; acedido 2-Maio-2018].

[32] PyQt. The most popular python bindings for the qt cross-platform c++ framework. Disponível em https://wiki.python.org/moin/PyQt. [Online; acedido 30-Maio-2018].

[33] Python. Web frameworks for python. Disponível em https://wiki.python.org/moin/WebFrameworks. [Online; acedido 30-Maio-2018].

[34] r0x0r. A lightweight cross-platform native wrapper around a webview component that allows to display html content in its own dedicated window. Disponível em https://github.com/r0x0r/pywebview. [Online; acedido 24-Maio-2018].

[35] A. A. Rahman, Y. B. Adamu, e P. Harun. Review on dashboard application from managerial perspective. Em *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, páginas 1–5, July 2017.

[36] G. s. Feng, W. Zhang, S. m. Jia, e H. s. Wu. Can bus application in automotive network control. Em *2010 International Conference on Measuring Technology and Mechatronics Automation*, volume 1, páginas 779–782, March 2010.

[37] A. A. Salunkhe, P. P. Kamble, e R. Jadhav. Design and implementation of can bus protocol for monitoring vehicle parameters. Em *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, páginas 301–304, May 2016.

[38] sqlite3. Appropriate uses for sqlite. Disponível em https://www.sqlite.org/whentouse.html. [Online; acedido 2-Janeiro-2018].

[39] Augmenting Humanity Thinkwik. The fastest and most reliable real-time engine. Disponível em https://socket.io/. [Online; acedido 20-Março-2018].

[40] Augmenting Humanity Thinkwik. why reactjs is gaining so much popularity these days. Disponível em https://medium.com/@thinkwik/why-reactjs-is-gaining-so-much-popularity-these-days-c3aa686ec0b3. [Online; acedido 24-Maio-2018].

[41] A. Tierno, M. M. Santos, B. A. Arruda, e J. N. H. da Rosa. Open issues for the automotive software testing. Em *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*, páginas 1–8, Nov 2016.

[42] Weepy. Weepy: Python framework. Disponível em http://weppy.org/docs/1.2. [Online; acedido 30-Maio-2018].