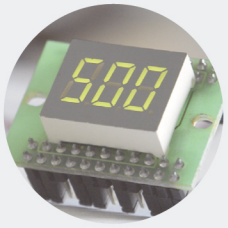
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






SISTEMA DE VALIDAÇÃO DE IMPLEMENTAÇÕES DO OCPP PARA POSTOS DE CARREGAMENTO DE VEÍCULOS ELÉTRICOS

LUÍS CARLOS OLIVEIRA LIMA
julho de 2020

POLITÉCNICO
DO PORTO

SISTEMA DE VALIDAÇÃO DE IMPLEMENTAÇÕES DO OCPP PARA POSTOS DE CARREGAMENTO DE VEÍCULOS ELÉTRICOS

Luís Carlos Oliveira Lima

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2020

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

Candidato: Luís Carlos Oliveira Lima, Nº 1150964, 1150964@isep.ipp.pt

Orientação científica: Professor Jorge Estrela da Silva, jes@isep.ipp.pt

Coorientação científica: Professor Vitor Manuel Rodrigues da Cunha, vrc@isep.ipp.pt

Empresa: Efacec Electric Mobility, S.A.

Supervisão: Carla Júlio, carla.julio@efacec.com, Edward Martinez, edward.martinez@efacec.com

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2020

Agradecimentos

Um especial agradecimento à coordenadora da equipa de software da Efacec Electric Mobility S.A., Carla Júlio e ao coordenador da equipa de Testes/DevOps, Edward Martinez por todo o apoio fornecido durante a realização deste projeto. Um agradecimento também à restante equipa de Investigação e Desenvolvimento da Efacec Electric Mobility S.A.

Aos engenheiros Jorge Estrela e Vítor Cunha, um enorme agradecimento por toda a orientação dada na elaboração deste documento.

Resumo

A Efacec Electric Mobility S.A. é uma empresa que desenvolve e produz carregadores para veículos elétricos, que são cada vez mais comuns na via pública através das redes de carregamento existentes. Geralmente estas redes são geridas com o auxílio de um sistema informático central, sendo o protocolo de comunicação OCPP utilizado para os carregadores e o sistema central trocarem informação.

Com o aumento de utilizadores de veículos elétricos, e o consequente crescimento das redes de carregamento, o cumprimento estrito do OCPP torna-se fulcral. O protocolo define formas de controlar quem pode efetuar um carregamento e também prevê o envio de todos os dados de todas as sessões de carregamento ao sistema central responsável pela rede. Para garantir que os seus carregadores cumprem o protocolo, a empresa propôs o desenvolvimento de um sistema que permitisse sistematizar os testes de validação do OCPP realizados aos carregadores.

Para responder à proposta da Efacec Electric Mobility S.A., este projeto baseia-se nos testes de validação até então realizados manualmente e desenvolve um sistema capaz de realizá-los automaticamente. Por forma a cumprir os objetivos, o novo sistema faz uso de uma biblioteca de OCPP para a linguagem Python e apresenta uma GUI em QT para criar a interface com o responsável pela realização dos testes.

O projeto desenvolvido cumpre com os requisitos definidos pela empresa pois automatiza os testes e assim a probabilidade de erros durante os mesmos diminui. Para além disso, a solução desenvolvida revelou-se suficientemente flexível para poder ser utilizada em outros cenários importantes para a empresa.

Palavras-Chave

Mobilidade elétrica, Redes de Carregamento, OCPP, CCS, CHAdeMO, AC Tipo 1 e 2

Abstract

The Efacec Electric Mobility S.A company develops and produces electric vehicle chargers, which are increasingly common on public roads through existing charging networks. Generally, these networks are managed with the support of a central computer system, with the OCPP communication protocol being used for the chargers and the central system to exchange information.

With the increase of electric vehicles, and the consequent growth of the charging networks, strict compliance with the OCPP becomes crucial. The protocol defines how to manage the access to a charger and also enables the upload of all the data from a charging session to the central system. To ensure that its chargers comply with the protocol, the company proposed the development of a system that would allow the automation of the OCPP validation tests performed on the chargers.

To answer the Efacec Electric Mobility S.A. proposal, this project builds upon the validation tests that were performed manually and develops a system capable of performing them automatically. In order to meet the objectives, the new system makes use of an OCPP library for the Python language and presents a QT GUI to create the interface with the tester.

The developed project fulfills the requirements defined by the company because it automates the validation tests and thus the probability of errors during the tests decreases. Moreover, the developed solution proved to be flexible enough to be used in additional scenarios that are important for the company.

Keywords

Electric Mobility, Charging Networks, OCPP, CCS, CHAdeMO, AC type 1 and 2

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AC – *Alternating Current*

BEV – *Battery Electric Vehicle*

CAN - *Controller Area Network*

CCS – *Combined Charging System*

CP – *Control Pilot*

DC – *Direct Current*

DNS – *Domain Name Service*

EMS – *Energy Management System*

EVCC – *Electric Vehicle Communication Controller*

FTP – *File Transfer Protocol*

GUI – *Graphical User Interface*

HTTP - *HyperText Transfer Protocol*

HTTPS – *HyperText Transfer Protocol Secure*

ICCID – *Integrated Circuit Card Identifier*

IMSI – *International Mobile Subscriber Identify*

IP – *Internet Protocol*

JSON – *JavaScript Object Notation*

MCI – Motor de Combustão Interna

OCA – *Open Charge Alliance*

OCPP - *Open Charge Point Protocol*

PLC - *Power Line Communication*

PP - *Proximity Pilot*

PWM - *Pulse Width Modulation*

RFID - *Radio Frequency Identification*

SECC - *Supply Equipment Communication Controller*

SIM - *Subscriber Identification Module*

SOAP - *Simple Object Access Protocol*

TLS - *Transport Layer Security*

URL - *Uniform Resource Locator*

V2G - *Vehicle To Grid*

VE - Veículo Elétrico

XML - *Extensible Markup Language*

1. INTRODUÇÃO

O presente documento apresenta o trabalho realizado no âmbito da unidade curricular Tese/Dissertação, do 2º ano do Mestrado em Engenharia Eletrotécnica e Computadores (MEEC), do Departamento de Engenharia Eletrotécnica (DEE), do Instituto Superior de Engenharia do Porto e foi realizado em parceria com a Efacec Electric Mobility S.A..

Na via pública é notório que a quantidade de veículos elétricos tem aumentado progressivamente, mas dúvidas e receios acerca dos carregamentos desses mesmos veículos mantêm-se. Ao longo dos anos têm sido criadas políticas por forma a incentivar a expansão das redes de carregamento rápido, mas a informação relacionada com a utilização dessas mesmas redes é geralmente apenas do conhecimento dos utilizadores de veículos elétricos. No sentido de colmatar esta situação, a presente dissertação realiza uma análise introdutória ao tema geral da mobilidade elétrica.

Para carregar um veículo elétrico num carregador público existem diversos protocolos de carregamento e tipos de carregadores. Estes carregadores diferenciam-se principalmente no número de conectores que possuem e na potência máxima de carregamento que conseguem atingir. No entanto, apesar da diversidade de características e de fornecedores todos os sistemas que constituem uma rede de carregamento comunicam em rede.

Tipicamente um carregador público está inserido numa rede de carregamento de variada abrangência, mas sempre gerida por uma entidade (e.g., MOBI.E). Para gerir estas extensas redes é utilizado um sistema informático central que, baseado num protocolo de comunicação, permite a gestão dos diversos postos de carregamento partilhando informação à distância e em tempo real. É neste ponto que surge o *Open Charge Point Protocol* (OCPP) ao ser o protocolo de comunicação que define as trocas de informação dentro de uma rede de carregamento. Por forma a garantir o funcionamento estável da rede, os fornecedores de carregadores focam-se no cumprimento estrito do protocolo.

A Efacec Electric Mobility S.A., sendo um dos fornecedores de carregadores para as redes de carregamento, preocupa-se em ter os seus equipamentos de acordo com o protocolo. Portanto, esta necessidade no cumprimento do OCPP deu o mote para o desenvolvimento deste projeto.

1.1. CONTEXTUALIZAÇÃO

As redes de carregamento são uma realidade recente na sociedade e mesmo os seus utilizadores, muitas vezes, não estão a par do diagrama geral de funcionamento das mesmas. Como ilustrado na Figura 1, uma rede de carregamento é constituída não apenas pelos seus carregadores, mas também por um sistema central. O OCPP surge então no sentido de uniformizar a forma como todos os carregadores e o sistema central comunicam, permitindo assim o controlo e monitorização à distância de todos os postos de carregamento.

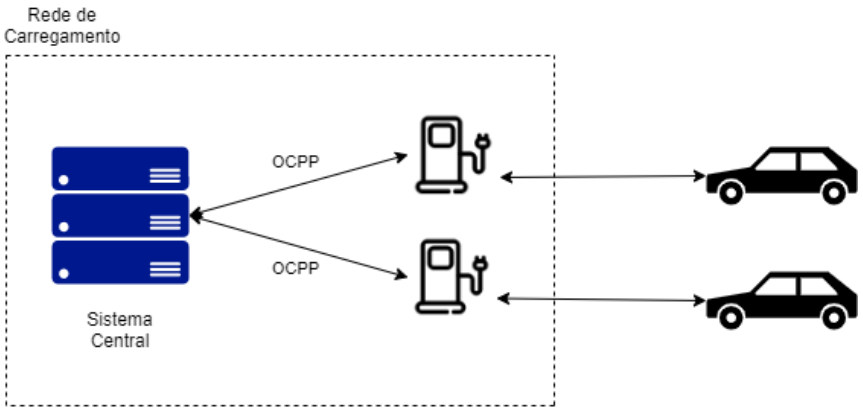


Figura 1 - Panorama geral de uma típica rede de carregamento

A Efacec Electric Mobility S.A. é uma empresa que desenvolve, produz e comercializa carregadores para veículos elétricos. O rápido crescimento do número de carregadores instalados no terreno tem sido acompanhado pelo aumento de erros reportados pelos compradores de carregadores. Neste contexto, surgiu a necessidade de agilizar os testes realizados no âmbito de validar a implementação do OCPP instalada nos carregadores. Anteriormente a este projeto, os procedimentos de testes exigiam a presença permanente de um técnico responsável. Esses testes baseavam-se no envio manual de mensagens do sistema central para o carregador e posterior análise do comportamento do carregador. Esse procedimento, além de moroso, era potenciador do desvio da atenção necessária para análise do cerne dos testes, que se prende com a análise do fluxo de mensagens da conversação OCPP.

A estratégia da empresa reorientou-se no sentido de encontrar mecanismos que ajudassem a que os testes passassem a ser realizados de uma forma mais eficiente.

Aquando do surgimento da necessidade de criar uma solução que ajudasse os técnicos responsáveis pelos testes, foi lançada publicamente uma biblioteca para a linguagem de programação Python que implementa o OCPP. Este facto acabou por influenciar a escolha da linguagem de programação a adotar no desenvolvimento da solução desejada. Ao longo do documento será evidenciado o papel que o protocolo tem nos carregadores presentes na via pública, por forma a ser percetível o interesse da empresa numa solução deste género.

1.2. DESCRIÇÃO DO PROJETO

O projeto proposto pela Efacec Electric Mobility S.A. tem como objetivo fundamental agilizar o processo de testes dos carregadores, no que ao OCPP diz respeito. Desta forma, a expectativa da empresa incide no desenvolvimento de uma aplicação que realize autonomamente sequências de testes por forma a auxiliar o responsável pela realização dos testes.

Com o principal objetivo definido, a empresa definiu os seguintes requisitos:

- Desenvolvimento de uma ferramenta capaz de realizar testes previamente concebidos;
- Os testes devem validar o fluxo de mensagens, o formato e conteúdo das mesmas de acordo com o OCPP;
- A interação com o responsável pela realização dos testes deverá ser feita com base numa *Graphical User Interface* (GUI);
- A aplicação tem que contemplar os testes atualmente realizados pela empresa em modo "manual";
- O código fonte de toda a aplicação deve ser acessível de forma a permitir futuras alterações.

Nesta dissertação será explicado todo o desenvolvimento necessário para o cumprimento dos requisitos definidos e, por conseguinte, atingir os objetivos obrigatórios do projeto.

1.3. PLANEAMENTO DO PROJETO

Previamente ao início do projeto foi tomada a decisão de utilizar a recém lançada biblioteca do protocolo para a linguagem Python. Esta situação levou a que, numa primeira fase, fosse imprescindível a realização de um estudo prévio de como deveria ser

usada a biblioteca e das formas como esta poderia contribuir para o objetivo do projeto. Apenas a partir desse ponto é que o desenvolvimento da aplicação propriamente dita poderia ser endereçado.

Para uma melhor organização e delineação do trabalho a realizar decidiu-se dividi-lo em pacotes de menor dimensão, nomeadamente:

- Estudo do protocolo e do funcionamento da biblioteca a utilizar;
- Implementação do envio e receção das mensagens OCPP;
- Desenvolvimento de uma GUI adequada aos objetivos e compatível com a linguagem de programação adotada;
- Associar os mecanismos Python das mensagens OCPP com a GUI;
- Estudo e implementação da metodologia a usar na realização dos testes automáticos.

Estes pacotes de trabalho deram origem à calendarização apresentada na Tabela 1 e que resume a evolução temporal de todo o projeto.

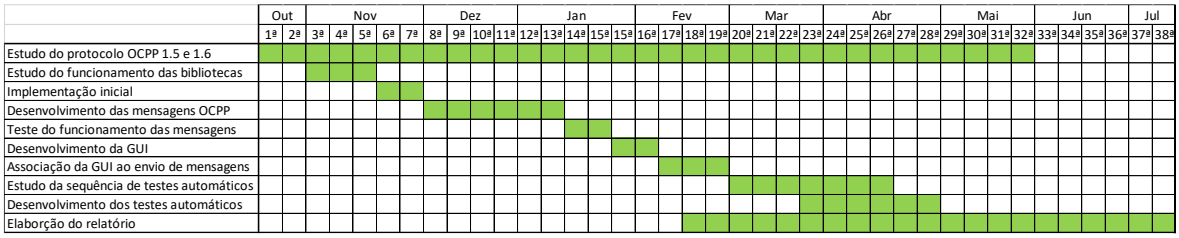


Tabela 1 - Planeamento do projeto

1.4. ORGANIZAÇÃO DA DISSERTAÇÃO

No Capítulo 1 deste documento é realizada a introdução ao projeto, bem como a sua contextualização, descrição e apresentação de objetivos. No segundo capítulo é exposto o conceito de mobilidade elétrica para que seja natural perceber onde se insere o OCPP no

universo das redes de carregamento de veículos elétricos. O protocolo e as ferramentas utilizadas no desenvolvimento do projeto são introduzidos e explicados no Capítulo 3, enquanto que o quarto e quinto capítulos dizem respeito, respetivamente, à implementação e demonstração de resultados. Para finalizar, as conclusões referentes ao projeto são realizadas no Capítulo 6.

2. MOBILIDADE ELÉTRICA

Nos últimos anos a sociedade tem vindo a assistir a uma mudança no que se refere ao tipo de mobilidade utilizada pelas pessoas no seu quotidiano. Os fabricantes de automóveis estão a desenvolver cada vez mais soluções de veículos a propulsão elétrica. Isto é consequência direta do atual contexto legislativo e normativo decorrente de um esforço de mitigação do impacto dos Motores de Combustão Interna (MCI) nas alterações climáticas.

Atualmente, a maioria dos veículos elétricos (VE) armazena a energia necessária para a sua propulsão em baterias elétricas. O presente capítulo tem como objetivo apresentar uma perspetiva geral da constituição de um veículo elétrico atual, como as baterias são recarregadas e quais os tipos de carregamento existentes. Por forma a descrever em maior detalhe o ecossistema onde se insere o OCPP, este capítulo também apresenta

alguns dos modelos de carregadores existentes no mercado, bem como as redes de carregamento em que se inserem.

2.1. CONSTITUIÇÃO GERAL DE UM BEV

Os VE que possuem energia armazenada em baterias de grande capacidade são denominados de *Battery Electric Vehicle* (*BEV*) ou de *Plug-In Hybrid Electric Vehicle* (PHEV). Sendo que, este segundo tipo é um sistema híbrido que intercala a sua operação elétrica com um MCI. De forma geral um BEV possui na sua constituição variadas semelhanças com os veículos de MCI. Todos estes veículos possuem sistemas de segurança (ativa e passiva), sistemas de transmissão de potência e caixa de velocidades (automática nos BEV), e a tradicional bateria de 12 V para alimentar a eletrónica de baixa potência.

As parcelas constituintes exclusivas dos BEV são as baterias de alta capacidade, o motor elétrico e a respetiva *drive* de potência, o *on-board charger* e o conector para carregamento das baterias. Esta constituição geral encontra-se ilustrada na Figura 2 [1].

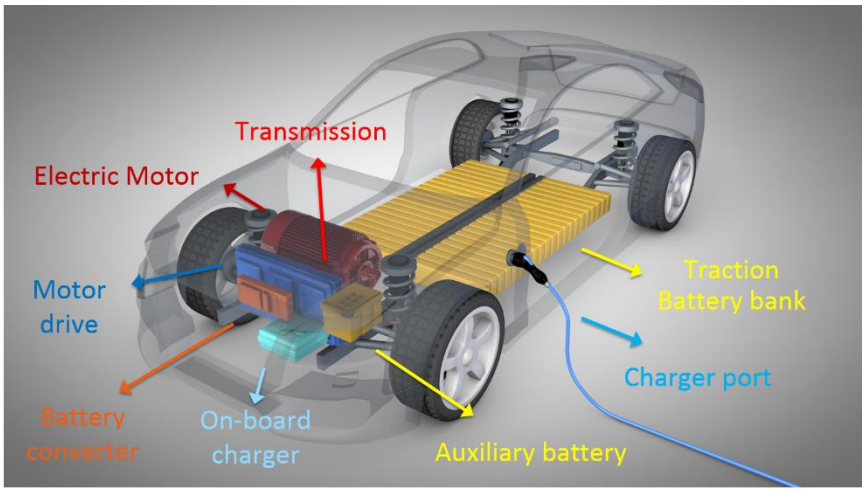


Figura 2 - Constituição geral de um VE [1]

É nas baterias que é armazenada toda a energia necessária para realizar a propulsão do veículo elétrico. Atualmente estas são formadas à base de iões de lítio, existindo modelos ligeiros de passageiros comercializados com capacidade de armazenamento até aos

100kWh [2]. Devido ao facto de serem a parte fulcral de um BEV várias organizações criaram protocolos para o controlo do carregamento das baterias, e que serão abordados em secções posteriores.

Relativamente aos motores elétricos usados nos VE, uma das soluções implementadas é a utilização de motores síncronos de ímanes permanentes. Este tipo de motor consegue ser compacto, ter um alto rendimento e permite um controlo de velocidade de rotação facilitado. Outra solução também implementada é a propulsão com base em motores assíncronos, que não são tão compactos como os motores síncronos, mas possuem um menor custo de produção. A Tesla, a Toyota e o grupo General Motors são exemplos de fabricantes que têm adotado esta tecnologia [3] [4].

Por forma a encontrar a autonomia do veículo, em distância, os fabricantes relacionam a capacidade de armazenamento de energia das baterias em kWh, com a eficiência do motor elétrico em kWh/100 km. Assim a autonomia de um VE, tipicamente o parâmetro mais determinante do ponto de vista do utilizador, não depende apenas da capacidade de armazenamento de energia nas baterias, mas também da eficiência do motor e sistema de transmissão de potência às rodas.

2.2. MODOS DE CARREGAMENTO E ON-BOARD CHARGER

Uma das principais mudanças que a mobilidade elétrica vai introduzir na sociedade é a forma de como os utilizadores vão "abastecer o depósito". Atualmente existem quatro modos de carregamento, todos eles com características e implicações diferentes, quer para o utilizador quer para o veículo. Estes modos de carregamento distinguem-se essencialmente pela potência de carregamento e pelo tipo de corrente de saída do carregador: *alternating current* (AC) ou *direct current* (DC) [5].

No modo 1 de carregamento é utilizada a tomada doméstica para efetuar o carregamento [6]. O modo 2 já implica a utilização de um cabo com proteção e permite que a tomada de alimentação seja monofásica ou trifásica. Por conseguinte, a potência de carregamento é superior ao modo 1 [6].

O terceiro modo de carregamento vai para além da simples ligação do veículo a uma tomada, como é o caso do modo 1 e 2. Neste modo, tem que existir uma tomada dedicada para o carregamento que, por sua vez, está ligada à rede elétrica e possui todos os mecanismos de proteção necessários. Este modo de carregamento é denominado de carregamento rápido em AC [6].

O modo de carregamento que mais se distingue dos restantes é o modo 4. Este modo está apenas presente em carregadores que têm aplicados protocolos de carregamento próprios com conectores (ou *plugs*) dedicados. Neste modo a saída do carregador é sempre em DC e consegue atingir valores de potência superiores a 50 kW, sendo por conseguinte também denominado de modo de carregamento rápido [7].

A Figura 3 representa graficamente os modos de carregamento 2, 3 e 4. De notar que devido às similaridades entre os modos 1 e 2, apenas está representado o modo 2 na figura.

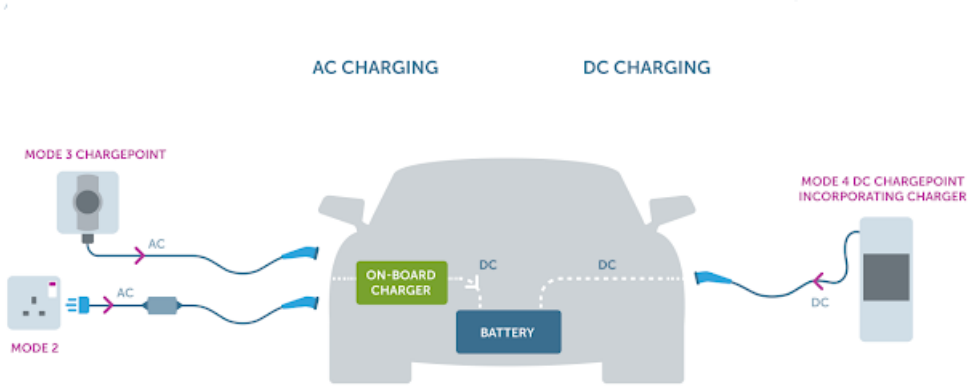


Figura 3 - Modos de carregamento de um VE [7]

Como referido anteriormente os modos de carregamento podem ter saída em AC ou DC, embora para efetivamente realizar o carregamento das baterias a energia fornecida a estas tem que ser em DC. Enquanto que no modo 4 a energia DC é transmitida diretamente às baterias, nos restantes modos (i.e., AC) é necessário o *on-board charger* [8].

O *on-board charger* tem a função de converter a energia proveniente do carregador de AC para DC. Ou seja, este dispositivo é um conversor AC/DC e é ele a razão pela qual o

carregamento em AC é limitado a uma potência mais baixa. Para eliminar esta restrição seria necessário um sistema de maiores dimensões e de maior custo, o que se tornaria inviável.

2.3. TIPOS DE CONECTORES PARA CARREGAMENTO AC

Destinado ao carregamento em AC existem vários tipos de conectores utilizados pelos fabricantes nos seus veículos elétricos. Estes conectores podem ser próprios de certas marcas ou então são adotados consoante o país de destino do VE. Atualmente os conectores existentes são denominados de AC tipo 1 e tipo 2, a *plug* GB/T e o conector proprietário da Tesla. Na Figura 4 é possível visualizar os vários conectores de carregamento AC.

Figura 4 - Da esquerda para a direita: conector AC tipo1 [8], AC tipo 2 [9], conector proprietário da Tesla [10] e GB/T [12]

O conector AC tipo 1 foi o primeiro a surgir e chegou a ser utilizado em todo o mundo. Este conector tem uma alimentação monofásica, pelo que permite uma potência de carregamento até 3.7 kW ou 7 kW. Atualmente este conector é utilizado na América do Norte e Japão [9].

Na Europa é utilizado o conector tipo 2 para carregamentos AC. Este conector permite limites de potência superiores ao do tipo 1, devido a possuir uma alimentação trifásica. O conector tipo 2 permite carregamentos até 3.7 kW, 7 kW, 11 kW, 22 kW ou 43 kW [9] [10]. Recordar que este limite máximo de carregamento é limitado pelo *on-board charger* existente no VE.

Os conectores AC dos tipos 1 e 2 utilizam o mesmo protocolo de comunicação entre o carregador e o veículo. Este protocolo é baseado num sinal de *Pulse Width Modulation* (PWM) fornecido pelo VE através do pino *Control Pilot* (CP). O sinal PWM tem sempre a frequência de 1 kHz, mas o veículo altera o *duty-cycle* e a amplitude do sinal, de modo a que o carregador possa saber em que estado do processo de carga o veiculo se encontra e qual o seu limite de corrente [11]. Na Figura 5 são apresentados os pinos pertencentes a este conector. Para além do CP, está presente também o pino *proximity pilot* (PP) que tem a função de detetar a presença do conector, os pinos das 3 fases, o neutro e a terra.

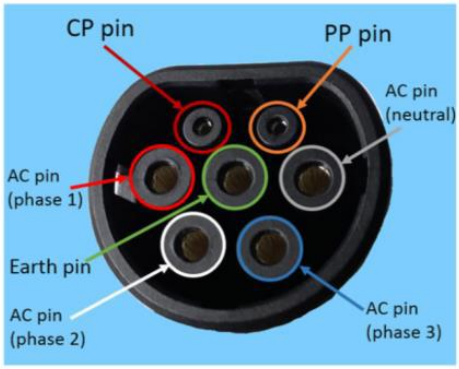


Figura 5 - Pinos do conector AC tipo 2 [11]

A *plug* GB/T é originária da China e assemelha-se ao sistema utilizado pelos conectores tipo 1 e 2, mas tem a particularidade de os pinos do conector serem fêmea e a tomada do carro serem do tipo macho. Esta configuração é exatamente o oposto da utilizada nos conectores tipo e tipo 2 [12].

O conector proprietário da Tesla é capaz de ser utilizado para carregamentos AC e DC. Em AC, a potência de carregamento é da mesma forma limitada pelo dispositivo *on-board charger* instalado no veículo [8].

2.4. TIPO DE CONECTORES PARA CARREGAMENTO DC

Relativamente ao carregamento DC de veículos elétricos existem quatro tipos de conectores atualmente. Estes conectores estabelecem protocolos distintos de

comunicação entre o carregador e o veículo, e que para além de diferenças ao nível físico, também proporcionam diferentes níveis de potência de carregamento.

Os conectores usados no carregamento DC são o CHAdeMO, o *Combined Charging System* (CCS), o GB/T e o conector próprio da Tesla.

O conector CHAdeMO começou a ser desenvolvido no Japão em 2005 [13] com o intuito de criar um protocolo de carregamento rápido DC para veículos elétricos. A CHAdeMO *Association* teve sempre como objetivo que o seu protocolo fosse seguro, que estivesse preparado para o futuro, que a sua aplicação fosse simples e que o protocolo fosse uniforme em todo o globo [14]. Por forma a que o protocolo fosse simples de implementar, a CHAdeMO *Association* utilizou a comunicação assente numa *Controller Area Network* (CAN) pois é um tipo de comunicação comum na indústria automóvel [15]. Atualmente na versão 2.0, o protocolo permite potências de carregamento até 400 kW [16]. Os protocolos CHAdeMO e GB/T, que sempre partilharam as suas especificações, uniram-se em 2018 com o objetivo de desenvolver o protocolo por forma a este chegar a uma potência de carregamento na ordem dos 900 kW [17]. A Figura 6 apresenta o conector CHAdeMO e o respetivo *inlet* [18].

Figura 6 - Conector e *inlet* CHAdeMO [18]

O protocolo de carregamento rápido DC CCS foi desenvolvido pela CharIN *Organisation* com o objetivo de cobrir todos os cenários possíveis de carregamento apenas com um produto. Isto é, com o mesmo *inlet* ter a possibilidade de efetuar carregamentos AC monofásico ou trifásico e carregamento rápido DC com potência de carregamento na ordem dos 350 kW [19]. O conector CCS combina fisicamente os conectores AC tipo 1 ou 2 com uma parte responsável por transmitir a potência em DC. Como o protocolo é utilizado em todo o mundo e os conectores AC são diferentes na América do Norte e na

Europa, por exemplo, foram criados o CCS tipo 1 e CCS tipo 2, representados na Figura 7 [12].

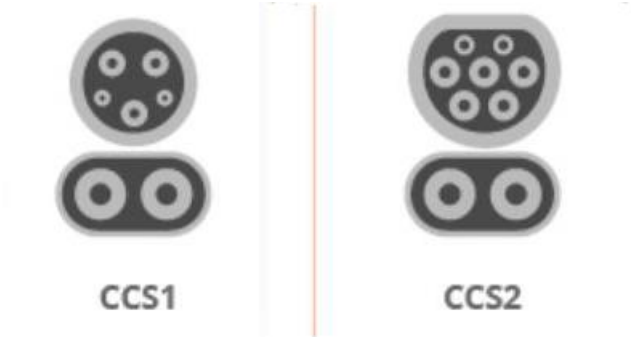


Figura 7 - Conector CCS tipo 1 e tipo 2 [12]

Assim, um veículo elétrico que adote o protocolo CCS pode efetuar carregamentos em AC utilizando apenas a parte superior do *inlet* ou pode realizar carregamentos DC onde o *inlet* é todo utilizado. No carregamento DC a parte superior do conector usa os pinos de PP, CP e a terra para *Power Line Communication (PLC)* presente no protocolo CCS [15] [20]. Portanto, os VE que possuam CCS apenas precisam de um *inlet*, enquanto que os VE com CHAdeMO necessitam de dois (um para carregamento DC e outro para AC).

O conector proprietário da Tesla é igual ao utilizado pela marca para os carregamentos AC, presente na Figura 3. A Tesla através do seu protocolo permite uma potência de carregamento na ordem dos 120 kW [21], nos seus carregadores designados de *Superchargers*. Por forma a permitir que os seus utilizadores possam efetuar carregamentos fora dos carregadores proprietários da Tesla, a marca criou adaptadores CHAdeMO e CCS [21]. Esta abordagem leva a que a Tesla seja a única marca onde é possível com o mesmo veículo ter três tipos de protocolos de carregamento DC.

Embora o maior objetivo dos protocolos de carregamento DC seja realizá-lo de forma mais rápida, muitos países ainda não adotaram um protocolo oficial. Nos Estados Unidos da América e Europa o protocolo oficial é o CCS, enquanto que no Japão é o CHAdeMO. A China adotou o GB/T, mas o real foco vai para o facto da maior parte dos países ainda não ter oficialmente escolhido um protocolo, como é possível ver no mapa da Figura 8 [22]. Esta escolha não irá definir que a totalidade dos carregadores DC de uma determinada

região terão apenas um protocolo de carregamento implementado, mas sim, o protocolo que as marcas devem adotar consoante a região onde o VE será comercializado.

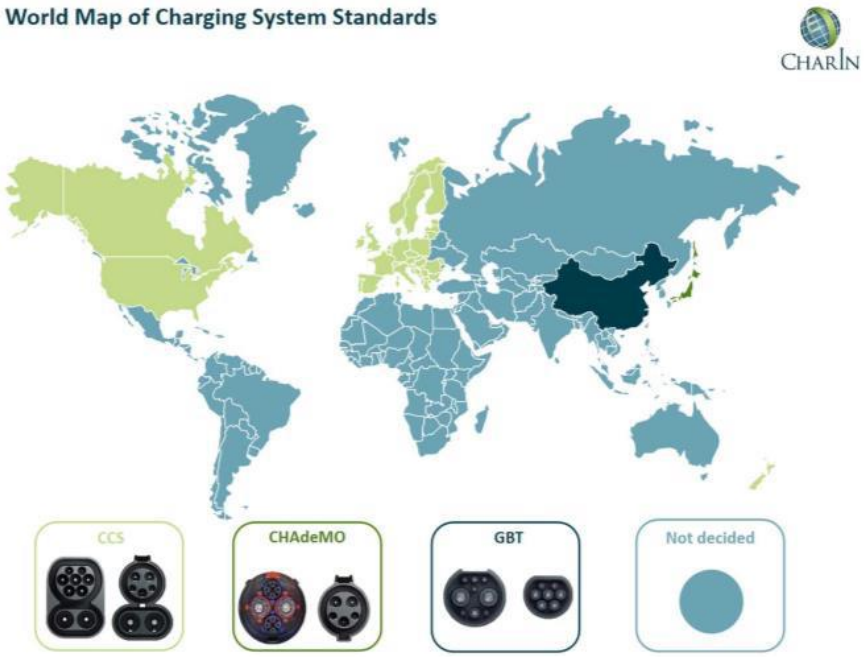


Figura 8 - Mapa da distribuição dos protocolos de carregamento DC [22]

2.5. TIPOS DE CARREGADORES PÚBLICOS DE VE

No mercado atual existem vários fabricantes de carregadores para veículos elétricos. Estes fabricantes podem ser as próprias marcas de automóveis que também oferecem soluções para carregamento doméstico ou então são empresas dedicadas exclusivamente à produção e desenvolvimento deste tipo de sistema.

O rápido crescimento da mobilidade elétrica levou à necessidade de criar e instalar carregadores para VE na via pública, por forma a que os utilizadores possam carregar os seus veículos fora de casa. Os fabricantes serviram-se dos tipos de conectores existentes e desenvolveram soluções de carregamento lento, rápido e ultrarrápido.

Os carregadores públicos considerados lentos possuem apenas conectores com saída AC tipo 1 ou tipo 2, como são os apresentados na Figura 9. Exemplos de fabricantes deste

tipo de carregadores são a Efacec [23] e a Magnum Cap [24]. Estes carregadores permitem atingir uma potência de carregamento de 43 kW, para determinados veículos (e.g., Renault Zoe) [10].

Figura 9 - Carregadores públicos lentos [23] [24]

Por forma a diminuir o tempo de carregamento das baterias dos VE, os fabricantes de carregadores desenvolveram sistemas com os protocolos de carregamento DC. Os carregadores rápidos são aqueles que conseguem atingir uma potência de carregamento na ordem dos 50 kW e possuem três saídas, duas com conectores DC (CHAdeMO e CCS) e uma com AC tipo 1 ou 2. Fabricantes deste tipo de carregadores são a ABB [25], a Efacec [26] e a Tritium [27], como é possível de visualizar na Figura 10.

Figura 10 - Carregadores rápidos [25] [26] [27]

Devido à evolução constante dos protocolos de carregamento rápido DC e à capacidade dos VE atingirem potências de carregamento superiores a 50 kW, foram desenvolvidos carregadores que acompanham esse progresso. Este tipo de carregadores já é denominado de ultrarrápido e implementa apenas protocolos de carregamento DC, com valores de potência de carregamento a atingir os 350 kW. Exemplos de carregadores deste tipo são os da Figura 11, fabricados pela Tritium [28], ABB [29] e Efacec [30].

Figura 11 - Carregadores ultrarrápidos [28] [29] [30]

2.6. REDES DE CARREGAMENTO DE VE

Na última década o número de veículos elétricos presentes nas estradas aumentou exponencialmente [31]. Esta evolução estimulou a instalação de carregadores de VE na via pública dos países, dando assim origem às redes de carregamento.

Em todo o mundo já existem múltiplas redes de carregamento, muitas delas cobrem não só um país, mas também um conjunto de países. O objetivo final das redes é o de criar as condições necessárias para que os utilizadores de VE possam viajar com a segurança de que irão ter um sítio para carregar o seu carro. O desenvolvimento destas redes de carregamento, aliado à cobrança do serviço prestado, levou a que estas atualmente possam ser mantidas e continuamente desenvolvidas.

Em Portugal a organização responsável pela gestão da rede de carregamento na via pública é a MOBI.E. Esta rede tem a distribuição representada na Figura 12 em que os postos de carregamento a azul escuro correspondem a carregadores lentos com saída em AC, enquanto que os postos a azul claro já possuem saída DC até 50 kW. Constata-se que a rede portuguesa está menos desenvolvida que a de outros países europeus e não apenas em quantidade, mas também na instalação de carregadores rápidos e ultrarrápidos [32].

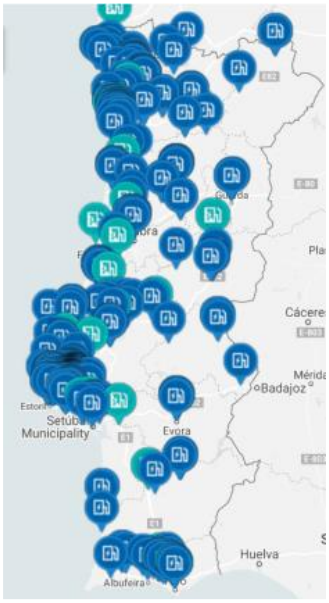


Figura 12 - Rede de carregamento da MOBI.E [7]

Por toda a Europa existem várias empresas responsáveis por desenvolver redes de carregamento, entre as quais a Allego [33], a Fortum [34] ou a Ionity [35]. Estas empresas desenvolveram redes que atravessam fronteiras e já possuem um elevado número de carregadores. O mapa da Figura 13 mostra a rede da Fortum [36], onde é possível observar que o número de carregadores presente na Finlândia (país onde é sediada) ultrapassa os 600.

Figura 13 - Rede de carregamento da Fortum [35]

Nos Estados Unidos da América uma das empresas responsáveis pelo desenvolvimento da rede é a Electrify America, cuja distribuição de carregadores se encontra ilustrada na Figura 14. Esta organização tem como premissa a instalação de carregadores com, pelo menos, 50 kW DC na sua saída. Portanto, todos os carregadores da Electrify America são rápidos ou ultrarrápidos [37].

Figura 14 - Rede de carregamento da Electrify America [36]

A Tesla, devido ao facto de ter um protocolo próprio de carregamento enveredou por criar a sua própria rede de carregamento. Esta característica estimulou a marca a instalar carregadores em grande número por todo o globo, por forma a conseguir criar as condições necessárias para que as pessoas adquiram os seus veículos. Na Península Ibérica o cenário de distribuição dos seus carregadores atualmente é o da Figura 15 [38].

Os postos a vermelho indicam carregadores com capacidade até 120 kW DC e os cinzentos possuem saída em AC até 22 kW. Os utilizadores de veículos Tesla possuem uma realidade bem diferente dos utilizadores de outras marcas pois podem usufruir de uma densa rede de carregamento, atualmente. Os utilizadores da Tesla ao adquirirem adaptadores CHAdeMO ou CCS [39] ou, nos modelos mais recentes que já possuem CCS integrado [40], aumentam ainda mais o número de carregadores onde podem carregar o seu VE.

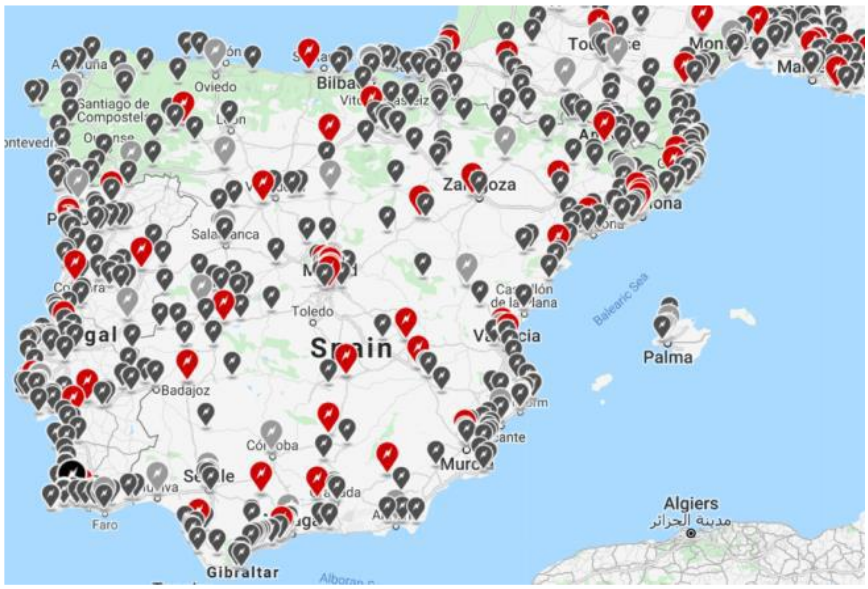


Figura 15 - Rede de carregamento Tesla na Península Ibérica [37]

O rápido crescimento da dimensão das redes de carregamento levou a que fosse necessário o desenvolvimento de ferramentas para a sua gestão. Os atuais sistemas de gestão universalmente adotados baseiam-se no OCPP que oferece uma forma de os carregadores comunicarem com um sistema central. Tendo os carregadores a comunicar em rede é possível, por exemplo, controlar o acesso aos mesmos, saber o seu estado ou contabilizar o consumo de energia.

2.7. DISCUSSÃO

As atuais redes de carregamento de veículos elétricos implementadas pelo mundo podem já ser consideradas extensas, mas a diversidade de modos de carregamento, protocolos,

conectores e potências de carregamento contribuem para a complexidade da mobilidade elétrica e, em particular, dos sistemas de carregamento. Neste sentido, o presente capítulo foi dedicado a contextualizar e clarificar este tema.

Uma rede de carregamento pode integrar inúmeros carregadores que podem chegar a diferentes níveis de potência. Esta situação adequa-se à existência de uma plataforma que proporcione a gestão automática da rede de carregamento. É neste ponto que o OCPP se enquadra ao disponibilizar as ferramentas necessárias para o estabelecimento de uma comunicação padronizada entre cada carregador e um sistema informático central responsável pela gestão de toda a rede. Portanto, o cumprimento do OCPP por parte dos carregadores torna-se fulcral para que possam ser facilmente adicionados a uma atual rede de carregamento gerida centralmente. É neste ponto que incidem os objetivos que este projeto pretende atingir.

3. O *OPEN CHARGE POINT PROTOCOL*

Na última década o número de postos de carregamento de veículos elétricos na via pública cresceu acentuadamente. Esta situação levou à criação de redes de carregamento responsáveis pela gestão desses mesmo carregadores.

Aquando do surgimento das redes de carregamento, os utilizadores de VE podiam carregar o seu veículo de forma grátis, por forma a estimular a sociedade a comprar veículos de propulsão elétrica. Com o passar do tempo, as redes de carregamento viram-se obrigadas a cobrar por carregamento pois, para além da vertente económica, surgiu também a necessidade de manter as estações de carregamento funcionais. Como

qualquer outro sistema, os carregadores de VE necessitam de manutenção para que estejam operacionais a maior parte do tempo.

Por forma a conectar as estações de carregamento a um sistema central capaz de gerir os carregamentos, foi criado o *Open Charge Point Protocol* (OCPP). A *Open Charge Aliance* (OCA) foi a entidade responsável pela criação do protocolo e será este o tema abordado neste terceiro capítulo [41].

O OCPP surgiu em 2009 com a versão 1.5, o qual evoluiu para a versão 1.6 em 2015 e em 2018 foi lançada a versão 2.0 [42]. O principal objetivo deste protocolo é definir a forma de comunicar entre um posto de carregamento e um sistema central que o vai controlar.

Em cada posto de carregamento existirá uma estrutura semelhante à da Figura 16. O VE comunica com o carregador através de um dos protocolos de carregamento abordados no capítulo anterior, e o posto de carregamento, por sua vez, comunica com o sistema central. O OCPP define que a conexão entre o carregador e o sistema central é possível desde que os intervenientes tenham uma conexão que suporte TCP/IP [43].

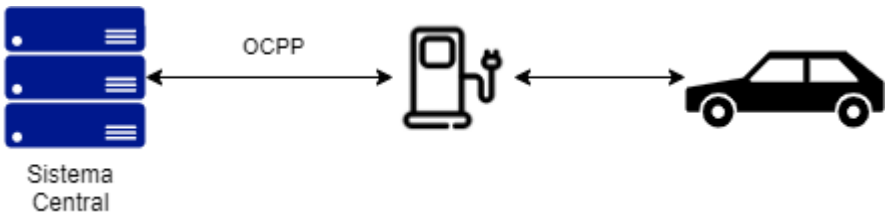


Figura 16 - Diagrama geral de um posto de carregamento

O protocolo incide na troca de mensagens entre o carregador e o sistema central. Esta troca de mensagens entra em funcionamento a partir do momento em que o carregador é ligado. As mensagens definidas pelo OCPP serão abordadas neste capítulo, inicialmente as definidas na versão 1.5 e posteriormente as da versão 1.6. Desta forma será possível descrever a evolução do protocolo entre estas duas versões. A versão 2.0 será explicada de forma complementar dado que o objeto do projeto desenvolvido se baseou na versão 1.6 do protocolo.

3.1. OCPP 1.5

A versão 1.5 do protocolo define que a troca de mensagens entre o carregador e o sistema central usa a *framework Simple Object Access Protocol* (SOAP) pois esta é capaz de enviar mensagens entre intervenientes na internet. A principal vantagem descrita pelo protocolo na utilização do SOAP é que este facilita o envio e receção de mensagens, bem como torna a implementação do protocolo mais rápida [44].

Por sua vez, o conteúdo da mensagem SOAP é desenhado de acordo com o *standard Extensible Markup Language* (XML). Para além do facto das mensagens serem enviadas em texto legível, o XML permite também o envio de imagens e código executável [44].

O OCPP 1.5 define 25 tipos de mensagens diferentes. Destas mensagens, 10 são iniciadas pelo carregador e as restantes 15 iniciadas pelo sistema central. A troca de mensagens é sempre iniciada com um *request*, ao qual é sempre enviado uma *confirmation*. A especificação denomina as mensagens de *request* e de *confirmation* como *OperationName.req*() e *OperationName.conf*(), respetivamente. A título de exemplo, a Figura 17 apresenta uma troca de mensagens do protocolo, por forma a explicar o fluxo da troca de mensagens [44].

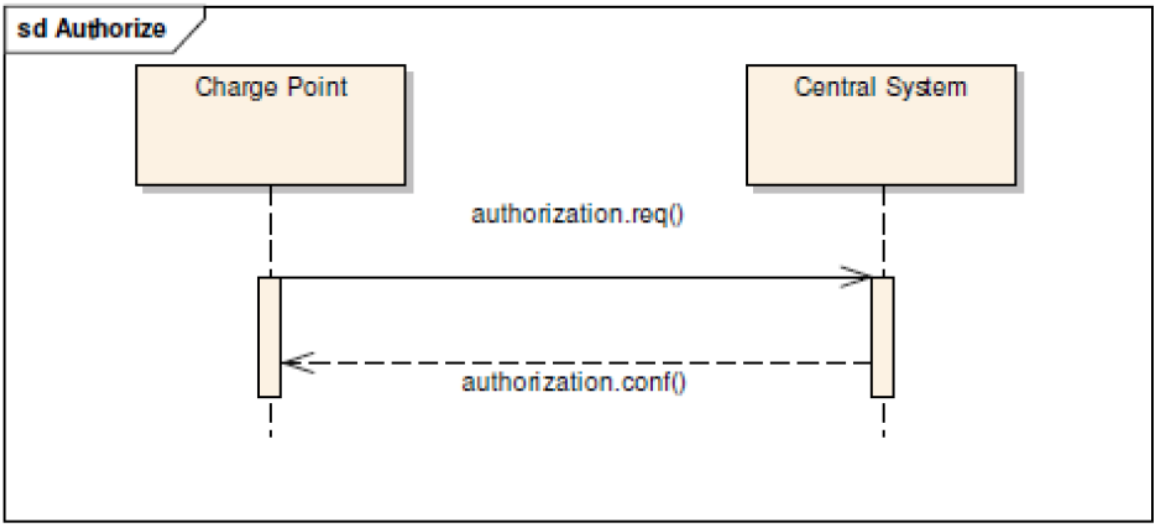


Figura 17 - Fluxo de mensagens do OCPP [43]

Por forma a que seja possível identificar cada carregador por parte do sistema central, o protocolo obriga a que cada carregador possua um *StationID* e o seu respetivo *endpoint.* Por outro lado, o sistema central apenas necessita de fornecer o seu *endpoint.* Estes dados são essenciais para que a troca de mensagens seja possível pois os *endpoints* são os destinos finais das mensagens e o *StationID* é um campo obrigatório nas mensagens enviadas do sistema central para o carregador. A Figura 18 mostra um exemplo de mensagem SOAP escrita em XML. É possível visualizar o campo *ChargeBoxIdentity* onde seria colocado o *StationID* e no corpo da mensagem surge o tipo de *request* (*clearCacheRequest*) feito, neste caso, pelo sistema central.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:ns="urn://Ocpp/Cp/2012/06/">
   <soap:Header>
      <ns:chargeBoxIdentity>?</ns:chargeBoxIdentity>
   </soap:Header>
   <soap:Body>
      <ns:clearCacheRequest/>
   </soap:Body>
</soap:Envelope>
```

Figura 18 - Exemplo de mensagem SOAP

3.1.1. DESCRIÇÃO FUNCIONAL DO OCPP

No OCPP é possível diferenciar vários momentos de utilização do mesmo. Existem comportamentos definidos para quando o carregador é ligado, para quando é necessário iniciar ou terminar uma sessão de carregamento, para quando é necessário realizar alterações na configuração do carregador ou atualizações do *firmware.*

Quando um carregador é iniciado, este é obrigado a entrar em contacto com sistema central e informá-lo acerca das suas características. Do outro lado, o sistema central pode verificar se o carregador pertence à sua rede e aceitar ou rejeitar que este se conecte. Se o carregador for aceite, o sistema central envia a sua data e hora, bem como o *heartbeat interval.* De seguida, o carregador tem de informar o sistema central do estado dos seus conectores [45].

Devido à necessidade por parte do sistema central em saber se os seus carregadores estão *online* ou não, existe a mensagem de *Heartbeat*. Esta mensagem é enviada pelo carregador para o sistema central periodicamente, respeitando sempre o *heartbeat interval* definido pelo sistema central quando o carregador é ligado. Desta forma se o

sistema central receber a mensagem de *heartbeat*, e o carregador a respetiva resposta, conclui-se que o carregador está *online* [45].

Para iniciar uma sessão de carga de um VE, referida como transação no protocolo, tudo vai depender se o utilizador tem autorização para realizá-la. Esta identificação do utilizador é realizada através de uma *tag* de *Radio Frequency IDentification* (RFID) que é enviada numa mensagem para o sistema central a pedir autorização para iniciar a transação. Durante a sessão de carga, o carregador pode enviar dados relativos ao carregamento em si e, no final, quando o utilizador termina o carregamento envia uma mensagem a pedir para terminar a transação [45].

O protocolo permite também, a qualquer momento, o sistema central modificar a configuração dos seus carregadores, pode também requerer um diagnóstico de algum mal funcionamento e ainda, se necessário, realizar um *reboot* ao carregador [45].

3.1.2. MENSAGENS INICIADAS PELO CARREGADOR

O protocolo inclui mensagens que são iniciadas pelo carregador e outras iniciadas pelo sistema central. Esta secção aborda as mensagens iniciadas pelo carregador e qual o seu papel em termos protocolares.

3.1.2.1. BOOTNOTIFICATION

A mensagem de BootNotification é apenas enviada pelo carregador em duas situações distintas: quando o carregador é ativado e quando o carregador passa do estado *offline* para *online.* Ou seja, se por algum motivo a ligação com o sistema central for interrompida, o carregador é obrigado a enviar uma mensagem de *BootNotification.req*() quando esta é reestabelecida [46].

Em termos de estrutura, a mensagem de *BootNotification.req*() tem dois campos obrigatórios, o modelo e o fabricante do carregador. Adicionalmente, a mensagem pode conter o número de série do carregador, o *Integrated Circuit Card Identifier* (ICCID) e o *International Mobile Subscriber Identify* (IMSI) associados ao cartão *Subscriber Identification Module* (SIM) que eventualmente esteja em uso no carregador, o tipo de contador de energia e o seu número de série [47].

Ao receber uma mensagem de *BootNotification.req*() o sistema central tem que responder com uma mensagem de *BootNotification.conf*(). Esta mensagem deve ter obrigatoriamente três campos, nomeadamente a data do sistema central, o *heartbeat interval* e um campo de estado que informa o carregador se está autorizado a interagir com o sistema central [47].

3.1.2.2. STATUSNOTIFICATION

Após a mensagem de *BootNotification.req*() ter sido confirmada, o carregador deve prosseguir com o envio do seu atual estado e dos seus conectores. Para além disso, o carregador deve reportar o seu estado sempre que um utilizador inicia ou termina uma sessão de carga ou exista um problema com a estação. Isso é feito através da mensagem *StatusNotification.req*() [46].

A mensagem de *StatusNotification.req*() tem três campos obrigatórios. O primeiro campo, *connectorID*, consiste num número inteiro não negativo que quando toma o valor 0 (zero) identifica a estação como um todo e, para os restantes valores, identifica um conector específico da estação [47]. O segundo campo é o código do evento que desencadeou o estado atual e o terceiro é o código do estado atual. Na versão 1.5 do protocolo, os estados previstos são os seguintes [48]:

- *Available* – quando o conector está disponível para carregar;
- *Occupied* – quando o conector está ocupado a carregar;
- *Reserved* – quando o conector está reservado para um utilizador;
- *Unavailable* – quando o conector está indisponível para carregar;
- *Faulted* – quando o conector está indisponível para carregar, mas devido a um erro.

Quando o sistema central recebe uma mensagem deste tipo, deve responder com uma mensagem de *StatusNotification.conf*() vazia [46].

3.1.2.3. AUTHORIZE

Para efetuar um carregamento, o utilizador tem que passar um cartão RFID no leitor do carregador. A etiqueta lida é enviada numa mensagem *Authorize.req*() para o sistema central por forma a que este verifique se o utilizador tem autorização para fazer o carregamento. Esta mensagem possui apenas um campo, a etiqueta RFID do cartão do utilizador. O protocolo permite que o carregador guarde em *cache* as etiquetas dos cartões que tenham sido autorizados anteriormente e, por conseguinte, não é obrigado a enviar a mensagem *Authorize.req*() caso a etiqueta esteja ainda armazenada em *cache*. Esta funcionalidade permite que o início do carregamento se processe de forma mais rápida [46] [49].

Por outro lado, o sistema central tem que responder com uma mensagem de *Authorize.conf*(). Esta mensagem tem um campo obrigatório e dois campos opcionais. O campo obrigatório, *AuthorizationStatus,* indica ao carregador se está autorizado a fazer o carregamento. O sistema central pode indicar um dos seguintes códigos no campo *AuthorizationStatus*: *Accepted, Blocked, Invalid, Expired* e *ConcurrentTx.* O carregador apenas pode iniciar uma sessão de carga se receber o conteúdo *Accepted.* Os dois campos opcionais são a data de validade da *etiqueta RFID* e a sua *tag* pai [48].

3.1.2.4. STARTTRANSACTION

Para dar início a uma transação, o que só deverá ser permitido depois do posto de carregamento ter recebido o código *Accepted* no campo *AuthorizationStatus* da mensagem *Authorize.conf*(), o utilizador deve escolher o conector que pretende usar. Após efetuada essa escolha, o carregador tem que enviar uma mensagem do tipo *StartTransaction.req*() para o sistema central.

A mensagem *StartTransaction.req*() tem quatro campos obrigatórios e um opcional. O campo opcional, *ReservationID*, apenas é preenchido quando o utilizador faz uso de uma reserva, procedimento que irá ser descrito no decorrer deste documento. Os campos obrigatórios na mensagem são [46] [47]:

- *ConnectorID* – identifica qual o conector escolhido pelo utilizador;

- *idTag* – a *etiqueta RFID* responsável pelo início da transação: o sistema central verifica o estado desta *tag*, pois esta pode estar assinalada como autorizada na *cache* do carregador, mas, entretanto, o seu estado ter mudado;

- *meterStart* – este campo é preenchido com o valor da energia, em Wh, no momento em que a transação se inicia;

- *Timestamp* – data e hora do início da transação.

O sistema central responde ao *StartTransaction.req*() com uma mensagem *StartTransaction.conf*(), que contém dois campos obrigatórios. O primeiro parâmetro, *idTagInfo,* informa se a etiqueta RFID continua autorizada ou não. O segundo, *transactionID,* é um número inteiro único, que identifica a transação [47].

3.1.2.5. METERVALUES

Durante uma transação, cabe ao carregador enviar medições de determinadas grandezas para o sistema central. A mensagem *MeterValues.req*() tem três campos, sendo o carregador obrigado a preencher apenas um, o campo *ConnectorID.* Contudo, é usual os outros dois campos serem também preenchidos. Estes campos são o identificador de transação, *transactionID,* e os valores das leituras efetuadas identificando sempre o tipo de grandeza, o seu valor e o *timestamp* do instante da medição [46] [47]. Como resposta, o sistema central deve enviar uma mensagem *MeterValues.conf*(), que não tem conteúdo.

O OCPP 1.5 define também quais as grandezas que podem ser enviadas do carregador para o sistema central. Em termos de energia e potência, o protocolo divide entre a que é fornecida ao VE e a que é fornecida pelo mesmo. O protocolo permite também enviar valores de tensão, corrente e temperatura [48].

3.1.2.6. STOPTRANSACTION

Uma transação pode terminar por diversos motivos: por ordem do utilizador, por ordem do veículo ou devido à deteção de uma avaria. Em qualquer uma das situações, o

carregador tem que informar o sistema central que a transação terminou, usando para tal a mensagem *StopTransaction.req*() [46].

O protocolo especifica que a mensagem do carregador deve incluir os campos *transactionID, timestamp* e *meterStop.* Esta informação permite que o sistema central identifique a transação que terminou, assim como a data e hora do respetivo término e a leitura final do medidor de energia. O carregador pode ainda especificar a etiqueta RFID associada à transação e outros dados para efeito de faturação [47].

O sistema central deverá responder com uma mensagem de *StopTransaction.conf*(), opcionalmente com informação relativa à etiqueta RFID associada à transação.

3.1.2.7. DIAGNOSTICS STATUS NOTIFICATION

O sistema central pode requerer ao carregador para enviar um ficheiro de diagnóstico. Normalmente esta situação acontece quando existe algum problema com o carregador, logo o sistema central pede este ficheiro ao carregador e este realiza um *upload* do ficheiro para um endereço previamente conhecido.

A mensagem de *DiagnosticsStatusNotification.req*() tem o único objetivo de informar o sistema central de como decorreu o *upload* [46]. A mensagem tem apenas um campo que pode ser preenchido com *Uploaded* ou *Upload Failed* [47] [50]. O sistema central responde com uma mensagem de *DiagnosticsStatusNotification.conf*() que não tem conteúdo [47].

3.1.2.8. FIRMWARE STATUS NOTIFICATION

O OCPP permite que sejam feitos remotamente *updates* de *firmware* do carregador. Depois de o carregador receber o pedido para esse efeito, este tem que realizar o *download* do *firmware* e instalá-lo, ao mesmo tempo que informa o sistema central do estado do *update.* É nesta situação que surge a mensagem de *FirmwareStatusNotification.req*() [46].

A mensagem *FirmwareStatusNotification.req*() tem um campo único e é onde o carregador indica o estado do *update* [47]. O carregador pode enviar nesse campo os estados *Downloaded, DownloadFailed, InstallationFailed* ou *Installed* [48]. Como

resposta, o sistema central envia uma mensagem de *FirmwareStatusNotification.conf*() sem conteúdo [47].

3.1.2.9. DATA TRANSFER

A mensagem de *DataTransfer.req*() é a única que pode ser enviada quer pelo carregador quer pelo sistema central. O objetivo desta mensagem é criar a possibilidade de o sistema central e o carregador trocarem informação não prevista no OCPP. O único campo obrigatório da mensagem é o *vendorID,* correspondente ao inverso do *Domain Name Service* (DNS) do fabricante do carregador. O restante conteúdo da mensagem é opcional, consistindo num identificador de mensagem, denominado *messageID,* e nos dados que se pretendem transmitir [51] [47].

A mensagem de confirmação tem um campo obrigatório que pode tomar um dos seguintes códigos: *Accepted, Rejected, UnknownMessageID* ou *UnknownVendorID*. Opcionalmente, a mensagem de confirmação também pode incluir um campo dados arbitrários [48].

Um exemplo de utilização desta mensagem pode ser encontrado na transferência dos preços praticados pelo operador da rede de carregamento. Em geral, o sistema central envia uma mensagem *DataTransfer.req*() com a informação do preçário, sendo essa informação apresentada no ecrã do posto de carregamento.

3.1.2.10. HEARTBEAT

A mensagem *Heartbeat.req*() tem como objetivo permitir ao sistema central identificar que carregadores se encontram online. Um carregador deve enviar ciclicamente esta mensagem, de acordo com o valor de *HeartbeatInterval* recebido na mensagem de *BootNotification.conf*() [46].

A mensagem *Heartbeat.req*() não inclui qualquer campo, enquanto que a mensagem de confirmação *Heartbeat.conf*() inclui a data e hora atual [47].

3.1.3. MENSAGENS INICIADAS PELO SISTEMA CENTRAL

Este subcapítulo será dedicado às mensagens iniciadas pelo sistema central. Estas mensagens possuem, na sua maioria, um cariz diferente das que são iniciadas pelo carregador. Usualmente estas mensagens são designadas como comandos remotos, ou seja, são pedidos que o sistema central faz ao carregador. Estes pedidos têm diferentes objetivos e deverão fazer com que o carregador realize diferentes ações.

3.1.3.1. CLEAR CACHE

Como referido anteriormente, o utilizador para iniciar uma transação tem que possuir uma etiqueta RFID e passá-la num leitor presente no carregador. De seguida, o carregador envia um *Authorize.req*() que pergunta ao sistema central o estado daquela *tag.* Mas o protocolo permite que o *Authorize.req*() não tenha que ser enviado, se o carregador tiver na sua memória *cache* a *tag* definida como autorizada.

A mensagem *ClearCache.req*(), que não necessita de nenhum parâmetro, tem o intuito de obrigar o carregador a limpar as *tags* RFID armazenadas na sua memória *cache* [52]. O carregador tem que responder com uma mensagem de *ClearCache.conf*(), aceitando ou rejeitando a mensagem enviada pelo sistema central [47].

3.1.3.2. SEND LOCAL LIST E GET LOCAL LIST VERSION

O protocolo permite que o sistema central envie uma lista de *tags* RFID autorizadas para o carregador. Deste modo, o carregador antes de enviar um *Authorize.req*() pode verificar, para além da sua *cache,* os cartões existentes na sua lista local. Para o sistema central enviar esta lista existe a mensagem *SendLocalList.req*().

A mensagem *SendLocalList.req*() tem que enviar sempre o número correspondente à versão da lista e dizer ao carregador se é para fazer um *update* total ou diferencial à sua lista local. Assim é substituir uma lista inteira ou apenas acrescentar *tags* à lista já existente. A mensagem pode ter a lista de *tags* autorizadas e um valor chamado de *hash* que é calculado através do conteúdo da lista de *tags* enviadas. Deste modo se existir algum problema com a mensagem, o carregador deteta-o ao verificar o valor do *hash* [52] [47].

A resposta que o carregador envia é a mensagem *SendLocalList.conf*() com, pelo menos, um parâmetro que indica o resultado do envio da lista local. Portanto o carregador pode responder com *Accepetd, Failed, HashError, NotSupported* ou *VersionMismatch* [48].

Por forma a que o sistema central consiga sincronizar e garantir que as listas inseridas nos carregadores são as corretas, foi criada a mensagem *GetLocalListVersion.req*() que requisita ao carregador o envio da versão da sua lista local para o sistema central [52]. O carregador responde ao pedido com a mensagem *GetLocalListVersion.conf*(), que incluirá um número correspondente à versão da lista local presente no equipamento [47].

3.1.3.3. RESET

O sistema central pode pedir ao carregador para realizar um *reset*, onde este pode ser de dois tipos: *soft* e *hard.* De maneira geral, um *reset* do tipo *soft* obriga a que o carregador retorne ao seu estado de *idle* e, se existir alguma transação a decorrer, esta tem que ser terminada normalmente. O *hard reset* também termina normalmente alguma transação que esteja a decorrer, mas depois de terminada o carregador tem de realizar um *reboot* [52].

Portanto, o sistema central ao enviar a mensagem de *Reset.req*() tem que especificar no campo *type* se o *reset* é *soft* ou *hard* [47]. O carregador responde ao pedido com a mensagem de *Reset.conf*(), aceitando ou rejeitando o pedido [48].

3.1.3.4. GET CONFIGURATION E CHANGE CONFIGURATION

Os carregadores possuem múltiplas variáveis que podem ser configuradas e estas podem estar relacionadas com o OCPP ou então podem ser relativas ao funcionamento geral do carregador. É sempre possível configurar o carregador ao aceder diretamente ao equipamento, mas essa possibilidade, muitas vezes, é contraproducente. Assim, o protocolo prevê a funcionalidade de obter, ou de alterar, o estado dessas variáveis remotamente.

Para obter o estado das variáveis, o protocolo definiu a mensagem de *GetConfiguration.req*(). Esta mensagem pode ser enviada vazia ou pode conter uma lista das variáveis que se quer saber o seu estado [52]. O carregador responde através da

mensagem *GetConfiguration.conf*() com o estado das variáveis requeridas, ou com o estado de todas as variáveis disponíveis caso a mensagem *GetConfiguration.req*() tenha sido enviada vazia [47].

Caso o sistema central pretenda mudar alguma variável específica pode fazê-lo através da mensagem *ChangeConfiguration.req*(). Nesta mensagem, o sistema central tem que especificar dois campos: a variável a atualizar e o seu novo valor. Em sentido contrário, o carregador responde com a mensagem *ChangeConfiguration.conf*() e indica se o comando remoto foi aceite ou não [53] [47].

3.1.3.5. CHANGE AVAILABILITY

O OCPP define que um carregador deverá estar num estado denominado de *Operative* quando este se encontra a carregar ou pronto a tal. Em qualquer outra situação o carregador deverá assumir o estado de *Inoperative*. O protocolo dá então a possibilidade ao sistema central de mudar a disponibilidade do carregador como um todo, ou por conector [53].

A mensagem que operacionaliza esta funcionalidade é a *ChangeAvailability.req*(), que possui dois campos obrigatórios. O primeiro campo é o *ConnectorID* que, tal como na mensagem *StatusNotification.req*(), se tomar o valor 0 (zero) refere-se ao carregador completo. O segundo campo corresponde ao *AvailabilityType* que permite ao sistema central definir a disponibilidade do dispositivo (*Operative* ou *Inoperative*) [47] [48].

O carregador tem que responder com a mensagem de *ChangeAvailability.conf*() e informa se o comando foi aceite ou não. Um caso especial acontece se a mensagem de *ChangeAvailability.req*() for enviada com uma transação a decorrer nesse conector. Se for o caso, o carregador deve responder *Scheduled* na sua mensagem de *ChangeAvailability.conf*() e obedecer ao comando no final da transação. Após o comando *ChangeAvailability.req*() ser aceite, o carregador tem que enviar para o sistema central uma mensagem de *StatusNotification.req*() que informa do estado atual do conector pois foi realizada uma mudança de estado.

3.1.3.6. REMOTE START TRANSACTION

Ao sistema central é dada a possibilidade de ordenar o carregador a iniciar uma transação num determinado conector. Esta funcionalidade foi criada tendo em vista a possibilidade de se associar aplicações de *smartphones* ao sistema central e que, por sua vez, possuem a conta do utilizador associada também. Deste modo, o utilizador pode iniciar a sua sessão de carregamento sem ser necessário a posse de um cartão com uma etiqueta RFID.

A mensagem criada para o efeito denomina-se *RemoteStartTransaction.req*() e é sempre acompanhada pela etiqueta RFID que pretende iniciar a transação e, possivelmente, de um conector em específico [52][47]. O carregador ao receber este comando tem que responder com uma mensagem de *RemoteStartTransaction.conf*() indicando se aceitou ou rejeitou o pedido. O carregador se aceitar o pedido de iniciar a transação, tem que de seguida proceder ao protocolado na mensagem de *StartTransaction.req*() [52].

3.1.3.7. REMOTE STOP TRANSACTION

Da mesma maneira que é possível iniciar uma transação remotamente, também é possível terminá-la. Com este fim encontra-se definida a mensagem *RemoteStopTransaction.req*(), onde apenas é necessário indicar, ao carregador, o *TransactionID* da transação que deverá ser terminada [52] [47]. O carregador ao responder com a mensagem *RemoteStopTransaction.conf*() também apenas tem que indicar se o comando foi aceite ou não [47]. Em caso afirmativo o carregador tem que cumprir com o protocolado na mensagem de *StopTransaction.req*() [52].

Esta funcionalidade como depende apenas do *TransactionID* pode ser utilizada para terminar transações que foram inicializadas localmente ou remotamente.

3.1.3.8. RESERVE NOW E CANCEL RESERVATION

O protocolo definiu que o sistema central pode pedir ao carregador para reservar um conector para um determinado utilizador. A reserva é feita a partir do momento em que é enviada a mensagem *ReserveNow.req*(), até à data e hora estipuladas no seu conteúdo. Outras formas de terminar uma reserva são o utilizador que fez a reserva começar a sua

transação, ou o conector definido numa reserva mudar o seu estado para *Faulted* ou *Unavailable* [52].

A mensagem *ReserveNow.req*() tem quatro parâmetros obrigatórios: data e hora de término da reserva, etiqueta RFID do utilizador que realizou a reserva, *ReservationID* e o *ConnectorID.* De notar que, se o *ConnectorID* for 0, o carregador tem que garantir que exista um conector disponível para aquele determinado utilizador. Se o *ConnectorID* for superior a 0 o utilizador estará a reservar um conector específico [47].

O carregador ao receber um pedido de reserva tem que responder com uma mensagem de *ReserveNow.conf*() e nesta apenas tem que informar o sistema central do resultado do pedido realizado. Neste caso o carregador tem a possibilidade de responder *Accepted, Faulted, Rejected, Occupied* ou *Unavailable* ao pedido de reserva [48].

O utilizador do VE pode querer cancelar a reserva e, para tal, no OCPP existe a mensagem *CancelReservation.req*(). O único parâmetro que esta mensagem necessita é o *ReservationID* gerado para a mensagem de *ReserveNow.req*(). Em sentido contrário, o carregador responde através da mensagem de *CancelReservation.conf*() e indica se aceita ou rejeita o comando remoto de cancelamento da reserva [52] [47].

3.1.3.9. UNLOCK CONNECTOR

Os carregadores com saída em AC podem possuir já um conector neles instalado ou então um *inlet* em que o utilizador pode conectar o cabo de carregamento que vem com o veículo. Nesta segunda situação, o carregador prende o cabo de carregamento no início da sessão de carregamento e assim se mantém até o utilizador mandar desbloquear o conector. Esta situação acontece para garantir que, se o carro for deixado a carregar e por algum motivo a transação terminar, o cabo não seja furtado.

Para desbloquear o conector existem duas possibilidades. A primeira é o utilizador desbloquear fisicamente com alguma opção que exista no carregador e a segunda é o sistema central enviar uma mensagem de *UnlockConnector.req*(). Esta mensagem indica qual o conector a ser desbloqueado através do seu *ConnectorID* e o carregador responde com a mensagem de *UnlockConnector.conf*() aceitando ou rejeitando comando [52] [47].

Um caso especial acontece quando este comando remoto é enviado com uma transação a decorrer. Neste caso, o carregador deve proceder ao término da carga de acordo com o que foi descrito na mensagem *StopTransaction.req*() [52].

3.1.3.10. GET DIAGNOSTICS

Em todos os sistemas ocorrem problemas e os carregadores de VE não são exceção. Para ajudar no diagnóstico do problema, normalmente, os fabricantes de carregadores criam um ficheiro com informação que lhes possa ser útil. De modo a facilitar o processo de obtenção desse ficheiro foi criada a mensagem de *GetDiagnostics.req*().

O objetivo da mensagem de *GetDiagnostics.req*() é fornecer um *link* de um diretório para o qual o carregador pode enviar o ficheiro através do *File Transfer Protocol* (FTP). Portanto a mensagem de *GetDiagnostics.req*() possui um campo obrigatório que é o *link* do diretório e outros quatro opcionais [52]. Na mensagem, o sistema central pode definir o número de tentativas que o carregador tem para enviar o ficheiro e o intervalo de tempo entre elas, a data de início e fim da informação recolhida pelo carregador. O carregador responde com a mensagem *GetDiagnostics.conf*() que pode ir vazia ou com o nome do ficheiro enviado [47].

Depois da receção da mensagem de *GetDiagnostics.req*(), para além do envio do ficheiro, o carregador tem que informar o sistema central do estado do envio. Para tal, tem que respeitar o abordado nas mensagens de *DiagnosticsStatusNotification.req*().

3.1.3.11. UPDATE FIRMWARE

Os carregadores são equipamentos que necessitam de realizar *updates* ao seu *firmware* com correções de erros ou com novas implementações. Para agilizar esse processo, o sistema central pode ordenar o carregador a realizar um *update*.

A mensagem de *UpdateFirmware.req*() trata de fornecer um *Uniform Resource Locator* (URL) para que o carregador faça *download* do ficheiro de *update*, quando este o pode realizar, o número de tentativas permitidas para tentar fazer o *update* e o intervalo de tempo entre cada tentativa. O carregador tem que responder com a mensagem *UpdateFirmware.conf*() que não tem conteúdo [52] [47].

Durante o processo de *update* do *firmware*, o carregador tem que informar o estado, como foi abordado nas mensagens de *FirmwareStatusNotification.req*().

3.1.4. COMPORTAMENTO OFFLINE

Por variados motivos, a ligação à internet por parte do carregador pode não ser possível em determinados períodos. O facto de o carregador estar *offline* não deve impedir que este realize as funções para as quais está destinado.

Como mencionado anteriormente, o objetivo da mensagem *Heartbeat.req*() é detetar que a ligação entre o sistema central e o carregador se mantém ativa. Quando o carregador verifica que não recebeu a resposta a esta mensagem, pode considerar que está *offline*. Embora esta mensagem tenha sido criada com este propósito, na verdade o carregador pode considerar que está *offline* se não receber resposta a qualquer outra mensagem OCPP.

Para que o carregador possa continuar a efetuar transações em modo *offline*, uma alternativa consiste em permitir que este aceite todos os pedidos de utilização do carregador, uma vez que não é possível perguntar ao sistema central se o utilizador é ou não autorizado. Uma segunda alternativa prevista no protocolo é o envio periódico, por parte do sistema central, de uma lista de utilizadores autorizados. Desta forma, se o carregador ficar *offline*, poderá verificar se o utilizador está autorizado ou não, com base na cópia local da lista de utilizadores autorizados [54].

A partir do momento em que o carregador deteta que está *offline*, este não deve tentar enviar nenhuma mensagem OCPP a não ser a de *BootNotification.req*(). Ciclicamente o carregador tenta estabelecer comunicação com o sistema central, usando esta mensagem. Durante o período que estiver sem comunicações com o sistema central, se ocorrer alguma transação, o carregador tem que guardar os dados de cada transação em memória [54].

Quando é finalmente obtida resposta à mensagem de *BootNotification.req*() e o carregador é aceite pelo sistema central, o carregador tem que enviar mensagens de *StatusNotification.req*() informando o sistema central do estado atual dos seus

conectores. No caso de existirem dados sobre transações ocorridas durante o período em *offline*, estes devem ser reportados pós o envio do estado dos conectores. Para reportar as transações, o carregador tem que enviar, para cada transação, uma mensagem de *StartTransaction.req*() com o *timestamp* correspondente ao instante em que a transação foi iniciada, as mensagens de *MeterValues.req*() e, se a transação já tiver entretanto terminado, a respetiva mensagem de *StopTransaction.req*() [54].

3.2. OCPP1.6

A versão 1.6 do OCPP foi lançado pela OCA em 2015. Esta versão do protocolo não elimina nenhuma das mensagens definidas na versão 1.5, mas altera alguns pormenores destas. Adicionalmente, a versão 1.6 inclui uma nova funcionalidade denominada *Smart Charging*, tendo sido para tal definido um conjunto de novas mensagens [55].

No entanto, é no formato de codificação das mensagens e na forma como estas são transmitidas que a versão 1.6 introduziu maiores alterações. Para além de continuar a prever a utilização do SOAP, o OCPP1.6 define a possibilidade de, alternativamente, as mensagens serem codificadas no formato *JavaScript Object Notation* (JSON). Se usado este formato o mecanismo de Websocket é usado como meio de transmissão das mensagens. Por forma a facilitar a distinção das diferentes variantes do protocolo, o protocolo é referido como OCPPJ quando usado com mensagens em formato JSON e OCPPS quando usado com mensagens SOAP [56].

3.2.1. OCPP1.6 SOAP

O OCPP16S define novas regras para a troca de mensagens entre os postos de carregamento e o sistema central, passando a ser obrigatório preencher o cabeçalho da mensagem com novos campos.

As mensagens iniciadas pelo sistema central devem identificar o *StationID* do carregador ao qual se destina a mensagem através de um elemento do tipo *ChargeBoxIdentity.* Para além de identificar o carregador a que se destina a mensagem, o sistema central tem que adicionar o campo *Action,* indicando o tipo de mensagem OCPP a enviar antecedido por

uma barra. Por exemplo, um campo *Action* com o valor "/ClearCache" indica que a mensagem transporta um pedido do tipo *ClearCache.req*(). No caso de ser o carregador a iniciar a troca de mensagens, o campo *Action* não deverá ser incluído, mantendo-se as restantes disposições[57].A Figura 19 apresenta um exemplo de uma mensagem do OCPP1.6 em formato SOAP.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:ns="urn://Ocpp/Cp/2015/10/">
   <soap:Header>
      <ns:chargeBoxIdentity>?</ns:chargeBoxIdentity>
      <ns:Action>/ClearCache</ns:Action>

   </soap:Header>
   <soap:Body>
      <ns:clearCacheRequest/>
   </soap:Body>
</soap:Envelope>
```

Figura 19 - Exemplo de mensagem SOAP OCPP1.6

3.2.2. OCPP1.6 JSON

A utilização do JSON permite a troca de mensagens num formato mais compacto do que o proporcionado pelo SOAP. Porém, em termos de OCPP, a utilização deste formato obriga a que existam regras diferentes na troca de mensagens.

Apesar da comunicação através de WebSocket ser *full-duplex*, o OCPP1.6J define um mecanismo síncrono para a realização de pedidos: nenhum novo *pedido* deve ser efetuado antes de ser recebida a confirmação ao pedido anterior [58].

A *framework* criada define que podem ser enviados três tipos de mensagens OCPP designados por: CALL, CALLRESULT e CALLERROR. Durante a troca de mensagens, o protocolo define que a mensagem de CALL é identificada pelo número 2, a de CALLRESULT pelo número 3 e a de CALLERROR pelo número 4. Logo, para cumprir o requisito de sincronismo, uma nova mensagem CALL só pode ser enviada quando for recebida a mensagem CALLRESULT ou CALLERROR referente ao pedido anterior [58].

3.2.2.1. MENSAGEM DO TIPO CALL

Uma mensagem do tipo CALL corresponde a um *pedido* efetuado por um dos intervenientes. Este tipo de mensagens contém quatro campos [58]:

- *MessageTypeID*: o número 2 que identifica que é uma mensagem CALL;

- *UniqueID*: identificador único para a troca de mensagens;
- *Action*: a mensagem OCPP que se pretende enviar;
- *Payload*: conteúdo JSON da mensagem OCPP.

A Figura 20 apresenta o formato padrão a que as mensagens CALL deverão obedecer e a Figura 21 um exemplo do envio de um pedido de BootNotification [58].

[<MessageTypeId>, "<UniqueId>", "<Action>", {<Payload>}]

Figura 20 – Formato padrão de uma mensagem do tipo CALL [57]

```
[2,
 "19223201",
 "BootNotification",
 {"chargePointVendor": "VendorX", "chargePointModel": "SingleSocketCharger"}
]
```

Figura 21 - Exemplo de uma mensagem do tipo BootNotification [57]

3.2.2.2. MENSAGEM DO TIPO CALLRESULT

As mensagens do tipo CALLRESULT são enviadas como resposta às mensagens do tipo CALL, caso não ocorram erros. Este tipo de mensagem contém três campos [58]:

- *MessageTypeID*: o número 3, que identifica que é uma mensagem do tipo CALLRESULT;
- *UniqueID*: identificador único para a troca de mensagens;
- *Payload*: conteúdo JSON da mensagem.

A Figura 22 apresenta o formato da mensagem CALLRESULT e a Figura 23 uma mensagem CALLRESULT que poderia ser enviada como resposta à mensagem da Figura 22 [58].

[<MessageTypeId>, "<UniqueId>", {<Payload>}]

Figura 22 - Trama da mensagem CALLRESULT [57]

```
[3,
 "19223201",
 {"status":"Accepted", "currentTime":"2013-02-01T20:53:32.486Z", "heartbeatInterval":300}
]
```

Figura 23 - Exemplo de uma mensagem CALLRESULT [57]

3.2.2.3. MENSAGEM DO TIPO CALLERROR

Na eventualidade do recetor detetar lgum erro aquando da receção de uma mensagem CALL, o OCPP define que a resposta deverá ser uma mensagem do tipo CALLERROR.

Esta mensagem é enviada quando a mensagem CALL enviada por um interveniente não cumpre as especificações. Ou seja, o que desencadeia o envio deste tipo de menagens é um erro num formato da mensagem recebida. Por exemplo, se uma mensagem CALL não possui o campo *MessageTypeID,* o recetor envia uma mensagem do tipo CALLERROR. Este tipo de mensagem, cujo formato é ilustrado Figura 24, tem cinco campos [58]:

- *MessageTypeID*: o número 4 que identifica que é uma mensagem CALLERROR;
- *UniqueID*: identificador único para a troca de mensagens;
- *ErrorCode*: código de erro definido no protocolo;
- *ErrorCodeDescription*: descrição do código de erro;
- *Error details*: detalhes do erro que não estão definidos, no protocolo.

[<MessageTypeId>, "<UniqueId>", "<errorCode>", "<errorDescription>", {<errorDetails>}]

Figura 24 - Trama da mensagem de CALLERROR [57]

3.2.3. PRINCIPAIS ALTERAÇÕES INTRODUZIDAS NO OCPP1.6

Com o OCPP1.6 foram introduzidas alterações nas regras de troca de mensagens. Estas alterações tiveram como objetivo principal criar a possibilidade de o carregador fornecer ao sistema central informação mais detalhada.

Além das mensagens criadas para a implementação do *Smart Charging,* o OCPP1.6 define uma nova mensagem designada TriggerMessage e modifica algumas das mensagens da versão 1.5. Essas alterações, com a exceção da funcionalidade de *Smart Charging*, são descritas na presente secção.

3.2.3.1. STATUS NOTIFICATION

Por forma a que o carregador possa fornecer mais informação acerca do seu estado, o OCPP1.6 introduziu novos estados que o carregador pode reportar. Para tal, o estado *ocuppied* foi eliminado e foram definidos os seguintes estados [59] [60]:

- *Preparing* –indica que que o conector está prestes a ser utilizado, mas ainda não existe nenhuma transação a decorrer; o carregador passa para este estado quando o utilizador insere um conector no carro, passa um cartão RFID válido ou até mesmo ocupar um lugar de estacionamento.

- *Charging* – indica que a bateria do veículo está a ser carregada;

- *Finishing* – indica que a transação já terminou, mas o conector continua ainda ligado ao veículo ou este continua a ocupar a posto de carregamento;

- *SuspendedEVSE* – devido às políticas de *SmartCharging* ou devido a intervenções na própria instalação, o carregador pode parar o fornecimento de energia ao veículo momentaneamente, sem terminar a transação. É nessa situação que o carregador reporta o estado *SuspendedEVSE;*

- *SuspendedEV* – este estado é reportado quando existe uma transação a decorrer, mas o veículo parou de receber energia, embora o carregador esteja apto para fornecer.

Estes cinco novos estados apenas podem ser reportados para uma entidade com *ConnectorID* maior que 0, visto que não faz sentido reportá-los para uma posto de carregamento [59].

Sublinhe-se que a estrutura das mensagens *StatusNotification.req*() *StatusNotification.conf*() não sofreu qualquer alteração [59] [61].

3.2.3.2. METERVALUES

No que toca à mensagem de MeterValues, a versão 1.6 define a possibilidade de transmitir grandezas adicionais ao sistema central. Define também a possibilidade do carregador transmitir ao sistema central os valores da sua potência máxima, da sua corrente máxima, da frequência da alimentação, das rotações por minuto de eventuais ventoinhas do sistema de arrefecimento, do fator de potência e do *State of Charge* [60].

Destas alterações, destaca-se a introdução da informação do *State of Charge*, que permite ao sistema central monitorizar a evolução do estado da bateria durante os carregamentos e registar os respetivos tempos de carga para posterior análise.

3.2.3.3. STOPTRANSACTION

Na versão 1.6, a mensagem de término da transação passa a ter mais um campo opcional (denominado *Reason*) que permite indicar a razão para o fim da transação [61].

O protocolo prevê diversas situações pelas quais é possível terminar uma transação, mas dá ênfase à razão denominada de *Local*. Este código indica que a transação foi terminada normalmente pelo utilizador, através da passagem do cartão RFID no leitor ou através de um botão existente no carregador para o efeito, correspondendo esta à situação mais frequente. O campo *Reason* permite também informar o sistema central que a transação foi terminada pelo botão de emergência, por falha de energia ou por um dos vários tipos de *reboot* [60].

3.2.3.4. TRIGGERMESSAGE

A única mensagem nova apresentada pela versão 1.6 do protocolo e que não é relacionada com o *SmartCharging* é a TriggerMessage. Esta mensagem é iniciada sempre pelo sistema central e surge com o objetivo de solicitar que o carregador envie certas mensagens obrigatoriamente [62].

O protocolo justifica a criação desta mensagem com o facto de que o carregador poder passar largos períodos sem reportar, por exemplo, o estado dos seus conectores. Se o sistema central considerar que passou demasiado tempo sem receber mensagens de StatusNotification, pode solicitá-la através da mensagem TriggerMessage [62].

A mensagem de *TriggerMessage.req*() possui um campo obrigatório e outro opcional. O campo obrigatório denomina-se de *RequestedMessage* e serve para o sistema central indicar qual a mensagem a ser enviada pelo carregador. O campo opcional é *ConnectorID* e é apenas utilizado caso a mensagem pretendida pelo sistema central se referir a um conector do carregador. O protocolo define que as mensagens que podem ser solicitadas pelo sistema central são as seguintes: BootNotification, DiagnosticsStatusNotification, FirmwareStatusNotification, HeartBeat, MeterValues e StatusNotification [61] [60].

Em sentido contrário, o carregador tem que responder à mensagem de *TriggerMessage.req*() com uma mensagem *TriggerMessage.conf*() informando o sistema central se aceita, rejeita ou se a funcionalidade não está implementada. Se o carregador aceitar, este só deve enviar a mensagem preterida após o envio da mensagem *TriggerMessage.conf*() [60].

3.2.4. SMARTCHARGING

Com o avanço das redes de carregamento de veículos elétricos, o impacto na rede elétrica dos países por parte destes sistemas de carregamento é cada vez maior. O *SmartCharging* surge com o objetivo de controlar o impacto que as redes de carregamento provocam nas redes elétricas dos países.

Portanto, as redes de carregamento podem controlar os seus carregadores ou certos utilizadores que possuam determinados veículos consoante, por exemplo, a capacidade da instalação elétrica da zona ou edifício onde são instalados os carregadores. Neste sentido, a versão 1.6 do OCPP permite que o sistema central possa aplicar limites de potência ou corrente aos carregadores durante determinados períodos do dia, denominando-os de perfis de carregamento [63].

Cada perfil de carregamento que é enviado para um carregador tem que ter obrigatoriamente um propósito e um *stack level*. Desta forma, o carregador pode guardar múltiplos perfis, ordenando-os pelo seu *stack level* e aplicando-os consoante o seu propósito. Os três propósitos definidos no *SmartCharging* são [63]:

- *ChargePointMaxProfile* – este propósito é direcionado ao carregador por completo. Neste caso, os limites de potência ou corrente têm que ser partilhados por todos os conectores existentes no carregador;

- *TxDefaultProfile* – como o próprio nome indica, este propósito define um perfil *default* a aplicar às transações. Este perfil pode ser definido ao carregador por inteiro (*ConnectorID* com valor 0) ou a um conector em específico (*ConnectorID* com valor maior que 0);

- *TxProfile* – este propósito é utilizado quando é necessário aplicar um perfil de carregamento a uma transação em específico. Desta forma, este perfil aplicado à transação sobrepõe qualquer outro que estivesse a ser utilizado naquele período.

Para além do *stack level* e propósito, um perfil de carregamento tem que ter um número de identificação único (*ChargingProfileID*) e informação se o perfil é do tipo *absolute, recurrent* ou *relative*. O tipo *absolute* define que o perfil de carregamento é para aplicar a um certo período no tempo, com uma data e hora de início e fim. O tipo *relative* define que o perfil deve ser aplicado consoante ocorra um evento em específico. Por último, o tipo *recurrent* designa que o objetivo do perfil é que se repita no tempo e, caso este seja o tipo de perfil escolhido, o sistema central tem que informar se o perfil se repete diariamente ou semanalmente [63].

Após as configurações iniciais do perfil de carregamento, o sistema central tem que definir o *ChargingSchedule*. Este parâmetro informa qual a duração do perfil em segundos, a grandeza a controlar (potência ou corrente), a data e hora de começo do perfil, e o *ChargingSchedulePeriod* [63] [60].

É no *ChargingSchedulePeriod* que o sistema central divide a duração total do perfil em três períodos. Cada período tem definido quando este começa em segundos, sendo que o primeiro tem que ter obrigatoriamente o valor 0. Em cada período é também informado o valor máximo que a grandeza a controlar pode atingir e o número de fases a controlar [63] [60]. A Figura 25 apresenta de forma esquemática um exemplo de um perfil de carregamento.

CHARGINGPROFILE			
chargingProfileId	**100**		
stackLevel	**0**		
chargingProfilePurpose	**TxDefaultProfile**		
chargingProfileKind	**Recurring**		
recurrencyKind	**Daily**		
chargingSchedule	(List of 1 ChargingSchedule elements)		
	ChargingSchedule		
	duration	**86400 (= 24 hours)**	
	startSchedule	**2013-01-01T00:00Z**	
	chargingRateUnit	**W**	
	chargingSchedulePeriod	(List of 3 ChargingSchedulePeriod elements)	
		ChargingSchedulePeriod	
		startPeriod	**0 (=00:00)**
		limit	**11000**
		numberPhases	3
		startPeriod	**28800 (=08:00)**
		limit	**6000**
		numberPhases	3
		startPeriod	**72000 (=20:00)**
		limit	**11000**
		numberPhases	3

Figura 25 - Exemplo de um perfil de carregamento [62]

3.2.4.1. SETCHARGINGPROFILE

As mensagens relativas ao *SmartCharging* são sempre iniciadas pelo sistema central, sendo a mensagem *SetChargingProfile.req*() a mais importante. É através desta mensagem que o sistema central envia o perfil de carregamento para o carregador [62].

O sistema central tem que preencher dois campos na mensagem de *SetChargingProfile.req*(), o *ConnectorID* e o perfil de carregamento. Em sentido contrário,

na mensagem de *SetChargingProfile.conf*(), o carregador tem que informar o sistema central se aceita, rejeita ou se não suporta o comando enviado [61].

3.2.4.2. GETCOMPOSITESCHEDULE

A qualquer momento o sistema central pode requisitar ao carregador que este envie o horário de perfis que possui, i.e., , tem que reportar quais os perfis que tem ativos durante determinado período [62].

A mensagem de *GetCompositeSchedule.req*() tem como campos obrigatórios o *ConnectorID* e a duração. O sistema central pode também requisitar que o carregador envie os limites máximos em potência ou corrente [61]. Por sua vez, o carregador ao receber a mensagem de *GetCompositeSchedule.req*() tem que enviar o seu *ChargingSchedule* que começa desde o momento em que recebe a mensagem, mais a duração indicada pelo sistema central. Na mensagem de *GetCompositeSchedule.conf*(), se o carregador não suportar esta funcionalidade, pode responder *Rejected*. Se o carregador aceitar o comando, tem que enviar o seu *ChargingSchedule* [61].

3.2.4.3. CLEARCHARGINGPROFILE

Por forma a controlar quais os perfis de carregamento que estão definidos no carregador, o sistema central tem a possibilidade de poder eliminar perfis específicos. Para esse efeito o OCPP criou a mensagem *ClearChargingProfile.req*(), em que o sistema central deve indicar obrigatoriamente o *ChargingProfileID* do perfil que pretende eliminar. Se for necessário, o sistema central pode ainda indicar o *ConnectorID,* o *stack level* e o propósito [62] [61].

O carregador tem que responder com a mensagem *ClearChargingSchedule.conf*() em que informa o sistema central se aceita o comando ou responder *Unknown*, caso o *ChargingProfileID* seja desconhecido para o carregador [61].

3.2.5. COMPORTAMENTO OFFLINE

Como acontece com a versão 1.5 do protocolo, o carregador tem que estar previamente preparado caso a sua ligação à internet deixe de funcionar. Na versão 1.6, o que está protocolado diferencia-se entre os dois formatos apresentados por esta nova versão.

Na versão OCPP1.6S, o protocolo mantém a mesma ideologia definida na versão 1.5. O carregador deteta que entrou em modo *offline* pela falha na receção das respostas às mensagens OCPP. Após isso, o carregador tenta enviar a mensagem de *BootNotification.req*() até receber resposta por parte do sistema central [57].

No que toca à versão JSON do OCPP1.6, como a comunicação tem por base a existência de um WebSocket, o carregador tem a possibilidade de implementar o sistema de *Ping* e *Pong* sempre existente nos WebSocket. Esta funcionalidade tem por base o envio de uma trama de *Ping* por parte do carregador e esperar o respetivo *Pong* enviado pelo sistema central. Com isto é possível detetar que o *endpoint* do sistema central ainda está responsivo e a ligação se mantém. Este mecanismo é repetido periodicamente, respeitando um intervalo definido do carregador e a partir do momento em que o WebSocket é estabelecido. Esta funcionalidade traz vantagens na deteção da estabilidade da ligação e consegue substituir grande parte das mensagens de *Heartbeat.req*(). De notar que a mensagem de *Heartbeat.req*() é sempre necessária para o carregador ter o seu relógio sincronizado com o do sistema central [64].

Aquando da reconexão, na versão 1.6J não é necessário enviar a mensagem de *BootNotification.req*() se nenhum parâmetro desta mensagem tiver alterado o seu valor durante o tempo que o carregador esteve *offline*. Por outro lado, a versão 1.6S necessita de enviar a mensagem completa de *BootNotification.req*(), exatamente como na versão 1.5. Após esta fase, tanto os protocolos 1.6S e 1.6J têm que reportar o estado dos seus conectores e os dados das eventuais transações que tenham decorrido em *offline*.

3.2.6. SEGURANÇA

O OCPP assenta no envio de mensagens entre dois participantes através de uma ligação internet. Como nenhum dos intervenientes quer que os seus dados sejam comprometidos é fulcral que a rede de carregamento possua mecanismos de cibersegurança. O protocolo define então dois mecanismos que podem ser utilizados no caso do OCPP1.6J e um para o OCPP1.6S.

O mecanismo aconselhado pelo protocolo para ser utilizado na versão 1.6S, mas que também pode ser utilizado no 1.6J é ao nível da rede. Ou seja, deve ser criada uma rede

privada entre o sistema central e o carregador. No caso do OCPP1.6S, ainda pode ser acrescentada outra camada de segurança, se forem utilizados pedidos *HyperText Transfer Protocol Secure* (HTTPS) na troca de mensagens [65] [66].

Para a versão 1.6J, onde não seja possível criar uma rede segura, o protocolo define o uso de mecanismos de encriptação e autenticação. O *standard* industrial para encriptação é o *Transport Layer Security* (TLS) e é este que pode ser utilizado no OCPP1.6J. O protocolo defende o uso do TLS devido ao elevado número de bibliotecas de TLS que trabalham com WebSocket e assim facilitar o desenvolvimento dos mecanismos de segurança. O sistema central pode ainda criar um *username* e *password* para cada carregador da sua rede e, desta forma, obrigar os seus carregadores a autenticarem-se aquando o estabelecimento do WebSocket [65].

3.3. OCPP 2.0

Em abril de 2018, a OCA lançou uma versão renovada do OCPP, a 2.0. A organização decretou no seu lançamento que esta nova versão não é retro compatível com as versões 1.5 e 1.6, mas que traria várias novas funcionalidades importantes.

Embora atualmente a versão do protocolo mais utilizada seja a OCPP1.6, o lançamento da versão 2.0 é considerada uma *milestone* fulcral na evolução da mobilidade elétrica. Esta nova versão traz múltiplas novidades aos mais variados níveis e prova que a aposta na mobilidade é cada vez maior. Com o propósito de preparar as redes de carregamento rápido de veículos elétricos para um crescimento acentuado de utilização, a versão 2.0 apresenta novas funcionalidades de controlo do carregamento e *SmartCharging.* Do ponto de vista do desenvolvimento de soluções, o protocolo pretende tornar mais ágil a sua implementação permitindo aos fabricantes de carregadores desenvolver funcionalidades inovadoras em torno do protocolo.

O grande foco da versão 2.0 do protocolo é a interoperabilidade com a ISO 15118 desenhada para comunicação *Vehicle To Grid* (V2G). No decorrer da presente secção será apresentado o impacto que esta nova funcionalidade poderá ter na mobilidade elétrica e

nas *smart grids.* Devido à dimensão que o protocolo toma nesta nova versão, não serão abordadas todas as mensagens em detalhe.

3.3.1. PROTOCOLOS DE TRANSPORTE E CIBERSEGURANÇA

Em termos de protocolo de transporte das mensagens OCPP, a versão 2.0 estabelece que o SOAP deixa de ser suportado. Com o aumento da complexidade do conteúdo das mensagens provocado por este *update* do protocolo, se estas continuassem a usar SOAP, tornar-se-iam complicadas de ler, a largura de banda usada seria superior e, por conseguinte, também o custo associado à transmissão dos dados por rede móvel [67].

Portanto, o OCPP 2.0 suporta apenas mensagens em formato JSON implementadas sobre WebSockets. Depois de na versão 1.6 a OCA verificar que a maioria das implementações do protocolo adotaram o formato JSON, levou a que a exclusão do formato SOAP se tornasse uma realidade. A utilização de WebSocket na comunicação já resultava na diminuição dos custos associados à quantidade de dados transmitidos pela rede móvel. No entanto, a versão 2.0 do protocolo apresenta suporte para *Websocket Compression*, o que pode reduzir ainda mais a quantidade de dados a transmitir [67][68].

Em termos de segurança o protocolo apresentou três perfis de segurança que podem ser aplicados. Estes três perfis têm por base a conjugação do TLS (já utilizado na versão 1.6), a criação de certificados e a respetiva aplicação dos mesmos no carregador, sistema central e na comunicação entre os dois intervenientes. O primeiro perfil é considerado inseguro e deve ser apenas implementado quando os intervenientes se encontram numa rede privada é caracterizado por implementar a básica autenticação HTTP (*HyperText Transfer Protocol*) do lado do carregador. O segundo já é considerado seguro pois implementa a utilização do TLS no transporte das mensagens, TLS e certificado do lado do sistema central e o carregador com autenticação básica HTTP. Por último, o perfil mais seguro é aquele que utiliza TLS com certificado no carregador e sistema central, bem como TLS no transporte das mensagens. A Tabela 2 apresenta os vários perfis possíveis [69].

Profile	Charging Station Authentication	CSMS Authentication	Communication Security
1. Unsecured Transport with Basic Authentication	HTTP Basic Authentication	-	-
2. TLS with Basic Authentication	HTTP Basic Authentication	TLS authentication using certificate	Transport Layer Security (TLS)
3. TLS with Client Side Certificates	TLS authentication using certificate	TLS authentication using certificate	Transport Layer Security (TLS)

Tabela 2 - Perfis de segurança definidos no OCPP 2.0 [68]

No que se refere à troca de mensagens, a *framework* utilizada continua a ser a já definida na versão 1.6. A lógica subjacente às mensagens de CALL, CALLRESULT e CALLERROR é a mesma, bem como o seu conteúdo e a sua identificação. Um caso particular ocorre quando o sistema central, ou o carregador, se querem certificar que estão a comunicar com o interveniente correto. Nesta situação OCPP 2.0 apresenta a opção de os intervenientes poderem assinar as suas mensagens, caso o requeiram. Por exemplo, o sistema central para requerer ao carregador para assinar as suas mensagens apenas tem que adicionar o texto "-Signed" no campo da ação que pretende realizar. Se a mensagem que o sistema central quer enviar tem a ação "ClearCache" e este quer que a resposta tenha assinatura, a ação tem que passar a ser "ClearCache-Signed" [70].

3.3.2. EVENTOS RELACIONADOS COM TRANSAÇÕES E MUDANÇAS DE ESTADO

O OCPP 2.0 continua a ser uma conversa entre dois intervenientes, mas a sua lógica foca-se mais em momentos do funcionamento e *use cases* do que simplesmente na abordagem das suas mensagens.

No campo das transações as mensagens de StartTransaction, StopTransaction e MeterValues deixaram de existir e a informação que estas eram encarregues de enviar é agora enviada na mensagem de TransactionEvent. Esta nova mensagem tem a seu encargo todos os envios de informação que o carregador tenha que realizar durante uma transação, incluindo as mudanças de estado de *Preparing*, *Charging* e *Finishing* existentes no OCPP1.6.

A mensagem de TransactionEventRequest tem cinco campos obrigatórios e sete opcionais [71]. O primeiro campo diz respeito ao *EventType* e é preenchido com *Started* na primeira mensagem da transação, *Updated* durante a transação e *Ended* na última mensagem [72]. Outro campo obrigatório é o número da sequência, ou seja, a primeira mensagem de

TransactionEvent enviada pelo carregador tem um número *seqNo* e, no decorrer da transação, este é incrementado tornando-se mais fácil controlar o fluxo das mensagens e verificar se nenhuma foi perdida [71]. O campo mais informativo e que permite a substituição das mensagens acima referidas é o TriggerReason. Ou seja, é neste campo que o carregador informa a razão pela qual está a enviar a mensagem de TransactioEventRequest. Os motivos que podem ser preenchidos no TriggerReason são apresentados na Tabela 3 [72].

Value	Description
Authorized	Charging is authorized, by any means. Might be an RFID, or other authorization means.
CablePluggedIn	Cable is plugged in and EVDetected.
ChargingRateChanged	Rate of charging changed by more than MaxLimitChangedSkipPercentage, or next period in charging schedule
ChargingStateChanged	Charging State changed.
Deauthorized	The transaction was stopped because of the authorization status in the response to a transactionEventRequest.
EnergyLimitReached	Maximum energy of charging reached. For example: in a pre-paid charging solution
EVCommunicationLost	Communication with EV lost, for example: cable disconnected.
EVConnectTimeout	EV not connected before the connection is timed out.
MeterValueClock	Needed to send a clock aligned meter value
MeterValuePeriodic	Needed to send a periodic meter value
TimeLimitReached	Maximum time of charging reached. For example: in a pre-paid charging solution
Trigger	Requested by the CSMS via a TriggerMessageRequest.
UnlockCommand	CSMS sent an Unlock Connector command.
StopAuthorized	An EV Driver has been authorized to stop charging. For example: By swiping an RFID card.
EVDeparted	EV departed. For example: When a departing EV triggers a parking bay detector.
EVDetected	EV detected. For example: When an arriving EV triggers a parking bay detector.
RemoteStop	A RequestStopTransactionRequest has been sent.
RemoteStart	A RequestStartTransactionRequest has been sent.

Tabela 3 - Tabela de TriggerReason [71]

Os últimos dois parâmetros que são obrigatórios no envio da mensagem de TransactionEvent são o *timestamp* e os dados da transação [71]. Neste campo de dados da transação, o carregador tem que colocar obrigatoriamente o *transactionID,* que nesta versão 2.0 é sempre gerado pelo carregador. Os outros campos dizem respeito ao estado da transação, a razão de esta ter parado, a duração e o *remoteStartID*, se a transação foi iniciada remotamente pelo sistema central [73].

A possibilidade de começar transações remotamente continua a ser uma realidade nesta nova versão do protocolo. A principal diferença é o nome das mensagens para começar e terminar as transações remotamente. A RequestStartTransactionRequest e a RequestStopTransactionRequest são as mensagens que o sistema central tem que enviar para iniciar e terminar transações remotamente [74]. O carregador responde com as mensagens RequestStartTransactionResponse e RequestStopTransactionResponse, mas

os campos a preencher são semelhantes aos existentes nas mensagens de RemoteStart.req, RemoStart.conf, RemoteStop.req e RemoteStop.conf do OCPP1.6 [75].

A mensagem de StatusNotification continua a existir no OCPP2.0, mas é enviada em menos ocasiões. O protocolo define que esta mensagem só deve ser enviada quando não existe nenhuma transação a decorrer no conector especificado [71]. Os estados *Preparing, Charging* e *Finishing* são reportados pelas mensagens de TransactionEvent. Em suma, a mensagem de StatusNotification é enviada para reportar os estados *Available, Occupied, Faulted, Reserved* e *Unavailable* à semelhança do que acontecia no OCPP1.5 [76].

Na Figura 26 é apresentado um exemplo de um diagrama temporal da realização de uma transação iniciada remotamente pelo utilizador [74]. Para completar o caso de uso, na Figura 27 é possível visualizar como deverá decorrer a troca de mensagens para o término de uma transação remotamente [74].

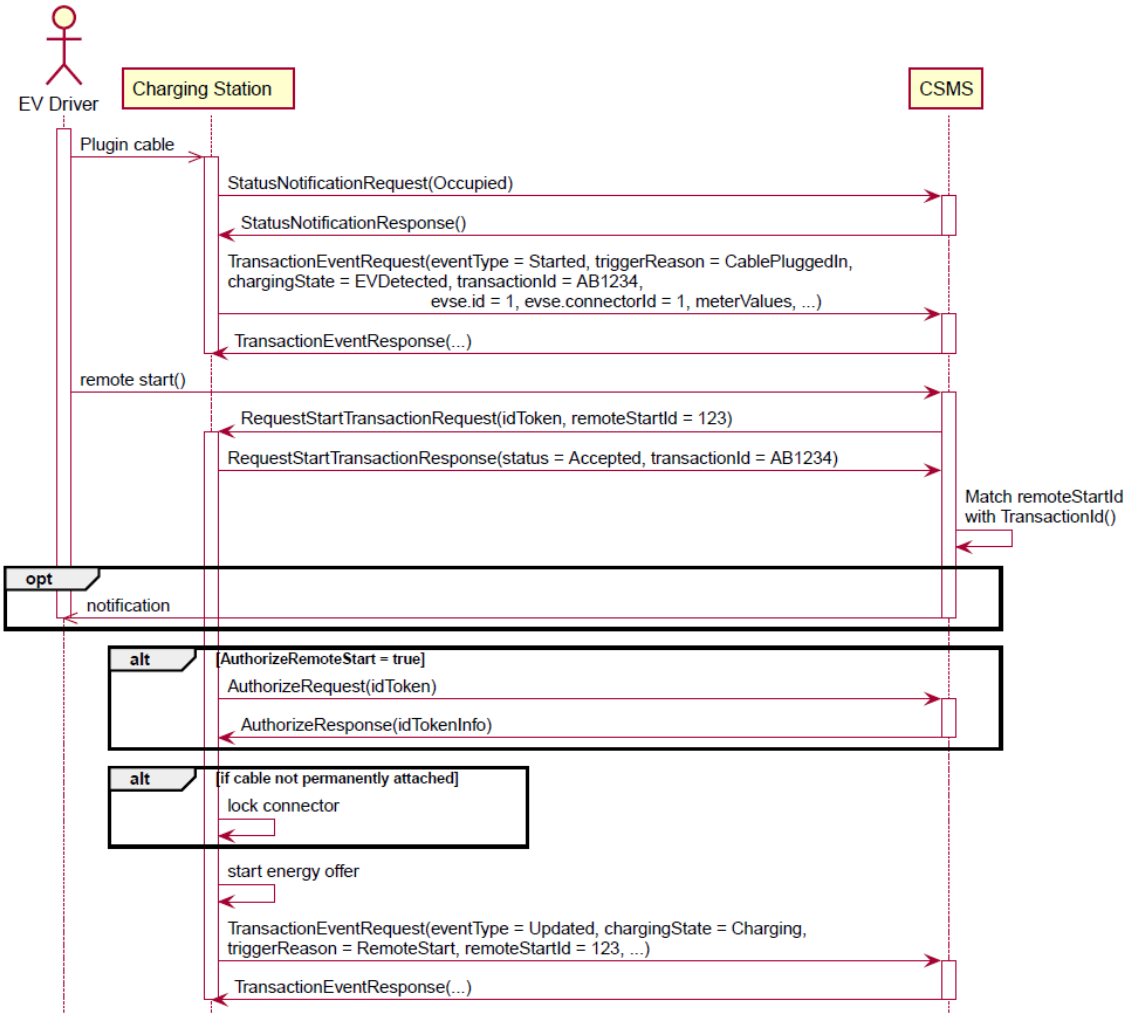


Figura 26 - Sequência de mensagens numa transação iniciada remotamente [73]

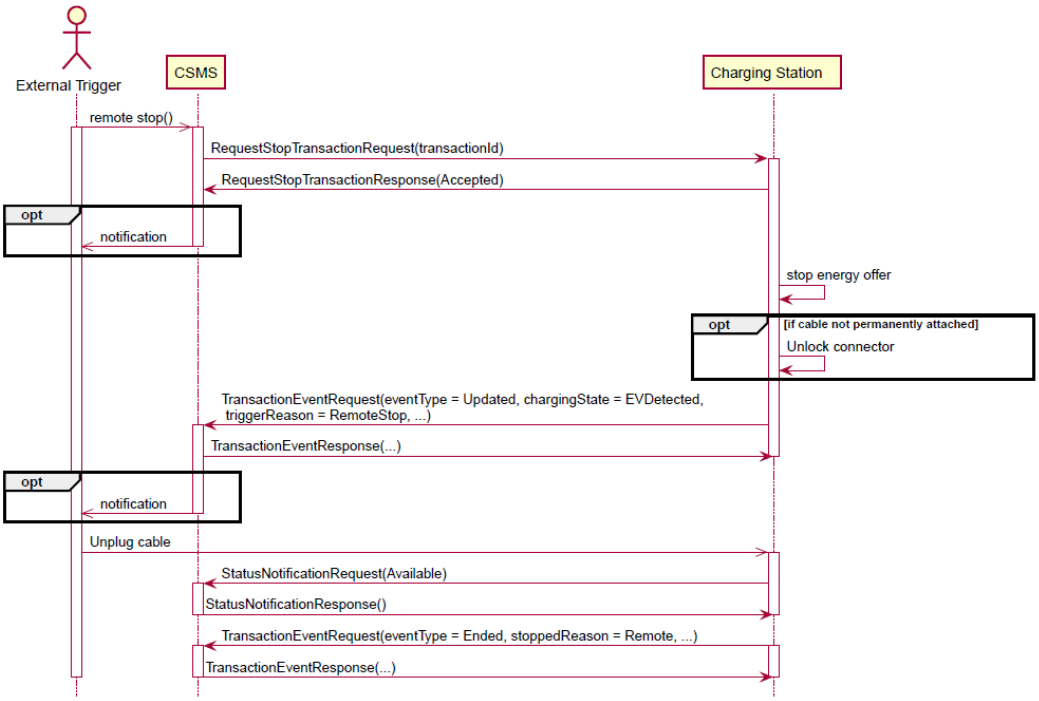


Figura 27 - Sequência de mensagens numa transação parada remotamente [73]

3.3.3. SMARTCHARGING

Relativamente ao *SmartCharging* apresentado no OCPP1.6, as diferenças no comportamento que a funcionalidade deve ter são praticamente inexistentes. A nova versão do protocolo prevê apenas novas formas de implementar o *SmartCharging.*

Enquanto que na versão 1.6 o OCPP previa que os dois únicos intervenientes no *SmartCharging* seriam o carregador e o sistema central, nesta nova versão são previstas duas novas topologias que podem ser implementadas. A primeira possibilidade prevê que possa existir um controlador local entre o sistema central e um ou vários carregadores, como ilustrado na Figura 28. De forma geral, o sistema central define um perfil máximo para aquele local e envia essa informação para o controlador local que, por sua vez, irá gerir o consumo de energia dos carregadores a si conectados [77].

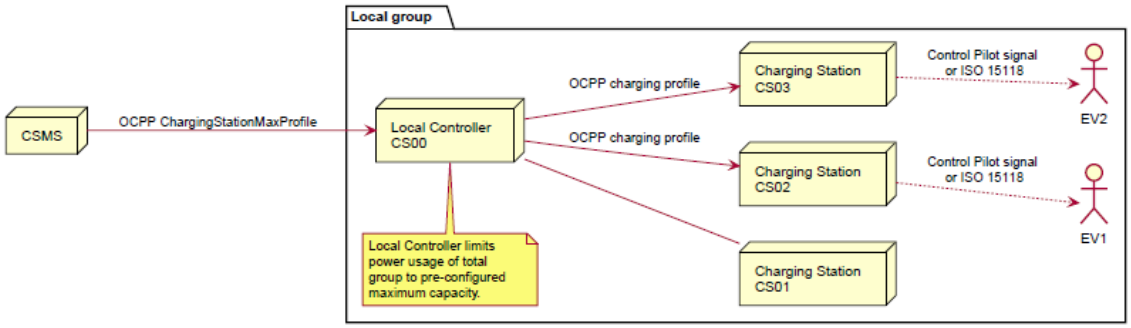


Figura 28 - SmartCharging com controlador local [76]

A segunda nova abordagem apresentada pelo OCPP 2.0 tem por base a existência de um quarto interveniente, para além do controlador local. Este quarto interveniente é um sistema de gestão de energia (ou *Energy Mangement System,* EMS) que é responsável por gerir os consumos de energia em tempo real de um determinado local. O EMS pode comunicar com o controlador local ou com os carregadores diretamente, mas através de um protocolo diferente do OCPP. As limitações impostas pelo EMS sobrepõem-se a qualquer perfil de carregamento que o sistema central tenha enviado para aquele local. A Figura 29 apresenta o diagrama de blocos desta quarta topologia de *SmartChargins*, no caso em que o EMS comunica com o controlador local [77].

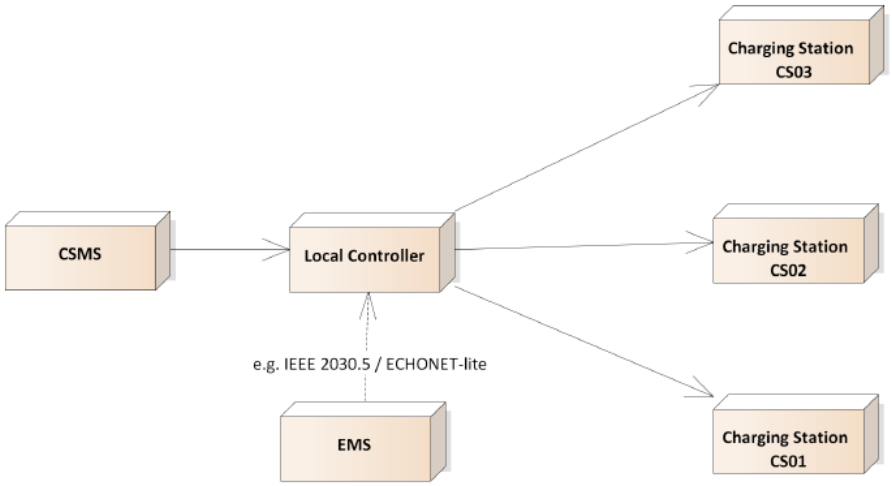


Figura 29 - SmartCharging com SEM [76]

Em termos de mensagens OCPP, a versão 2.0 não apresenta diferenças no que toca à gestão dos perfis de carregamento. O próprio perfil de carregamento enviado para o controlador local ou para o carregador é igual ao apresentado na versão 1.6. A principal

diferença surge quando é necessário a articulação do *SmartCharging* com a ISO 15118, tema que será abordado na secção seguinte.

3.3.4. ISO 15118: PLUG&CHARGE E SMARTCHARGING

A ISO 15118 é um protocolo de comunicação com objetivo de aumentar a interoperabilidade entre o VE, o carregador e a rede energética. Com o crescimento acentuado das redes de carregamento, o impacto na rede elétrica será significativo. Portanto, o principal objetivo da norma é criar um mecanismo que concilie a proteção da rede elétrica com a eficácia do carregamento de veículos elétricos [78].

No caso das redes de carregamento de VE é o sistema central que tem a capacidade de saber o estado da rede elétrica. O carregador tem então a capacidade de comunicar com o VE através da ISO 15118, o que não era possível no OCPP1.6. O OCPP 2.0 evoluiu nesse aspeto e já é possível ao carregador comunicar com o sistema central sobre eventos relacionados com a ISO 15118.

Relacionando a ISO 15118 e o OCPP 2.0 é possível criar um mecanismo de *SmartCharging* dinâmico. Até então, o *SmartCharging* permitia atribuir perfis de carregamento ao carregador ou conector, mas apenas baseados em previsões ou na capacidade limite da rede elétrica no local. Com a interligação da norma com a versão 2.0 do OCPP passa a ser possível adaptar o carregamento ao VE.

3.3.4.1. SMARTCHARGING

Neste contexto o OCPP 2.0 apresenta novas mensagens criadas especificamente para lidar com a ISO 15118 na parte de carregamento inteligente. Estas mensagens tentam estabelecer um perfil de carregamento adaptado a cada veículo.

A partir do momento em que o conector é colocado no VE, o carregador e o VE começam a comunicar. É nesta altura que o veículo informa o carregador das suas características de carregamento, ou seja, carregamento AC mono ou trifásico, ou DC. O veículo calcula também a energia que necessita para efetuar o carregamento, quanto tempo necessita, quais os valores máximo e mínimo de potência/corrente que consegue atingir. Após este cálculo, o veículo propõe um perfil de carregamento e envia-o para o carregador. Com

estes dados, o carregador envia a mensagem NotifyEVChargingNeedsRequest para o sistema central informando as características do carregamento e a mensagem NotifyEVChargingScheduleRequest com o perfil proposto pelo VE. O sistema central informa o carregador se conseguiu processar a informação e pode enviar a mensagem NotifyCentralChargingNeedsRequest que informa qual a potência máxima que aquele carregamento específico pode atingir enquanto decorrer [79] [80].

Adicionalmente, esta funcionalidade dinâmica permite que possa haver uma renegociação do perfil de carregamento durante a transação. O sistema central pode iniciar este processo enviando a mensagem Renegotiate15118ScheduleRequest. O carregador ao receber esta mensagem, comunica ao veículo que é necessária uma renegociação e este então recalcula os parâmetros para o carregamento. Esta informação é passada ao carregador e este, através das mensagens NotifyEVChargingNeedsRequest e NotifyEVChargingScheduleRequest, volta a informar o sistema central do que o VE ainda necessita para concluir o carregamento [79] [80].

A Figura 30 apresenta a troca de mensagens entre o VE, o carregador e o sistema central que coloca em funcionamento o *SmartCharging* através da ISO 15118 [79].

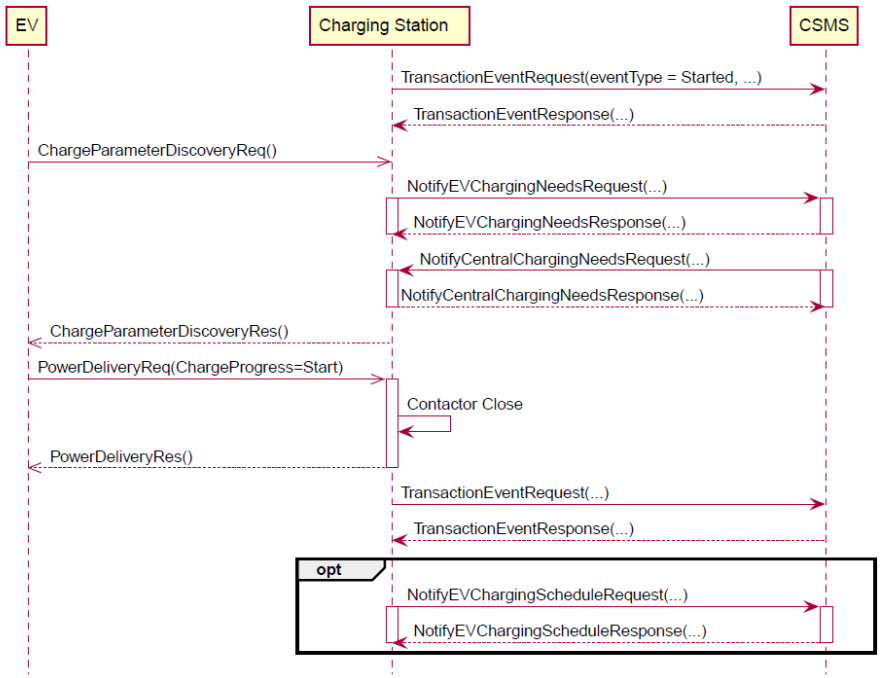


Figura 30 - ISO 15118 SmartCharging [78]

3.3.4.2. PLUG&CHARGE

A ISO 15118 prevê que tanto o VE como o carregador possuam controladores de comunicação, denominados, respetivamente, de *Electric Vehicle Communication Controller* (EVCC) e *Supply Equipment Communication Controller* (SECC). Estes controladores de comunicação, por sua vez, têm números de identificação únicos (EVCCID e SECCID) que podem ser utilizados para identificar o VE ou o carregador. A partir do momento em que o conector é colocado no veículo, o carregador pode pedir o seu certificado que contém toda a informação necessária para a sua identificação, da qual faz parte o EVCCID. Com esta informação pode ser colocado em funcionamento o *Plug&Charge* [81].

O *Plug&Charge* consiste na possibilidade de efetuar o carregamento apenas colocando o conector no veículo, i.e., todo o processo de autorização, carregamento e cobrança é realizado automaticamente. O carregador utiliza o certificado do VE para identificação e pergunta ao sistema central se o certificado tem autorização para iniciar uma transação. Depois de reconhecido o certificado, o sistema central autoriza o carregamento e a transação decorre de forma normal. Se o certificado do veículo estiver num repositório comum a diferentes redes de carregamento, o utilizador não necessita de ter uma etiqueta RFID para cada rede pois o seu veículo irá ser identificado de forma transversal.

Em suma, os principais objetivos do *Plug&Charge* são aumentar a comodidade no carregamento pois o utilizador apenas tem de colocar o conector e o carregamento é processado automaticamente. O segundo ponto é o facto de o utilizador poder carregar o seu veículo independentemente da rede de carregamento disponível.

Com o OCPP 2.0 o carregador, depois de obter o certificado do VE, apenas tem que enviar uma mensagem de AuthorizeRequest para o sistema central e este responde se o veículo pode carregar ou não. A Figura 31 demonstra a troca de mensagens entre o VE, o carregador e o sistema central [82].

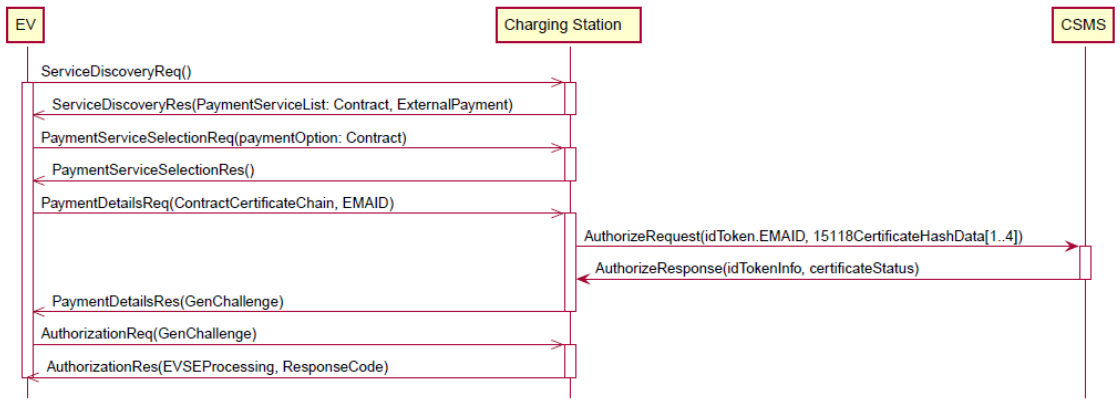


Figura 31 - Mensagens referentes ao Plug&Charge [81]

3.4. FERRAMENTA DE TESTES DA OCA E A BIBLIOTECA DE OCPP EM LINGUAGEM PYTHON

A organização que criou o OCPP, a OCA, no sentido de sistematizar o processo de validação de implantações do protocolo desenvolveu uma aplicação que permite realizar testes a carregadores de VE [83].

Esta ferramenta é capaz de realizar sequências de testes a todas as versões do protocolo e assim, com base nos resultados, os fabricantes dos carregadores podem afirmar que estão certificados pela OCA. Desta forma, os carregadores e os sistemas centrais ao cumprirem com os testes disponibilizados pela OCA tornam a probabilidade da existência de erros protocolares cada vez menor.

Por outro lado, a Efacec Electric Mobility S.A. adquiriu esta ferramenta e foi com base na sua utilização que surgiu parte da ideia para o presente projeto. Embora os testes realizados pela ferramenta da OCA permitam verificar o bom funcionamento do carregador no que toca ao protocolo, esta não é capaz de realizar testes que possam gerar erros numa marca de carregadores específica. Adicionalmente, também não é possível adicionar novos testes nem aceder ao seu código fonte, pelo que a utilização desta ferramenta fica limitada aos testes idealizados pela OCA.

Para colmatar as limitações identificadas na aplicação da OCA, este projeto propõe-se a desenvolver uma aplicação alternativa que permita a criação de testes específicos para os

carregadores da Efacec Electric Mobility S.A.. Tendo em consideração este objetivo geral foi identificada uma recente biblioteca de OCPP [84] desenvolvida para a linguagem Python.

A versão alfa desta biblioteca que implementa o protocolo em Python surgiu a fevereiro de 2019 e encontra-se, atualmente, na versão 0.6.4. A biblioteca apresenta então a capacidade de criar ambos os intervenientes de uma troca de mensagens OCPP, mas apenas suporta as versões 1.6 e 2.0 do protocolo. Adicionalmente, apenas é possível implementar o OCPP1.6 em formato JSON utilizando WebSockets, o que resulta numa limitação adicional. Mesmo considerando estas limitações, a utilização da biblioteca é uma mais valia devido às funcionalidades implementadas.

3.5. APRESENTAÇÃO DO FUNCIONAMENTO GERAL DA BIBLIOTECA

A biblioteca utilizada como suporte ao projeto é de acesso livre, estando disponíveis exemplos e documentação da sua utilização [85]. A biblioteca pode ser usada quer no sistema central quer no carregador, mas, nesta dissertação, o foco está na sua utilização do lado do sistema central.

O sistema central comunica com os carregadores através de WebSockets, assumindo o papel de servidor. Sempre que é estabelecida uma ligação é criada uma instância da classe *ChargePoint*. Este objeto é usado para gerir a comunicação, sendo através dos seus métodos que a biblioteca deteta a receção de mensagens, verifica se estas cumprem o formato correto e filtra-as consoante o tipo de mensagem e a sua ação. Este objeto é identificado pelo *StationID* do carregador que se liga e pelo WebSocket criado.

A biblioteca permite a criação das mensagens OCPP de uma forma direta, sem ser necessário pensar no seu formato JSON. Para tal, a biblioteca disponibiliza classes com métodos para criar cada mensagem CALL e CALLRESULT e preencher o seu *payload* quando são chamadas.

A partir deste ponto é possível enviar mensagens para o carregador e verificar a sua resposta. Deste modo, conseguindo saber se as mensagens recebidas estão bem

construídas, se a ação é a esperada e o conteúdo o correto, cria um ambiente ideal para um cenário de testes.

Embora o exemplo fornecido pelos criadores da biblioteca permita criar a base necessária para uma troca de mensagens OCPP, é necessário implementar a forma de enviar as mensagens CALL e CALLRESULT, assim como o preenchimento do *payload* das mesmas. Esta implementação é apresentada nesta dissertação, no Capítulo 4.

3.6. DISCUSSÃO

O OCPP, apresentado neste capítulo, revela-se abrangente e suficientemente versátil para que através dele possam ser criados múltiplos casos de uso dos carregadores. No entanto a sua extensão e complexidade pode originar um aumento da probabilidade de existência de erros no que toca ao *software* que implementa o protocolo.

Os sistemas centrais (clientes de empresas como a Efacec Electric Mobility), para além de requererem que os carregadores inseridos nas suas redes consigam carregar os VE dos seus utilizadores, impõem que toda a informação sobre os carregamentos seja reportada. Esta informação é preciosa para os sistemas centrais pois é através dela que estes vão identificar os utilizadores que realizaram os carregamentos, saber quanta energia foi fornecida na sessão de carga ou quanto tempo decorreu e, por fim, cobrar uma determinada importância. Esta é então uma visão aproximada do modelo de negócio das entidades gestoras das redes de carregamento.

Do ponto de vista do utilizador, ao ser cobrado pelo carregamento, este exige o bom funcionamento do carregador em termos físicos, mas também em termos protocolares. Isto implica que o carregador tenha que conseguir lidar bem com as *tags* RFID em todas as suas transações e respeitar a sua configuração.

Embora o protocolo já vá na sua versão 2.0, verifica-se que em termos de implementações do protocolo em uso, a versão 1.6J ainda é a mais comum. Como a biblioteca de OCPP apresentada é compatível com a versão 1.6J, encontra-se desta forma criada a condição ideal para levar a cabo o projeto. A implementação da aplicação de

testes irá então usar a versão 1.6J do protocolo que, como foi apresentado, possui a mesma lógica de funcionamento da versão 1.5, mas com algumas novas introduções. Portanto, o conhecimento prévio de como o protocolo funciona, quais as mensagens existentes e quando são enviadas, é essencial para entender os testes que a aplicação deverá fazer para validar a implementação do OCPP num carregador. No capítulo seguinte, para além de apresentar como foi desenvolvida a aplicação, serão também apresentados alguns testes de exemplo. Os resultados obtidos após a colocação da aplicação em funcionamento em ambiente laboratorial dependem também da compreensão geral do OCPP.

4. IMPLEMENTAÇÃO DO PROJETO

A solução desenvolvida no decorrer deste projeto visa a automatização do processo de validação da implementação do OCPP nos carregadores de veículos elétricos produzidos na Efacec Electric Mobility S.A., empresa na qual este projeto foi desenvolvido. Nesse sentido, a aplicação informática desenvolvida simula algumas das funcionalidades dum sistema central. A necessidade desta aplicação, desenvolvida à medida das necessidades da empresa, derivou essencialmente do facto da ferramenta disponibilizada pela OCA não permitir parametrizar novos testes. Uma vez que o código fonte da mesma não é disponibilizado, a possibilidade de adaptar a ferramenta também ficou excluída.

Este capítulo aborda o desenvolvimento da aplicação, incluindo exemplos de utilização da biblioteca OCPP no desenvolvimento da mesma. Esta aplicação foi desenvolvida

utilizando a linguagem Python e faz uso da biblioteca OCPP [84]. É também apresentada a GUI desenvolvida para a aplicação.

Tendo em consideração que um dos objetivos da aplicação é permitir a realização de testes de validação de forma automática, uma das tarefas foi a definição da lógica de funcionamento dos mesmos. A aplicação permite o envio de mensagens OCPP isoladas e a realização de sequências de teste pré-definidas. As sequências de testes automáticos implementadas na aplicação baseiam-se em trocas de mensagens OCPP. Foram preparadas duas sequências de teste de modo a permitir verificar que os carregadores cumprem a lógica protocolar desejada. Para além da sequência de mensagens ter que cumprir a especificação do protocolo, a aplicação verifica também o formato e o conteúdo das mesmas.

4.1. ARQUITETURA DA APLICAÇÃO

A aplicação desenvolvida neste projeto divide-se em dois blocos principais. O primeiro bloco diz respeito ao servidor de Websockets e o segundo é a interface gráfica com o utilizador. A Figura 32 apresenta essa arquitetura de forma esquemática.

O servidor de WebSockets é criado com o objetivo de possibilitar o estabelecimento de uma ligação que permita uma comunicação *full-duplex* com o carregador. Depois de estabelecida a comunicação entre o carregador e a aplicação, estes podem começar a troca de mensagens OCPP. Por forma a respeitar o sincronismo característico do OCPP, o servidor utiliza a biblioteca asyncio para a implementação da espera pela resposta do carregador antes do envio de uma nova mensagem.

Este bloco da aplicação é iniciado numa nova *thread* e é através dos eventos gerados no contexto do mesmo que é desencadeado o envio de mensagens isoladas ou o início de sequências de testes.

A GUI desenvolvida para esta aplicação contempla dois modos de utilização, disponibilizadas em dois separadores. O primeiro separador permite o envio isolado de mensagens do OCPP1.6, excetuando as relativas ao *SmartCharging, TriggerMessage* e

DataTransfer. Neste modo, cabe ao utilizador decidir que mensagem deseja enviar e o seu conteúdo. O segundo separador dá acesso às sequências de teste.

Uma vez que a versão do OCPP atualmente mais utilizada nos carregadores da empresa é a OCPP1.6J, a aplicação foi desenvolvida apenas para esta versão.

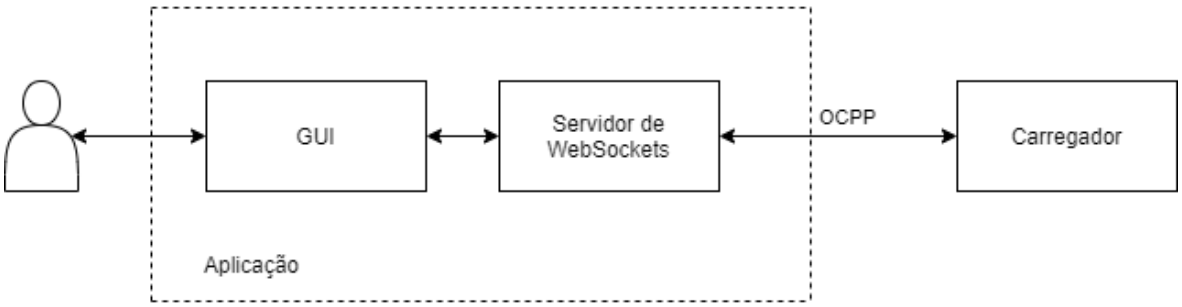


Figura 32 - Arquitetura da aplicação

4.2. DESENVOLVIMENTO DA GUI

A interface da aplicação foi desenvolvida com recurso à biblioteca PyQt5 [86] e à ferramenta QtDesigner [87]. Esta última permite desenhar o *layout* da GUI, inserir *widgets* e associar eventos aos mesmos. Após definido o *layout,* o QtDesigner, permite gerar o código Python que implementa a GUI. A partir deste ponto, associa-se aos eventos dos botões da GUI a função que trata do envio das mensagens OCPP, a par com a passagem dos dados que sejam necessários.

A GUI desenvolvida para esta aplicação contempla dois separadores. O primeiro separador, denominado de "Manual", disponibiliza um botão para o envio de cada uma das mensagens do OCPP1.6, excetuando as relativas ao *SmartCharging, TriggerMessage* e *DataTransfer.* Este separador, apresentado na Figura 33, é a vista de *default* quando a aplicação é iniciada.

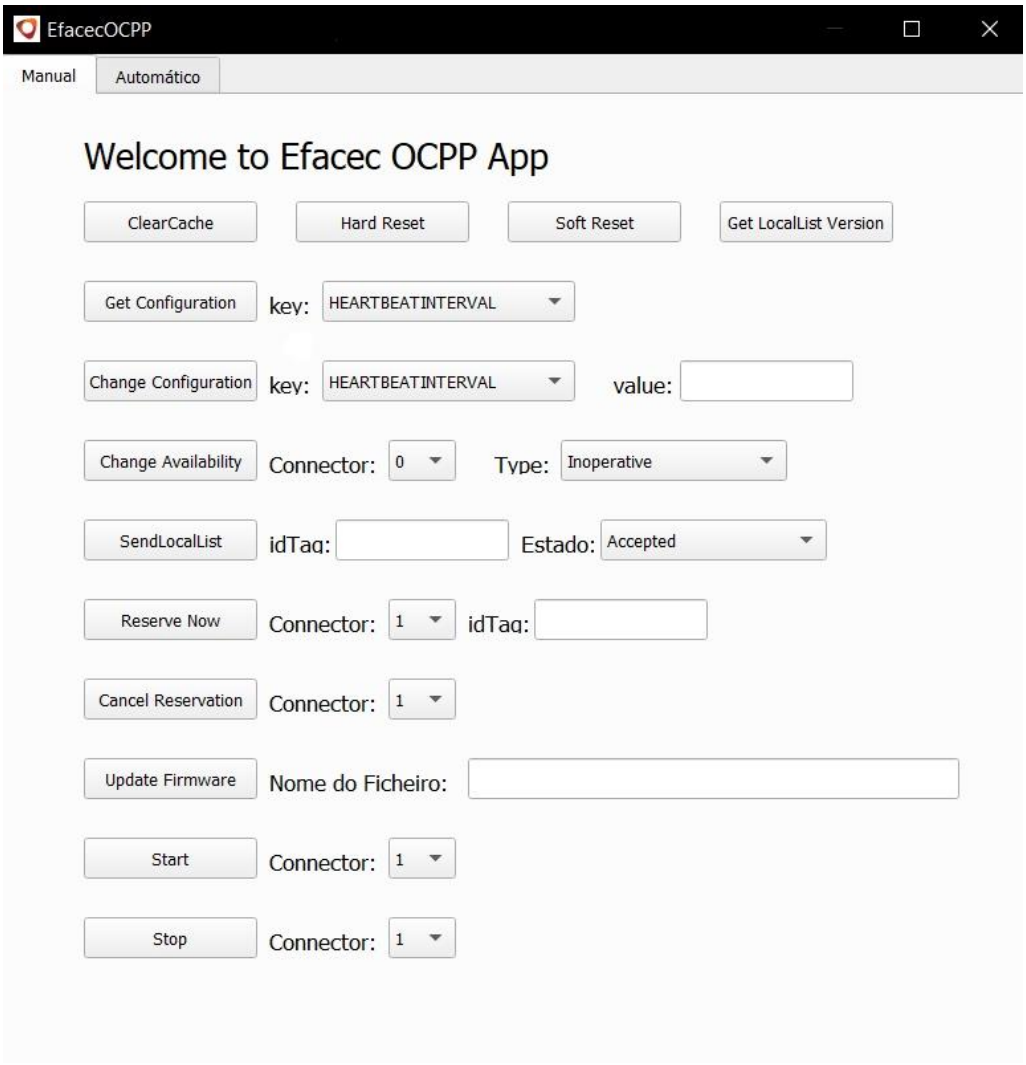


Figura 33 - Separador “Manual” da aplicação desenvolvida

O segundo separador, apresentado na Figura 34, surge com o intuito de permitir ao utilizador escolher qual a sequência de teste que deseja realizar. Para acionar um conjunto de testes a realizar deverá ser pressionado o botão “Start”. Se o utilizador desejar parar os testes, pode pressionar no botão “Stop”. Assim que a GUI é lançada, a aplicação também cria uma consola onde são apresentados dados pertinentes acerca das mensagens trocadas. Por exemplo, na Figura 35, é apresentada a consola com os dados enviados na mensagem *MeterValues*.

Figura 34 - Separador “Automático” da aplicação desenvolvida

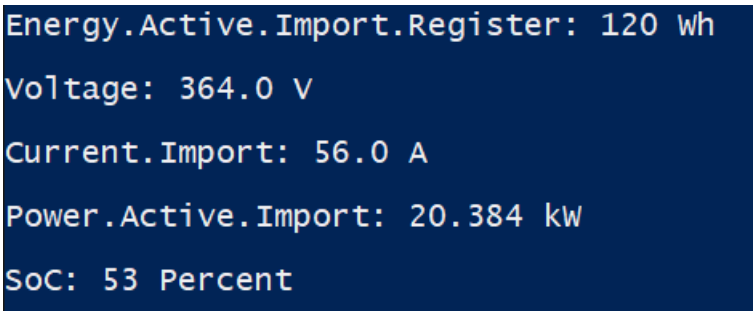


Figura 35 - Informação recolhida nas mensagens de MeterValues

4.3. INTEGRAÇÃO DA BIBLIOTECA OCPP

O envio e receção de mensagens é feito com base na biblioteca OCPP [84]. Esta biblioteca requer a utilização de código adicional para a implementação dos WebSockets, tendo sido utilizado para tal a biblioteca WebSockets do Python [88].

A utilização da biblioteca OCPP segue a lógica apresentada no fluxograma da Figura 36. A primeira etapa consiste na criação de um servidor de WebSockets associado a um determinado endereço porto TCP da máquina hospedeira. Este servidor simula o sistema central. Para o estabelecimento da comunicação por WebSockets, o cliente (carregador) especifica um endereço da forma “ws://ipaddr:port/*StationID*”, onde **ipaddr** é o endereço Internet Protocol (IP) do carregador, **port** é o respetivo porto TCP e *StationID* é o identificador do carregador no contexto do OCPP. Só depois de estabelecida a ligação

entre o servidor e um cliente, a aplicação passa a recorrer à biblioteca de OCPP. Nessa altura, é criada uma instância da classe *ChargePoint* da biblioteca, que fica associada à ligação WebSocket e ao StationID do carregador. A partir desse momento, o servidor fica à escuta de mensagens OCPP provenientes do(s) cliente(s).

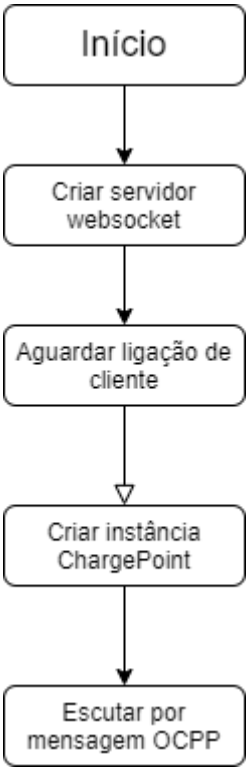


Figura 36 - Fluxograma do arranque de uma aplicação que use a biblioteca OCPP

4.4. INTEGRAÇÃO DO MÓDULO DE TROCA DE MENSAGENS COM A GUI

Após o desenvolvimento da GUI, foi necessário interligá-la com a parte da aplicação responsável pelo envio, receção e análise das mensagens OCPP.

A Figura 37 representa a sequência de arranque da aplicação, no que diz respeito à parte que trata do OCPP. Neste ponto, é adicionada uma ramificação ao fluxo de execução no sentido de criar a GUI aquando da ligação de um carregador à aplicação. Quando a conexão ocorre, e depois de criada a instância *ChargePoint*, a GUI é lançada numa *thread* que corre em simultâneo com a restante aplicação. Desta forma é possível à aplicação

escutar por possíveis mensagens provenientes do carregador e enviar mensagens, se for gerado algum evento através da GUI. Na Figura 37 é apresentada de forma gráfica a evolução da Figura 36 para acomodar este funcionamento da aplicação.

De notar que, se a ligação ao cliente se perder, a GUI é terminada e fechada automaticamente. A aplicação continua a esperar por novos clientes, sem ser necessário reiniciar. Aquando da ligação de um novo cliente o ciclo repete-se.

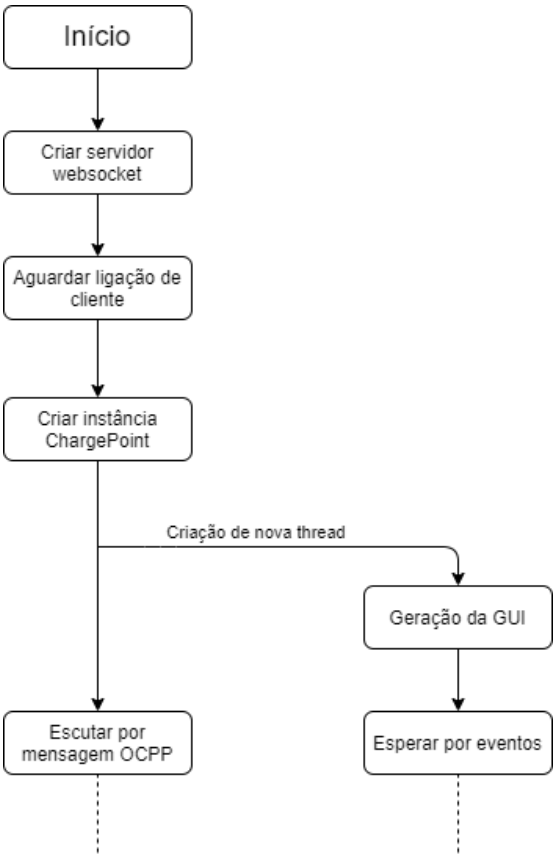


Figura 37 – Integração da GUI com o módulo de troca de mensagens

4.5. TRATAMENTO DAS MENSAGENS RECEBIDAS

Tendo em consideração o procedimento descrito na Secção 4.3, depois de ser criada a instância "*ChargePoint"* a aplicação fica à escuta de mensagens. A primeira tarefa após a receção de uma mensagem é decifrar se esta é do tipo CALL, CALLRESULT ou CALLERROR.

Caso a mensagem recebida seja do tipo CALL, o que significa que corresponde a um pedido iniciado pelo carregador, é necessário apurar qual a ação requerida pela mesma. A biblioteca decifra automaticamente qual a ação requerida, sendo assim possível interpretar os dados enviados pelo carregador e avançar para a criação da respetiva mensagem de resposta, do tipo CALLRESULT. A biblioteca tem um papel fulcral na construção das mensagens pois, para responder ao carregador, apenas é necessário o preenchimento dos campos definidos no protocolo. A biblioteca encarrega-se de formar a mensagem OCPP em formato JSON e de a enviar para o carregador. A Figura 38 apresenta o código relativo à construção de uma resposta à mensagem de BootNotification. Na linha 8 da figura é chamada da biblioteca OCPP a função responsável por criar e enviar a mensagem CALLRRESULT do tipo BootNotification. A esta função são passados por parâmetro o atual *timestamp*, o *HeartbeatInterval* desejado e o estado que indica se o carregador é aceite ou não. Como é possível verificar, a biblioteca tem um papel fulcral na construção e envio das mensagens, mas também, na linha 11, verifica-se que a biblioteca disponibiliza todos os estados de registo possíveis de preencher chamando a classe RegistrationStatus e escolhendo o parâmetro desejado.

```
1   @on(Action.BootNotification)
2      def on_boot_notitication(self, charge_point_vendor, charge_point_model, firmware_version, **kwargs):
3
4          print("Vendor: " + charge_point_vendor)
5          print("Model: " + charge_point_model)
6          print("Software Version: " + firmware_version)
7
8          return call_result.BootNotificationPayload(
9              current_time=datetime.utcnow().isoformat(timespec='milliseconds')+"Z",
10             interval=15,
11             status=RegistrationStatus.accepted
12         )
```

Figura 38 - Construção da mensagem de *BootNotification.conf*()

Se a mensagem for do tipo CALLRESULT, esta pode ser analisada. Note-se que, em condições normais estes tipos de mensagens só serão recebidos como resposta às mensagens do tipo CALL enviadas pela própria aplicação, tal como descrito na próxima secção. No exemplo da Figura 39, apresenta-se o código relativo ao tratamento da resposta a uma mensagem de *ClearCache*. Nesta situação é possível determinar se a resposta do carregador à mensagem enviada foi "*Accepted*" ou "*Rejected*". Esta é a metodologia utilizada no tratamento das mensagens de CALLRESULT, embora a abordagem seja diferente consoante a mensagem recebida.

```
response = await self.call(call.ClearCachePayload())

    if response.status == ClearCacheStatus.accepted:
            print("Clear Cache Accepted \r\n")

    else:
        if response.status == ClearCacheStatus.rejected:
                print("Clear Cache Rejected \r\n")
```

Figura 39 - Extrato de código de análise de uma mensagem CALLRESULT

A lógica utilizada no processamento das mensagens recebidas pela aplicação desenvolvida com o auxílio da biblioteca OCPP é apresentada no fluxograma da Figura 40. De notar que, depois de uma mensagem ser analisada a aplicação fica novamente a aguardar pela chegada de novas mensagens OCPP.

Se a mensagem recebida for do tipo CALLERROR, esta é imediatamente descartada porque significa que houve um erro do lado do cliente.

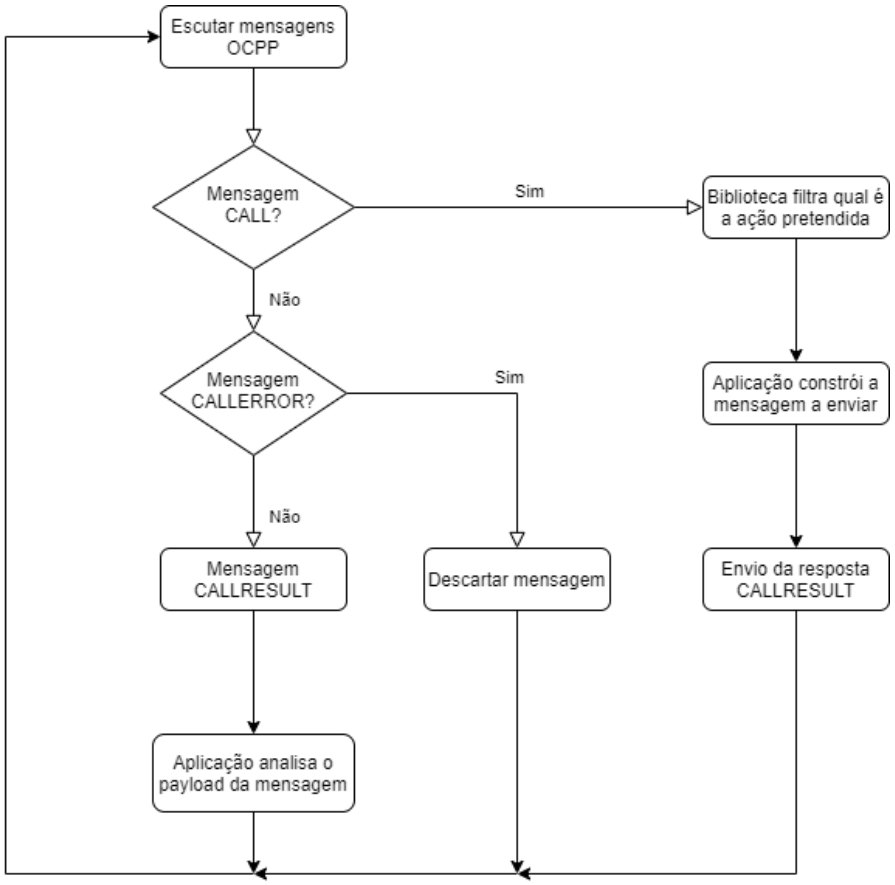


Figura 40 - Fluxograma do tratamento das mensagens recebidas

4.6. ENVIO DE MENSAGENS DO TIPO CALL

Em condições normais, para que a aplicação receba uma mensagem do tipo CALLRESULT, primeiramente tem que ser enviada uma mensagem do tipo CALL. Mais uma vez, a aplicação recorre à biblioteca OCPP para a construção e envio desse tipo de mensagem. Para implementar uma mensagem do tipo CALL é necessário chamar a função *send_call*(), que permite gerar mensagens deste tipo e preencher os campos necessários para o envio da mesma. Enquanto que na linha 8 da Figura 38 se observa que o método chamado para criar a mensagem provém de um módulo chamado "*call_result*", na Figura 41 o método utilizado pertence a um módulo denominado "*call*". Estes módulos agrupam as diferentes classes responsáveis por gerar os seus respetivos *payloads*. Na Figura 41, como exemplo, é gerada a mensagem *Reset.req*(), escolhendo o tipo *Soft* e, portando, requer ao carregador que realize um *soft reset.*

Como referido na Secção 3.2.2, associada a uma mensagem de *request* deverá existir sempre uma de *confirmation*. Mais especificamente, nenhuma nova mensagem CALL deverá ser enviada até que seja recebido o respetivo CALLRESULT ou CALLERROR de uma qualquer mensagem CALL anteriormente enviada (comportamento síncrono). Para cumprir com este aspeto do protocolo, depois de uma mensagem CALL ser enviada a biblioteca bloqueia o envio de novas mensagens desse tipo. Se uma mensagem CALLRESULT for entretanto recebida, o sistema comporta-se de acordo com o abordado no ponto 4.5. O sistema tem um comportamento diferente quando essa mensagem não é recebida dentro de um tempo limite. Se este limite de tempo for atingido, é assumido que existe um problema com a ligação e o WebSocket é quebrado, ficando a aplicação novamente à espera de que um carregador se conecte. Esta lógica é apresentada de forma esquemática no fluxograma da Figura 42.

```
async def send_call(self, **kwargs):

    await self.call(call.ResetPayload(
                        type="Soft"
                    ))
```

Figura 41 - Excerto de código da construção de uma mensagem CALL

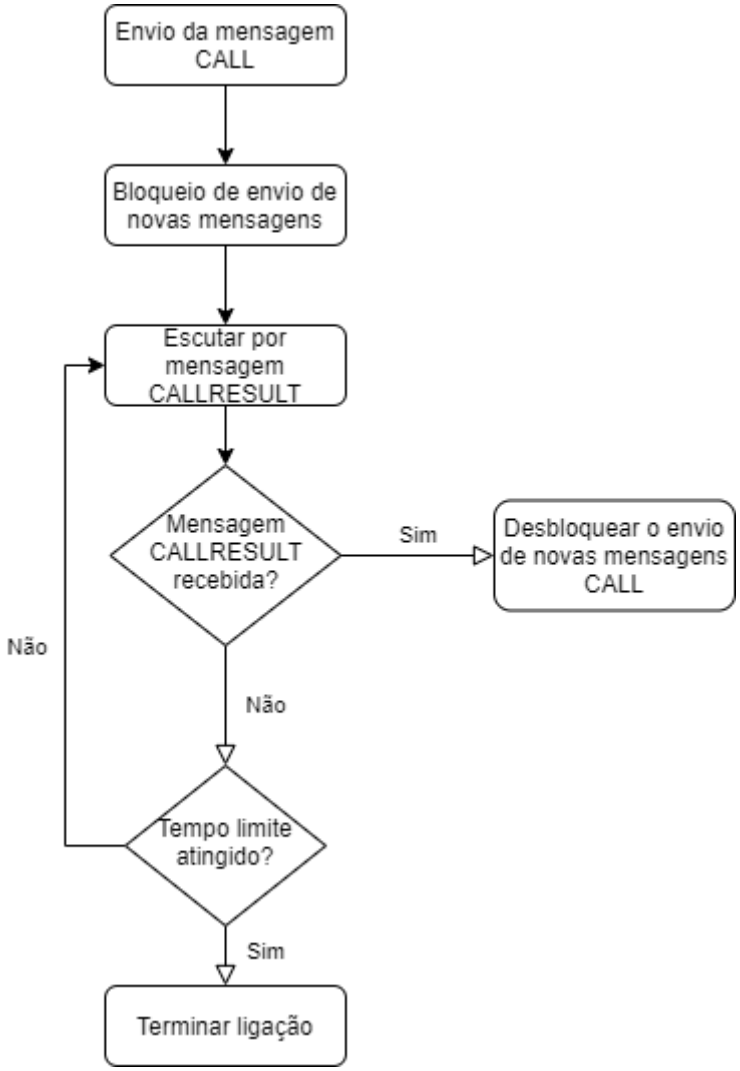


Figura 42 - Fluxograma do envio de uma mensagem CALL

4.7. PRINCÍPIO DE FUNCIONAMENTO DOS TESTES AUTOMÁTICOS

Dos dois separadores implementados na GUI, o segundo é aquele que permite iniciar os testes automáticos. O utilizador pode escolher uma das sequências de testes pré-definidas e pressionar o botão “Start”.

Após o utilizador dar início aos testes, a aplicação encarrega-se de enviar as mensagens OCPP e esperar pelas respostas enviadas pelo carregador. A aplicação apenas envia a

mensagem seguinte após ter recebido a resposta à mensagem anterior. Portanto, cada envio de mensagem é um teste ao que o carregador é capaz de processar ao receber determinada mensagem CALL.

Para além do facto de cada mensagem CALL ser apenas enviada após a receção de uma CALLRESULT, existe um intervalo de tempo entre cada envio de mensagens. Como as mensagens de *Heartbeat* são periódicas, a abordagem utilizada foi enviar uma mensagem CALL após a receção de uma mensagem *Heartbeat.req*() e do envio da resposta. Desta forma, para além de não ser necessário criar um temporizador na aplicação, é testado se o carregador cumpre com as mensagens de *Heartbeat.* O carregador deve respeitar o valor de *HeartbeatInterval* definido na resposta à mensagem de *BootNotification.*

A Figura 43 apresenta a lógica implementada para a realização dos testes automáticos. A partir do momento em que é acionado o começo dos testes, a aplicação espera o próximo *Heartbeat* e envia a primeira mensagem da sequência de teste. De seguida é esperada a resposta e analisado o conteúdo para que se possa concluir se o comportamento está correto ou não. O contador de mensagens enviadas é incrementado e testado com o intuito de saber se o teste chegou ao fim. Caso não seja recebida resposta dentro de um intervalo de tempo pré-definido, o teste será terminado.

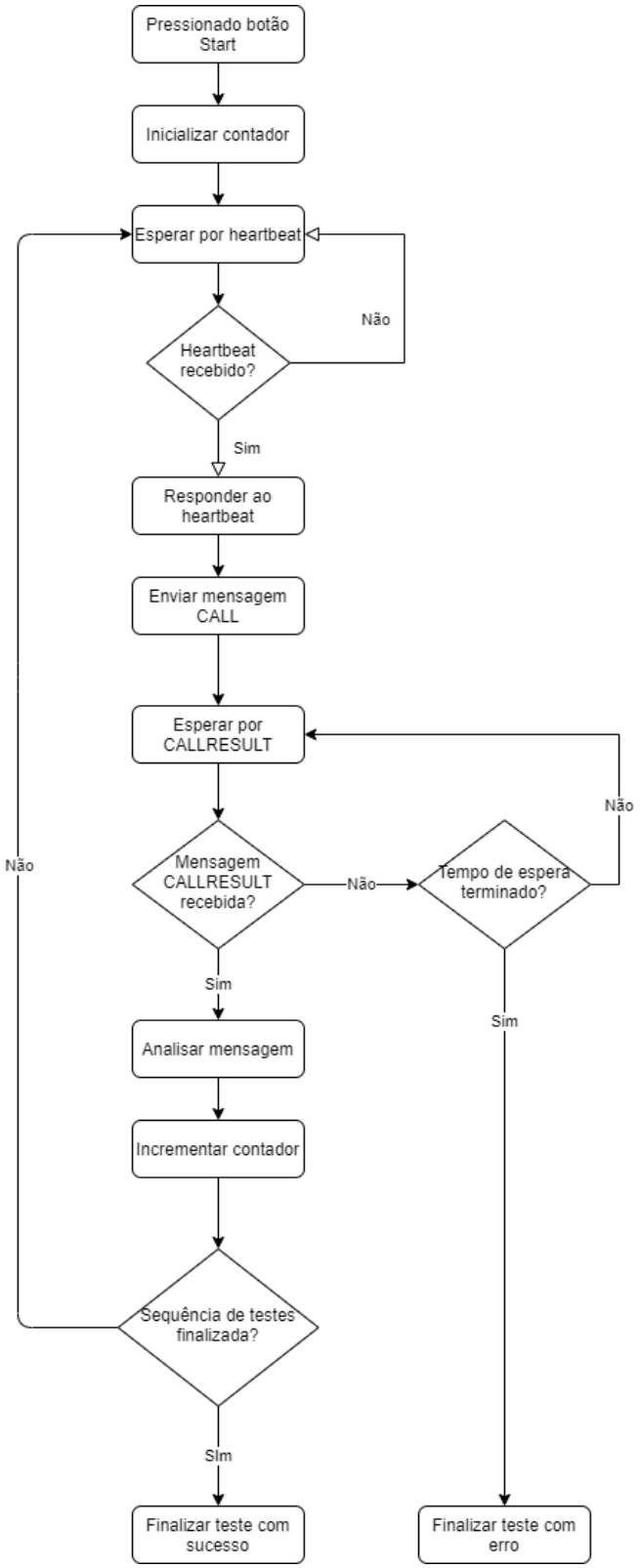


Figura 43 - Implementação dos testes automáticos

4.7.1. ANÁLISE DAS RESPOSTAS DO CARREGADOR

A Figura 43 apresenta o bloco denominado de "Analisar Mensagem" que representa a etapa onde é testado o comportamento do carregador. De maneira geral, as sequências de testes vão criar situações de uso especificas que vão obrigar o carregador a enviar determinadas mensagens, no tempo correto e com o conteúdo e formato protocolado.

Para conseguir verificar todos os pontos necessários durante os testes, foram idealizados três níveis de verificação que são chamados a cada receção de uma mensagem. O primeiro nível trata de observar a trama recebida e verificar se o que o carregador enviou respeita as regras definidas no protocolo. Neste caso, a trama tem que possuir a identificação se é uma mensagem CALL, CALLRESULT ou CALERROR, o seu *MessageID,* o campo *Action* e, por último, o *payload* da mensagem JSON (Figura 20 e Figura 22). É neste ponto que a biblioteca utilizada no projeto traz um novo benefício pois na receção das mensagens, se estas não tiverem o formato obrigatório ou o *MessageID* não estiver correto, a biblioteca envia imediatamente uma mensagem de CALLERROR para o carregador. No que se refere ao teste, caso a aplicação receba uma mensagem mal construída termina o teste e gera um aviso para o utilizador com o motivo do teste ter terminado.

O segundo nível de verificação diz respeito ao teste que está a ser realizado. Durante a sequência de teste é esperado que o carregador respeite o protocolo e envie certas mensagens nos *timings* corretos. Aquando da idealização da sequência de teste é definido a sequência de mensagens que o carregador deverá enviar obrigatoriamente. Portanto, após uma mensagem ser recebida e passar no primeiro nível de verificação, é analisado qual a mensagem recebida e se era esta que o teste esperava. Logicamente, ao ser a mensagem que o teste esperava, este continua para a terceira verificação e, caso contrário, a aplicação termina o teste e apresenta a razão ao utilizador.

Por último, a aplicação tem que conseguir obter o conteúdo do *payload* recebido na mensagem e analisá-lo. Neste sentido o terceiro nível de verificação confronta o que era esperado receber no *payload* da mensagem com o que realmente foi recebido. Ao estar dentro do que era esperado, o teste prossegue para a etapa seguinte. Caso contrário, o

teste é terminado e é apresentada a razão ao utilizador. No diagrama da Figura 44 são representados os três níveis de teste aplicados dentro do bloco “Analisar Mensagem”.

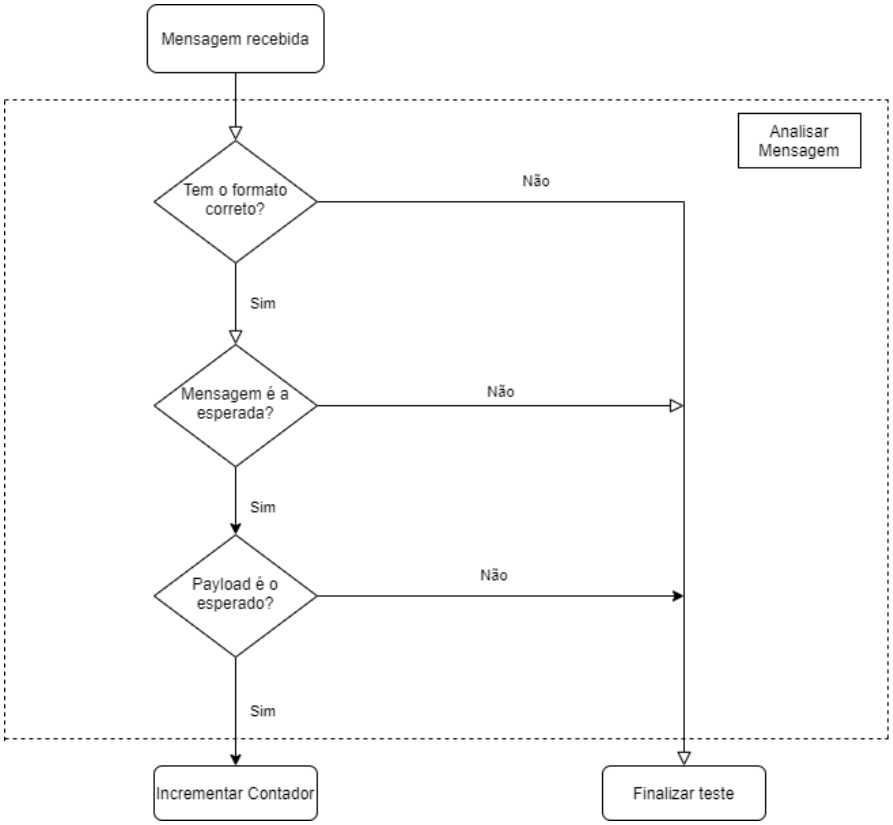


Figura 44 - Diagrama dos três níveis de teste implementados

4.8. ENVIO DE MENSAGENS PROPRIETÁRIAS

Os sistemas centrais podem trabalhar com mensagens específicas e que são enviadas no corpo das mensagens de DataTransfer. Esta troca de mensagens pode ser iniciada tanto pelo carregador, como pelo sistema central. É no caso em que a conversa é iniciada pelo sistema central que surge o benefício da aplicação.

A equipa de desenvolvimento da Efacec Electric Mobility S. A., por norma, para testar estas situações específicas consegue ligar-se ao sistema central e iniciar a comunicação OCPP. Porém, para verificar que o carregador responde corretamente a mensagens CALL de DataTransfer é necessário gerar o evento que desencadeia o seu envio. O entrave

surge então no facto de os sistemas centrais não disponibilizarem à equipa da Efacec a possibilidade de desencadear os eventos necessários através da sua página online.

Portanto, sabendo à partida quais as mensagens que têm que ser trocadas, associou-se um botão da GUI ao envio dessas mesmas mensagens de DataTransfer. Desta forma, simula-se o comportamento do sistema central que permite testar os desenvolvimentos realizados.

4.9. CRIAÇÃO DE SEQUÊNCIAS DE TESTES E SEU USO

Após a elaboração da lógica de funcionamento dos testes automáticos da aplicação, foram criadas sequências de testes do OCPP.

Uma sequência de teste do protocolo consiste no envio sequencial de mensagens especificas que, por sua vez, desencadeiam comportamentos no carregador o que permitirá validar a implementação do protocolo no carregador. As sequências de teste foram estudadas e elaboradas em conjunto com os engenheiros da Efacec Electric Mobility S.A.. Devido à extensão do protocolo, a sequência das mensagens a enviar pode ter múltiplas combinações, o que levou a que as sequências de teste tivessem duas finalidades distintas. A primeira procura testar a configuração do posto de carregamento e a segunda está orientada para testar as transações e estados do carregador.

Para além de ser definida a sequência de mensagens a enviar pela aplicação, também é necessário perceber quais são as respostas que o carregador deve enviar em determinado momento. A análise realizada às mensagens enviadas pelo carregador verifica qual a mensagem enviada, o preenchimento dos campos necessários e o seu conteúdo. Portanto, as sequências de testes têm que criar situações que testem a robustez do sistema, no que toca ao protocolo.

Nesta fase do projeto, todas as sequências de teste existentes na aplicação estão implementadas no código fonte. Essa é a razão pela qual a empresa requer acesso ao código e assim focar o desenvolvimento do projeto na implementação dos testes. Um

futuro acréscimo ao projeto será a criação de uma forma intuitiva de criação de novos testes.

4.9.1. SEQUÊNCIAS DE TESTES ORIENTADA À CONFIGURAÇÃO

O principal objetivo dos testes à configuração do carregador é verificar a robustez do sistema na sequência da alteração das suas variáveis. De forma geral, estas sequências de testes deverão criar situações resultantes de eventuais erros por parte do sistema central, às quais o carregador tem que conseguir reagir sem comprometer o seu normal funcionamento, isto é, sem entrar num estado não previsto.

São utilizadas principalmente duas mensagens do OCPP, a *GetConfiguration.req*() e a *ChangeConfiguration.req*(). A mensagem de *GetConfiguration.req*() permite obter a configuração de todas as variáveis definidas no carregador ou apenas as especificadas na mensagem. Por outro lado, a mensagem *ChangeConfiguration.req*() permite alterar a configuração do carregador. Logo, com a possibilidade de colocar qualquer conteúdo no corpo da mensagem é possível criar situações extremas com as quais o carregador terá que lidar.

De seguida, ilustra-se a lógica do teste usando uma das sequências implementadas. Após o utilizador pressionar o botão *Start*, a aplicação começa por enviar uma mensagem *GetConfiguration.req*() sem conteúdo, o que implica o carregador ter que enviar a configuração de todas as variáveis na mensagem de *GetConfiguration.conf*(). A mensagem seguinte é também uma mensagem de *GetConfiguration.req*(), mas desta feita especificando algumas variáveis (nomeadamente, “HeartbeatInterval” e “MeterValuesSampleInterval”), ao que o carregador tem que responder o comando e enviar apenas o conteúdo dessas mesmas variáveis. A aplicação é capaz de verificar automaticamente o preenchimento dos campos necessários, bem como o conteúdo da resposta do carregador e, se a resposta não estiver correta, gerar uma mensagem de erro.

O passo seguinte da sequência foca-se na mensagem de *ChangeConfiguration.req*(). Aqui é necessário sempre especificar qual a variável a modificar e o respetivo novo conteúdo. É na capacidade de enviar qualquer conteúdo para uma variável que surge a vantagem da

utilização da aplicação, no que toca a testes de configuração. Na sequência definida, são modificados três tipos de variáveis que diferem no conteúdo que cada uma deve aceitar. As variáveis utilizadas são: o *HeartbeatInterval,* que é um valor de tempo em segundos, o *Max.Power,* que diz respeito ao valor máximo de potência que o carregador pode atingir, em quilowatts (kW) e arredondado às unidades, e o *AuthorizeTxRequests* que aceita o conteúdo *True/False* e define se o carregador aceita ou não autorização remota para começar transações. A sequência de teste continua através do envio de mensagens de ChangeConfiguration para estas variáveis e testar a resposta do carregador.

Em primeiro lugar é testada a configuração da variável *HeartbeatInterval*. Para tal são enviadas sequencialmente quatro mensagens de *ChangeConfiguration* com o conteúdo 30, -30, 0 e 30.5. Como o *HeartbeatInterval* é uma variável que define o intervalo de tempo entre cada *Heartbeat* esta apenas pode ter valores superiores a 0. Adicionalmente, o protocolo especifica que o intervalo de tempo deve ser um valor inteiro. Logo, o carregador deve aceitar apenas a primeira mensagem de configuração, com o conteúdo 30. O teste à variável *Max.Power* é semelhante, pois também deve apenas aceitar valores inteiros, mas pode receber o valor 0. Então as mensagens ChangeConfiguration para a variável *Max.Power* terão como conteúdo 30, -30, 0 e 30.5 e o carregador deve responder às mensagens aceitando a primeira e terceira mensagem e rejeitando as restantes. Por fim, as mensagens ChangeConfiguration para a variável *AuthorizeTxRequests* terão o conteúdo *True, False*, *true, false* e "Treu". Como o conteúdo das mensagens OCPP não é *case sensitive*, o carregador deve aceitar o comando para as quatro primeiras mensagens e rejeitar a última com o erro ortográfico.

Em suma, esta sequência de mensagens testa a robustez do carregador a possíveis mensagens com erro enviadas pelo sistema central. Como a aplicação verifica se as mensagens estão bem construídas e inclusive o seu conteúdo, é possível concluir se o carregador faz as verificações esperadas das mensagens enviadas pelo sistema central. Desta forma, garante-se que o carregador não entrará em estados imprevistos na eventualidade de receber mensagens com erros da parte do sistema central.

4.9.2. SEQUÊNCIA DE TESTES FOCADA NAS TRANSAÇÕES

A segunda finalidade dos testes idealizados em conjunto com os engenheiros da Efacec Electric Mobility S.A. foca-se em torno das transações realizadas. Desde o momento em que um utilizador coloca um conector no seu veículo até o retirar, muitas informações têm que ser reportadas para o sistema central. Portanto, esta sequência de testes verifica principalmente a sequência de envio de determinadas mensagens e se o seu conteúdo é o esperado ao longo de uma transação.

As mensagens StatusNotification, StartTransaction, MeterValues e StopTransaction são as usadas mais frequentemente neste tipo de testes, embora numa perspetiva diferente dos testes de configuração. Estas mensagens são sempre iniciadas pelo carregador, logo foi necessário criar situações de teste que obrigam o carregador a enviá-las. Uma outra diferença deste tipo de testes para os de configuração é o facto de que, para além de testar a construção das mensagens e o seu conteúdo, é testado também a sequência de mensagens enviadas pelo carregador. Ou seja, ao criar as situações de teste, o carregador tem que seguir o envio de mensagens que o protocolo define.

De forma análoga à secção anterior, a lógica e funcionamento deste tipo de testes é exemplificada com base numa das sequências de teste implementadas.

Após o utilizador pressionar no botão *Start* da aplicação, esta envia uma mensagem de *RemoteStartTransaction.req*() para o carregador e define qual a etiqueta RFID e o *ConnectorID*. Após o envio desta mensagem é esperado que o carregador responda com uma mensagem de *RemoteStartTransaction.conf*() e que o pedido realizado tenha sido aceite. No seguimento deste pedido, é esperado que o carregador envie uma mensagem de *StatusNotification.req*() com o estado *Preparing*, à qual a aplicação responde. A partir deste ponto, um dos elementos da equipa de teste tem que colocar o conector no veículo. Esta ação deverá desencadear o envio, pelo carregador, da mensagem de *StartTransaction.req*() com a etiqueta RFID fornecida, o *ConnectorID*, o *meterStart* e o *timestamp.* A aplicação responde com a mensagem *StartTransaction.conf*(), aceitando o início da transação e definindo um *TransactionID*, e aguarda que o carregador reporte o estado *Charging* numa nova mensagem de StatusNotification. Com o decorrer da sessão de carga, o teste espera a receção de mensagens de MeterValues com o respetivo

ConnectorID e *TransactionID*, os valores da energia gasta e os valores instantâneos da potência, corrente, tensão e SoC. Depois de recebidos três mensagens de MeterValues, a aplicação envia uma mensagem de *RemoteStopTransaction.req*() para o *TransactionID* da transação que está a decorrer e espera que o carregador responda com a mensagem *RemoteStopTransaction.conf*() aceitando o comando enviado. O protocolo define que, ao aceitar este comando, o carregador tem que enviar uma mensagem de *StopTransaction.req*() referindo o *TransactionID, ConnectorID, meterStop* e *timestamp.* Enquanto o conector estiver colocado no veículo, o carregador tem que reportar o estado *Finishing* e só depois de retirado o conector é que o estado pode voltar a *Available*. A Figura 45 representa de forma esquemática a troca de mensagens realizada durante o decorrer desta sequência de teste. Uma vez mais, a aplicação verifica se esta é a sequência de mensagens e emite uma mensagem de erro em caso contrário. O erro também pode surgir se, por exemplo, o carregador não respeitar o *TransactionID* ou *ConnectorID* que foi atribuído à transação, ou se as mensagens não estiverem de acordo com o protocolo.

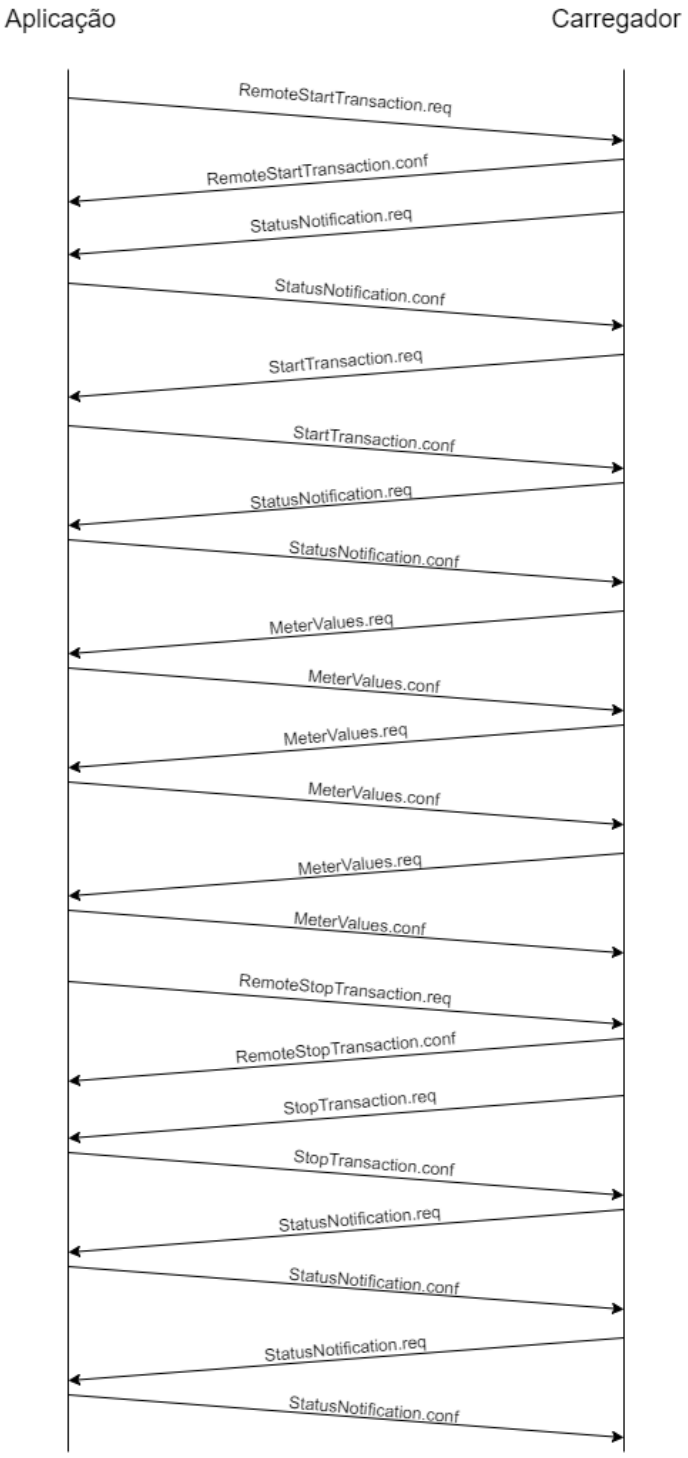


Figura 45 - Exemplo de sequência de testes focada nas transações

Uma segunda situação que pode surgir a partir de uma ligeira alteração ao teste anterior. Para isso é necessário utilizar a mensagem de ChangeAvailability. Esta mensagem tem o intuito de mudar a disponibilidade de um conector específico e possui uma particularidade caso seja enviada durante uma transação. Considerando o teste anterior,

antes de a aplicação mandar terminar a sessão de carga é enviada a mensagem *ChangeAvailability.req*() com o conteúdo *Inoperative* para o *ConnectorID* que está a realizar a transação. O protocolo, nesta situação, indica que o carregador não deve aceitar nem negar o comando enviado, mas sim responder *Scheduled* na mensagem *ChangeAvailability.conf*(). Significando isto que, depois de a transação terminar e o estado *Finishing* ser enviado, uma mensagem de StatusNotification com o estado *Unavailable* deve ser enviada para o *ConnectorID* requerido pela aplicação.

Com estes exemplos é percetível a diferença entre os dois tipos de testes idealizado pelos engenheiros da Efacec Electric Mobility S.A.. Esta diferenciação surge também do facto de o próprio protocolo criar mensagens com funcionalidades diferentes e com *timings* de funcionamento díspares de umas para as outras.

5. DEMONSTRAÇÃO DE RESULTADOS

Aquando da redação do presente documento a aplicação desenvolvida possui uma versão estável que cumpre os requisitos estipulados pela empresa. Esta encontra-se em produção na Efacec Electric Mobility S.A. para realizar testes aos carregadores, o que adicionalmente resulta em testes à robustez da própria aplicação. Os vários módulos da solução desenvolvida podem, contudo, ainda possuir erros de funcionamento e que desta forma poderão ser detetados e corrigidos no futuro.

Os resultados apresentados neste capítulo referem-se ao que foi, até ao momento, possível apurar durante os testes de validação realizados aos carregadores produzidos

pela empresa. Além disso, será também apresentado um caso de uso real, bem como serão discutidas mais-valias adicionais que foram identificadas durante os testes.

5.1. INSTALAÇÃO NO LABORATÓRIO DE ENSAIOS

O objetivo de apresentar este caso de uso, para além de descrever de como o responsável pelos testes deverá proceder para colocar em funcionamento o sistema, é também o de criar uma imagem mais clara de um cenário real do laboratório de ensaios da Efacec Electric Mobility S.A..

A Figura 46 apresenta um cenário normal de utilização da aplicação desenvolvida, com esta a funcionar num computador do laboratório. Nesta situação o carregador está a realizar um carregamento CCS ao veículo, o que pode ser verificado através do ecrã do equipamento de carregamento (canto superior esquerdo da imagem). À parte do carregador apenas é necessário um cabo *ethernet*, em termos de *hardware.* Para prosseguir é configurado o IP do computador, o *endpoint* ao qual o carregador se vai ligar e o respetivo *StationID*. Após o estabelecimento do WebSocket e o envio da mensagem de BootNotification, a aplicação está pronta a ser utilizada.

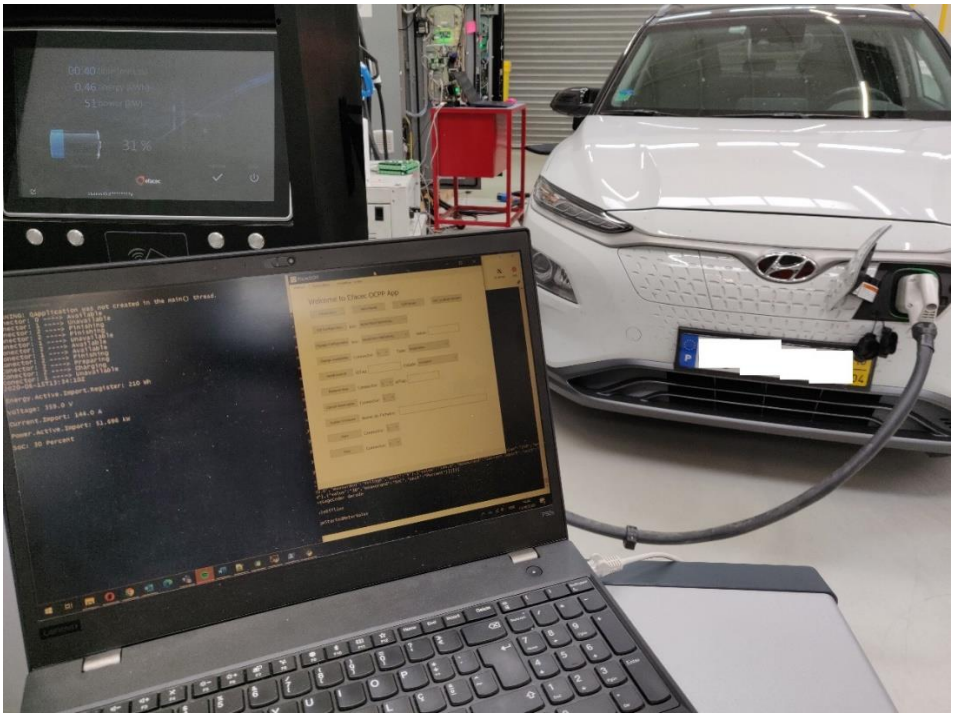


Figura 46 - Caso de uso real da aplicação desenvolvida

5.2. FALHAS ENCONTRADOS PELA APLICAÇÃO

A partir do momento em que a aplicação desenvolvida neste projeto passou a ser utilizada nos ensaios realizados aos carregadores em laboratório, esta já detetou algumas situações de erro na implementação do OCPP nos equipamentos produzidos pela Efacec Electric Mobility S.A..

5.2.1. FALHA NO TRATAMENTO DE MENSAGENS DANOSAS DE CHANGECONFIGURATION

No que aos testes de configuração diz respeito, a aplicação foi capaz de detetar falhas quando são forçados valores errados nas mensagens de *ChangeConfiguration.req*(). Os carregadores defendem-se ao rejeitar valores que possuam casas decimais, mas quando são enviados valores negativos o carregador aceita-os. Este erro provocava, por exemplo, a paragem no envio das mensagens *Heartbeat.req*(), caso o *ChangeConfiguration.req*() fosse enviado para a variável *HeartbeatInterval*. Comportamentos semelhantes aconteciam para variáveis que definissem tempo, provando que o sistema necessitava de ser trabalhado ao nível da robustez e proteção em caso de falha do sistema central.

5.2.2. GESTÃO ERRADA DO NÚMERO DE CASAS DECIMAIS DO *TIMESTAMP*

Um dos primeiros erros encontrados no sistema da Efacec Electric Mobility S.A. provocava a alteração das horas do sistema a cada mensagem de *Heartbeat.conf*() que a aplicação enviava como resposta ao *Heartbeat.req*(). Esta situação surgia, devido à mensagem *Heartbeat.conf*() possuir no seu conteúdo o *timestamp,* mas aquando da implementação da mensagem na aplicação os segundos foram arredondados a cinco casas decimais. Verificou-se que o carregador não estava preparado para receber este número de casas decimais e atualizava a sua hora para um valor errado. Posto isto, este erro resultou no envio da mensagem *StartTransaction.req*() com um *timestamp* superior ao enviado no *StopTransaction.req*() da mesma transação. O sistema central recebia a informação de que o término da carga teria ocorrido antes do início. O OCPP não define o número máximo de casas decimais que os segundos devem ter, logo o código no carregador tem que se melhorado para precaver tal situação.

5.2.3. MENSAGENS DE STATUSNOTIFICATION REPETIDAS

Devido ao elevado fluxo de mensagens que pode surgir durante uma transação e o facto do técnico que realiza os testes ter que estar atento a múltiplas situações enquanto a sessão de carga se desenrola, pormenores acerca do estado do carregador podem passar despercebidos.

A aplicação imprime na consola todos os estados que o carregador reporta durante os testes, com base nesta funcionalidade foi possível verificar que o carregador enviava mensagens de *StatusNotification.req*() repetidas. O OCPP define que as mensagens de StatusNotification devem apenas ser enviadas se o estado de um conector mudar. Portanto, nunca devem ser enviadas mensagens repetidas de *StatusNotification.req*().

5.2.4. GETCONFIGURATION NO ARRANQUE IMPEDE O ACESSO À BASE DE DADOS

O carregador para armazenar toda a sua configuração possui uma base de dados. Neste sentido foi concebido um teste com o intuito de enviar uma mensagem de *GetConfiguration.req*() vazia logo após a resposta ao BootNotification.

A mensagem de *GetConfiguration.req*() quando é enviada vazia requer que o carregador envie o conteúdo de todas as suas variáveis. Num sistema como os carregadores da Efacec Electric Mobility S.A., o número de variáveis pode ascender às dezenas e quando o levantamento de todas as variáveis foi realizado no arranque do mesmo, o *software* sofreu um erro e não conseguiu realizar o acesso à base de dados.

5.3. DESENVOLVIMENTO EM TELETRABALHO

Uma situação adversa abateu-se sobre a sociedade em geral durante o desenrolar deste projeto e a equipa da Efacec Electric Mobility S.A. que trata de desenvolver o protocolo, tal como todas as equipas em geral, teve que cingir as suas funções ao teletrabalho.

A equipa não estando preparada para uma situação destas e, embora pudesse desenvolver novas características nos carregadores, não teve como as testar. Este projeto surgiu no *timing* acertado pois, como a aplicação pode funcionar em qualquer

computador e também como um sistema central, a equipa de desenvolvimento pode utilizá-la para testar o bom funcionamento dessas novas características pois conseguem simular o carregador (com simuladores existentes na empresa) e o sistema central com a aplicação..

6. CONCLUSÃO

Tendo em conta o que foi apresentado ao longo desta dissertação conclui-se que o tema da mobilidade elétrica, nos dias que correm, é cada vez mais impactante. A quantidade de carregadores de veículos elétricos existentes na via pública é progressivamente maior e, devido ao seu modo de funcionamento, necessitam de sistemas de gestão específicos. Estes sistemas para a grande parte da sociedade são ainda algo desconhecido e, portanto, este projeto fez jus em apresentar uma perspetiva geral do seu comportamento e da sua utilização.

Por forma a que fosse percetível a lógica de funcionamento do OCPP, foi necessário abordar em primeiro lugar o tema da mobilidade elétrica. Como foi exposto, atualmente existem várias formas de os utilizadores de VE carregarem os veículos. A evolução da tecnologia levou a que existisse uma diferenciação entre carregamento AC e DC que, por

sua vez, resulta em valores de potência de carregamento diferentes. O desenvolvimento do carregamento rápido em DC proporcionou a oportunidade de se instalarem carregadores rápidos na via pública e assim formar redes de carregamento. As redes de carregamento rápido têm o principal objetivo de permitirem aos utilizadores de VE realizarem carregamentos mais rápidos e devido à sua abrangência permitir a realização de viagens com maior distância.

Como evidenciado ao longo do documento, os carregadores instalados nas redes de carregamento têm que ser desenvolvidos de forma a poderem funcionar de forma automática. Um carregador tem que ser capaz de realizar o carregamento de veículo propriamente dito, enviar a informação acerca dos carregamentos e reportar o seu estado de funcionamento. Para tal foi criado o OCPP, um protocolo que foi pensado para a troca de informação entre o carregador e o sistema que gere a rede de carregamento. Na versão 1.5, o protocolo criou o conjunto de mensagens necessárias para o controlo da rede de carregamento. Na versão 1.6 que lhe sucedeu, a maior alteração foi a possibilidade de enviar mensagens através de WebSockets e em formato JSON. O mecanismo de WebSockets permite adotar medidas de cibersegurança que até então não existiam (i.e., TLS). Em termos do protocolo, a maior novidade foi a introdução do *SmartCharging* por forma a ser possível controlar o impacto dos carregamentos na rede elétrica. A versão 2.0 do OCPP aumenta a quantidade de informação partilhada na rede e, no âmbito desta dissertação, procurou-se apenas demonstrar o futuro que se avizinha relativamente às redes de carregamento.

O OCPP desempenha um papel fulcral na gestão das redes de carregamento. Embora o protocolo seja impercetível para o utilizador comum, este é essencial para o bom funcionamento da rede. Por conseguinte, os carregadores instalados na via pública têm que possuir versões de *software* que cumpram o protocolo. É nessa perspetiva que a Efacec Electric Mobility S.A. decidiu propor este projeto, onde o principal objetivo incidia no desenvolvimento de uma solução que permitisse validar as suas implementações do OCPP instaladas nos carregadores que produz.

A aplicação resultante deste projeto cumpre os objetivos inicialmente definidos, tendo sido também identificadas mais-valias adicionais decorrentes da sua utilização. A

implementação final consegue usufruir das capacidades disponibilizadas pela biblioteca de OCPP e funciona como um sistema central no seu estado *default.* Quando acionados os testes automáticos, o técnico responsável pela sua realização apenas necessita de acompanhar a troca de mensagens e verificar o resultado final dos testes. Sem esta aplicação, teria que ser o responsável a dar ordem de envio de mensagens através do acesso a um sistema central, o que aumenta a probabilidade de erro no teste pois desvia o foco da atenção do técnico. Adicionalmente, para o responsável que está a realizar o teste o processo torna-se menos moroso e mais rápido, aumentando a sua eficiência.

Portanto, o projeto inserido nesta dissertação cumpriu todos os requisitos definidos pela empresa e trouxe funcionalidades adicionais que lhe aumentaram o valor. Esta dissertação marca uma mudança no processo de realização dos testes aos carregadores, diminuindo assim a probabilidade de mau funcionamento do sistema no terreno.

6.1. POSSÍVEIS MELHORIAS DO SISTEMA

Como em qualquer projeto e embora os objetivos levantados no início do mesmo tenham sido cumpridos, existem pontos que podem ser acrescentados ou melhorados.

No que diz respeito ao que a aplicação é capaz de fazer, a empresa concluiu que um futuro desenvolvimento necessário será a criação de testes através da GUI. De momento é possível adicionar no código fonte, mas essa abordagem pode ser facilitada se for realizada através da aplicação. Um ponto que pode ser adicionado é também a criação de um documento com o resultado dos testes, aquando da sua finalização.

Relativamente ao OCPP, nos carregadores da empresa não estão implementadas todas as mensagens da versão 1.6. Portanto, essas mensagens ainda não foram criadas na aplicação, mas quando os carregadores estiverem preparados prevê-se qua a aplicação consiga testar essas mesmas funcionalidades. A longo prazo, a Efacec Electric Mobility S.A. antevê a implementação da versão 2.0 nos seus carregadores e aí a aplicação terá que ser atualizada para suportar essa versão mais recente do OCPP.

Em suma, o sistema criado cumpre com os requisitos solicitados, mas existem pontos a melhorar a curto prazo. Por outro lado, e com a evolução constante da mobilidade elétrica, planeia-se a evolução do protocolo e dos sistemas a longo prazo.

Referências Documentais

[1] "2.1.2 Lecture Notes - TU Delft OCW." [Online]. Available: https://ocw.tudelft.nl/course-readings/2-1-2-lecture-notes/. [Accessed: 18-Feb-2020].

[2] Y. Miao, P. Hynan, A. Von Jouanne, and A. Yokochi, "Current li-ion battery technologies in electric vehicles and opportunities for advancements," *Energies*, vol. 12, no. 6, 2019, doi: 10.3390/en12061074.

[3] J. J. Makrygiorgou and A. T. Alexandridis, "Power electronic control design for stable EV motor and battery operation during a route," *Energies*, vol. 12, no. 10, 2019, doi: 10.3390/en12101990.

[4] "Induction Versus DC Brushless Motors | Tesla." [Online]. Available: https://www.tesla.com/pt_PT/blog/induction-versus-dc-brushless-motors?_ga=2.135226097.1148069356.1503696154-1737370490.1503696154&redirect=no. [Accessed: 28-Jul-2020].

[5] "Understanding EV Chargers." [Online]. Available: https://www.spiritenergy.co.uk/kb-ev-charging-understanding. [Accessed: 25-Feb-2020].

[6] M. C. Falvo, D. Sbordone, I. S. Bayram, and M. Devetsikiotis, "EV charging stations and modes: International standards," *2014 Int. Symp. Power Electron. Electr. Drives, Autom. Motion, SPEEDAM 2014*, no. June, pp. 1134–1139, 2014, doi: 10.1109/SPEEDAM.2014.6872107.

[7] "DC Fast Charging Explained - EV Safe Charge." [Online]. Available: https://evsafecharge.com/dc-fast-charging-explained/. [Accessed: 25-Feb-2020].

[8] "Onboard Charger | Tesla." [Online]. Available: https://www.tesla.com/support/home-charging-installation/onboard-charger. [Accessed: 25-Feb-2020].

[9] "EV Charging Connector Types and Speeds | Pod Point." [Online]. Available: https://pod-point.com/guides/driver/ev-connector-types-speed. [Accessed: 25-Feb-2020].

[10] "Renault Zoe Q210 (2013-2017) price and specifications - EV Database." [Online].

Available: https://ev-database.org/car/1026/Renault-Zoe-Q210#charge-table. [Accessed: 25-Feb-2020].

[11] K. Kersting, “2013 11 12 IECON Validation tests for DC charging stations [Modo de compatibilidad].”

[12] “EV Charging Connector Types | Enel X.” [Online]. Available: https://evcharging.enelx.com/eu/about/news/blog/552-ev-charging-connector-types. [Accessed: 25-Feb-2020].

[13] “History & Timeline – Chademo Association.” [Online]. Available: https://www.chademo.com/about-us/history-and-timeline/#sc-tabs-1583183120739. [Accessed: 02-Mar-2020].

[14] “Technology Overview – Chademo Association.” [Online]. Available: https://www.chademo.com/technology/technology-overview/. [Accessed: 02-Mar-2020].

[15] “CHAdeMO officially recognised as international DC Fast Charging Standard published by IEC – Chademo Association.” [Online]. Available: https://www.chademo.com/chademo-officially-recognised-as-international-dc-fast-charging-standard-published-by-iec/. [Accessed: 02-Mar-2020].

[16] “Protocol Development – Chademo Association.” [Online]. Available: https://www.chademo.com/activities/protocol-development/. [Accessed: 02-Mar-2020].

[17] “History & Timeline – Chademo Association.” [Online]. Available: https://www.chademo.com/about-us/history-and-timeline/#sc-tabs-1583184084579. [Accessed: 02-Mar-2020].

[18] “Nissan Leaf price and specifications - EV Database.” [Online]. Available: https://ev-database.org/car/1106/Nissan-Leaf. [Accessed: 03-Jun-2020].

[19] “Why we’re different!: Charging Interface Initiative e. V. (CharIN e. V.).” [Online]. Available: https://www.charinev.org/about-us/why-were-different/. [Accessed: 02-Mar-2020].

[20] “CCS Specification: Charging Interface Initiative e. V. (CharIN e. V.).” [Online]. Available: https://www.charinev.org/ccs-at-a-glance/ccs-specification/. [Accessed: 02-Mar-2020].

[21] “Carregadores eléctricos | Tesla.” [Online]. Available:

https://www.tesla.com/pt_PT/support/charging-connectors?redirect=no. [Accessed: 02-Mar-2020].

[22] "South Korea To Standardize U.S Version Of CCS/Combo Fast Charge Standard | InsideEVs Photos." [Online]. Available: https://insideevs.com/photos/674182/south-korea-to-standardize-us-version-of-ccs-combo-fast-charge-standard/. [Accessed: 03-Mar-2020].

[23] Efacec, "Public Charging Station," 2019.

[24] "MCCWB-MS - Magnum Cap." [Online]. Available: https://magnumcap.com/professional-line/mccwb-ms/. [Accessed: 03-Mar-2020].

[25] "Terra 54 CJG - Multi-Standard DC Charging Station | DC Fast Charger | ABB." [Online]. Available: https://new.abb.com/ev-charging/products/car-charging/multi-standard/terra-54-cjg. [Accessed: 03-Mar-2020].

[26] Efacec, "Quick CHarge Station QC45," pp. 1–2.

[27] Tritium, "50kW DC Fast Charger."

[28] Tritium, "High Power Charging," 2015.

[29] "High Power Charging | High Power Fast Chargers | ABB." [Online]. Available: https://new.abb.com/ev-charging/products/car-charging/high-power-charging. [Accessed: 03-Mar-2020].

[30] Efacec, "Datasheet-Ultra Fast Charge Station Product description," pp. 350–351, 2019.

[31] "Vendas de Veículos Elétricos em 2019. Novo máximo! - UVE." [Online]. Available: https://www.uve.pt/page/vendas-ve-2019/. [Accessed: 02-Jun-2020].

[32] "MOBI.E." [Online]. Available: https://www.mobie.pt/map. [Accessed: 03-Jun-2020].

[33] "Allego for business - Allego." [Online]. Available: https://www.allego.eu/. [Accessed: 04-Mar-2020].

[34] "SaaS Platform for EV Charging Network Management | Fortum." [Online]. Available: https://www.fortum.com/products-and-services/vehicle-charging/business-services/saas-platform-ev-charging-network-management. [Accessed: 04-Mar-2020].

[35] "IONITY - WHERE & HOW." [Online]. Available: https://ionity.eu/en/where-and-how.html. [Accessed: 04-Mar-2020].

[36] “Fortum Charge & Drive.” [Online]. Available: https://map.chargedrive.com/en/. [Accessed: 04-Mar-2020].

[37] “Locate a charger | Electrify America.” [Online]. Available: https://www.electrifyamerica.com/locate-charger. [Accessed: 04-Mar-2020].

[38] “Supercharger | Tesla.” [Online]. Available: https://www.tesla.com/pt_PT/supercharger?redirect=no. [Accessed: 04-Mar-2020].

[39] “Carregadores eléctricos | Tesla.” [Online]. Available: https://www.tesla.com/pt_PT/support/charging-connectors?redirect=no. [Accessed: 04-Mar-2020].

[40] “Tesla Model 3 Standard Range price and specifications - EV Database.” [Online]. Available: https://ev-database.org/car/1060/Tesla-Model-3-Standard-Range#charge-table. [Accessed: 04-Mar-2020].

[41] “Open Charge Alliance - Global Platform For Open Protocols.” [Online]. Available: https://www.openchargealliance.org/. [Accessed: 09-Mar-2020].

[42] “Background, About us - Open Charge Alliance.” [Online]. Available: https://www.openchargealliance.org/about-us/background/. [Accessed: 09-Mar-2020].

[43] OCA, “Open Charge Point Protocol 1.5,” 2012, pp. 9–10.

[44] e-laad.nl, “OCPP v1.5 A functional description,” 2013, pp. 3–4.

[45] e-laad.nl, “OCPP v1.5 A functional description,” 2013, pp. 4–6.

[46] OCA, “Open Charge Point Protocol 1.5,” 2012.

[47] OCA, “Open Charge Point Protocol 1.5,” 2012.

[48] OCA, “Open Charge Point Protocol 1.5,” 2012.

[49] OCA, “Open Charge Point Protocol 1.5,” 2012, p. 6.

[50] OCA, “Open Charge Point Protocol 1.5,” 2012, p. 7.

[51] OCA, “Open Charge Point Protocol 1.5,” 2012, p. 20.

[52] OCA, “Open Charge Point Protocol 1.5,” 2012.

[53] OCA, “Open Charge Point Protocol 1.5,” 2012, p. 5.

[54] OCA, “Open Charge Point Protocol 1.5,” 2012.

[55] OCA, “Open Charge Point Protocol 1.6,” 2017, pp. 114–116.

[56] OCA, “Open Charge Point Protocol 1.6,” 2017, p. 13.

[57] OCA, *Open Charge Point Protocol SOAP 1.6, OCPP-S 1.6 Specification*. 2015.

[58] OCA, “Open Charge Point Protocol JSON 1.6, OCPP-J 1.6 Specification,” 2015, pp. 10–14.

[59] OCA, “Open Charge Point Protocol 1.6,” 2017.

[60] OCA, “Open Charge Point Protocol 1.6,” 2017.

[61] OCA, “Open Charge Point Protocol 1.6,” 2017.

[62] OCA, “Open Charge Point Protocol 1.6,” 2017.

[63] OCA, “Open Charge Point Protocol 1.6,” 2017.

[64] OCA, “Open Charge Point Protocol JSON 1.6, OCPP-J 1.6 Specification,” 2015, pp. 14–15.

[65] OCA, “Open Charge Point Protocol JSON 1.6, OCPP-J 1.6 Specification,” 2015, pp. 15–20.

[66] OCA, “Open Charge Point Protocol SOAP 1.6, OCPP-S 1.6 Specification,” 2015, p. 11.

[67] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 0,” no. April, 2018, pp. 3–4.

[68] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 4,” no. April, p. 5, 2018.

[69] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” no. April, 2018, pp. 13–26.

[70] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 4,” 2018, pp. 15–16.

[71] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 323–325.

[72] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 372–373.

[73] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, p. 346.

[74] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 144–154.

[75] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 316–317.

[76] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, p. 357.

[77] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 200–202.

[78] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 0,” 2018, p. 4.

[79] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 231–236.

[80] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 311–313.

[81] “ISO 15118-1:2019(en), Road vehicles — Vehicle to grid communication interface — Part 1: General information and use-case definition.” [Online]. Available:

https://www.iso.org/obp/ui/#iso:std:iso:15118:-1:ed-2:v1:en:term:3.1.31. [Accessed: 30-Apr-2020].

[82] OCA, “OCA: Open Charge Point Protocol - OCPP 2.0 - Part 2,” 2018, pp. 76–79.

[83] “Test Tool, Protocols, Home - Open Charge Alliance.” [Online]. Available: https://www.openchargealliance.org/protocols/test-tool/. [Accessed: 14-Jun-2020].

[84] “ocpp · PyPI.” [Online]. Available: https://pypi.org/project/ocpp/. [Accessed: 11-May-2020].

[85] “GitHub - mobilityhouse/ocpp: Python implementation of the Open Charge Point Protocol (OCPP).” [Online]. Available: https://github.com/mobilityhouse/ocpp. [Accessed: 30-Jun-2020].

[86] “PyQt5 · PyPI.” [Online]. Available: https://pypi.org/project/PyQt5/. [Accessed: 19-May-2020].

[87] “Qt Designer Manual.” [Online]. Available: https://doc.qt.io/qt-5/qtdesigner-manual.html. [Accessed: 19-May-2020].

[88] “websockets · PyPI.” [Online]. Available: https://pypi.org/project/websockets/. [Accessed: 11-May-2020].