UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

João Rafael Caye

Geração automática de arquivos para simulação em bancada de mensagens de diagnósticos veiculares adquiridas da rede CAN

Florianópolis
2023

João Rafael Caye

Geração automática de arquivos para simulação em bancada de mensagens de diagnósticos veiculares adquiridas da rede CAN

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Carlos Barros Montez, Dr.
Supervisor: Renê Engels, Eng.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Caye, João Rafael
Geração automática de arquivos para simulação em bancada de mensagens de diagnósticos veiculares adquiridas da rede CAN / João Rafael Caye ; orientador, Carlos Montez, 2023.
57 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Rede CAN. I. Montez, Carlos . II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

João Rafael Caye

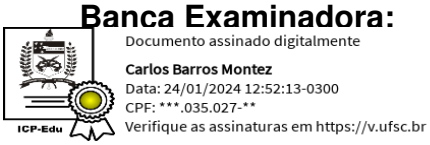
Geração automática de arquivos para simulação em bancada de mensagens de diagnósticos veiculares adquiridas da rede CAN

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

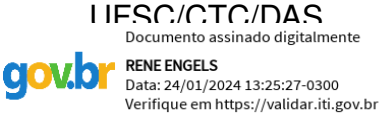
Florianópolis,11 de Dezembro de 2023.

Prof. Marcelo de Lellis, Dr.
Coordenador do Curso

Banca Examinadora:



Prof. Carlos Barros Montez, Dr.
Orientador
UFSC/CTC/DAS



Renê Engels, Eng.
Supervisor
Doutor-IE

Prof. Rodrigo Lange, Dr.
Avaliador
IFRS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Este trabalho é dedicado aos meus colegas de classe,
amigos e familiares.

AGRADECIMENTOS

A conclusão dessa etapa só foi possível graças à vocês que estiveram comigo desde o início. Aos meus pais Luís e Sandra, que sempre oportunizaram o acesso à leitura e educação. Aos meus avós, Osvaldo (*in memorian*) e Onorina, que abriram as portas e me receberam durante todo o período da graduação. Ao meu irmão Júlio, meu primeiro amigo e referência profissional.

Obrigado por compartilharem comigo mais essa etapa, família!

“ Victory depends on your ability to distinguish vulnerable points from invulnerable ones.”
(Funakoshi, 1938)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 29 de Novembro de 2023.

Na condição de representante da Empresa Doutor-IE na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

gov.br
Documento assinado digitalmente
RENE ENGELS
Data: 24/01/2024 13:27:26-0300
Verifique em https://validar.iti.gov.br

Renê Engels
Doutor-IE

RESUMO

Este relatório apresenta o processo de desenvolvimento de um aplicativo para interpretação do protocolo de transporte de mensagens de diagnóstico da rede CAN. O projeto foi voltado para a equipe de engenharia da Empresa Doutor-IE, que possuía um processo muito lento para a reprodução dos *frames* das mensagens do barramento CAN. Foram utilizadas para a implementação da solução a linguagem Python, CAPL, e as normas KWP 2000 e UDS. São descritos os casos de uso, abordados os tópicos que embasaram a construção da solução e o apontamento de requisitos de sistema. Por fim, as funcionalidades desenvolvidas e resultados do conteúdo abordado são apresentadas. Este relatório resume o trabalho de um Engenheiro de Controle e Automação em algumas de suas atribuições, visando otimizar processos empresariais.

Palavras-chave: CAPL. CAN. CANoe. OBD.

ABSTRACT

This report outlines the development process of an application for interpreting the transport protocol of CAN networks. The project was aimed at the engineering team of Doutor-IE Company, which had a very slow process for reproducing message frames. Python language, CAPL, and the KWP 2000 and UDS standards were used for the implementation of the solution. The use cases are described, the topics that underpinned the construction of the solution are addressed, and the system requirements are identified. Finally, the developed functionalities and results of the content covered are presented. This report summarizes the work of a Control and Automation Engineer in some of their duties, aiming to optimize business processes.

Keywords: CAPL. CAN. CANoe. OBD.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

BLF	*Binary Logging Format*
CAN	*Controller Area Network*
CAPL	*Communication Access Programming Language*
DTC	*Diagnostic Trouble Code*
KWP	*Key Word Protocol*
OBD	*On Board Diagnostic*
OBD-II	*On Board Diagnostic II*
PID	*Parameter Identifier*
UCE	Unidade de Comando Eletrônico
UDS	*Unified Diagnostic Services*

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A frota de veículos leves – automóveis – praticamente triplicou nos últimos 15 anos (IBGE, 2023), acompanhado das crescentes restrições nas normas de emissões e regulamentações automotivas (NORMA OBDBR2/3). Essas restrições trazem consigo desafios substanciais para a eficiência operacional das oficinas mecânicas. Nesse cenário, o desenvolvimento de um equipamento de diagnóstico ganha relevância como uma resposta estratégica para auxiliar as oficinas na detecção e resolução precisa das múltiplas falhas e problemas apresentados pelos veículos modernos.

À medida que os veículos se tornam cada vez mais complexos em termos de eletrônica, sistemas de controle e mecânica avançada, as oficinas mecânicas enfrentam a necessidade de aprimorar suas capacidades de diagnóstico e reparo. A correta identificação e resolução de falhas não apenas são cruciais para a satisfação dos clientes, mas também para a segurança viária e o cumprimento das regulamentações de segurança e emissões.

Além disso, os grandes *players* do mercado de equipamentos de dianóstico automotivos são estrangeiros, não possuindo uma tradução adequada dos nomes dos sistemas e funcionalidades. Por consequência, observou-se a oportunidade de entregar uma ferramenta disruptiva: um *scanner* automotivo feito para o mercado brasileiro.

Para o desenvolvimento do equipamento, é necessária a engenharia reversa dos protocolos de comunicação, foco deste trabalho.

A motivação deste trabalho é conceber uma etapa do desenvolvimento desse grande projeto, a simulação em bancada das mensagens de diagnóstico das Unidade de Comando Eletrônico (UCE) do veículo. O projeto em questão envolve o conhecimento de protocolos de comunicação, aquisição de sinais, arquitetura de software e gerenciamento de projeto. Gerenciar, interpretar normas, projetar e executar são etapas fundamentais para o êxito do projeto.

As características apresentadas anteriormente e descritas ao longo deste Projeto de Fim de Curso vão ao encontro das características pretendidas para um egresso do curso de Engenharia de Controle e Automação, uma vez que visa o desenvolvimento de uma nova tecnologia não disponível no mercado.

1.2 OBJETIVOS

Diante da motivação exposta, pretende-se criar uma aplicação em linguagem Python capaz de interpretar os *frames* das mensagens de diagnóstico e gerar um arquivo principal em linguagem CAPL (INFORMATIK, 2021), para que seja possível

simular em bancada a comunicação de diagnóstico entre *scanner* e central do veículo utilizando o equipamento Vector.

Para isso, este projeto terá os seguintes desafios:

- Compreender o funcionamento de um equipamento de diagnóstico automotivo;
- Compreender o funcionamento de uma rede CAN;
- Criar código em Python capaz de extrair mensagens do *log* recebido;
- Criar código Python para formatar as mensagens conforme modelo de documento em CAPL;
- Criar código em Python para gerar os *arrays* com os *bytes* da mensagem lida;
- Criar código em Python para gerar os documentos de simulação completos em CAPL.

O presente escopo consiste em um projeto de engenharia reversa do protocolo de transporte das mensagens de diagnóstico e simulação em bancada. Muito além do desenvolvimento de qualquer sistema, especificar e projetar as funcionalidades que vão de fato atender reais necessidades de uma equipe é papel do Engenheiro de Controle e Automação.

1.3 ESTRUTURA DO DOCUMENTO

A seção inicial deste documento expõe os princípios e a base teórica necessária para compreender a solução desenvolvida ao longo do projeto. Em seguida, são apresentados os estudos de caso e a análise dos requisitos essenciais para a concepção do produto. Após isso, descrevem-se os procedimentos de implementação do sistema e o método de desenvolvimento. Por fim, são realizadas análises dos resultados obtidos, seguidas de uma breve demonstração da solução finalizada, juntamente com conclusões e perspectivas para o futuro.

2 A DOUTOR-IE

A Doutor-IE é uma empresa de tecnologia sediada em Florianópolis, nas imediações da Universidade Federal de Santa Catarina. Iniciou suas atividades na década de 90 desenvolvendo manuais técnicos sobre a eletrônica embarcada em veículos automotores. Em 2010 lançou a **Enciclopédia Online Doutor-IE**, a primeira plataforma online de manuais técnicos automotivos do Brasil. Em 2020, esta empresa lança sua primeira versão em aplicativo para *smartphone*. Desde então ela vem inovando, com diversas funcionalidades novas relacionadas à informação técnica automotiva.

A empresa é líder no segmento no mercado nacional. Contudo, para que mantenha-se em tal condição, o espírito da inovação está presente no dia-a-dia dos colaboradores.

O conteúdo desse trabalho é parte de mais um projeto de inovação da empresa, buscando quebrar paradigmas e entregar produtos com qualidade jamais vista no mercado nacional.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo apresentam-se os conceitos necessários para entender o desenvolvimento do projeto. Abordam-se desde as normas regulamentadoras até os equipamentos e tecnologias utilizadas.

3.1 NORMA OBD-II

O sistema OBD-II é um protocolo de diagnóstico automotivo padronizado que desempenha um papel crucial no monitoramento e controle das emissões de veículos. Ele foi introduzido nos Estados Unidos por volta de 1994 (CALIFÓRNIA, 1993) inicialmente no estado da Califórnia, como resposta às crescentes preocupações com a poluição do ar e seus efeitos adversos na saúde pública e no meio ambiente. O OBD-II desde então se tornou um padrão global, adotado pela maioria dos países para reduzir as emissões de veículos e melhorar a qualidade do ar.

O OBD-II é essencialmente um sistema de diagnóstico avançado integrado em veículos modernos, principalmente carros a gasolina e caminhonetes leves. Seu objetivo principal é detectar e relatar mau funcionamento no motor do veículo e nos sistemas de controle de emissões. O sistema é composto por vários sensores, uma UCE e um conector trapezoidal padronizado localizado sob o painel de instrumentos.

A seguir pontuam-se alguns tópicos relativos ao funcionamento do sistema OBD-II e seus efeitos na redução de poluição:

- **Monitoramento em Tempo Real**: O sistema OBD-II utiliza os sinais do sensores utilizados pela central para o controle do motor, para monitorar continuamente o desempenho do motor e dos sistemas de controle de emissões. A partir dos dados de vários sensores, como o sensor de oxigênio, o sensor de fluxo de massa de ar e o sensor de temperatura do conversor catalítico, utiliza monitores para verificar se o sistema atende aos requisitos mínimos de emissões projetados.

- ***Diagnostic Trouble Code* (DTC)**: Se a UCE identificar que algum dos monitores foi reprovado nos testes anteriores, caso ocorra um mau funcionamento nos sistemas do motor ou de controle de emissões, ele pode gravar na memória um ou mais DTCs. Esses códigos fornecem informações específicas sobre o problema, tornando mais fácil para os mecânicos diagnosticar e reparar o problema.

- **Luz de Verificação do Motor**: Quando um DTC crítico para o funcionamento dos principais sistemas é gerado, o sistema OBD-II ativa a luz de alerta para verificação do motor no painel de instrumentos do veículo, conforme Figura 1. Isso serve como uma indicação visual para o motorista de que há um problema com o motor ou circuitos eletrônicos do veículo. Ignorar este aviso pode levar a

uma maior poluição e a uma menor eficiência do consumo de combustível devido aos problemas presentes.

- **Controle de Emissões**: O OBD-II desempenha um papel vital em garantir que os sistemas de controle de emissões, como o conversor catalítico e o sistema de recirculação dos gases de escape (EGR), estejam funcionando corretamente. Esses sistemas ajudam a reduzir poluentes nocivos, como óxidos de nitrogênio (NOx), monóxido de carbono (CO) e hidrocarbonetos (HC), emitidos pelo escapamento do veículo.

- **Conformidade Regulatória**: O OBD-II está intimamente ligado às regulamentações de emissões. Governos de todo o mundo estabeleceram limites rigorosos para as emissões de veículos a fim de combater a poluição do ar e seus problemas de saúde associados. Os sistemas OBD-II ajudam a garantir que os veículos atendam a essas regulamentações, monitorando e relatando continuamente problemas relacionados às emissões.

- **Impacto Ambiental**: A ampla adoção do OBD-II teve um impacto positivo significativo na redução da poluição do ar conforme (LIU *et al.*, 2019). Ao detectar e resolver prontamente os problemas no motor e nos sistemas de controle de emissões, ele impede que os veículos emitam poluentes em excesso na atmosfera. Isso, por sua vez, ajuda a melhorar a qualidade do ar e a reduzir os efeitos adversos à saúde associados à má qualidade do ar, como doenças respiratórias e problemas cardiovasculares.

- **Eficiência do consumo de Combustível**: O OBD-II também pode contribuir para melhorar a eficiência de combustível. Quando o motor opera de forma otimizada, ele queima combustível de maneira mais eficiente, reduzindo o consumo de combustível e, consequentemente, as emissões. Isso beneficia tanto o meio ambiente quanto o proprietário do veículo, economizando nos custos de combustível.

Em resumo, a norma OBD-II tem sido uma ferramenta crucial na luta contra a poluição do ar relacionada a veículos. Ao monitorar e diagnosticar continuamente problemas no motor e nos sistemas de controle de emissões, ela garante que os veículos cumpram as regulamentações de emissões e operem de forma eficiente. Isso, por sua vez, leva à redução dos níveis de poluição, à melhoria da qualidade do ar e à promoção da saúde pública. À medida que a indústria automobilística continua a evoluir, o protocolo OBD-II permanece como um componente fundamental nos esforços contínuos para mitigar o impacto ambiental dos veículos em escala global.

Figura 1 – Luz de mau funcionamento do motor.

Fonte: El País, 2022.

3.2 EQUIPAMENTO DE DIAGNÓSTICO AUTOMOTIVO

Em 1908 Henry Ford lança o Ford T, o mais popular veículo automotor da época. Tinha sua partida à manivela, e enfrentava dificuldades para funcionar em dias frios. Também possuía uma velocidade bastante limitada e baixa. Não possuía redes de comunicação, ou qualquer dispositivo eletrônico.

Com a ocasião das Grandes Guerras, houveram avanços tecnológicos significativos. Em 1935 é lançado o primeiro Volkswagen Fusca, veículo moderno para a época, com partida a manivela. Rapidamente recebeu novos componentes, passou a contar com motor de partida, dínamo para carregar a bateria, luzes indicadoras de direção, faróis, interruptor de pressão de óleo, buzina. Foi um grande avanço, passando a trazer diversos dispositivos que auxiliassem o motorista. No entanto, caso a lâmpada indicadora de direção viesse a queimar, o motorista não teria nenhum *feedback* da falha. Em 1975 foram lançados os primeiros veículos com catalisador. Entretando, como garantir que estes ainda eram eficientes dez anos depois? Não havia medição para que tal informação fosse conhecida.

Em 1988 é lançado o primeiro veículo brasileiro com sistema de injeção eletrônica de combustível. Logo após, vieram sistemas de freio anti-bloqueantes, sistemas de *arbag*, sistemas de som, ar-condicionado, etc.

No Fusca, quando a lâmpada queimava, bastava que alguém observasse o veículo e informasse ao motorista que havia um problema na lâmpada.

Mas agora, com tantos circuitos eletrônicos nos veículos e dezenas de metros de fios elétricos para conduzir a energia e os sinais dos sensores, como diagnosticar

as falhas?

É aí que entra o equipamento de diagnóstico automotivo, o *Scanner*.

A era digital transformou a indústria automotiva, integrando sistemas complexos de diagnóstico que proporcionam uma manutenção preditiva e resolução de problemas com precisão sem precedentes (SUDSOMBOON, 2007). Essa inovação é encapsulada pelo uso de ferramentas de diagnóstico automotivo, dispositivos essenciais que operam como uma ponte de comunicação entre a engenharia avançada de um veículo e o técnico ou o proprietário. Ao ligar o aparelho ao conector trapezoidal de um veículo, essas ferramentas não apenas recuperam informações vitais sobre o status e o desempenho do veículo, mas também oferecem uma análise detalhada e *insights* sobre o funcionamento dos sistemas eletrônicos. A adoção generalizada dessas ferramentas representa um marco na manutenção automotiva, elevando a detecção e correção de falhas a um nível de eficiência que redefine o conceito de cuidados com o veículo. A Figura 2 apresenta um equipamento de diagnóstico.

O funcionamento de um *Scanner* automotivo pode ser descrito da seguinte forma:

Figura 2 – *Scanner* automotivo.

Fonte: Car and Driver, 2023.

- **Conexão à Porta OBD**: O primeiro passo para utilizar o equipamento é localizar

a porta OBD – ou conector trapezoidal – no veículo. Essa porta geralmente está localizada no *cockpit* do motorista, próximo à coluna de direção. Uma vez localizada, a ferramenta é conectada a essa porta usando um cabo compatível.

- **Recuperação de Dados**: Após a conexão do cabo, o *scanner* comunica-se com o sistema de computador a bordo do veículo. O equipamento identifica as diversas centrais e comunica-se com os sistemas eletrônicos do veículo.
- **Interpretação dos Dados**: O equipamento então interpreta os dados recebidos dos sensores e sistemas do veículo. Esses dados incluem informações sobre o desempenho do motor, emissões, eficiência de combustível, sistemas de airbag, ar-condicionado, freios ABS, etc. A ferramenta converte os dados binários lidos em valores e leituras que podem ser compreendidas. Ela é capaz também de interpretar os códigos de falhas enviados pelas centrais.
- **Exibição e Análise**: Cada equipamento possui as suas particularidades, mas de modo geral exibem os dados e apresentam o significado dos mesmos. Podem ser código de falhas, leitura de velocidade, rotação, pressão ou temperatura.
- **Teste de atuadores**: É possível controlar atuadores através de testes como a abertura do corpo de borboletas eletrônico, acoplamento da embreagem do ar-condicionado, pulsar os bicos injetores entre outras tarefas.
- **Funções especiais**: As funções especiais têm a função de configurar as centrais. São utilizadas para atualizações de software, ajuste de configurações ou rotinas de aprendizado.
- **Limpeza de DTCs**: Com a resolução dos problemas, o equipamento pode enviar as mensagens de limpeza de código de falhas. Essa função solicita à UCE a remoção dos códigos da sua memória. Com a exclusão dos códigos, a luz de mau-funcionamento do motor é apagada.

Com todas essas funções, o equipamento é de suma importância no dia-a-dia das oficinas mecânicas. Ele é de extrema importância para os reparadores, pois permite:

- **Diagnósticos Eficientes**: Esses equipamentos reduzem significativamente o tempo e o esforço necessários para diagnosticar problemas em veículos. Os mecânicos podem identificar rapidamente problemas e suas causas, o que leva a reparos mais rápidos.
- **Economia de Custos**: Os proprietários de veículos também se beneficiam. Ao identificar problemas precocemente, e assim evitar reparos mais extensos e caros no futuro.

- **Testes de Emissões**: *Scanners* desempenham um papel crucial nos testes e na conformidade com as normas de emissões. Essas normas garantem que os veículos atendam aos padrões ambientais, reduzindo a poluição do ar.

- **Melhor Eficiência de Consumo de Combustível**: Ao monitorar e diagnosticar o desempenho do motor, os *scanners* contribuem para uma melhor economia de combustível, economizando dinheiro aos proprietários de veículos. Contribuem também para um menor volume de gases poluentes lançados na atmosfera, poluindo menos.

Os *Scanners* automotivos revolucionaram a forma como é feito o diagnóstico veicular. Sua capacidade de recuperar e interpretar rapidamente dados de diagnóstico as tornou parte das ferramentas das oficinas mecânicas. Com seu uso é possível manter o veículo em ótimo estado e garantir sua conformidade com os padrões ambientais.

3.3 REDE CAN

O barramento CAN é um protocolo de comunicação amplamente adotado, conhecido por sua confiabilidade e eficiência em diversas indústrias. Apresenta-se neste documento a camada física e de enlace de dados, uma vez que o nível lógico de troca de dados será apresentado nas normas mais adiante.

Este protocolo revolucionou a comunicação em setores onde a troca de dados em tempo real e a robustez são essenciais, como automotivo, automação industrial e aeroespacial. A camada física do protocolo possui as seguintes características apresentadas a seguir, e podem ser visualizadas na Figura 3.

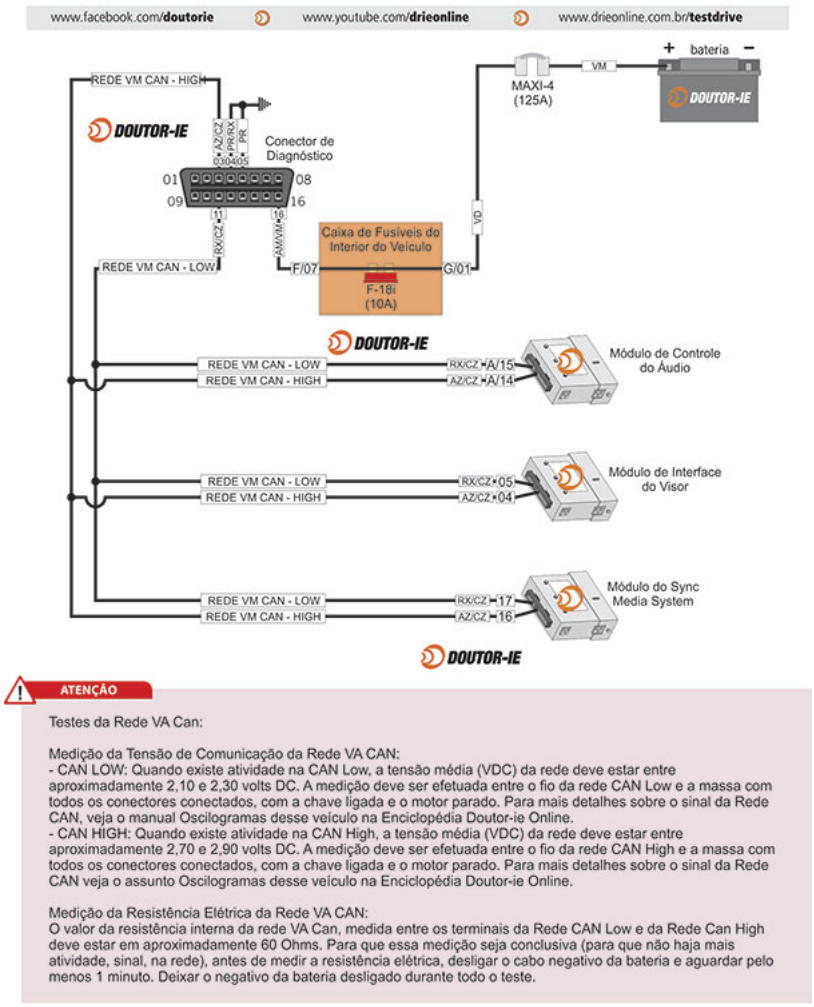
Criado pela fabricante de centrais *Bosch* (OSCH; SMOLKA, 2001), este padrão de rede passou a estar presente no mercado a partir de 1989.

- **Componentes de Hardware:** A camada física do Barramento CAN compreende os componentes físicos que possibilitam a comunicação. Isso inclui, em geral, o meio de transmissão de dois fios (CAN High e CAN Low) e os nós, que geralmente são microcontroladores ou unidades de controle eletrônico (UCEs) na rede.

- **Sinalização Diferencial:** A comunicação no Barramento CAN utiliza sinalização diferencial, onde os dados são transmitidos como uma diferença de tensão entre CAN High e CAN Low. Essa abordagem diferencial melhora a imunidade a ruídos e permite a transmissão confiável de dados, mesmo em ambientes eletricamente ruidosos.

- **Temporização de Bits e Sincronização:** O Barramento CAN convencional opera com uma taxa de bits fixa – o *baud rate* – e todos os nós na rede devem aderir a

Figura 3 – Diagrama de redes de comunicação do Ford Ka.



Fonte: Oficina Brasil, 2016.

especificações precisas de temporização para garantir a sincronização. A sincronização da temporização de bits é crucial, e envolve os nós se alinhando com o início e o fim de cada intervalo de bit para uma recepção consistente de dados.

A camada de enlace de dados é composta pelas seguintes características (FAVERI TRON *et al.*, 2022):

- **Estrutura das Mensagens:** A camada de enlace de dados é responsável por segmentar os dados em quadros para transmissão. As mensagens são divididas em quadros, que incluem o ID da mensagem, carga de dados e bits de controle. Essa estrutura permite a transmissão eficiente de dados na rede.
- **Arbitragem e Prioridade:** A camada de enlace de dados gerencia o processo de arbitragem, no qual os nós competem pelo acesso ao barramento quando vários nós desejam transmitir simultaneamente. Mensagens com valores de IDs

de prioridade mais baixa têm acesso ao barramento primeiro, garantindo uma transmissão ordenada de dados.

- **Detecção e Tratamento de Erros:** Para manter a integridade dos dados, a camada de enlace de dados incorpora mecanismos de detecção de erros, como o *Cyclic Redundancy Check* (CRC). Se um erro é detectado, a camada de enlace de dados toma medidas, que podem incluir a retransmissão da mensagem.
- **Controle de Fluxo:** Mecanismos de controle de fluxo garantem que os dados sejam transmitidos a uma taxa que o nó receptor possa manipular, evitando sobrecarga de dados e congestionamento na rede.

Essa é uma visão geral que contempla os principais aspectos da rede CAN considerados no trabalho desenvolvido. Existem outras topologias físicas de rede CAN que não foram abordadas neste trabalho.

3.4 NORMA KWP 2000

O *Key Word Protocol* (KWP) é um protocolo antigo, desenvolvido para a indústria automotiva para diagnóstico do sistema *On Board Diagnostic* (OBD). É utilizado para padronizar a comunicação entre equipamento de diagnóstico e as centrais do veículo. Ele padroniza os serviços utilizados nas centrais, bem como a estrutura de suas mensagens. Isso permite que empresas não fabricantes possam desenvolver seus próprios equipamentos para atender o mercado.

Esse protocolo foi introduzido em 1996, fazendo parte dos primeiros veículos da norma OBD. Ele foi desenvolvido para o uso na rede de comunicação Linha K, que é um protocolo serial mestre-escravo. Posteriormente utilizou-se esse protocolo em conjunto com a rede CAN.

Alguns pontos-chave sobre o protocolo KWP:

- **Padrão de Comunicação:** O KWP é um protocolo de comunicação padrão que especifica como os dados são trocados entre *scanner* e as ECUs de um veículo. Ele define o formato, o tempo e a estrutura das mensagens enviadas entre os dois.
- **Funções de Diagnóstico:** O KWP permite que o *scanner* acesse e obtenha informações de várias centrais dentro de um veículo. Isso inclui a leitura e exclusão de códigos de falha, o monitoramento de dados de sensores e a realização de vários testes de atuadores e funções especiais de diagnóstico.
- **Padrão ISO:** O KWP é padronizado pela ISO 14230, que especifica os requisitos e detalhes de implementação do protocolo. A ISO 14230 inclui várias partes,

sendo que cada parte se concentra em aspectos diferentes da comunicação KWP – como camada física e camada de enlace de dados.

- **Conformidade com OBD-II:** O KWP é usado em conjunto com o padrão OBD-II, que é obrigatório para a maioria dos veículos fabricados após 1996 nos Estados Unidos e 2010 no Brasil.

A norma KWP refere-se ao conjunto de normas e especificações que definem como as ferramentas de diagnóstico se comunicam com as centrais eletrônicas de um veículo usando o protocolo KWP. Isso desempenha um papel crucial em diagnósticos e manutenção de veículos, permitindo que técnicos acessem e interpretem informações dos sistemas de computadores a bordo de um veículo de forma eficiente.

A Figura 4 apresenta o serviço 0x21 dessa norma, com as respectivas configurações da mensagem. A Figura 5 mostra a estrutura da mensagem dos códigos de falhas, impressíndíveis pra o diagnóstico veicular efetivo.

Figura 4 – Estrutura do serviço 0x21.

7.1.2 - Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
1	**readDataByLocalIdentifier Request Service Id**	M	21	RDBLID
2	recordLocalIdentifier	M	xx	RLOCID
3	transmissionMode=[single, slow, medium, fast, stop]	U	xx=[01, 02, 03, 04, 05]	TXM
4	maximumNumberOfResponsesToSend	U/C1	xx	MNORTS

Table 7.1.2.1 - readDataByLocalIdentifier Request Message

C1 = condition: transmissionMode parameter must be present

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
1	**readDataByLocalIdentifier Positive Response Service Id**	S	61	RDBLIDPR
2	recordLocalIdentifier	M	xx	RLOCID
3	recordValue#1	M	xx	RECVAL
:	:	:	:	
n	recordValue#m	M	xx	

Table 7.1.2.2 - readDataByLocalIdentifier Positive Response Message

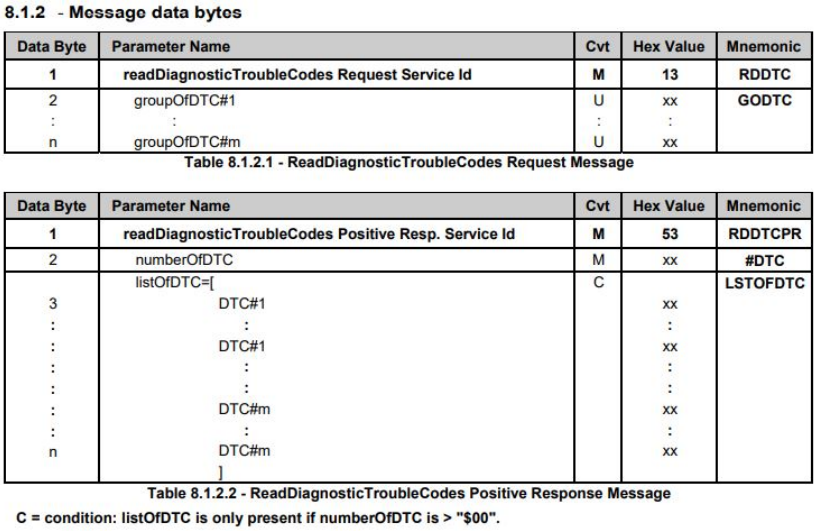
Fonte: *Key Word Protocol 2000*, 1996.

3.5 NORMA UDS

A norma *Unified Diagnostic Services* (UDS) é um protocolo de diagnóstico padronizado usado na indústria automotiva para se comunicar e diagnosticar unidades de controle eletrônico (UCE) em veículos. A normaUDSUDS é uma evolução do protocolo de diagnóstico KWP e é considerada uma abordagem mais avançada e padronizada para diagnóstico de veículos.

Alguns pontos-chave para sobre a norma UDS:

Figura 5 – Estrutura do serviço 0x13.



8.1.2 - Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
1	readDiagnosticTroubleCodes Request Service Id	M	13	RDDTC
2	groupOfDTC#1	U	xx	GODTC
:	:	:	:	
n	groupOfDTC#m	U	xx	

Table 8.1.2.1 - ReadDiagnosticTroubleCodes Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
1	readDiagnosticTroubleCodes Positive Resp. Service Id	M	53	RDDTCPR
2	numberOfDTC	M	xx	#DTC
	listOfDTC=[	C		LSTOFDTC
3	DTC#1		xx	
:	:		:	
:	DTC#1		xx	
:	:		:	
:	:		:	
:	DTC#m		xx	
:	:		:	
n	DTC#m		xx	
	]			

Table 8.1.2.2 - ReadDiagnosticTroubleCodes Positive Response Message

C = condition: listOfDTC is only present if numberOfDTC is > "$00".

Fonte: *Key Word Protocol 2000*, 1996.

- **Padronização:** A norma UDS é um padrão reconhecido internacionalmente especificado na ISO 14229-1. Essa padronização garante que ferramentas de diagnóstico e UCEs de veículos possam se comunicar efetivamente em diferentes marcas e modelos de veículos.
- **Serviços de Diagnóstico:** A norma UDS define um conjunto de serviços ou funções de diagnóstico que os *scanners* podem solicitar às UCEs do veículo. Esses serviços abrangem uma ampla gama de funções de diagnóstico, incluindo leitura e exclusão de códigos de falha, acesso a informações de identificação da UCE, obtenção de dados de sensores, realização de testes e rotinas.
- **Comunicação Bidirecional:**A norma UDS suporta comunicação bidirecional entre *scanner* e as centrais do veículo. Isso permite capacidades de diagnóstico mais avançadas, como reprogramação de ECUs ou configuração de parâmetros específicos.
- **Códigos de Erro Padronizados:** A norma UDS utiliza códigos de falha de diagnóstico padronizados (DTC) para relatar problemas nos sistemas do veículo. Esses códigos são definidos na ISO 15031-6 e são comuns entre vários fabricantes de veículos, facilitando a identificação e a resolução de problemas para técnicos.
- **Segurança:** A norma UDS inclui recursos de segurança para proteger informações sensíveis do veículo e impedir o acesso não autorizado. Ela utiliza meca-

nismos de autenticação e controle de acesso para garantir que apenas usuários autorizados possam realizar determinadas funções de diagnóstico.

- **Estrutura Modular:** A norma UDS possui uma estrutura modular que permite aos fabricantes de veículos definir serviços de diagnóstico personalizados específicos para seus veículos, mantendo-se compatíveis com a norma UDS para serviços básicos. Essa flexibilidade é útil para fabricantes adicionar funções ou recursos de diagnóstico exclusivos.
- **Protocolos de Transporte:** A norma UDS pode ser implementada em vários protocolos de comunicação, incluindo CAN, K-Line e outros, dependendo da arquitetura e requisitos do veículo.
- **Evolução:** A norma UDS continua a evoluir com a indústria automotiva, incorporando novos recursos e aprimoramentos à medida que a tecnologia avança. Essas atualizações ajudam a garantir que a UDS permaneça relevante e eficaz no diagnóstico de veículos modernos com sistemas eletrônicos cada vez mais complexos.

3.6 EQUIPAMENTO VECTOR

A empresa alemã Vector atua no segmento de engenharia automotiva à 35 anos. Ela produz ferramentas feitas por engenheiros e para engenheiros. Suas ferramentas mais difundidas no mercado são os softwares Canoe e CANalyzer, que são ferramentas de simulação e análise de protocolos de comunicação. Ambos são configurados a partir da linguagem CAPL, proprietária da empresa Vector.

Foram utilizados equipamentos da linha VN – apresentados na Figura 6 – para o desenvolvimento do projeto. Eles são capazes de gerar um log das mensagens no carro, bem como atuar na simulação da central, simulando a comunicação entre *Scanner* e carro.

Esses equipamentos são extremamente versáteis e programáveis, permitindo ao engenheiro a simulação de diversos cenários de transferência de dados.

3.7 LINGUAGEM CAPL

A linguagem CAPL, que significa Linguagem de Protocolo Assistida por Computador (INFORMATIK, 2021), é uma linguagem de *script* amplamente utilizada na indústria automotiva, especialmente para simular e testar protocolos de comunicação. Ela funciona em conjunto com os programas Vector CANoe e CANalyzer, para auxiliar no desenvolvimento e validação de sistemas automotivos.

Dentro do ambiente CAPL, os *scripts* são escritos e executados. Os programas descritos anteriormente oferecem uma interface amigável para tarefas como definição

Figura 6 – Equipamentos Vector da linha VN.

Fonte: Vector, 2023.

de cenários de teste, configuração de parâmetros de comunicação de rede e execução de simulações.

Os engenheiros podem definir mensagens em CAPL, replicando pacotes de dados transmitidos pela rede de comunicação. Essas mensagens incluem detalhes essenciais, como ID da mensagem, comprimento dos dados e o próprio conteúdo dos dados.

Os *scripts* em CAPL operam com base em eventos, ou seja, respondem a eventos ou gatilhos específicos que ocorrem na simulação. Os eventos podem envolver a recepção de uma mensagem na rede, a expiração de um temporizador ou eventos personalizados definidos pelo usuário.

O gerenciamento de mensagens é um aspecto fundamental da linguagem CAPL. Os engenheiros podem programar como as mensagens de entrada e saída são tratadas. Por exemplo, ao receber uma mensagem específica, o *script* pode ditar a resposta, que pode envolver a alteração do conteúdo da mensagem, a geração de novas mensagens ou a ativação de eventos adicionais.

A linguagem CAPL suporta *timers* (temporizadores) e atrasos, permitindo a introdução de restrições de tempo e a emulação de cenários de temporização do mundo real. Os engenheiros podem definir temporizadores para transmitir periodicamente mensagens, replicando o comportamento em tempo real. Oferece também a capacidade de definir variáveis e realizar várias operações de manipulação de dados, incluindo operações aritméticas, operações *bitwise* e conversões de tipos de dados.

Essas capacidades são essenciais para implementar lógica complexa nos *scripts*.

Todos esses fatores tornam a linguagem essencial para o desenvolvimento do presente projeto. Todos os arquivos de programação gerados a partir da linguagem Python estarão escritos na linguagem CAPL.

3.8 LINGUAGEM PYTHON

Python é uma linguagem de programação de alto nível conhecida por sua simplicidade e legibilidade. Ela oferece uma sintaxe limpa e direta que atrai tanto iniciantes quanto desenvolvedores experientes. A linguagem Python suporta vários paradigmas de programação, como programação procedural, orientada a objetos e funcional.

A linguagem Python é uma linguagem interpretada, o que significa que não há necessidade de uma etapa de compilação antes de executar o código. Essa linguagem característica permite a codificação interativa e o desenvolvimento rápido. Ele também suporta a tipagem dinâmica, ou seja, os tipos de variáveis são determinados em tempo de execução, reduzindo a necessidade de declarações explícitas de tipos.

A versatilidade do Python se destaca em suas aplicações em diferentes domínios, incluindo desenvolvimento web, análise de dados, aprendizado de máquina, computação científica, automação, desenvolvimento de jogos e muito mais. Sua sintaxe limpa baseada em indentação promove código legível e de fácil manutenção.

O Python fornece estruturas de dados de alto nível, como listas, dicionários e tuplas, simplificando a manipulação de dados complexos. Ele segue o processo de Proposta de Melhoria do Python (PEP) para desenvolvimento orientado pela comunidade, garantindo mudanças e aprimoramentos bem documentados e discutidos pela comunidade.

Em face das características apresentadas anteriormente, o projeto desenvolvido teve a linguagem como base para o desenvolvimento, com o uso de bibliotecas de interface visual.

4 CASOS DE USO

Neste capítulo são apresentadas as especificações e as necessidades dos futuros usuários do sistema, oriundos de discussões com os engenheiros da equipe. Os problemas apresentados a seguir são discutidos um a um nesta seção, uma vez que eles estão correlacionados com o sistema desenvolvido.

4.1 GERAÇÃO DE ARQUIVOS DE SIMULAÇÃO

Os engenheiros da equipe, em entrevista, relataram que há grande dificuldade e morosidade na reescrita dos *frames* das mensagens respondidas pela central. Isso porquê, no método utilizado, eles deveriam redigitar todos os *bytes* encontrados no log. Conforme ilustra a Figura 7 – que contém menos de 1 segundo de troca de mensagens – o número de dados é bastante elevado, permitindo portanto que erros aconteçam por desatenção do usuário.

Esses erros podem ser responsáveis pela incompatibilidade dos dados que se tenta reproduzir, uma vez que as mensagens em nível binário trazem todas as informações do carro para o equipamento de diagnóstico.

Havia uma opção anterior para mensagens longas que compreendia o uso da ferramenta Excel. A solução resolvia parcialmente o problema de digitação, uma vez que era efetiva apenas em mensagens com múltiplos *frames*. Por outro lado, não resolvia o problema do tempo que se leva para reescrever a mensagem.

Outro grande problema da equipe compreendia a manutenção dos códigos. Como no modelo anterior cada engenheiro desenvolvia à sua forma, ao passar dos anos, os códigos antigos se tornavam de compreensão difícil.

4.1.1 CASOS DE USO DA GERAÇÃO AUTOMÁTICA DE SIMULAÇÃO

1. **Extração dos dados**: Com base no relato dos futuros usuários do sistema, o ponto de partida do programa é a extração dos dados. Para que os usuários possam utilizar os dados gerados, deve-se gerar um arquivo identificando qual o PID de *request* através de um *hash* e um array, iniciando no índice zero, com a mensagem de diagnóstico completa.

2. **Geração do arquivo de simulaçao**: Dado o desafio significativo que a equipe enfrenta em relação à padronização do código, é imperativo que se estabeleça um processo padronizado para a criação de um arquivo de simulação. A automação desse processo de geração de arquivos de simulação contribuirá substancialmente para a uniformidade no desenvolvimento, o que, por sua vez, facilitará a realização de futuras manutenções no código com uma maior eficiência e coerência.

Figura 7 – Log CAN gerado pelo Canalyzer.

```
0.003423 1  230          Rx   d 8 F0 1C 00 00 4B 00 00 00
0.003676 1  41           Rx   d 8 00 00 00 00 00 00 00 00
0.004028 1  91           Rx   d 8 FF FE FF FE 03 00 50 00
0.004270 1  92           Rx   d 8 3F FE 3F FE 3F FE A5 00
0.004510 1  326          Rx   d 8 01 A0 02 39 81 D0 00 00
0.006095 1  82           Rx   d 8 82 00 14 38 A8 00 00 00
0.006336 1  7E           Rx   d 8 7C 8B 3E 80 EF 94 00 00
0.009274 1  217          Rx   d 8 00 00 00 00 00 00 00 00
0.009524 1  49           Rx   d 8 40 00 00 00 00 00 00 00
0.009792 1  77           Rx   d 8 00 00 00 00 7F F0 00 00
0.010057 1  7D           Rx   d 8 00 00 FF 00 00 3F E0 00
0.010317 1  213          Rx   d 8 FF FF 00 00 80 01 FF 00
0.010581 1  216          Rx   d 8 00 01 00 00 00 00 00 00
0.010839 1  415          Rx   d 8 00 00 F4 EF 00 00 00 00
0.012084 1  167          Rx   d 8 72 80 01 00 00 19 F4 00
0.012326 1  171          Rx   d 8 F0 E0 00 00 00 00 00 00
0.012574 1  204          Rx   d 8 C0 00 7D 01 CC 00 00 00
0.012810 1  42F          Rx   d 8 7D 0C 00 39 00 00 01 CC
0.014006 1  91           Rx   d 8 FF FE FF FE 03 00 50 00
0.014248 1  92           Rx   d 8 3F FE 3F FE 3F FE A5 00
0.015677 1  7E           Rx   d 8 7C 8B 3E 80 EE A4 00 00
0.019302 1  217          Rx   d 8 00 00 00 00 00 00 00 00
0.019548 1  4B0          Rx   d 8 FE 00 80 00 00 00 04 FF
0.021894 1  47           Rx   d 8 20 00 00 00 00 00 00 00
0.022140 1  165          Rx   d 8 10 C1 00 00 00 00 00 00
0.022377 1  167          Rx   d 8 72 80 01 00 00 19 F3 00
0.022629 1  200          Rx   d 8 00 00 7F FF 7F FF 00 00
0.022871 1  202          Rx   d 8 04 F3 48 00 60 00 00 00
0.023119 1  204          Rx   d 8 C0 00 7D 01 CC 00 00 00
0.023360 1  230          Rx   d 8 F0 1C 00 00 4B 00 00 00
0.024021 1  91           Rx   d 8 FF FE FF FE 03 00 50 00
0.024273 1  41           Rx   d 8 00 00 00 00 00 00 00 00
0.024516 1  92           Rx   d 8 3F FE 3F FE 3F FE A5 00
0.024758 1  242          Rx   d 8 01 C8 20 82 E8 00 00 01
0.024996 1  40A          Rx   d 8 C0 00 00 32 3D 90 95 00
0.025238 1  82           Rx   d 8 82 00 14 38 AB 00 00 00
0.025476 1  7E           Rx   d 8 7C 8B 3E 80 ED B4 00 00
```

Fonte: Autor, 2023.

4.2 REQUISITOS DA GERAÇÃO AUTOMÁTICA DE SIMULAÇÃO

Considerando os casos de uso apresentados para desempenho e funcionalidade da geração automática, consideram-se requisitos funcionais:

- Realizar o upload do arquivo na extensão .blf;
- Possuir campos para a inserção dos identificadores da central;
- Permitir a inserção de identificador de *gateway*;
- Interpretar apenas *requests* com respostas positivas;
- Gerar arquivos individuais para cada SID presente na central;
- Abrir a pasta com os resultados ao final da execução do programa.

Consideram-se requisitos não funcionais:

- O programa deve levar menos de 1 minuto durante a execução;
- O programa deve ser ter seu uso intuitivo;
- O programa deve ser de fácil compartilhamento;
- O programa deve poder ser executado em todas as máquinas da equipe.

4.3 LOGS DO ARQUIVO CONVERTIDO E COMPARAÇÃO ENTRE CENTRAIS

Os arquivos processados possuem características particulares, mas também são capazes de compartilhar similaridades com centrais presentes em outros veículos. A equipe relatou que, por diversas vezes, as centrais respondem negativamente. Em face da resposta negativa não ser observada com atenção pelo desenvolvedor, durante a análise erros são decorrentes da ausência dessa resposta positiva e ocasionam horas de debug por parte do desenvolvedor para identificar tais erros.

Em relação à similaridade, perdem-se semanas de trabalho quando é realizado o desenvolvimento de duas centrais em paralelo que, no entanto, possuem os mesmos parâmetros de comunicação. Seria de grande valor para a equipe se houvesse um método de comparar as diferentes centrais.

4.3.1 CASOS DE USO DA GERAÇÃO AUTOMÁTICA DOS LOGS

1. **Log de respostas negativas**: Durante a comunicação, existem algumas condições que devem ser cumpridas para que o teste seja executado ou que a central seja capaz de responder à pergunta realizada. A ausência de resposta positiva induz, segundo os desenvolvedores, à grande perda de tempo de trabalho uma vez que nem sempre a identificação desse erro durante a simulação em bancada é feita de forma rápida. Portanto, caso o programa identifique uma resposta negativa, deve ser gerado um log em formato .CSV relacionando o *hash* do *request* com o valor hexadecimal da resposta negativa - para que posteriormente o desenvolvedor possa verificar na norma o significado da mesma.
2. **Log do tamanho das mensagens**: Para que seja possível comparar as centrais, deve ser criado um arquivo que registre todas as mensagens positivas da central, que será equivalente à uma "fotografia"da mesma. O *log* deve conter as características da central e das mensagens de diagnóstico trocadas entre *scanner* e veículo.

4.4 REQUISITOS FUNCIONAIS DA GERAÇÃO DOS LOGS

Considerando os casos de uso apresentados para desempenho e funcionalidade da geração dos LOGs, consideram-se requisitos funcionais:

- O log deve conter um hash com capacidade de identificar de forma única o PID lido;
- O log deve conter os identificadores de pergunta e resposta da central;
- O log deve conter os tamanhos de pergunta e resposta da mensagem;
- A listagem deve ser feita em ordem crescente.

Consideram-se os seguintes requisitos não funcionais:

- Os logs devem ser processados rapidamente;
- Os logs devem ser de fácil compreensão.

5 IMPLEMENTAÇÃO

Neste capítulo são apresentados os processos de desenvolvimento e implementação do projeto. O projeto foi desenvolvido segundo a metodologia ágil, mesmo que contando apenas com um desenvolvedor. A estrutura, passo-a-passo de desenvolvimento e a evolução do projeto são descritas a seguir.

5.1 A ESTRUTURA DO PROJETO

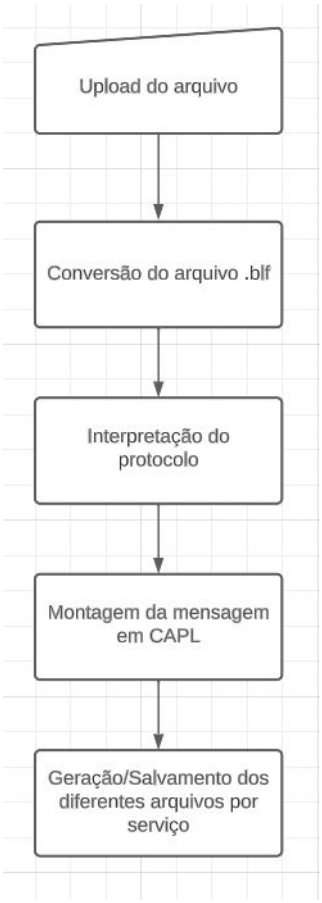
Inicialmente, a fim de estruturar o projeto a ser desenvolvido, aplicaram-se os conhecimentos de engenharia para o desenvolvimento de um fluxograma da programação, que é a ferramenta base para o sucesso do projeto. O fluxograma apresentado na Figura 8 identifica todas as etapas que devem ser desenvolvidas.

Dada a complexidade do projeto, o desenvolvimento ocorreu de forma gradual, de forma que novas funcionalidades ou recursos foram adicionados conforme as entregas ocorreram aos *stakeholders*.

Os arquivos que compreendem os códigos desenvolvidos estão na seção de apêndice do documento.

As tarefas descritas pelo fluxograma serão quebradas em processos menores, conforme descrito nos tópicos a seguir.

Figura 8 – Fluxograma do sistema.



Fonte: Autor, 2023.

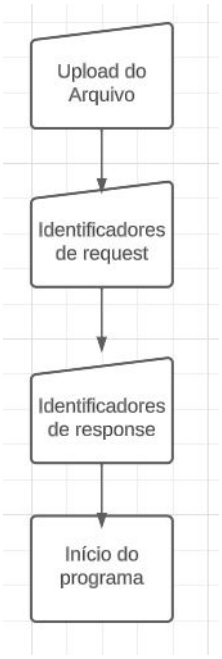
5.1.1 CARREGAMENTO DO ARQUIVO

A ação de upload do arquivo .blf gerado pelo equipamento VECTOR possui diversas particularidades. Essa ação compreende, também, a inserção dos dados relativos à central e equipamento de diagnóstico em que se está comunicando.

Conforme apresenta o fluxograma da tela na Figura 9, ao fazer o upload, o usuário deve inserir os identificadores de *request* e *response*. Esses dados são suficientes para o início do programa. A interface visual utilizada na etapa inicial do projeto foi o *tkinter*, biblioteca nativa do Python para o desenvolvimento de interfaces gráficas. O uso dela garantiu ótimo desempenho da interface ao longo do projeto.

A primeira versão da interface é apresentada pela Figura 10. Ela contém exatamente o apresentado pelo fluxograma.

Figura 9 – Fluxograma da tela.

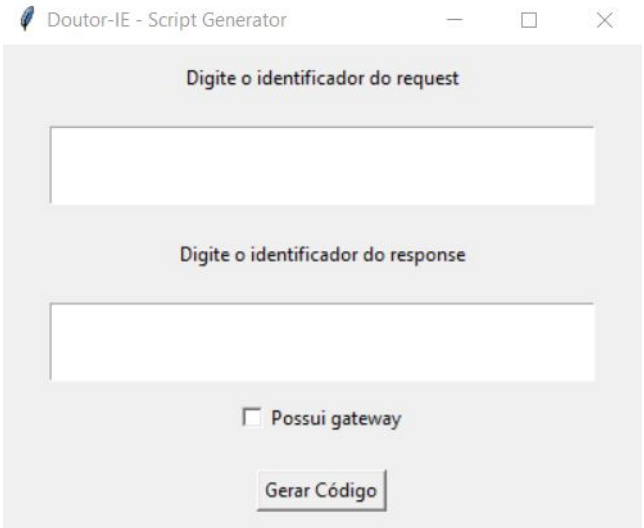


Fonte: Autor, 2023.

5.1.2 CONVERSÃO DO ARQUIVO BLF

O arquivo *Binary Logging Format* (BLF) é a extensão e tipo de arquivo utilizado pela ferramenta VECTOR de trabalho dos engenheiros da equipe. Portanto, é necessário que o programa seja capaz de interpretar esse tipo de arquivo. A biblioteca CAN, do

Figura 10 – Interface inicial do programa.



Fonte: Autor, 2023.

Python, possui uma série de funcionalidades implementadas pela empresa VECTOR e por isso foi utilizada no projeto.

Essa biblioteca contém a função de interpretação dos dados em formato BLF. No entanto, para que o usuário possa verificar se os dados convertidos estão corretos, o programa deve converter esse formato para ASCII.

Outra grande vantagem em realizar essa conversão é a possibilidade de gerar um arquivo único a partir de múltiplos arquivos. Isso permite que os arquivos convertidos tenham a totalidade de mensagens presentes no veículo, mesmo que o engenheiro tenha salvado em múltiplos arquivos durante a aquisição dos dados.

O passo-a-passo da conversão é apresentada no fluxograma da Figura 11.

Figura 11 – Fluxograma da conversão.

Fonte: Autor, 2023.

Até essa etapa, o único arquivo gerado foi o próprio log convertido.

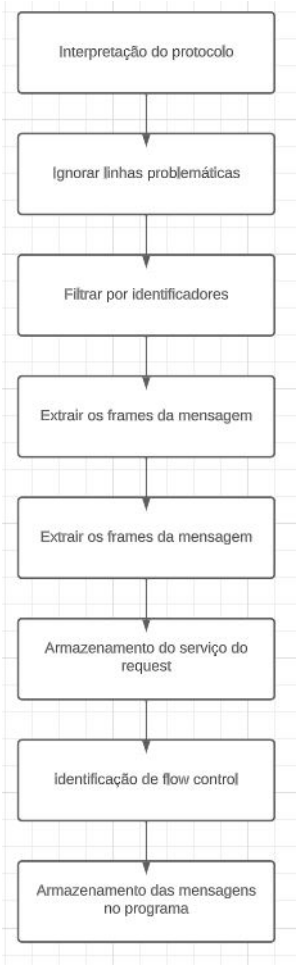
5.1.3 Interpretação do protocolo de transporte

Essa etapa contém o núcleo do programa. O fluxo das informações no programa é apresentado pelo fluxograma da Figura 12.

O programa desenvolvido foi modularizado em funções, portanto as tarefas apresentadas no fluxograma são desempenhadas por inúmeras funções.

Esta etapa do projeto, o núcleo, manteve-se praticamente igual durante todo o desenvolvimento uma vez que quebrado o protocolo e registradas as mensagens, os arquivos de texto a serem gerados podem ser manipulados futuramente.

Figura 12 – Fluxograma do programa.



Fonte: Autor, 2023.

O programa realiza o processamento linha por linha. A ideia é que embora o programa não faça parte da troca de mensanges no veículo, o comportamento do processamento das mensagens será muito similar. A cada mensagem que chega para a "central", o programa, é processada. Caso a linha seja problemática, contenha erros ou textos que não sejam parte de uma estrutura normal da mensagem, devem ser ignorados. Um exemplo de erro é apresentado na Figura 13.

Figura 13 – Mensagem de erro no log CAN.

```
0.046803 CAN 1 Status:chip status error active
0.047115 1  39             Rx   d 3 00 00 1B  Length = 0 BitCount = 77 ID = 57
```

Fonte: Autor, 2023.

Após a filtragem inicial das mensagens, a primeira condição do programa é garantir que o identificador do primeiro frame de mensagens seja o que foi inserido pelo usuário e que seja o identificador de *request*. Caso seja *response*, isso indica que o log perdeu algum *frame* anterior e por isso essa mensagem é ignorada.

Caso a mensagem passe por esses filtros, então passamos à extração da mensagem. As etapas descritas a seguir fazem parte da extração do protocolo de transporte (ISO TP) segundo as normas da rede CAN convencional.

Precisam-se considerar dois casos distintos para o *request*: Caso a mensagem seja menor ou igual a sete *bytes* a mensagem começa na posição um do *frame* – como pode ser visto na Figura de exemplo 14. O *byte* 0x03, significa que aquele *frame* possui 3 *bytes* de comprimento. Portanto, as mensagens subsequentes serão de resposta da central.

Figura 14 – *Frame* com 3 bytes de tamanho.

```
616.662077 1  18DA0EF1x       Rx   d 8 03 22 26 10 00 00 00 00  Length = 0 BitCount = 140 ID = 416943857x
```

Fonte: Autor, 2023.

Por outro lado, a mensagem pode ter tamanho maior que um frame e, por isso, possuir um protocolo de transporte mais extenso. Na Figura 15 podemos ver que os dois *bytes* iniciais são 10 0B. Segundo a camada de enlace CAN, esse é o formato para mensagens com múltiplos *frames*. Logo, o tamanho da mensagem será 0x0B, equivalente a 11 em decimal. Portanto, as mesagens subsequentes, embora contenham o identificador de *request* não serão novas mensagens. Dessa forma, o programa deve aguardar o fim das mensagens de *request* para receber e processar as mensagens de *response*.

Figura 15 – *Frame* com 11 bytes de tamanho.

```
182.149798 1  18DA0EF1x       Rx   d 8 10 0B 22 26 60 26 61 26  Length = 0 BitCount = 135 ID = 416943857x
```

Fonte: Autor, 2023.

Sabendo que após a mensagem de pergunta iniciarão as respostas, precisa-se

registrar qual a pergunta que foi feita. Para isso, utilizou-se um dicionário. Neste, a chave foi composta por um *hash*, oriundo dos *bytes* iniciais da mensagem de *request*.

Para realizar também a ordenação das mensagens que serão processadas, a biblioteca *OrderedDict* do Python foi utilizada, uma vez que é capaz de fazer a ordenação automática ao realizar a inserção de mais elementos no dicionário.

De modo geral, o *hash* é composto por até 3 *bytes* de *request*. Futuramente isso será alterado para atender necessidades específicas da equipe.

Após a conclusão das requisições, a central inicia a resposta. Como é o dado mais importante, possui diferenças em relação ao tratamento dos dados de pergunta. Com a chegada da primeira mensagem de resposta, armazena-se o tamanho da mensagem em sua totalidade. Conforme os chamados *consecutive frames* são interpretados pelo programa, diminui-se o tamanho do tamanho total. Isso é bastante importante, uma vez que os frames tem sempre o tamanho de 8 *bytes* – nos *consecutive frames* são 7 *bytes* de dados e um de tamanho – e caso a mensagem termine com apenas mais 2 *bytes* do *frame*, então os demais são descartados, uma vez que são utilizados apenas para completar a mensagem.

Tanto para o *request* quanto para o *response* deve-se tomar cuidado com a mensagem de controle de fluxo de dados. Essa mensagem é utilizada para configurações de tempo de resposta, e não carrega nenhum dado. Portanto, quando ela ocorre está relacionada com a arbitragem de tempo da central mas não carrega dados relacionados à mensagem.

Nesta etapa, o núcleo das mensagens foi extraído. Agora estão armazenados em um dicionário, ordenados por serviços e parâmetros.

5.1.4 MONTAGEM DA MENSAGEM EM CAPL

Dada a modularidade do sistema, existem arquivos distintos para a realização de cada módulo.

Para a montagem da mensagem em CAPL foi desenvolvido pela equipe um documento padrão de montagem das centrais. A partir desse documento, os códigos abaixo foram desenvolvidos.

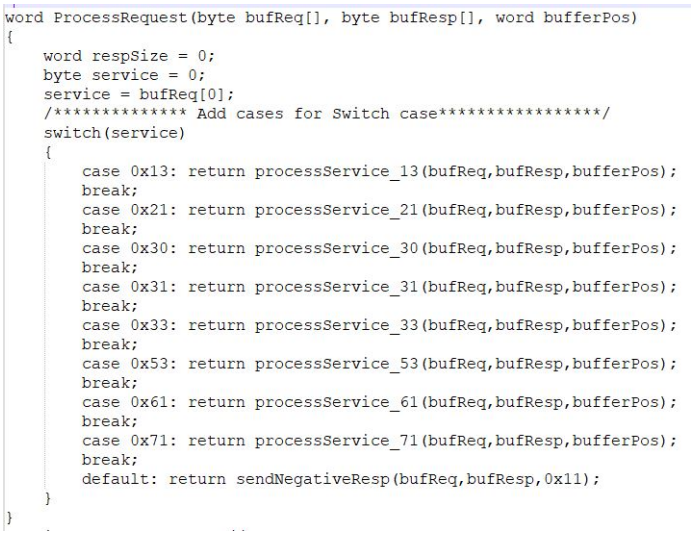
O resultado final opera da seguinte maneira: Uma função principal, no arquivo *mainFile.can* utiliza a função *switch* e *case* da linguagem C para chamar os respectivos serviços. Os serviços listados nessa função condicional são exatamente os existentes no arquivo que está sendo processado.

Caso o arquivo possua apenas dois serviços, então dois serviços aparecerão.

Caso o arquivo possua dez serviços, dez serviços aparecerão. Na Figura 16 mostra-se a estrutura descrita.

Da mesma forma funciona para cada serviço. Cada função chamada anteriormente possui uma mesma estrutura de *switch* e *case* para cada PID existente no log.

Figura 16 – Função presente no arquivo *mainFIle.can*.

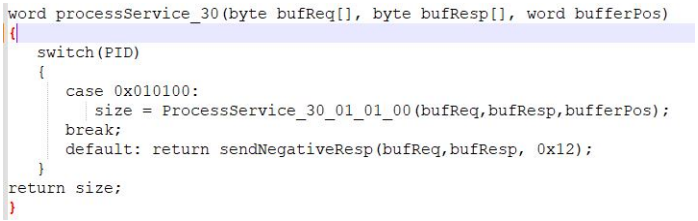


```
word ProcessRequest(byte bufReq[], byte bufResp[], word bufferPos)
{
    word respSize = 0;
    byte service = 0;
    service = bufReq[0];
    /************** Add cases for Switch case*****************/
    switch(service)
    {
        case 0x13: return processService_13(bufReq,bufResp,bufferPos);
        break;
        case 0x21: return processService_21(bufReq,bufResp,bufferPos);
        break;
        case 0x30: return processService_30(bufReq,bufResp,bufferPos);
        break;
        case 0x31: return processService_31(bufReq,bufResp,bufferPos);
        break;
        case 0x33: return processService_33(bufReq,bufResp,bufferPos);
        break;
        case 0x53: return processService_53(bufReq,bufResp,bufferPos);
        break;
        case 0x61: return processService_61(bufReq,bufResp,bufferPos);
        break;
        case 0x71: return processService_71(bufReq,bufResp,bufferPos);
        break;
        default: return sendNegativeResp(bufReq,bufResp,0x11);
    }
}
```

Fonte: Autor, 2023.

Da mesma forma, apenas aparecem os *PID's* existentes no log daquela central. A Figura 17 apresenta a estrutura descrita.

Figura 17 – Função presente nos arquivos de serviços.



```
word processService_30(byte bufReq[], byte bufResp[], word bufferPos)
{
   switch(PID)
   {
      case 0x010100:
         size = ProcessService_30_01_01_00(bufReq,bufResp,bufferPos);
      break;
      default: return sendNegativeResp(bufReq,bufResp, 0x12);
   }
return size;
}
```

Fonte: Autor, 2023.

A estrutura de cada função funciona da seguinte forma: Cada *byte* da mensagem é armazenado em uma posição do vetor. Esse vetor possui como valor final no seu índice N, que representa o tamanho da mensagem. Para indicar o fim da função retorna-se N+1, calculado pelo programa desenvolvido. Esse valor indica o tamanho do *buffer* de resposta. A Figura 18 apresenta a estrutura descrita.

Essa é a estrutura básica dos arquivos gerados. As informações necessárias para a construção dos arquivos apresentados foi oriunda do dicionário criado na fase de extração das mensagens.

As mensagens montadas não são diretamente armazenadas em arquivos. Elas são primeiramente armazenadas em variáveis. Isso é feito para manter a modularidade do sistema, uma vez que cada etapa possui a sua função. A lista que é gerada com

Figura 18 – Função para cada PID.

```
word ProcessService_21_14_01(byte bufReq[], byte bufResp[], word bufferPos)
{
   bufResp[0] = 0x61;
   bufResp[1] = 0x14;
   bufResp[2] = 0x30;
   bufResp[3] = 0x38;
   bufResp[4] = 0x30;
   bufResp[5] = 0x54;
   bufResp[6] = 0x41;
   bufResp[7] = 0xAA;
   return 8;
}
```

Fonte: Autor, 2023.

os dados possui em cada termo o conteúdo de uma linha. Assim, na fase seguinte os dados já estarão ordenados e prontos para serem escritos nos arquivos.

5.1.5 GERAÇÃO/SALVAMENTO DOS DIFERENTES ARQUIVOS POR SERVIÇO

Essa é a última etapa do programa. Nela, chamamos o arquivo com suas respectivas funções para a geração dos arquivos finais. As estruturas de repetição, como a sequência de *bytes*, foi gerada anteriormente. Para a geração, o dicionário é passado como parâmetro para a função final. Na Figura 19 podemos ver quão simplificado é o resultado final, considerando todos os ajustes realizados anteriormente.

Figura 19 – Função de escrita dos arquivos.

```
def writeToFile(dict):
    for item in sorted(dict):
      results = open('_process_service_'+ item[0:2].upper() +'.cin','a')
      for line in dict[item]:
          results.write(line)
          results.write('\n')
      results.close()
```

Fonte: Autor, 2023.

5.1.6 NOVAS FUNCIONALIDADES

Sempre em contato com a equipe, foram relatados alguns desejos para que o resultado final fosse ainda mais otimizado. O primeiro deles, foi realizar o gerenciamento de sessão. Isso porque, tanto na norma UDS e KWP o serviço 10 representa a sessão em que o equipamento de diagnóstico se encontra. Logo, a cada troca de sessão deve ser registrado no programa, e no início de cada função verificar se a sessão em que se encontra está correta. O armazenamento da condição é feito através de *flags* que conservam o estado anterior, até a próxima troca. Logo, durante a montagem

da mensagem foi adicionada uma etapa a mais, que é a verificação da sessão que o equipamento se encontra. Mais detalhes poderão ser vistos na seção de análise dos resultados.

Outra reclamação da equipe foi que existem serviços proprietários, que não operam dentro das normas, que são bastante desgastantes para o engenheiro desenvolver. No caso particular, foi identificado que o serviço 0xA8, das montadoras Toyota e Lexus era bastante extenso e complexo para que o desenvolvimento individual fosse realizado para toda central. Com isso, foi implementado a geração desse serviço. Em suma, a diferença principal em relação à estrutura original do programa é que esse serviço não respeita a regra de *request* e *response* – Ou seja, não segue a norma CAN segundo o ISO TP. Nesse caso particular, a central faz uma pergunta, e recebe como resposta múltiplos pacotes de dados de PIDs diferentes.

Conforme pode ser visto nos códigos em apêndice, ao identificar um *request* do serviço 0xA8 uma *flag* é habilitada e armazena essa informação. Dessa forma, quando houver um *request* cujo serviço seja equivalente, então o programa registrará em um dicionário à parte dos demais, exclusivo para esse serviço. Isso acarretará em adição de funções no arquivo de escrita, uma vez que a estrutura em CAPL será bastente diferente das demais. Os resultados de tal alteração serão apresentados na seção seguinte.

5.2 PROGRAMA DE COMPARAÇÃO

Por fim, após a geração dos arquivos estar concluída é necessário comparar diferentes centrais e obter um relatório que descreva a similaridade entre centrais. O arquivo gerado na etapa anterior possui 4 colunas. Nelas são armazenadas as seguintes informações: Serviço, PID, tamanho da pergunta e tamanho da resposta. Ao realizar o processo em duas centrais distintas, obteremos dois arquivos. Com o uso da biblioteca Pandas, em Python, é feita a comparação entre os arquivos. São considerados compatíveis as linhas que contém as mesmas informações. Nesse caso, no relatório final essa linha aparecerá apenas uma vez, acompanhada do texto "compatível". Caso os valores de serviço e PID sejam iguais, mas o tamanho de pergunta ou resposta seja diferente, o relatório mostrará as informações como "Incompatível", e irá exibir os tamanhos incompatíveis separados por uma barra. Caso seja encontrado o serviço e PID em apenas uma das tabelas, então o texto "Primeira entrada"e "Segunda entrada"aparecerá. Os métodos utilizados para realizar a comparação podem ser vistos no apêndice do documento. Por fim, serão contados quantas vezes cada um dos eventos anteriores aconteceu, e será exibida uma tabela com os percentuais de compatibilidade entre as centrais.

5.3 O ARQUIVO EXECUTÁVEL

Para disponibilizar o programa para a equipe, foi utilizada a biblioteca *auto-py-to-exe* do Python. Com essa biblioteca, basta fazer o upload do arquivo principal e apresentar o diretório que contém os demais arquivos para que seja criado um arquivo executável, que contenha todas as dependências para o uso dos desenvolvedores. Foi adicionado também uma imagem de início, que possui a função de indicar o início do programa. Todos esses ajustes são realizados pela biblioteca apresentada.

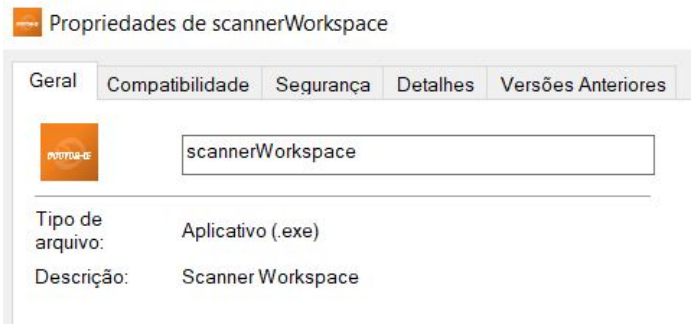
6 ANÁLISE DOS RESULTADOS

Por fim, neste capítulo são apresentados os resultados dos requisitos apurados no Capítulo 4 e das duas funcionalidades que compõem a solução.

6.1 O ARQUIVO GERADO

Após a conclusão dos códigos, o arquivo executável gerado é apresentado pela Figura 20. Como pode ser observado, a aplicação passou a ser chamada de *Scanner Workspace* por englobar mais de uma função na mesma aplicação. Esse arquivo foi distribuído aos engenheiros ao longo do desenvolvimento para que os mesmos realizassem os testes. A versão apresentada é a última versão, estável e funcional para o uso da equipe.

Figura 20 – Aplicativo *scannerWorkspace*.



Fonte: Autor, 2023.

Ao executar a aplicação, ela leva alguns segundos para abrir sua interface. Isso ocasionava alguma dificuldade por parte da equipe, então foi adicionada uma tela de início – a *splash screen* que aparece logo que se inicia a aplicação – que pode ser observada na figura 21.

A aplicação leva cerca de dez segundos para iniciar, sendo um tempo em conformidade com os requisitos apresentados anteriormente.

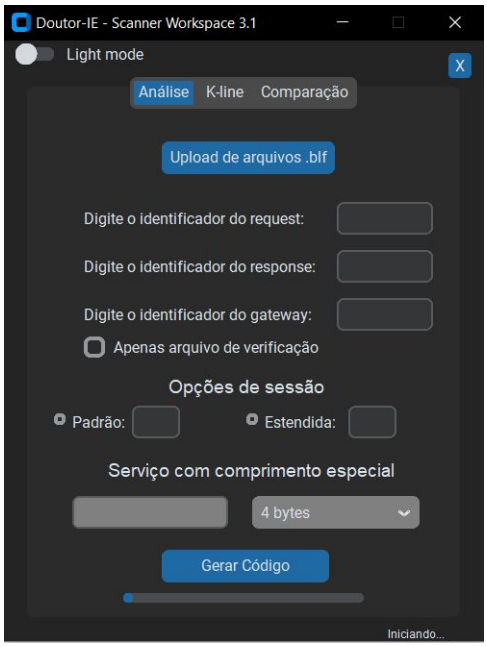
6.2 A APLICAÇÃO

A aplicação é composta por 3 abas: Análise, K-Line e Comparação. As abas Análise e Comparação foram desenvolvidas pelo autor e são o escopo do trabalho. A tela inicial da aplicação é a apresentada pela Figura 22. Na parte superior, o usuário pode fazer o *upload* do arquivo para ser analisado. A tela de *upload* é a tela padrão do sistema operacional Windows. O diretório da pasta de *upload* é o mesmo da pasta do arquivo executável.

Figura 21 – Tela de carregamento.

Fonte: Autor, 2023.

Figura 22 – Interface do aplicativo.



Fonte: Autor, 2023.

Logo abaixo o usuário pode inserir os identificadores de *request* e *response*. Esses são campos essenciais para o funcionamento do programa. Chamamos de *gateway* o comportamento de sistemas que endereçam as centrais pelo primeiro *byte* do *frame* das mensagens. Logo, caso exista, o usuário pode inserir um ou mais iden-

tificadores de *gateway*. Caso exista, a mensagem será tratada considerando essa condição.

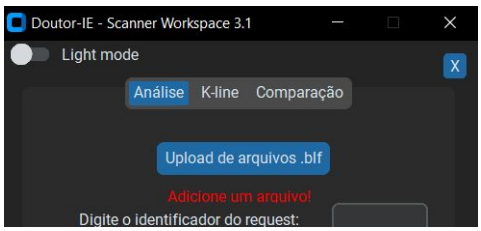
Abaixo há um *checkbox* que pode ser marcado caso o usuário queira apenas os arquivos para obter o relatório de comparação entre as centrais.

Há também o espaço "Opções de sessão"que é um cadastro opcional, mas faz com que o programa gere o controle de sessão na estrutura dos códigos gerados.

O campo "Serviço com comprimento especial"refere-se ao tamanho do *hash* utilizado para segregar o PID. Através de um menu de seleção o usuário pode escolher até 7 *bytes* como tamanho máximo. Isso fez-se necessário pois existem serviços da norma que o *byte* de controle é o último da mensagem de *request*. Sem essa condição, eram tratados como iguais mensagens que na prática eram diferentes apenas para além da terceira posição.

Caso os campos obrigatórios sejam preenchidos de forma incorreta, mensagens de erro são exibidas, conforme apresenta a Figura 23. Esse tratamento impede que erros ocorram no *backend* da aplicação.

Figura 23 – Mensagem de erro.



Fonte: Autor, 2023.

Por fim, uma barra de carregamento foi adicionada e texto no canto inferior direito informando qual o estado o programa se encontra.

Ao preencher todos os dados corretamente, basta apertar sobre "Gerar código"e aguardar a geração dos arquivos.

6.3 EXECUTANDO O PROGRAMA

Usaremos como exemplo a conversão de um *log* da central do motor de um Honda Civic 2010. No exemplo, o desenvolvedor inseriu os identificadores de *request*, *response* e as informações de sessão. A Figura 24 apresenta esses dados.

Clica-se em "Gerar código"e aguarda-se o fim da execução. Em conjunto com a conclusão do processo, a barra de carregamento é preenchida e o texto "Concluído"é apresentado na tela. No mesmo instante, o diretório com todos os arquivos criados é aberto, conforme apresentado na Figura 25.

Figura 24 – Informando os dados da central.

Fonte: Autor, 2023.

Figura 25 – Pasta com os arquivos gerados.

Nome	Status	Data de modificação	Tipo
_process_service_2E.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_2F.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_3E.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_10.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_19.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_22.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_27.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_31.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_service_A4.cin	✓	29/10/2023 18:31	Arquivo CIN
_process_supported.cin	✓	29/10/2023 18:31	Arquivo CIN
log_pidSize	✓	29/10/2023 18:31	Arquivo CSV
mainFile.can	✓	29/10/2023 18:31	Arquivo CAN
negativeReponse	✓	29/10/2023 18:31	Documento de Te...

Fonte: Autor, 2023.

O arquivo principal, *mainFile.can* contém as chamadas de função para os demais arquivos. Todos os arquivos gerados são incluídos no arquivo principal, conforme apresenta a Figura 26. Graças à ordenação realizada na programação, os serviços são listados em ordem crescente, permitindo fácil navegação pelo código.

A Figura 27 apresenta a estrutura gerada para o serviço 0x22. Para esse veículo o desenvolvedor, antes do *scannerWorkspace*, teria que realizar a cópia dos 281 *bytes* de forma manual. Na imagem foram ocultadas as linhas intermediárias.

Para esse serviço, a Figura 28 apresenta a quantidade de PIDs para essa central.

Figura 26 – Arquivo principal.

```
/*@!Encoding:1252*/
includes
{
    #include "..\..\..\0-Painel_analise_modulos\variavel_analise_parametros.cin"
    #include "..\..\..\1-libs\utils.cin"
    #include "..\..\..\1-libs\tp_lib_2.cin"
    #include "_process_supported.cin"
    #include "_process_service_10.cin"
    #include "_process_service_19.cin"
    #include "_process_service_22.cin"
    #include "_process_service_27.cin"
    #include "_process_service_2E.cin"
    #include "_process_service_2F.cin"
    #include "_process_service_31.cin"
    #include "_process_service_3E.cin"
    #include "_process_service_A4.cin"
}

word ProcessRequest(byte bufReq[], byte bufResp[], word bufferPos)
{
    word respSize = 0;
    byte service = 0;
    service = bufReq[0];
    /************** Add cases for Switch case*****************/
    switch(service)
    {
        case 0x10: return processService_10(bufReq,bufResp,bufferPos);
        break;
        case 0x19: return processService_19(bufReq,bufResp,bufferPos);
        break;
        case 0x22: return processService_22(bufReq,bufResp,bufferPos);
        break;
        case 0x27: return processService_27(bufReq,bufResp,bufferPos);
        break;
        case 0x2E: return processService_2E(bufReq,bufResp,bufferPos);
        break;
        case 0x2F: return processService_2F(bufReq,bufResp,bufferPos);
        break;
        case 0x31: return processService_31(bufReq,bufResp,bufferPos);
        break;
        case 0x3E: return processService_3E(bufReq,bufResp,bufferPos);
        break;
        case 0xA4: return processService_A4(bufReq,bufResp,bufferPos);
        break;
        default: return sendNegativeResp(bufReq,bufResp,0x11);
    }
    return 0;
}
On DiagRequest ECU.*{ }//Replace ECU by the node's name
```

Fonte: Autor, 2023.

Figura 27 – Mensagem com tamanho longo.

```
89  word ProcessService_22_26_10_26(byte bufReq[], byte bufResp[], word bufferPos)
90  {
91     if((bufferPos !=11)) return sendNegativeResp(bufReq,bufResp,0x13/*UDS*/);
92     if(currentSession != 0x01 && currentSession != 0x03) return sendNegativeResp(bufReq, bufResp, 0x7F/*UDS*/);
93
94     bufResp[0] = 0x62;
95     bufResp[1] = 0x26;
374    bufResp[280] = 0x00;
375    return 281;
376 }
377
```

Fonte: Autor, 2023.

Ao total são apresentados 18 PIDs distintos. Esse é um dos grandes benefícios do programa. Ele faz de forma assertiva e em segundos o trabalho que um engenheiro da equipe levava dias, quiçá semanas dado o tamanho da central.

6.3.1 EXEMPLO DE CONVERSÃO

Para ilustrar ainda mais a qualidade dos resultados obtidos, apresenta-se o resultado do método implementado para o serviço proprietário 0xA8 das montadoras

Figura 28 – *Switch e Case* do serviço 0x22.

```
switch(PID)
{
   case 0x2600:
      size = ProcessService_22_26_00(bufReq,bufResp,bufferPos);
   break;
   case 0x2610:
      size = ProcessService_22_26_10(bufReq,bufResp,bufferPos);
   break;
   case 0x261026:
      size = ProcessService_22_26_10_26(bufReq,bufResp,bufferPos);
   break;
   case 0x2611:
      size = ProcessService_22_26_11(bufReq,bufResp,bufferPos);
   break;
   case 0x2612:
      size = ProcessService_22_26_12(bufReq,bufResp,bufferPos);
   break;
   case 0x262026:
      size = ProcessService_22_26_20_26(bufReq,bufResp,bufferPos);
   break;
   case 0x2630:
      size = ProcessService_22_26_30(bufReq,bufResp,bufferPos);
   break;
   case 0x2660:
      size = ProcessService_22_26_60(bufReq,bufResp,bufferPos);
   break;
   case 0x266026:
      size = ProcessService_22_26_60_26(bufReq,bufResp,bufferPos);
   break;
   case 0x2661:
      size = ProcessService_22_26_61(bufReq,bufResp,bufferPos);
   break;
   case 0x266226:
      size = ProcessService_22_26_62_26(bufReq,bufResp,bufferPos);
   break;
   case 0x2663:
      size = ProcessService_22_26_63(bufReq,bufResp,bufferPos);
   break;
   case 0x2668:
      size = ProcessService_22_26_68(bufReq,bufResp,bufferPos);
   break;
   case 0x266F26:
      size = ProcessService_22_26_6F_26(bufReq,bufResp,bufferPos);
   break;
   case 0x2673:
      size = ProcessService_22_26_73(bufReq,bufResp,bufferPos);
   break;
   case 0x267F:
      size = ProcessService_22_26_7F(bufReq,bufResp,bufferPos);
   break;
   case 0xE814:
      size = ProcessService_22_E8_14(bufReq,bufResp,bufferPos);
   break;
   case 0xF190:
      size = ProcessService_22_F1_90(bufReq,bufResp,bufferPos);
   break;
   default:
```

Fonte: Autor, 2023.

Toyota e Lexus.

A Figura 29 apresenta o início dos vetores de dados do serviço 0xA8. A Figura 30 contém a continuação da primeira imagem. São 82 vetores contendo até 62 *bytes* cada. Dados levantados pela equipe indicam que antes do desenvolvimento do projeto o tempo de desenvolvimento apenas para gerar a simulação da central era de 15 dias trabalhados. Além da morosidade no processo, ainda era absolutamente sujeito a erros.

Cada *byte* está sujeito ao erro humano. Centrais podem conter até dez mil *bytes*! Portanto, são até dez mil erros evitados com o uso do *scannerWorkspace*.

Figura 29 – Vetores do serviço 0xA8.

```
word ApplicationCar(char application[], byte dataReturn[])
{
    long result = 0;
    byte A801_0[15] = { 0xE8, 0x01, 0x00, 0x04, 0xFF, 0xFF,
    byte A801_1[62] = { 0xE8, 0x01, 0x03, 0x37, 0xFF, 0xFF,
    byte A801_10[49] = { 0xE8, 0x01, 0x66, 0x2D, 0xFF, 0xFF,
    byte A801_11[63] = { 0xE8, 0x01, 0x67, 0x3B, 0xFF, 0xFF,
    byte A801_12[58] = { 0xE8, 0x01, 0x68, 0x36, 0xFF, 0xFF,
    byte A801_13[49] = { 0xE8, 0x01, 0x6A, 0x2D, 0xFF, 0xFF,
    byte A801_14[63] = { 0xE8, 0x01, 0x6B, 0x3B, 0xFF, 0xFF,
    byte A801_15[58] = { 0xE8, 0x01, 0x6C, 0x36, 0xFF, 0xFF,
    byte A801_16[49] = { 0xE8, 0x01, 0x6E, 0x2D, 0xFF, 0xFF,
    byte A801_17[63] = { 0xE8, 0x01, 0x6F, 0x3B, 0xFF, 0xFF,
    byte A801_18[58] = { 0xE8, 0x01, 0x70, 0x36, 0xFF, 0xFF,
    byte A801_19[14] = { 0xE8, 0x01, 0x80, 0x04, 0xFF, 0xFF,
    byte A801_2[63] = { 0xE8, 0x01, 0x06, 0x02, 0xFF, 0xFF,
    byte A801_20[63] = { 0xE8, 0x01, 0xA5, 0x3B, 0xFF, 0xFF,
    byte A801_21[55] = { 0xE8, 0x01, 0xA6, 0x33, 0xFF, 0xFF,
    byte A801_22[63] = { 0xE8, 0x01, 0xAD, 0x3B, 0xFF, 0xFF,
    byte A801_23[55] = { 0xE8, 0x01, 0xAE, 0x33, 0xFF, 0xFF,
    byte A801_24[63] = { 0xE8, 0x01, 0xB1, 0x3B, 0xFF, 0xFF,
    byte A801_25[63] = { 0xE8, 0x01, 0xB2, 0x3B, 0xFF, 0xFF,
    byte A801_26[54] = { 0xE8, 0x01, 0xB3, 0x32, 0xFF, 0xFF,
    byte A801_27[60] = { 0xE8, 0x01, 0xB4, 0x38, 0xFF, 0xFF,
    byte A801_28[60] = { 0xE8, 0x01, 0xB5, 0x38, 0xFF, 0xFF,
    byte A801_29[60] = { 0xE8, 0x01, 0xB6, 0x38, 0xFF, 0xFF,
    byte A801_3[61] = { 0xE8, 0x01, 0x0A, 0x01, 0xFF, 0x0B,
    byte A801_30[60] = { 0xE8, 0x01, 0xB7, 0x38, 0xFF, 0xFF,
    byte A801_31[32] = { 0xE8, 0x01, 0xB8, 0x1C, 0xFF, 0xFF,
    byte A801_32[54] = { 0xE8, 0x01, 0xB9, 0x32, 0xFF, 0xFF,
    byte A801_33[60] = { 0xE8, 0x01, 0xBA, 0x38, 0xFF, 0xFF,
    byte A801_34[60] = { 0xE8, 0x01, 0xBB, 0x38, 0xFF, 0xFF,
    byte A801_35[60] = { 0xE8, 0x01, 0xBC, 0x38, 0xFF, 0xFF,
    byte A801_36[60] = { 0xE8, 0x01, 0xBD, 0x38, 0xFF, 0xFF,
    byte A801_37[38] = { 0xE8, 0x01, 0xBE, 0x1C, 0xFF, 0xFF,
    byte A801_38[54] = { 0xE8, 0x01, 0xC1, 0x32, 0xFF, 0xFF,
    byte A801_39[60] = { 0xE8, 0x01, 0xC2, 0x38, 0xFF, 0xFF,
    byte A801_4[60] = { 0xE8, 0x01, 0x22, 0x06, 0xFF, 0xFF,
    byte A801_40[60] = { 0xE8, 0x01, 0xC3, 0x38, 0xFF, 0xFF,
    byte A801_41[60] = { 0xE8, 0x01, 0xC4, 0x38, 0xFF, 0xFF,
    byte A801_42[60] = { 0xE8, 0x01, 0xC5, 0x38, 0xFF, 0xFF,
    byte A801_43[32] = { 0xE8, 0x01, 0xC6, 0x1C, 0xFF, 0xFF,
    byte A801_44[54] = { 0xE8, 0x01, 0xC7, 0x32, 0xFF, 0xFF,
    byte A801_45[60] = { 0xE8, 0x01, 0xC8, 0x38, 0xFF, 0xFF,
    byte A801_46[60] = { 0xE8, 0x01, 0xC9, 0x38, 0xFF, 0xFF,
    byte A801_47[60] = { 0xE8, 0x01, 0xCA, 0x38, 0xFF, 0xFF,
    byte A801_48[60] = { 0xE8, 0x01, 0xCB, 0x38, 0xFF, 0xFF,
    byte A801_49[32] = { 0xE8, 0x01, 0xCC, 0x1C, 0xFF, 0xFF,
    byte A801_5[25] = { 0xE8, 0x01, 0x25, 0x09, 0xFF, 0xFF,
    byte A801_50[49] = { 0xE8, 0x01, 0xCD, 0x2D, 0xFF, 0xFF,
    byte A801_51[56] = { 0xE8, 0x01, 0xCE, 0x34, 0xFF, 0xFF,
    byte A801_52[51] = { 0xE8, 0x01, 0xCF, 0x2F, 0xFF, 0xFF,
    byte A801_53[48] = { 0xE8, 0x01, 0xD0, 0x2C, 0xFF, 0xFF,
    byte A801_54[48] = { 0xE8, 0x01, 0xD1, 0x2C, 0xFF, 0xFF,
    byte A801_55[49] = { 0xE8, 0x01, 0xD2, 0x2D, 0xFF, 0xFF,
    byte A801_56[56] = { 0xE8, 0x01, 0xD3, 0x34, 0xFF, 0xFF,
```

Fonte: Autor, 2023.

6.4 COMPARAÇÃO DE CENTRAIS

Para a comparação entre centrais, a Figura 31 apresenta a interface do programa. O primeiro botão permite o *upload* de tabelas presente no banco de dados da empresa. O segundo botão permite o *upload* da tabela que ser quer comparar, permitindo a comparação de um com múltiplos arquivos. No campo seguinte pode ser inserido o nome do arquivo do relatório que será gerado.

As tabelas geradas se comportam como a impressão digital de uma central. Ao longo dos testes, as características apresentadas foram sufiecientes para identificar as centrais que realmente são compatíveis.

O arquivo chamado *pidSize*, a impressão digital, é apresentado pela Figura 32.

Figura 30 – Vetores do serviço 0xA8(2).

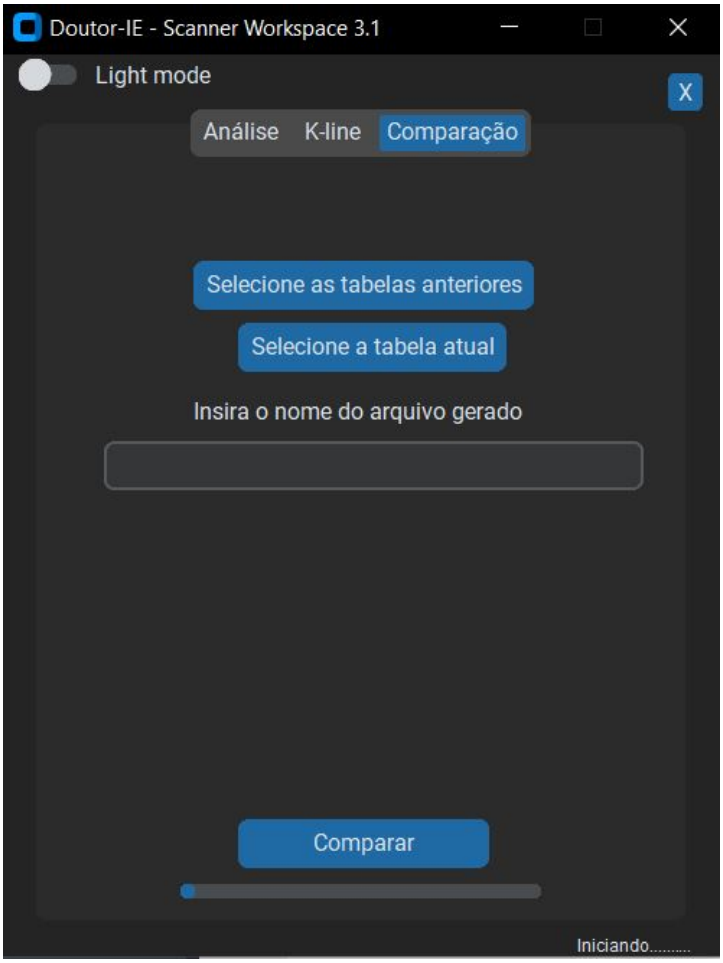
```
66      byte A801_56[56] = { 0xE8, 0x01, 0xD3, 0x
67      byte A801_57[51] = { 0xE8, 0x01, 0xD4, 0x
68      byte A801_58[48] = { 0xE8, 0x01, 0xD5, 0x
69      byte A801_59[48] = { 0xE8, 0x01, 0xD6, 0x
70      byte A801_6[58] = { 0xE8, 0x01, 0x61, 0x3
71      byte A801_60[49] = { 0xE8, 0x01, 0xD7, 0x
72      byte A801_61[56] = { 0xE8, 0x01, 0xD8, 0x
73      byte A801_62[51] = { 0xE8, 0x01, 0xD9, 0x
74      byte A801_63[48] = { 0xE8, 0x01, 0xDA, 0x
75      byte A801_64[54] = { 0xE8, 0x01, 0xDB, 0x
76      byte A801_65[44] = { 0xE8, 0x01, 0xE1, 0x
77      byte A801_66[29] = { 0xE8, 0x01, 0xE5, 0x
78      byte A801_67[49] = { 0xE8, 0x01, 0xF0, 0x
79      byte A801_68[56] = { 0xE8, 0x01, 0xF1, 0x
80      byte A801_69[51] = { 0xE8, 0x01, 0xF2, 0x
81      byte A801_7[49] = { 0xE8, 0x01, 0x62, 0x2
82      byte A801_70[48] = { 0xE8, 0x01, 0xF3, 0x
83      byte A801_71[48] = { 0xE8, 0x01, 0xF4, 0x
84      byte A801_72[49] = { 0xE8, 0x01, 0xF5, 0x
85      byte A801_73[56] = { 0xE8, 0x01, 0xF6, 0x
86      byte A801_74[51] = { 0xE8, 0x01, 0xF7, 0x
87      byte A801_75[48] = { 0xE8, 0x01, 0xF8, 0x
88      byte A801_76[48] = { 0xE8, 0x01, 0xF9, 0x
89      byte A801_77[49] = { 0xE8, 0x01, 0xFA, 0x
90      byte A801_78[56] = { 0xE8, 0x01, 0xFB, 0x
91      byte A801_79[51] = { 0xE8, 0x01, 0xFC, 0x
92      byte A801_8[63] = { 0xE8, 0x01, 0x63, 0x3
93      byte A801_80[48] = { 0xE8, 0x01, 0xFD, 0x
94      byte A801_81[48] = { 0xE8, 0x01, 0xFE, 0x
95      byte A801_9[58] = { 0xE8, 0x01, 0x64, 0x3
96      byte NumberOfResponses_A801[1] = {0x52};
```

Fonte: Autor, 2023.

São esses os arquivos criados pelo *scannerWorkspace* e que devem ser carregados na etapa de comparação.

Após carregar os arquivos e apertar em "Comparar", é gerado um relatório no formato .xlsx. O relatório gerado pela comparação de duas centrais é apresentado na Figura 33. No relatório são apresentadas as condições de cada um dos PIDs. À direita, são apresentados os percentuais da segunda entrada em relação a todos os PIDs individuais analisados.

Figura 31 – Interface de comparação do *scannerWorkspace*



Fonte: Autor, 2023.

Figura 32 – Arquivo *pidSize*.

```
ID,Service,PID,requestSize,ResponseSize
18da0ef1,0,0,0,0
18daf10e,1,0,0,0
X,22,2600,3,13
X,22,F190,3,24
X,19,0208,3,3
X,19,0204,3,3
X,19,0A,2,603
X,22,2673,3,57
X,22,2630,3,6
X,22,262026,11,281
X,22,2610,3,57
X,22,2661,3,57
X,A4,10,2,2
X,22,261026,11,281
X,22,266026,11,281
X,22,266226,11,281
X,22,266F26,9,157
X,10,03,2,2
X,2F,013803,6,4
X,2F,013800,4,4
X,10,01,2,2
X,22,2668,3,57
X,3E,80,2,1
X,2F,014E03,6,4
X,22,267F,3,13
X,22,2612,3,57
X,2F,014E00,4,4
X,2F,014F03,6,4
```

Fonte: Autor, 2023.

Figura 33 – Relatório de comparação.

	A	B	C	D	E	F	G	H	J	K	L	M
1		**ID**	**Service**	**PID**	**requestSize**	**ResponseSize**	**Comparacao**	**Obs**				
2	**0**	18da0ef1	0	0	0	0	Compativel		**Compativel**	**Incompativel**	**Somente Primeira entrada**	**Somente Segunda entrada**
3	**1**	18daf10e	1	0	0	0	Compativel		85%	12%		2%
4	**2**	X	10	01	2	2	Compativel		41 PIDs	6 PIDs	1 PIDs	1 PIDs
5	**3**	X	10	03	2	2	Compativel					
6	**4**	X	19	0204	3	3	Compativel					
7	**6**	X	19	0208	3 / 4	3 / 3	Incompativel					
8	**8**	X	19	0a	2 / 2	603 / 605	Incompativel					
9	**9**	X	22	26	2	2	Segunda entrada					
10	**10**	X	22	2600	3	13	Compativel					
11	**11**	X	22	2610	3	57	Primeira entrada					
12	**13**	X	22	261026	9 / 11	281 / 281	Incompativel					
13	**14**	X	22	2611	3	57	Compativel					
14	**15**	X	22	2612	3	57	Compativel					
15	**16**	X	22	262026	11	281	Compativel					
16	**17**	X	22	2630	3	6	Compativel					
17	**19**	X	22	2660	3 / 3	57 / 59	Incompativel					

Fonte: Autor, 2023.

7 CONCLUSÕES E PERSPECTIVAS

7.1 CONCLUSÃO

Os requisitos levantados durante o desenvolvimento do projeto foram de suma importância para o êxito do mesmo. As entrevistas com a equipe de engenharia e desenvolvimento proporcionaram o pleno entendimento das necessidades e posteriormente a efetiva implementação dos programas. Isso fez com que o produto final fosse capaz de atender as necessidades dos mesmos.

Conforme apresentado, a metodologia de desenvolvimento com entregas periódicas permitiu que ajustes e novas funcionalidades fossem adicionadas ao projeto durante o seu desenvolvimento, ocasionando uma melhor adequação às necessidades apresentadas pela equipe.

Ao nível do setor, a implementação do novo método de geração de simulações utilizando o *scannerWorkspace* permitiu um aumento da produtividade e dinamicidade das funções da equipe de engenharia, uma vez que estes deixaram de realizar as tarefas repetitivas e passaram a focar em atividades de análise, interpretação e engenharia reversa das mensagens.

Ao nível estratégico da empresa, a alta gerência, a implantação da aplicação permitiu o aumento da produtividade da equipe, deixando de levar semanas para o desenvolvimento para alguns dias, ou até mesmo horas. Por consequência, o custo por veículo desenvolvido na empresa foi reduzido drásticamente.

De acordo com dados levantados pelo setor, a produtividade da equipe – mensurada pelas suas entregas – aumentou em até 10 vezes.

7.2 PRÓXIMOS PASSOS E TRABALHOS FUTUROS

A solução apresentada, embora contemple boa parte dos veículos em circulação ainda possui um longo caminho de desenvolvimento pela frente. Inúmeras montadoras possuem serviços proprietários tal qual o 0xA8 apresentado nos capítulos anteriores, e precisam de desenvolvimento específico para que seja possível a automação da geração da simulação.

Com a tecnologia embarcada dos veículos cada vez mais complexa, é provável que cada vez mais particularidades surjam nos veículos, fazendo com que o programa tenha a necessidade de estar em constante atualização.

REFERÊNCIAS

[*S.l.*]: EL PAÍS. https://english.elpais.com/society/2022-10-12/what-does-it-mean-when-a-cars-malfunction-indicator-light-mil-goes-on.html. Acesso: 03/10/2023.

[*S.l.*]: CAR e DRIVER. https://www.caranddriver.com/car-accessories/g42938164/best-obd2-car-scanners-tested/. Acesso: 03/10/2023.

CALIFÓRNIA, ESTADO DA. [*S.l.*: *s.n.*], 1993. https://ww2.arb.ca.gov/sites/default/files/barcu/regact/1993/obdiiford93/fsor.pdf. Acesso: 05/11/2023.

FAVERI TRON, Alvise de; LONGARI, Stefano; CARMINATI, Michele; POLINO, Mario; ZANERO, Stefano. CANflict: Exploiting Peripheral Conflicts for Data-Link Layer Attacks on Automotive Networks. **Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security**, 2022. Disponível em: https://api.semanticscholar.org/CorpusID:252383481.

IBGE. [*S.l.*: *s.n.*], 2023. https://cidades.ibge.gov.br/brasil/pesquisa/22/28120?tipo=grafico. Acesso: 05/11/2023.

INFORMATIK, Vector. Tips and Tricks for the Use of CAPL. VECTOR, 2021.

LIU, Le; WANG, Lihui; ZHAO, Wei; QIN, Hongyu; WANG, Yuwei. Experimental Study on Effects of OBD II Diagnostics on of Emissions for Light Vehicles. *In*: [*S.l.*: *s.n.*], mar. 2019. P. 1201–1212. ISBN 978-3-030-15738-8. DOI: 10.1007/978-3-030-15740-1_152.

OSCH, Michiel; SMOLKA, Scott A. Finite-State Analysis of the CAN Bus Protocol. IEEE, 2001.

RAVAGNANI, Válter. [*S.l.*]: OFICINA BRASIL. https://www.oficinabrasil.com.br/noticia/tecnicas/diagrama-eletrico-da-rede-de-comunicacao-va-can-do-modelo-ford-ka-1-5-16v-sigma-flex. Acesso: 03/10/2023.

SUDSOMBOON, Weerayute. Effects of an Automotive Scan Tools Training Material Package on Problem Solving Skills for Undergraduate Students, 2007.

VECTOR. [*S.l.*: *s.n.*]. https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/successor-products/. Acesso: 03/10/2023.