Gisele Maria Almeida Carvalho
Maria Eduarda da Silva Sá

DESENVOLVIMENTO DE *SOFTWARE* PARA PARAMETRIZAÇÃO DE MÓDULOS AUTOMOTIVOS

Resende - RJ
2024

ASSOCIAÇÃO EDUCACIONAL DOM BOSCO
FACULDADE DE ENGENHARIA DE RESENDE

Gisele Maria Almeida Carvalho
Maria Eduarda da Silva Sá

DESENVOLVIMENTO DE *SOFTWARE* PARA PARAMETRIZAÇÃO DE MÓDULOS AUTOMOTIVOS

Trabalho de Graduação apresentado à Associação Educacional Dom Bosco, Faculdade de Engenharia de Resende, Curso de Engenharia Elétrica/Eletrônica, como requisito parcial para obtenção do diploma de Bacharel em Engenharia Elétrica/Eletrônica.

Resende - RJ
2024

Catalogação na fonte
Biblioteca Central da Associação Educacional Dom Bosco – Resende-RJ

C331 Carvalho, Gisele Maria Almeida

Desenvolvimento de *software* para parametrização de módulos automotivos / Gisele Maria Almeida Carvalho; Maria Eduarda da Silva Sá - 2024.

68f.

Orientador: Tiago Duarte Amorim

Trabalho de conclusão de curso apresentado como requisito parcial à finalização do curso de Engenharia Elétrica da Faculdade de Engenharia de Resende da Associação Educacional Dom Bosco.

1. Engenharia elétrica. 2. Software. 3. Visual Studio. 4. Módulo automotivo. I. Sá, Maria Eduarda da Silva. II. Amorim, Tiago Duarte. III. Faculdade de Engenharia de Resende. IV. Associação Educacional Dom Bosco. V. Título.

CDU 004.42(043)

FACULDADES DOM BOSCO

Gisele Maria Almeida Carvalho
Maria Eduarda Da Silva Sá

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO ENGENHARIA ELÉTRICA/ELETRÔNICA."**

APROVADO EM SUA FORMA FINAL PELA BANCA EXAMINADORA

BANCA EXAMINADORA:

Prof. (a).: TIAGO DUARTE AMORIM
Orientador

Prof. (a).: LUIZ FERNANDO RIBAS MONTEIRO
Membro da Banca

Prof. (a).: BIANCA AZEVEDO SALGADO
Membro da Banca

Novembro 2024

Dedicamos este trabalho às nossas famílias, cuja presença e apoio foram fundamentais em nossa jornada acadêmica.

AGRADECIMENTOS

Agradecimentos de Gisele

Primeiramente, agradeço a Deus, cuja luz e sabedoria me guiaram em cada passo desta jornada. Agradeço a minha vida, minha família e meus amigos.

Agradeço à minha mãe, Palmira Rosenis de Almeida Carvalho, e ao meu pai, Elenicio Ribeiro de Carvalho, pela força, amor incondicional, conselhos, apoio, incentivo constante e compreensão, por sempre serem meu alicerce e me motivarem a buscar o melhor. Agradeço por cada sacrifício que vocês fizeram para que eu pudesse ter as oportunidades que tenho hoje, e por sempre acreditarem em mim, mesmo quando eu duvidava.

Agradeço a minha irmã, Giovana Maria Almeida Carvalho, por me inspirar com sua alegria contagiante e curiosidade, lembrando-me da importância de sonhar. Obrigado por ser uma fonte de motivação, com seu olhar otimista e sua risada, que sempre trazem luz aos meus dias.

Agradeço ao meu namorado, Ítallo Rocha Generozo, por estar ao meu lado, por me ajudar em tudo que foi possível. Obrigada por me oferecer amor e compreensão nos momentos desafiadores,

Um agradecimento especial ao meu orientador, Tiago Amorim, cuja orientação e paciência foram fundamentais para o desenvolvimento deste projeto.

Por fim, sou grato a todo o corpo docente da minha faculdade, que contribuiu com seu conhecimento e dedicação, enriquecendo minha formação e meu crescimento pessoal e profissional.

Agradecimentos de Maria Eduarda

Primeiramente, quero expressar minha profunda gratidão a Deus, cuja graça e misericórdia sempre me acompanharam. Sem sua presença, este objetivo não teria sido alcançado. Toda honra e glória seja dada a Ele.

Agradeço à minha mãe, Marlei Fátima da Silva, por seu apoio incondicional em todas as etapas da minha jornada. Seu incentivo constante e amor inabalável foram fundamentais para que eu pudesse superar desafios ao longo desses cinco anos.

À minha irmã, Paula Fernanda de Carvalho, sou imensamente grata por todo o apoio que me deu neste último ano. Sua ajuda foi essencial para a conclusão desta etapa, e sou grata por tê-la ao meu lado.

Um agradecimento especial ao meu padrasto, José Marcos Domingos, que sempre me ajudou imensamente em meus estudos. Sua dedicação e suporte foram valiosos e muito apreciados.

Quero também agradecer ao meu namorado, Matheus Martins da Silva, por ter ficado ao meu lado e por me ajudar em tudo que foi possível. Sua presença e apoio foram fundamentais durante este processo.

Um agradecimento especial ao meu orientador, Tiago Amorim, por sua dedicação e conselhos valiosos. Sua orientação foi fundamental para o desenvolvimento deste trabalho.

Agradeço a todos os meus professores, cujos ensinamentos e orientações foram cruciais para meu crescimento acadêmico e pessoal.

Por fim, agradeço aos meus amigos e amigas, que me acompanharam nessa jornada, oferecendo apoio e motivação. Vocês tornaram essa experiência mais leve e divertida.

“Caráter tem a ver com os valores que cultivamos no íntimo, os quais nos obrigamos a praticar mesmo sem ninguém para observar.”

Luciano Subirá

RESUMO

Este trabalho apresenta o desenvolvimento de um *software* para parametrização de módulos automotivos, seguindo etapas como planejamento, definição de objetivos e escopo, escolha da linguagem C# e do Visual Studio 2022 como IDE (*Integrated Development Environment*), *design* da interface, análise da comunicação CAN (*Controller Area Network*), integração entre *software* e *hardware*, além de testes e validação dos resultados. O Visual Studio 2022 foi escolhido por sua robustez e suporte a ferramentas avançadas, enquanto a integração com WPF (*Windows Presentation Foundation*) facilitou a comunicação entre a interface gráfica e a lógica do sistema, garantindo uma operação eficaz. A utilização da CANcase como interface de *hardware* permitiu a comunicação eficiente com os componentes automotivos através do barramento CAN, essencial para a leitura e escrita de dados em tempo real. Como resultado, o projeto entregou um *software* intuitivo e eficaz, capaz de realizar a parametrização dos módulos com precisão e confiabilidade.

PALAVRAS-CHAVE: *Software*. Rede CAN. Módulo automotivo. Visual Studio. C#.

ABSTRACT

This work presents the development of a software for the parameterization of automotive modules, following stages such as planning, definition of objectives and scope, choice of C# language and Visual Studio 2022 as the IDE (Integrated Development Environment), interface design, analysis of CAN (Controller Area Network) communication, integration between software and hardware, as well as testing and validation of results. Visual Studio 2022 was chosen for its robustness and support for advanced tools, while the integration with WPF (Windows Presentation Foundation) facilitated communication between the graphical interface and the system logic, ensuring efficient operation. The use of CANcase as the hardware interface enabled efficient communication with automotive components through the CAN bus, essential for real-time data reading and writing. As a result, the project delivered an intuitive and effective software capable of accurately and reliably parameterizing the modules.

KEYWORDS: Software. CAN network. Automotive module. Visual Studio. C#.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	*Anti-lock Braking System*
BCM	*Body Control Module*
CAN	*Controller Area Network*
CAN FD	*CAN Flexible Data-rate*
CANoe	*CAN Open Environment*
CAPL	*Communication Access Programming Language*
CSMA/CA	*Carrier Sense Multiple Access/Collision Avoidance*
DID	*Data Identifier*
ECU	*Eletronic Central Unit*
E/E	*Eletroeletrônicos*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
LIN	*Local Interconnect Network*
MOST	*Media Oriented Systems Transport*
NRZ	*Non Return to Zero*
OSI	*Open Systems Interconnection*
OTA	*Over-the-Air*
PDM	*Power Distribution Module*
TCU	*Transmission Control Unit*
UDS	*Unified Diagnostic Services*
WPF	*Windows Presentation Foundation*

SUMÁRIO

1 INTRODUÇÃO

Um dos principais autores do livro **Indústria Automobilística Brasileira: 50 Anos** relatou em uma de suas edições comemorativas que o primeiro brasileiro a obter um carro foi o milionário Henrique Dumont. O veículo, um Peugeot comprado em Paris, foi adquirido por Dumont na região de Ribeirão Preto, em São Paulo, no ano de 1893. Uma curiosidade interessante é que ele era pai de Alberto Santos Dumont, que se tornou conhecido como o Pai da Aviação (TAFFAREL, 2015).

Com o passar dos anos, os veículos automotores (carros de passeios, ônibus, caminhões, motocicletas etc.) se tornaram cada vez mais presentes e indispensáveis no cotidiano da população brasileira.

Visando a adição de novas funcionalidades e um melhor desempenho, esses meios de transporte têm evoluído em complexidade. A partir do avanço tecnológico e científico, o sistema, antes baseado apenas em uma máquina de combustão interna, tornou-se um mecanismo sofisticado que integra mecânica com eletrônica embarcada. Um sistema é considerado embarcado quando é projetado para executar uma única tarefa e interagir continuamente com o ambiente ao seu redor por meio de sensores e atuadores, permitindo maior controle e eficiência dos veículos (CAVALCANTE, 2018).

> Acompanhando o desenvolvimento de novas tecnologias e na tentativa de atender um consumidor que busca cada vez mais modernidade, conforto e segurança, os carros passaram a contar com sistemas computacionais gerenciando os módulos veiculares. Sistemas eletrônicos conseguem identificar simultaneamente o estado do motor, dos sistemas de iluminação e combustível, por exemplo. Sistemas de localização, como o GPS (*Global Positioning System*), também fazem parte das tecnologias atualmente presentes nos automóveis do mundo todo (CAVALCANTE, 2018, p. 17).

A integração desses sistemas eletrônicos nos veículos trouxe inúmeros benefícios, como maior segurança, melhor desempenho e eficiência energética. No entanto, também impôs novos desafios, como a necessidade de desenvolver ferramentas avançadas para a configuração e manutenção dos módulos eletrônicos embarcados. A crescente complexidade dos veículos modernos exige que engenheiros e técnicos tenham acesso a tecnologias de diagnóstico e parametrização capazes de lidar com diferentes padrões e protocolos de comunicação. Nos primeiros automóveis, o desenvolvimento estava centrado principalmente nos sistemas mecânicos. Com o avanço das redes de computadores e a redução dos custos de

hardware e *software*, tornou-se possível aprimorar diversas conexões ponto a ponto por redes de comunicação internas.

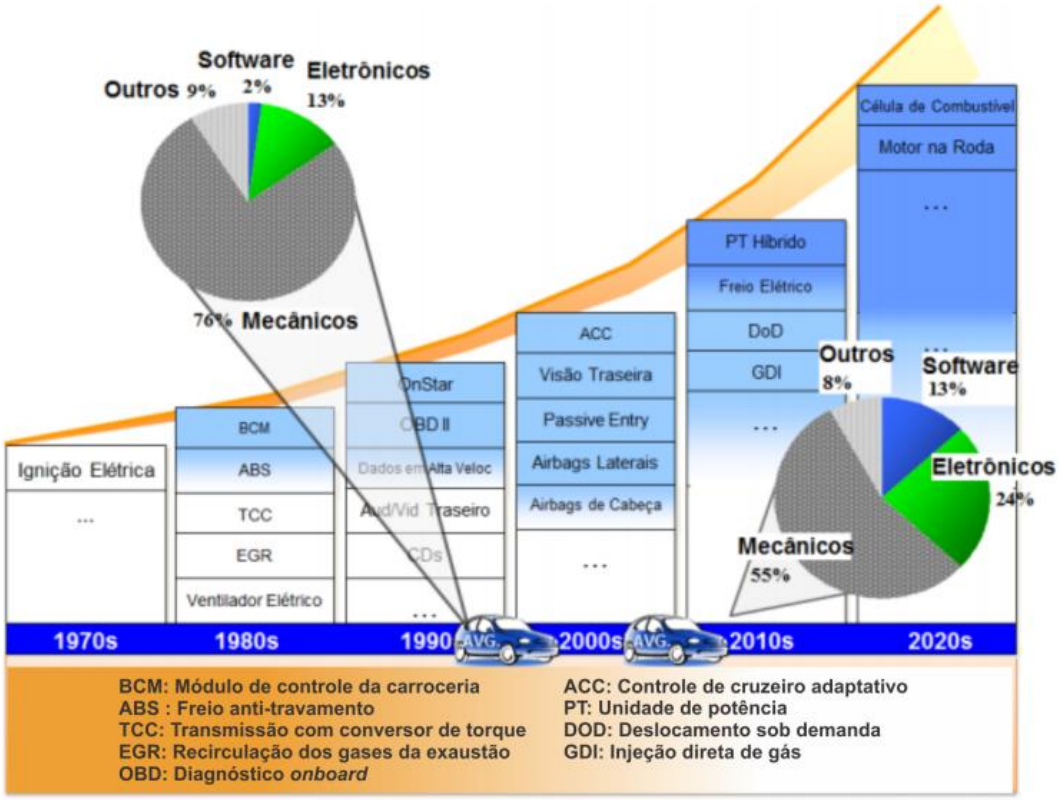
Com o desenvolvimento de unidades de controle eletrônicas, ou ECUs (*Eletronic Central Unit*) houve aumento da quantidade de conexões e cabos dos veículos. Esse sistema de comunicação das unidades de controle resultou em erros frequentes e dificuldade de manutenção. Em resposta a esses problemas, surgiu a necessidade de simplificar o cabeamento, mantendo a eficiência e segurança do sistema. Com o objetivo de utilização de um sistema de baixo custo de implementação e eficiente para comunicar-se com os módulos eletrônicos, assim, foi desenvolvido o protocolo CAN, que posteriormente foi padronizado pelas normas ISO 11898-1, ISO 11898-2 e ISO 11898-3 (TIRONI et al., 2018).

Diferente das arquiteturas tradicionais, na qual é necessário um computador central, *Hubs* e endereçamento IP, a rede CAN dispensa a utilização de tais equipamentos, aplicando um barramento de pares trançados com sinais opostos para equalizar campo magnético e distribuindo as informações neste barramento, seu protocolo de comunicação é enviar mensagens com *source address*, sendo necessário programar os equipamentos para identificar os locais das mensagens fontes. Após a padronização, o protocolo utilizado em sistemas embarcados tornou-se conhecido mundialmente. Com esse avanço, os componentes evoluíram para módulos, reduzindo assim os custos e aumentando a eficácia do sistema.

No cenário atual, as empresas precisam estar em constante busca por inovação, independentemente do setor em que atuam. No setor automobilístico, essa inovação está diretamente relacionada ao uso de sistemas eletroeletrônicos (E/E), uma vez que as principais funcionalidades introduzidas nos veículos atualmente estão diretas ou indiretamente ligadas a esses sistemas (SILVA, 2017).

A figura 1 ilustra a contribuição dos diferentes sistemas automotivos para o custo total do veículo, evidenciando um aumento significativo no uso de *software* e sistemas E/E ao longo dos anos, em contraste com a redução de outros subsistemas.

Figura 1: Parcela dos subsistemas no custo total do veículo.



Fonte: Rafael Rodrigues da Silva (2017).

> O desenvolvimento de sistemas embarcados elevou a complexidade e consequentemente o custo dos *softwares* automotivos. De fato, a utilização de *software* para controle de sistemas automotivos tem crescido consideravelmente nas últimas décadas (BROY et al., 2013). Krüger e Meisinger (2006) afirmam que em média 90% de todas as inovações que ocorrem em sistemas automotivos são diretas ou indiretamente ligadas aos *softwares* embarcados. Os automóveis atuais chegam a conter mais de 2500 funções controladas podendo conter, em alguns casos até 10 milhões de linhas de códigos (LIU; ZHANG; ZHU, 2016). Esses números mostram o quão complexo tem se tornado o desenvolvimento de *software* automotivo nos veículos modernos. (SILVA, 2017, p. 2).

Nesse contexto, o desenvolvimento de *softwares* específicos para a configuração de módulos automotivos se torna essencial. Tais ferramentas possibilitam a programação e adaptação de componentes eletrônicos de acordo com as necessidades específicas de cada veículo, garantindo a compatibilidade e o correto funcionamento dos diversos sistemas integrados. Com isso, busca-se não apenas melhorar a experiência do usuário final, mas também reduzir o tempo e o custo de manutenção e reparos, contribuindo para a evolução contínua da indústria automotiva. É essencial que um módulo possa ser utilizado em diversos modelos. Isso requer um *software* configurável que permite definir as melhores aplicações para cada veículo. A

configuração é feita por meio de parâmetros que definem as entradas, saídas e o gerenciamento dele.

Atualmente, uma empresa automobilística enfrenta desafios significativos em relação à parametrização de módulos em seus veículos, dependendo de um *software* externo específico para realizar essa atividade. Esse *software* possui uma licença de custo elevado e é disponibilizado a um número limitado de funcionários, o que torna o processo mais lento e ineficiente. Com o desenvolvimento de um *software* interno, a organização poderá economizar recursos financeiros e melhorar consideravelmente sua eficiência interna na configuração dos módulos dos veículos. Essa solução permitirá um processo mais ágil e acessível, reduzindo a dependência de recursos externos e aumentando a autonomia da empresa em suas operações.

O estudo será estruturado nas seguintes seções: revisão de literatura, materiais utilizados, metodologia aplicada, apresentação e discussão dos resultados, e, por fim, a conclusão.

1.1 OBJETIVO GERAL

O presente trabalho propõe o desenvolvimento de um *software* intuitivo que, ao ser conectado à rede CAN do automóvel, possibilite o acesso ao módulo automotivo para parametrização. Para isso, serão necessárias as seguintes etapas principais: planejamento, definição dos objetivos e escopo, escolha da linguagem de programação e IDE, *design* da interface, análise de comunicação CAN, conectividade entre *software* e *hardware*, testes e validação dos resultados.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos incluem: criar uma interface de usuário simples e intuitiva que facilite a parametrização de diferentes modelos de veículos; implementar testes para garantir o correto funcionamento do s*oftware* e validar sua eficácia; e elaborar uma documentação detalhada com as instruções desse sistema.

2 REVISÃO DA LITERATURA

2.1 MÓDULOS ELETRÔNICOS

Desde o surgimento dos primeiros protótipos de automóveis no século XVIII até os carros modernos dos últimos 10 anos, o foco na melhoria de desempenho, segurança e conforto tem sido constante. Novas tecnologias foram integradas aos sistemas internos dos veículos, muitas vezes invisíveis para os usuários, mas que trouxeram mais informações e facilidade de uso para motoristas e passageiros. A grande revolução tecnológica na indústria automotiva, intensificada nos últimos 15 anos, foi marcada pela introdução de dispositivos eletroeletrônicos. Nos anos 1970, dois fatores aceleraram o uso de sistemas eletrônicos nos automóveis: as regulamentações governamentais sobre emissões de poluentes e economia de combustível, e o avanço da eletrônica digital de estado sólido, que reduziu os custos. Para melhorar o controle do motor, a eletrônica tornou-se essencial, com a crescente preocupação ambiental e a busca por eficiência. Na década de 1980, essas demandas culminaram na introdução dos módulos eletrônicos veiculares, conhecidos como ECUs (GIRARDI, 2015).

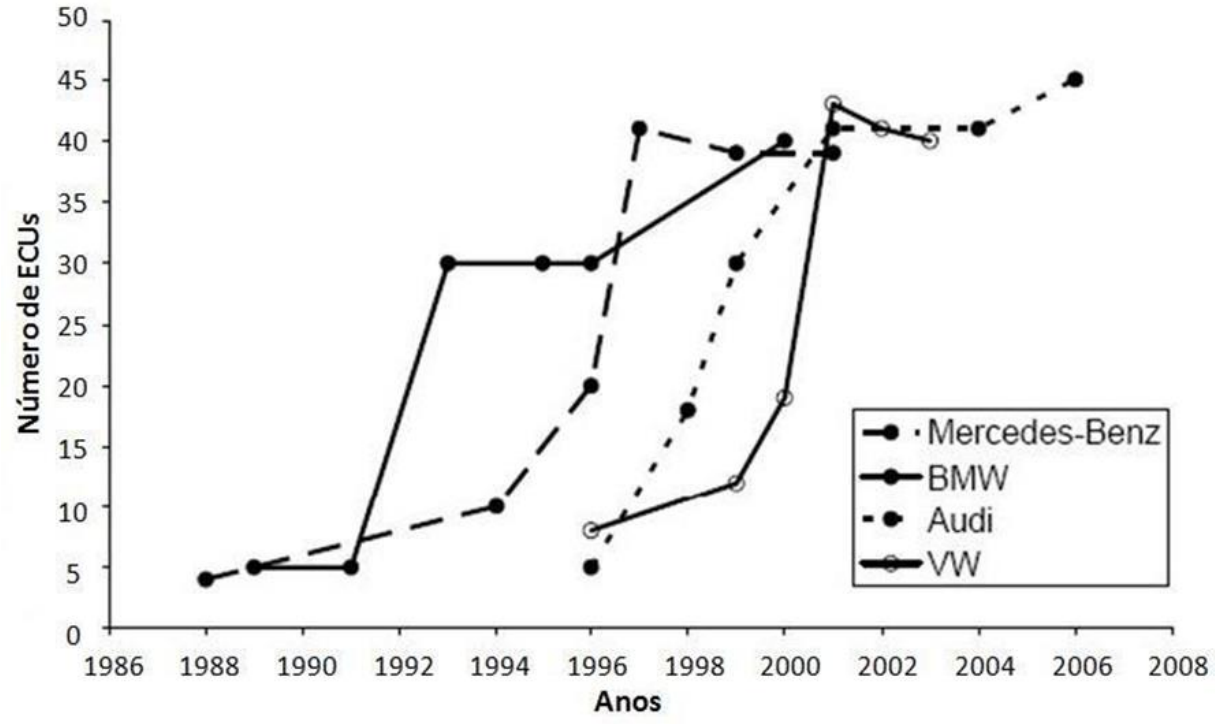
Módulos eletrônicos são componentes computacionais, instalados em veículos modernos, responsáveis por controlar e gerenciar diversas funções e sistemas. Cada módulo eletrônico é especializado em uma determinada área, utilizando sensores, atuadores e algoritmos para monitorar e ajustar parâmetros automaticamente. Esses módulos processam informações em tempo real para otimizar o desempenho, a segurança, a eficiência e o conforto do veículo.

> Como os dispositivos eletrônicos são rápidos e fáceis de controlar, aceitam variações em sua configuração de parâmetros sem a necessidade de remover partes físicas do veículo, fornecem boa precisão e possuem um desempenho superior aos sistemas hidráulicos e mecânicos convencionais, os módulos logo tiveram a sua aplicação estendida a outros diferentes sistemas em função da crescente demanda por segurança, conforto, tecnologia e redução do impacto ambiental (GIRARDI, 2015, p. 11).

A figura 2 ilustra a evolução do uso de ECUs ao longo dos anos por diversas empresas do setor automotivo em mercados desenvolvidos. Conforme apresentado abaixo, o maior crescimento na adoção desses módulos eletrônicos ocorreu durante a década de 1990, com a implementação inicial realizada por empresas que tradicionalmente atendem consumidores de maior poder aquisitivo, cujas demandas

por tecnologia, conforto e desempenho são mais elevadas. Já no ano 2000, a média de módulos eletrônicos utilizados por veículo era de aproximadamente 40 unidades.

Figura 2: Número de ECU’s em mercados desenvolvidos.



Fonte: GIRARDI (2015).

Entre os diversos tipos de módulos eletrônicos presentes nos veículos modernos, além da ECU, destacam-se outros importantes sistemas que desempenham funções específicas e essenciais. Exemplos incluem o TCU (*Transmission Control Unit*), responsável pela gestão da transmissão do veículo, o BCM (*Body Control Module*), que controla diversas funções elétricas da carroceria, o ABS (*Anti-lock Braking System*), que assegura o controle dos freios, e o PDM (*Power Distribution Module*), que distribui a energia elétrica para diferentes sistemas. Esses módulos demonstram a ampla gama de funções controladas eletronicamente, refletindo a complexidade dos veículos modernos e sua dependência da eletrônica avançada.

2.1.1 Unidade de Controle da Transmissão

A transmissão de um veículo é um dos sistemas mais importantes para o seu funcionamento, sendo responsável por transferir a potência gerada pelo motor para as rodas, permitindo o movimento do veículo. Além disso, a transmissão desempenha

um papel fundamental no controle da velocidade e no ajuste do torque (força de tração) de acordo com as condições de condução, como aceleração, desaceleração ou mudanças no terreno.

Seu principal objetivo é permitir que o motor opere em sua faixa de rotações mais eficiente, ajustando a relação de marchas para otimizar tanto o consumo de combustível quanto o desempenho. Durante a aceleração, a transmissão altera as engrenagens para fornecer mais potência, enquanto, em velocidades mais altas, ela reduz a rotação do motor para economizar combustível. Em terrenos íngremes ou escorregadios, ela pode ajustar a distribuição de torque para garantir que as rodas não percam tração.

> Assim como em todas as transmissões automáticas a seleção das marchas é feita de forma pré-definida e de forma sequencial, obedecendo um comando lógico dado por uma unidade de controle TCU- *Transmission Control Unit* (Unidade de Controle da Transmissão), que através de um sistema CAN-BUS (Definição de Can - Bus) faz a leitura e integra todos os parâmetros necessários para esta ação. (MACEDO; LOPES, 2020, p. 9).

A TCU, um dispositivo eletrônico crucial, opera como o "centro de comando" da transmissão do veículo. Ela coleta informações de múltiplos sensores e sistemas eletrônicos, processando esses dados para determinar as ações necessárias. Com base nessa análise, a TCU envia sinais de controle para os atuadores responsáveis pela operação da transmissão. Além disso, ela também coordena e repassa instruções para outros sistemas do veículo, garantindo um funcionamento integrado e eficiente de todo o conjunto mecânico e eletrônico. (NAUNHEIMER et. al., 2011).

Por fim, a transmissão se mostra vital para o funcionamento eficiente do motor e para ajustar o torque entregue às rodas, garantindo que o veículo se adapte a diferentes condições de direção com precisão e segurança.

2.1.2 Módulo de Controle da Carroceria

O Módulo de Controle da Carroceria é um componente eletrônico fundamental em veículos modernos, responsável por gerenciar diversas funções eletrônicas na carroceria.

Ele processa informações de sensores simples, como interruptores de comando distribuídos por várias partes do veículo. Esses sensores fornecem dados que o BCM utiliza para acionar dispositivos de saída, como lâmpadas, motores

elétricos, travas e vidros, garantindo que as funções do veículo sejam realizadas de maneira eficiente e confiável, proporcionando ao usuário uma experiência segura e confortável (SANTOS, 2012).

> Ele é também fundamental para o funcionamento geral do automóvel, pois monitora diversos sensores instalados para medir variáveis como temperaturas, velocidade ou pressões, e converte esses sinais em informações que são mostradas no painel do carro. Há um importante papel de verificação de falhas no sistema eletrônico, de forma que ao detectar algum possível erro, o BCM solicita um sinal do módulo que pode conter a falha, e caso não haja resposta, ele cria um código de erro e passa a agir para corrigir o problema (BOCCI, 2020, p. 30).

Ao monitorar sensores que medem variáveis essenciais, como temperatura, velocidade e pressão, o BCM converte esses dados em informações visíveis para o motorista. Além disso, é o BCM é importante na verificação de falhas no sistema eletrônico. Ao detectar um erro, ele solicita um sinal do módulo relacionado e, se não obtiver resposta, gera um código de erro e inicia ações para corrigir a falha. Isso demonstra não apenas a complexidade dos sistemas eletrônicos dos veículos modernos, mas também como o BCM contribui para a segurança e confiabilidade do automóvel.

Essa compreensão é crucial para o desenvolvimento de um *software* que busca otimizar a parametrização desses módulos, destacando a interconexão entre eles.

2.1.3 Sistemas de Freio Antibloqueio

O módulo ABS é um sistema eletrônico presente nos veículos que evita o travamento das rodas durante uma frenagem brusca. Sua função é garantir que o motorista mantenha o controle da direção, permitindo que as rodas continuem girando, em vez de deslizarem, quando o freio é aplicado com força. Isso ajuda a melhorar a estabilidade e a capacidade de manobra do veículo, principalmente em condições de baixa aderência (LIMA et al., 2018).

O primeiro sistema semelhante ao ABS foi criado em 1936 pela Bosch para evitar o escorregamento de rodas em veículos ferroviários. Em 1970, a primeira versão do ABS, chamada "ABS 1", foi lançada, ainda grande e mecânica. Em 1978, o "ABS 2", mais compacto e confiável, estreou no Mercedes Classe S. Ao longo dos anos, o ABS evoluiu rapidamente, com o "ABS 2S" em 1983, que era menor e mais leve. Ainda na década de 1980, o "ABS 2E" começou a usar memória programável, e o "ABS 5.0"

de 1993 aumentou ainda mais a capacidade de memória. Em 2005, o "ABS 8.1" trouxe mais redução de peso, e as versões "ABS 9" e "ABS 10" foram lançadas para motocicletas em 2009 e 2016 (DUTRA et al., 2020).

O ABS possui sensores que recebem a pressão exercida pelo condutor e, com base nisso, distribui a pressão de frenagem para cada roda, evitando seu travamento individual. Isso melhora a eficiência da frenagem, reduz a distância de parada e mantém a dirigibilidade em diferentes terrenos, aumentando a segurança e diminuindo o risco de acidentes causados por perda de controle ou travamento das rodas. Além disso, em acidentes inevitáveis, o ABS reduz a gravidade dos impactos (DUTRA et al., 2020).

Em suma, o módulo ABS tornou-se um dos principais avanços tecnológicos em segurança veicular, proporcionando maior controle e estabilidade aos motoristas em emergências.

2.1.4 Módulo de Distribuição de Potência

Os módulos de distribuição de potência são projetados para simplificar o gerenciamento de sistemas elétricos, substituindo relés e fusíveis de forma eficiente. São aplicáveis em diferentes tipos de veículos, desde alta performance até uso cotidiano, e suportam tanto sistemas de 12V quanto de 24V. Esses módulos alimentam uma variedade de componentes elétricos, como sistemas de ignição, combustível, controle de potência, entre outros. Geralmente, possuem mecanismos para isolamento e absorção de vibrações, garantindo maior durabilidade e confiabilidade (GTI Racing Parts, 2024).

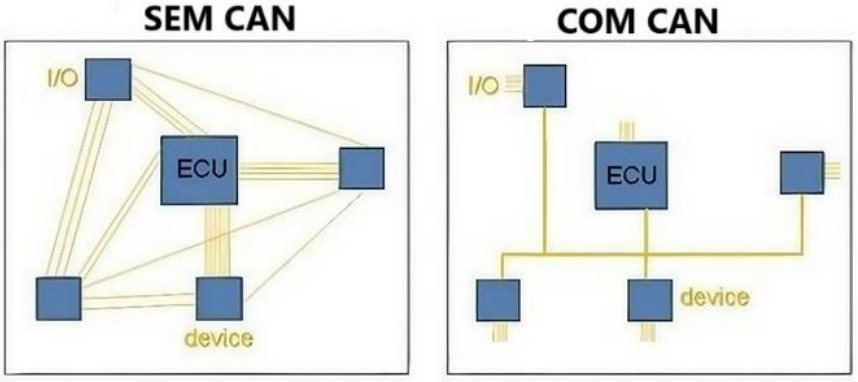
O Módulo de Distribuição de Potência é responsável por gerenciar a alimentação dos módulos eletrônicos, sensores e atuadores de um veículo. Ele utiliza circuitos integrados inteligentes para controlar o fornecimento de energia aos periféricos, monitorando o consumo de corrente. Além disso, o módulo detecta condições de curto-circuito e circuitos abertos, e possui fusíveis rearmáveis, que aumentam a segurança e a eficiência do sistema elétrico do veículo (SOUSA, 2022).

Portanto, os módulos de distribuição de potência são fundamentais para a eficiência e segurança dos sistemas elétricos dos veículos, substituindo relés e fusíveis, monitorando o consumo de energia e detectando falhas.

2.2 REDE CAN

Com o avanço tecnológico e a crescente demanda por conforto e segurança, o número de ECUs aumentou. Isso resultou em um aumento significativo na quantidade de cabos e conexões nos veículos. Sendo assim, tornou-se essencial desenvolver um sistema que otimizasse essa complexa estrutura eletrônica, é possível observar uma comparação da estrutura de cabos sem a rede CAN e com essa tecnologia na figura 3 abaixo.

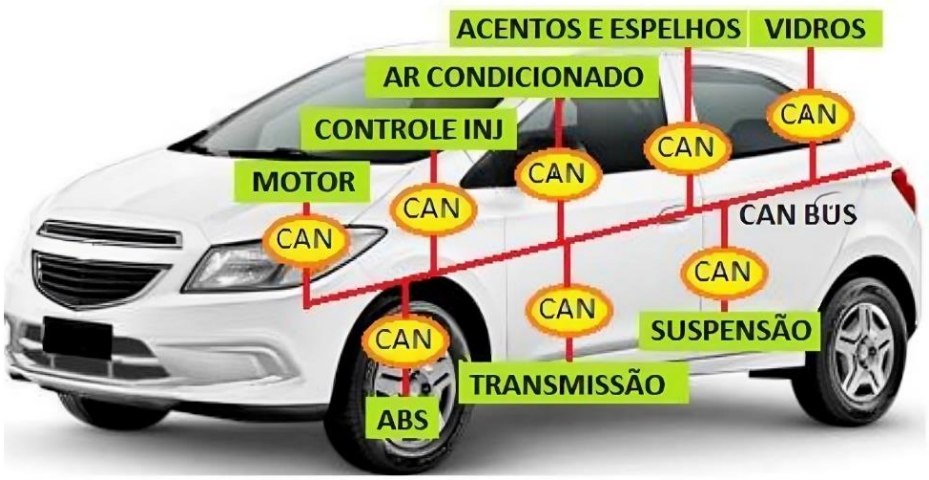
Figura 3: Redução de fios elétricos entre os módulos.



Fonte: VITORINO et. al. (2022).

Diante desse contexto, a multinacional alemã Bosch desenvolveu, na década de 1980, o protocolo *Controller Area Network* com o objetivo de melhorar a eficiência na comunicação entre as centrais eletrônicas dos automóveis, conforme ilustrado na figura 4. (TIRONI et al., 2018).

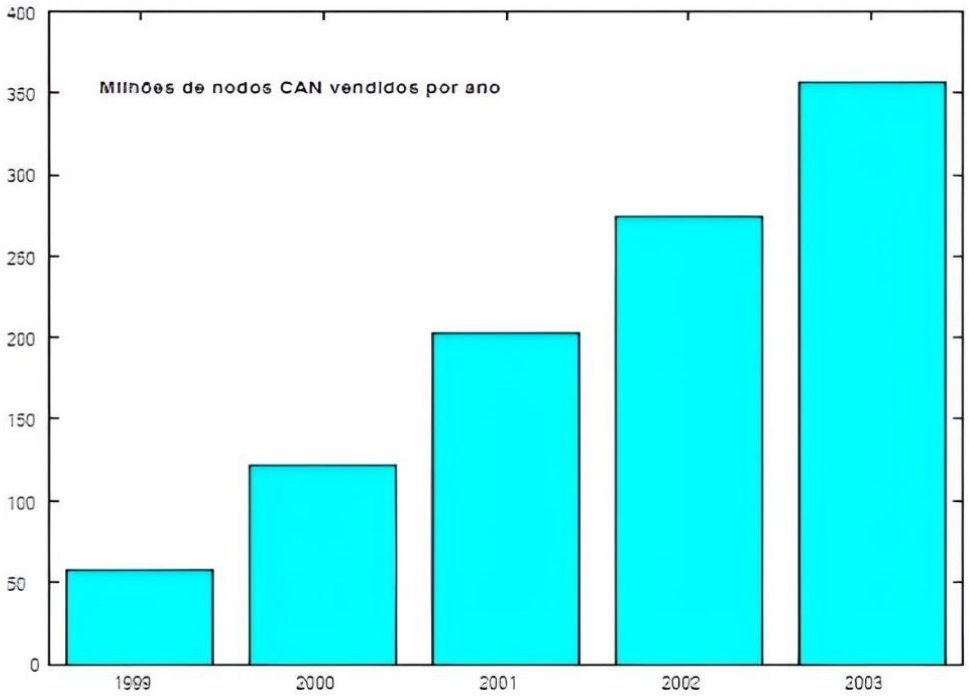
Figura 4: Esquema exemplificado da rede CAN em um veículo.



Fonte: TIRONI et al. (2018).

> Como consequência do desenvolvimento do protocolo, já em 1987 empresas como Intel e Philips iniciaram a produção de controladores que representa o crescimento de vendas de chips do módulo CAN ao longo do tempo, tal qual demonstrado na figura 5. Primeiramente, a empresa americana Intel desenvolveu o chip controlador CAN modelo 82526 e pouco tempo mais tarde a empresa Philips Semiconductors lançou no mercado o 82C200, estes possibilitaram, em 1991, o desenvolvimento do primeiro veículo com a presença da rede CAN integrada. Implementado pela montadora Mercedes, o modelo S-W140 tinha por base cinco unidades de controle eletrônico. (TIRONI et al., 2018, p.1).

Figura 5: Venda de Chips do módulo CAN por ano.

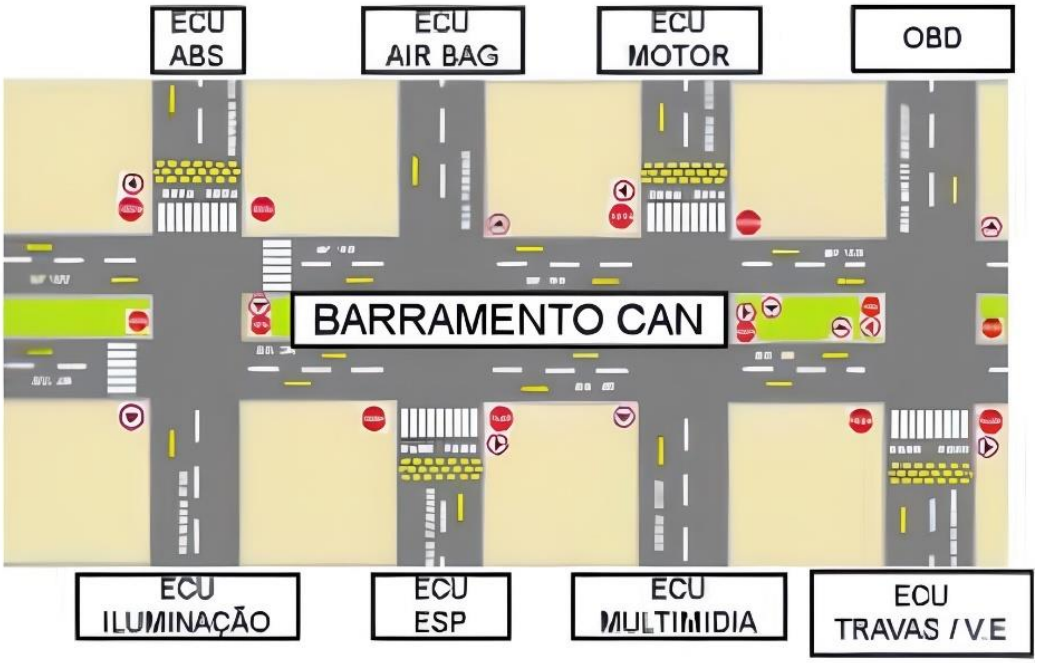


Fonte: TIRONI et al. (2018).

Com o sucesso da implementação do protocolo CAN no modelo S-W140 da Mercedes, outras montadoras começaram a adotar essa tecnologia em suas linhas de produção. O uso do CAN expandiu-se rapidamente para diversos sistemas automotivos, como controle de motor, transmissão, freios e sistemas de segurança, devido à sua confiabilidade e capacidade de reduzir a complexidade das redes elétricas. Ao longo do tempo, o protocolo CAN tornou-se o padrão em praticamente todos os veículos modernos, sendo utilizado não apenas em automóveis, mas também em caminhões, ônibus e até mesmo em equipamentos industriais e agrícolas. Essa expansão reflete a versatilidade e eficiência do CAN, que continua sendo aprimorado e adaptado para atender às novas demandas tecnológicas e regulatórias, como a necessidade de integração com veículos elétricos e sistemas autônomos.

De forma semelhante, um barramento CAN pode ser visto como uma avenida principal movimentada, onde o fluxo de veículos é constante, enquanto as ECUs estão conectadas por ruas arteriais que oferecem acesso a essa via. O tráfego de dados pode fluir da ECU para a avenida (da rua para a via principal) ou do barramento para a ECU (da avenida principal para a rua). O protocolo coordena todo esse tráfego, assim como as leis de trânsito, a sinalização e os semáforos controlam o fluxo de veículos. Na figura 6 é visto essa representação (FERREIRA et. Al., 2018).

Figura 6: Protocolo CAN – Analogia com a organização de tráfego terrestre.



Fonte: FERREIRA et. Al. (2018).

De acordo com BORTH (2016) as redes automotivas são classificadas em três classes: A, B e C, dependendo de suas aplicações. São elas:

a) Classe A: usada para comunicação geral, como diagnóstico e controles secundários, com baixa velocidade (10 Kbps) e baixo custo. O protocolo LIN (*Local Interconnect Network*) é o mais utilizado.

b) Classe B: aplicada em sistemas não críticos, com velocidade de 10 Kbps a 125 Kbps, suportando transmissões assíncronas e periódicas. O CAN, conforme a norma ISO 11898-3 (velocidade até 125 Kbps, com detecção de falhas e transmissão por um único fio), é o protocolo mais comum.

c) Classe C: destinada a sistemas em tempo real, como controle do motor, com velocidades de 125 Kbps a 1 Mbps. O CAN é amplamente usado, regulamentado pela J1939 (250 Kbps, aplicação em ônibus e caminhões) e ISO 11898-2 2 (500 Kbps, aplicação em veículos leves).

Além disso, importante ressaltar algumas características fundamentais da rede *Controller Area Network* que a tornam uma escolha preferencial na comunicação entre ECUs em veículos:

a) Acessibilidade a microcontroladores e controladores CAN a preços baixos;

b) Utilização de cabos de comunicação de baixo custo;

c) Flexibilidade da rede CAN para novas configurações, permitindo a adição de novos nós sem necessidade de modificar os nós já existentes.

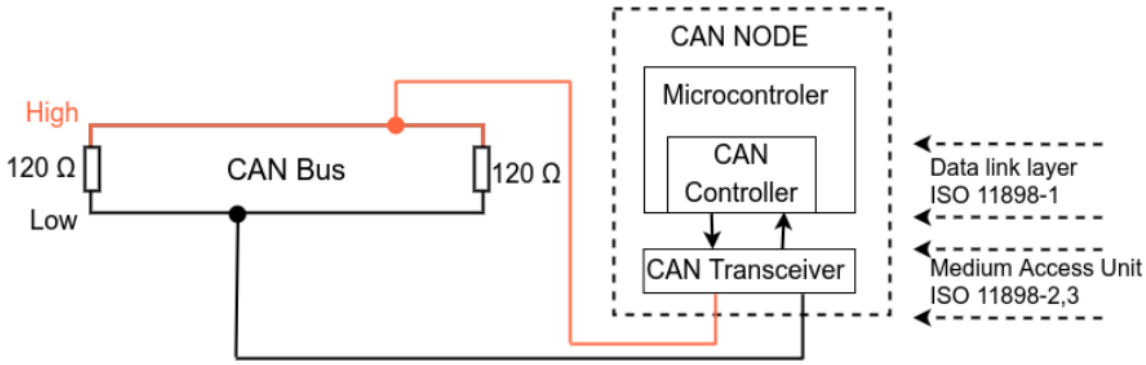
O protocolo CAN adota a técnica CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*) para o gerenciamento de mensagens dentro da rede. Isso significa que, antes de iniciar a transmissão, o controlador CAN verifica se o barramento está livre. Se dois ou mais módulos tentarem transmitir ao mesmo tempo, a mensagem com o menor valor numérico no campo de arbitragem terá prioridade. Quando o barramento retornar ao estado inativo, novas tentativas de envio das mensagens em espera serão realizadas. Além disso, o CAN utiliza o formato NRZ (*Non Return to Zero*), onde os bits são representados por níveis de tensão distintos de zero. Como outros protocolos de comunicação, o CAN segue o modelo OSI (*Open Systems Interconnection*), que é dividido em sete camadas diferentes. Contudo, a norma ISO aborda apenas as camadas inferiores desse modelo, que incluem a camada física e a camada de enlace de dados (BORTH, 2016).

A camada física, que é a primeira do modelo OSI, desempenha um papel crucial na transmissão de dados brutos entre dispositivos. Ela converte informações digitais em sinais elétricos, ópticos ou de rádio e estabelece as especificações para o meio físico, como níveis de tensão e tipos de conectores. Por sua vez, a camada de enlace de dados, que ocupa a segunda posição no modelo, é responsável por garantir a transferência confiável de dados dentro da mesma rede local. Ela organiza os dados em quadros, controla o acesso ao meio para evitar colisões e implementa mecanismos

de detecção e correção de erros. Além disso, essa camada atribui endereços físicos aos dispositivos, permitindo sua identificação única na rede. Juntas, essas camadas asseguram uma comunicação eficiente e segura entre os dispositivos, fundamental para o funcionamento adequado das redes. Além disso, com o intuito de proteger a rede, o CAN é composto por pares trançados ou blindagem para anular o campo eletromagnético que pode ser gerado.

> Um módulo CAN é caracterizado como um Node conectado ao CAN Bus, responsável por requisitar e transmitir mensagens a outros módulos conectados ao barramento. A comunicação entre os Nodes é estruturada com três componentes: microcontrolador, responsável pelo processamento da mensagem; CAN *Transceiver,* responsável por disponibilizar a mensagem ao CAN Bus; e o CAN *Controller,* responsável por realizar a interface para a aplicação dos filtros do protocolo, disponibilização das mensagens e proporcionar a manipulação delas. Estes componentes podem ser observados na figura 7, a qual apresenta a estrutura de um módulo CAN e a forma como é conectado ao CAN Bus. (VITORINO et. al., 2022).

Figura 7: Estrutura de um módulo CAN.



Fonte: VITORINO et. al., 2022 (2022).

Portanto, é possível notar que a rede CAN desempenha um papel essencial na estrutura eletrônica dos veículos automotivos. Sua capacidade de garantir comunicação eficiente e confiável entre os diversos módulos eletrônicos é fundamental para o funcionamento harmonioso de sistemas críticos, como controle de motor, segurança e conforto. A integração dessa tecnologia não apenas otimiza a complexidade da fiação e reduz custos, mas também permite que os veículos modernos atendam às exigências crescentes de desempenho e segurança, tornando a rede CAN uma escolha indispensável na engenharia automotiva.

2.2.1 Tipos de Mensagens

O protocolo CAN define diferentes tipos de mensagens que desempenham papéis específicos na comunicação entre os nós da rede. Segundo BRANDÃO (2022) são elas:

Data Frame: Este é o tipo mais comum de mensagem, contendo campos que incluem um identificador, dados, um código de verificação e um campo de reconhecimento. O identificador pode variar em tamanho, sendo de 11 bits no formato padrão e de 29 bits no formato estendido. O campo de dados pode conter até 8 bytes.

Remote Frame: Este tipo de mensagem é utilizado para solicitar a transmissão de dados de outro nó. Embora semelhante ao *data frame*, é identificado por um bit específico e não contém dados.

Error Frame: Mensagem enviada quando um nó detecta um erro na comunicação. Essa mensagem ativa uma resposta de erro em outros nós, e o transmissor original retransmite a mensagem com erro.

Overload Frame: Semelhante ao *error frame*, este tipo de mensagem é enviado por um nó que está ocupado, servindo para introduzir um atraso na transmissão de mensagens.

Valid Frame: Uma mensagem é considerada livre de erros quando recebida corretamente. Se um erro for detectado, o transmissor deve repetir a transmissão.

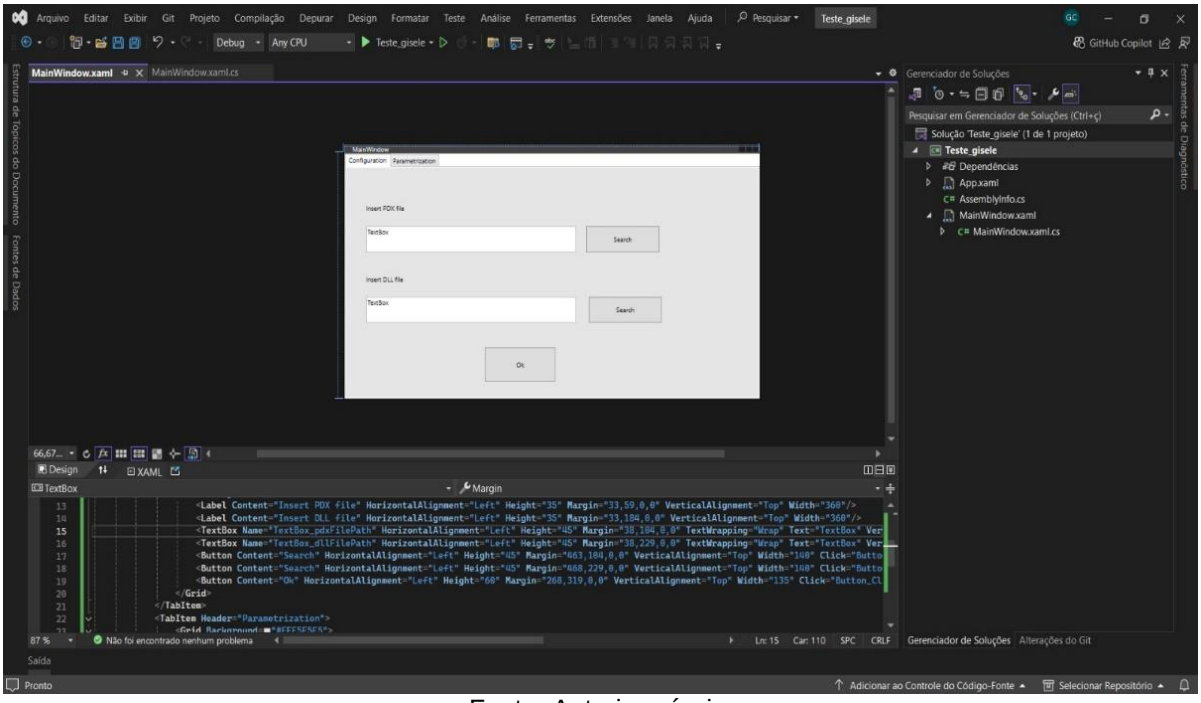
2.3 VISUAL STUDIO

O Visual Studio é um Ambiente de Desenvolvimento Integrado amplamente utilizado para desenvolver aplicativos em várias linguagens de programação. Ele fornece uma série de compiladores, ferramentas para autocompletar código, funcionalidades de controle de versão, opções de extensões e diversos outros recursos valiosos. A IDE oferece um suporte abrangente para o desenvolvimento de aplicações .NET, possibilitando a criação de *software* para *desktop*, *web*, dispositivos móveis, jogos e IoT (*Internet of Things*). Os aplicativos .NET podem ser desenvolvidos utilizando as linguagens C#, F# ou Visual Basic (CARDOSO, 2024).

De acordo com OLIVEIRA (2024, p. 30), “é uma ferramenta poderosa que oferece recursos avançados de edição, depuração e compilação, facilitando o processo de criação de aplicativos e jogos.” Uma das principais características que torna o Visual Studio uma escolha preferencial entre desenvolvedores é sua interface

intuitiva e personalizável, que se adapta às necessidades de diferentes projetos e fluxos de trabalho. Além disso, facilita a navegação entre as diversas partes de um projeto, reduzindo o tempo necessário para localizar e modificar o código. Na figura 8 é possível visualizar a interface inicial dessa plataforma.

Figura 8: Interface inicial da plataforma Microsoft Visual Studio.

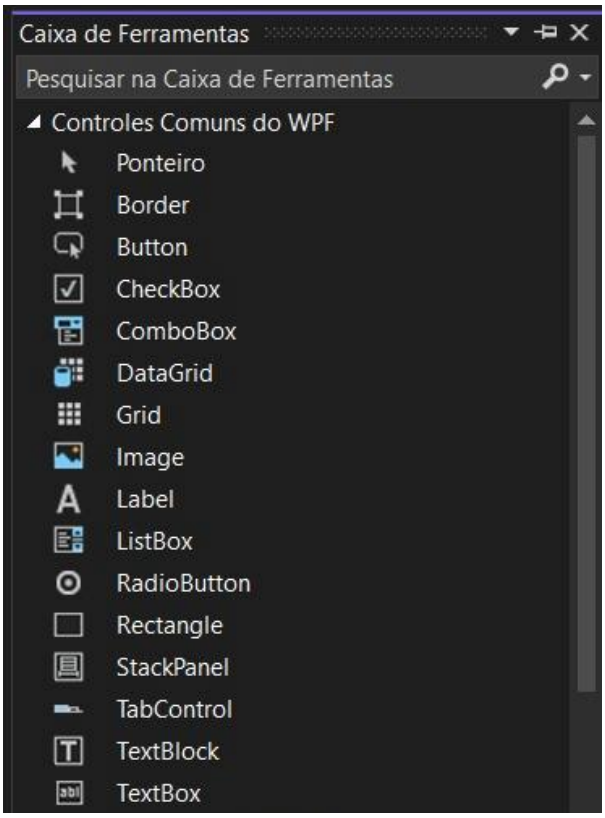


Fonte: Autoria própria.

O Visual Studio permite identificar e corrigir problemas no código de maneira eficaz, uma vez que possui ferramentas de depuração e testes avançadas. Isso não apenas facilita a identificação de falhas antes que o *software* seja implantado, mas também a manutenção do código, garantindo que alterações futuras não introduzam novos problemas. Consequentemente, os usuários podem entender melhor o comportamento de suas aplicações e garantir que elas funcionem conforme o esperado.

A capacidade de testar e validar o código em um ambiente controlado é um componente chave na prática de desenvolvimento ágil e na entrega contínua de *software*. Essa combinação de recursos fez com que o Visual Studio se tornasse o ideal para o projeto em questão, no qual a precisão e a confiabilidade são cruciais. A figura 9 a seguir oferece uma visão mais detalhada das ferramentas disponíveis nesta plataforma.

Figura 9: Caixa de ferramentas da plataforma Microsoft Visual Studio.



Fonte: Autoria própria.

Assim, a vasta biblioteca de extensões do Visual Studio permite que os desenvolvedores personalizem sua IDE com ferramentas que atendem às suas necessidades específicas. Além disso, é possível adicionar extensões que ampliam as capacidades da IDE permitindo que os programadores se concentrem nas tarefas mais importantes sem se preocupar com limitações de funcionalidade.

2.4 LINGUAGEM DE PROGRAMAÇÃO C#

Para desenvolver um *software*, utilizam-se diferentes linguagens de programação. Hoje, há inúmeras opções, mas algumas das mais comuns incluem: Assembly, C, C++, C#, Java, entre outras. Essas linguagens são classificadas em três tipos: linguagens de baixo nível (ou de máquina), linguagens intermediárias como Assembly, e as chamadas linguagens de alto nível. A linguagem de máquina é a única que o computador consegue processar diretamente. Ela se resume a uma série de códigos numéricos que instruem o processador sobre quais tarefas executar. Cada processador é compatível apenas com sua própria linguagem de máquina, e esses códigos são extremamente difíceis de serem interpretados por pessoas, pois são

compostos de números que representam operações muito específicas e pré-definidas (FILHO, 2015).

Com o avanço de linguagens como C++ e Java e o crescimento dos dispositivos móveis, surgiram novas necessidades, como a criação de aplicativos *web*. Os desenvolvedores perceberam que os usuários não estavam mais limitados a *desktops*, exigindo *softwares* acessíveis para diferentes dispositivos. Em resposta a essa demanda, a Microsoft lançou a plataforma .NET, junto com a linguagem de programação C# (pronuncia-se “Cê Charp” em português) (FILHO, 2015).

> C# é uma linguagem de programação que foi baseada em C++ e sofreu grandes influências da linguagem de programação Java. A linguagem de programação C# é orientada a objetos e é considerada como simples e de grande desempenho, pois aproveita de características de outras linguagens que foi originada (OLIVEIRA et al., 2016, p. 2).

Segundo GOMES e OLIVEIRA (2015) algumas características do C# são:

a) Baixa complexidade;
b) Possui classes, atributos, métodos e objetos como qualquer outra linguagem orientada a objetos;
c) Evita erros de atribuições;
d) Linguagem gerenciada;
e) Controle de versões.

2.5 BIBLIOTECA *DRIVER VECTOR*

Para a realizar as diferentes conexões entre *software* e *hardware* é necessário que no código de programação exista algumas bibliotecas, muitas vezes disponibilizadas pelo próprio fabricante do equipamento. Existem diversas bibliotecas para inúmeras aplicações.

A *Driver Library* permite o desenvolvimento de aplicações próprias para CAN, CAN FD (*CAN Flexible Data-rate*), LIN, MOST (*Media Oriented Systems Transport*), Ethernet e entrada/saída digital/analógica.

Ao utilizar essa biblioteca é permitida a comunicação entre a CANcase e o *softwares* programados por outros desenvolvedores, não limitando apenas ao sistema do fabricante. Além disso, é possível definir as configurações de *hardware* necessárias, como a atribuição de canal físico etc. As aplicações possibilitam ler os

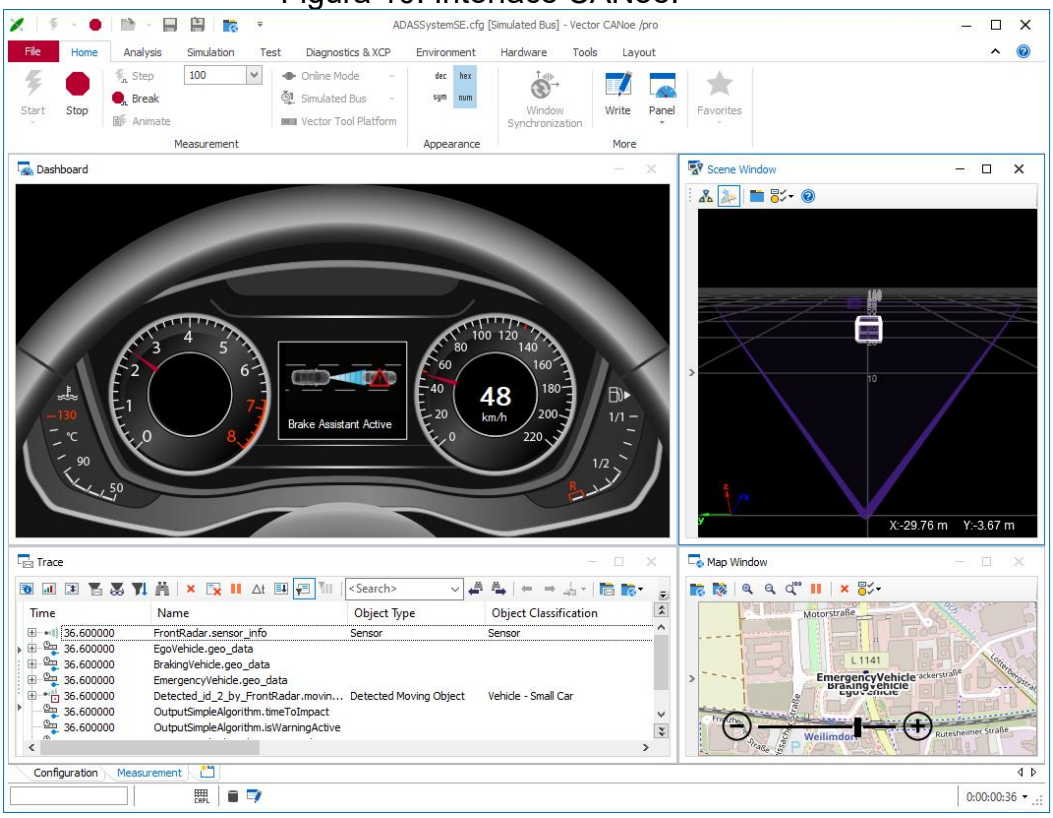
parâmetros em tempo de execução através de um nome de aplicação definido pelo usuário.

Segundo o Manual da Vector (2020) o uso dessa biblioteca pode ser dividido em três etapas principais: inicialização do driver, configuração do canal e tarefas de barramento.

2.6 CANOE

O CANoe (*CAN Open Environment*) é uma ferramenta de *software* desenvolvida pela Vector Informatik GmbH, voltada para o desenvolvimento e teste de redes de ECUs no setor automotivo. Utilizada extensivamente por montadoras e fornecedores de ECUs, a ferramenta auxilia em tarefas como desenvolvimento, simulação, diagnóstico, análise e inicialização de redes e módulos. Devido ao seu amplo suporte para diversos sistemas de barramento de veículos, o CANoe é altamente eficiente tanto no desenvolvimento de ECUs para veículos convencionais quanto para veículos híbridos e elétricos. A ferramenta é compatível com vários tipos de barramento, incluindo CAN, LIN, Ethernet e MOST, além de suportar outros protocolos baseados em CAN (PEREIRA, 2020).

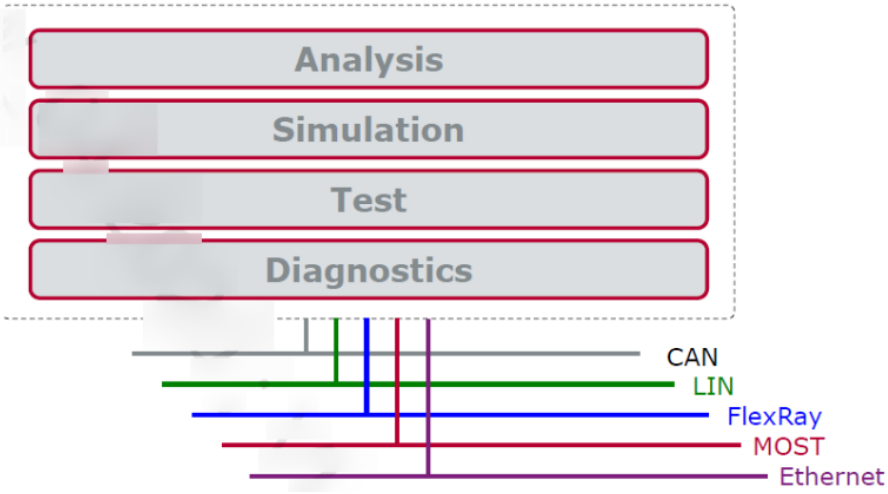
Figura 10: Interface CANoe.



Fonte: Vector (2024).

O CANoeb (interface mostrada na figura 10) permite a simulação de toda a comunicação entre componentes eletrônicos de um veículo, possibilitando testes completos do sistema sem que o veículo esteja fisicamente presente. Ela permite que os dados sejam exibidos de forma bruta (como sinais elétricos ou dados binários) ou de forma simbólica, onde os dados são apresentados de forma mais compreensível.

Figura 11: Conexões de barramento.



Fonte: PEREIRA (2020).

A imagem 11 ilustra as conexões de barramento (*bus connections*) que o CANoe pode analisar e gerenciar em uma rede automotiva, destacando as principais funcionalidades suportadas pela ferramenta: Análise, Simulação, Teste e Diagnóstico. Essas funcionalidades são organizadas em camadas, sugerindo que o CANoe pode atuar em diferentes níveis de profundidade e propósito na rede de comunicação. O *software* CANoe dá suporte ao desenvolvimento de redes de veículos e das ECUs relacionadas desde as primeiras fases de desenvolvimento até as etapas de testes posteriores. Além de viabilizar uma simulação completa de todos os componentes, ele também permite a simulação parcial da rede, incorporando ECUs reais ao ambiente simulado.

> O Vector CAPL Browser é uma ferramenta que acompanha o CANoe, sendo utilizada para escrever código que será empregado posteriormente pelo CANoe durante os testes. O código criado no navegador pode, por exemplo, ser usado para gerar relatórios das atividades dos casos de teste ou para controlar certas partes do processo de teste no CANoe. Essa ferramenta é bastante semelhante a outros ambientes de desenvolvimento integrados (IDE) e realiza a compilação do código CAPL antes de o CANoe poder utilizá-lo. Antes da criação do vTESTStudio, o CAPL Browser era a principal ferramenta para a criação de testes. Neste trabalho, ele foi empregado para ajustar e configurar partes antigas da configuração do CANoe, desenvolvidas anteriormente. O CAPL é uma linguagem fácil de usar para quem tem experiência com outras linguagens de programação, oferecendo

> funcionalidades adicionais que são úteis para trabalhar com ECUs e realizar comunicações via CAN (ESPFORS, 2018, p.19).

O uso do Vector CAPL Browser foi essencial para ajustar configurações antigas no CANoe, oferecendo controle preciso dos testes e comunicação eficiente com módulos automotivos via CAN. Mesmo com a introdução do vTESTStudio, o CAPL Browser permanece relevante por sua simplicidade e capacidade de personalização, além de garantir uma integração segura e eficaz entre o *software* e as ECUs durante os testes.

Os testes podem ser programados em CAPL (*Communication Access Programming Language),* XML ou C#, permitindo flexibilidade para os engenheiros configurarem e automatizarem os testes conforme necessário.

A CAPL, é utilizada nos programas CANoe e CANalyzer da Vector. Baseada na linguagem C, CAPL adiciona recursos específicos para o desenvolvimento de sistemas embarcados baseados em CAN. Ela é uma linguagem orientada a eventos, permitindo o desenvolvimento de aplicações que respondem a diversos eventos do sistema, como pressionamento de teclas, temporizadores de *software* e mensagens CAN, executando rotinas de forma similar a uma interrupção (PENDRILL, 2016).

Com isso, a CAPL se estabelece como uma ferramenta essencial para o desenvolvimento e teste de sistemas embarcados automotivos, oferecendo flexibilidade e eficiência na comunicação e no controle de eventos específicos do sistema.

3 MATERIAIS E MÉTODOS

3.1 MATERIAIS

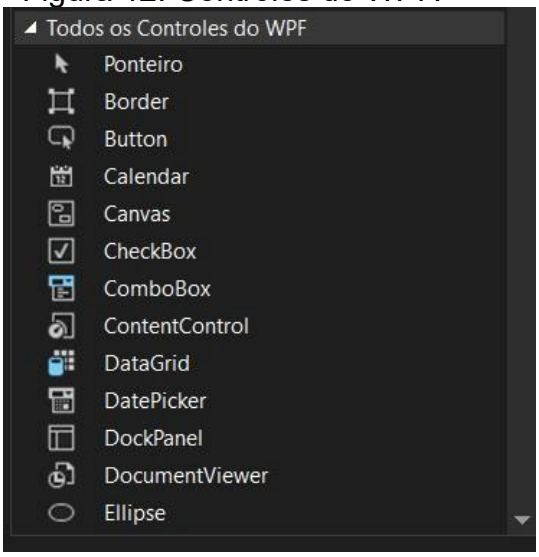
Os materiais necessários para a concepção deste projeto podem ser separados em duas áreas. A primeira delas refere-se aos *hardwares* utilizados para a realização dos testes de integração e comunicação, enquanto a segunda ao *softwa*re que irá concatenar as informações codificadas e transferir as informações desejadas para o dispositivo automotivo.

3.1.1 Software

O Visual Studio 2022, um ambiente de desenvolvimento integrado versátil, ideal para criar a aplicação desejada, oferece diferentes versões, podendo ser gratuito ou pago (com recursos adicionais). No desenvolvimento deste trabalho, foi utilizada a versão Visual Studio *Community*, que é totalmente gratuita e adequada para projetos individuais, acadêmicos e de pequeno porte.

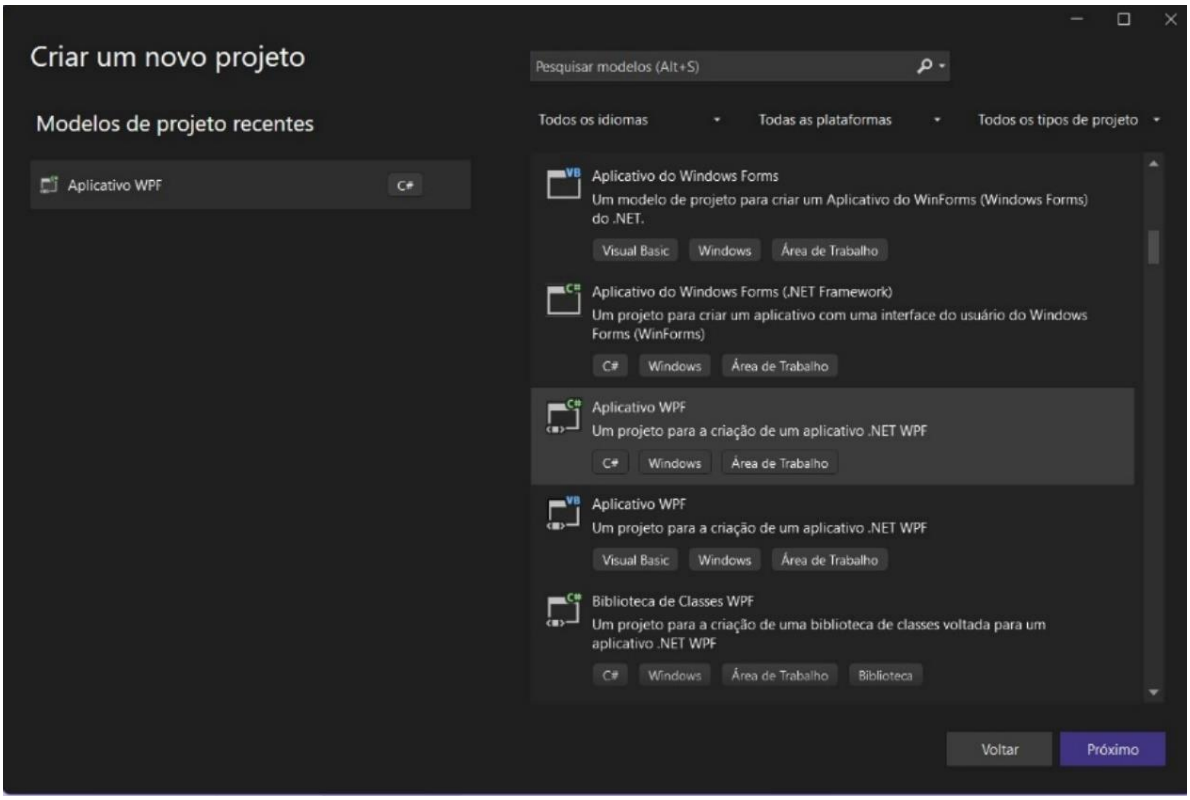
A linguagem utilizada para a estruturação da lógica foi o C#, pois é uma linguagem desenvolvida pela Microsoft capaz de agregar funcionalidades do C++ e do Java, e o Visual Basic .NET, para integrar o *front-end* e *back-end* do projeto. Por fim, em comum com as linguagens de desenvolvimento, fez-se uso da biblioteca WPF (figuras 12 e 13), capaz de fornecer ferramentas práticas para o desenvolvimento das interfaces gráficas, e do *framework* .NET, por se apresentar como uma plataforma robusta de desenvolvimento de *software*.

Figura 12: Controles do WPF.



Fonte: Autoria própria.

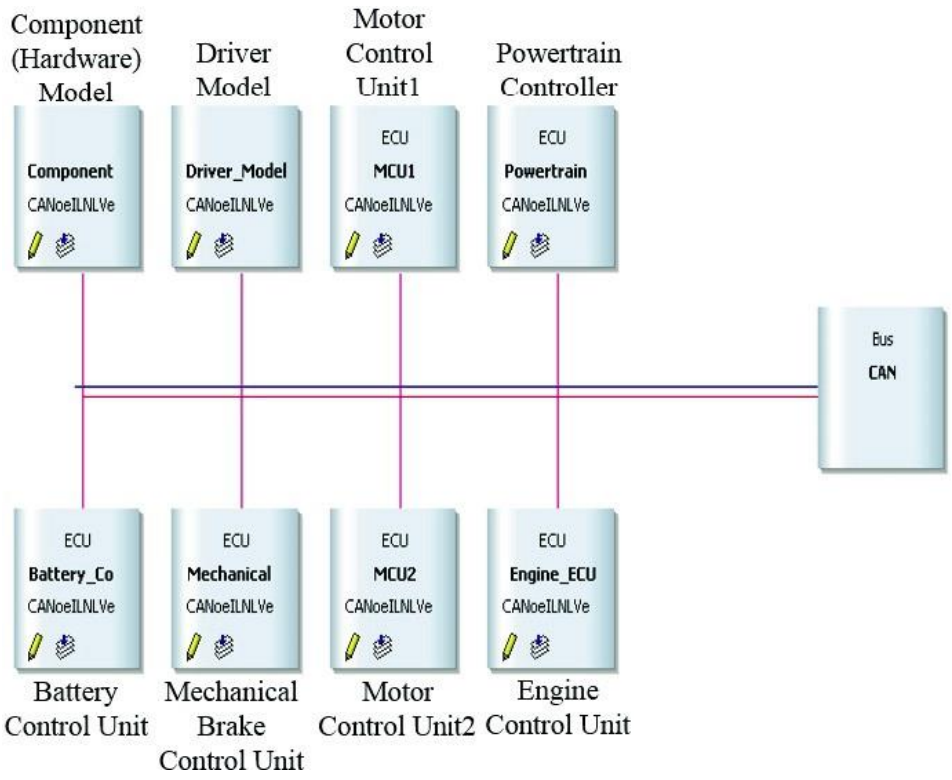
Figura 13: Biblioteca WPF.



Fonte: Autoria própria.

Os testes contaram com suporte extensivo do CANoe, uma ferramenta amplamente utilizada no desenvolvimento, análise e teste de componentes de *software*, desde subsistemas individuais até redes completas (figura 14). Esse *software* versátil foi essencial para a criação de uma simulação detalhada de uma ECU, permitindo a comunicação CAN bidirecional necessária para a leitura e escrita de parâmetros de forma realista e controlada. Além disso, o CANoe facilitou a validação e depuração da comunicação, proporcionando um ambiente robusto para testar o comportamento do sistema em condições simuladas, assegurando que o sistema funcione conforme esperado antes de ser integrado a um ambiente real de operação. Com essa configuração, foi possível realizar testes detalhados, monitorando o comportamento da rede e validando a integridade das mensagens CAN.

Figura 14: Ambiente de simulação de ECUs no CANoe.



Fonte: SANKAVARAM et. al. (2012).

3.1.2 Hardware

Para a realização dos testes, foi utilizada uma CANcase, apresentada na figura 15, um dispositivo de *hardware* especializado que permite a comunicação eficiente entre o computador e a rede CAN. Esse equipamento é fundamental para viabilizar a interface entre o *software* em desenvolvimento e a rede CAN, possibilitando o envio e recepção de mensagens CAN de forma precisa e confiável.

Figura 15: CANcase.

Fonte: Vector (2024).

Adicionalmente, um módulo automotivo foi empregado como receptor das mensagens CAN, sendo o principal destinatário dos dados estruturados e transmitidos pelo *software*. Esse módulo simula as condições reais de funcionamento de um sistema automotivo, permitindo avaliar a funcionalidade e a performance das mensagens enviadas e recebidas dentro da rede CAN.

Na tabela 1 é mostrado a custo de cada equipamento utilizado no projeto.

Tabela 1: Estimativa de Custo.

ESTIMATIVA DE CUSTO	
Equipamentos	**Preço**
CANcase	R$ 2.000,00
Módulo Automotivo	R$ 2.000,00
Total	R$ 4.000,00

Fonte: Autoria própria.

3.2 METODOLOGIA

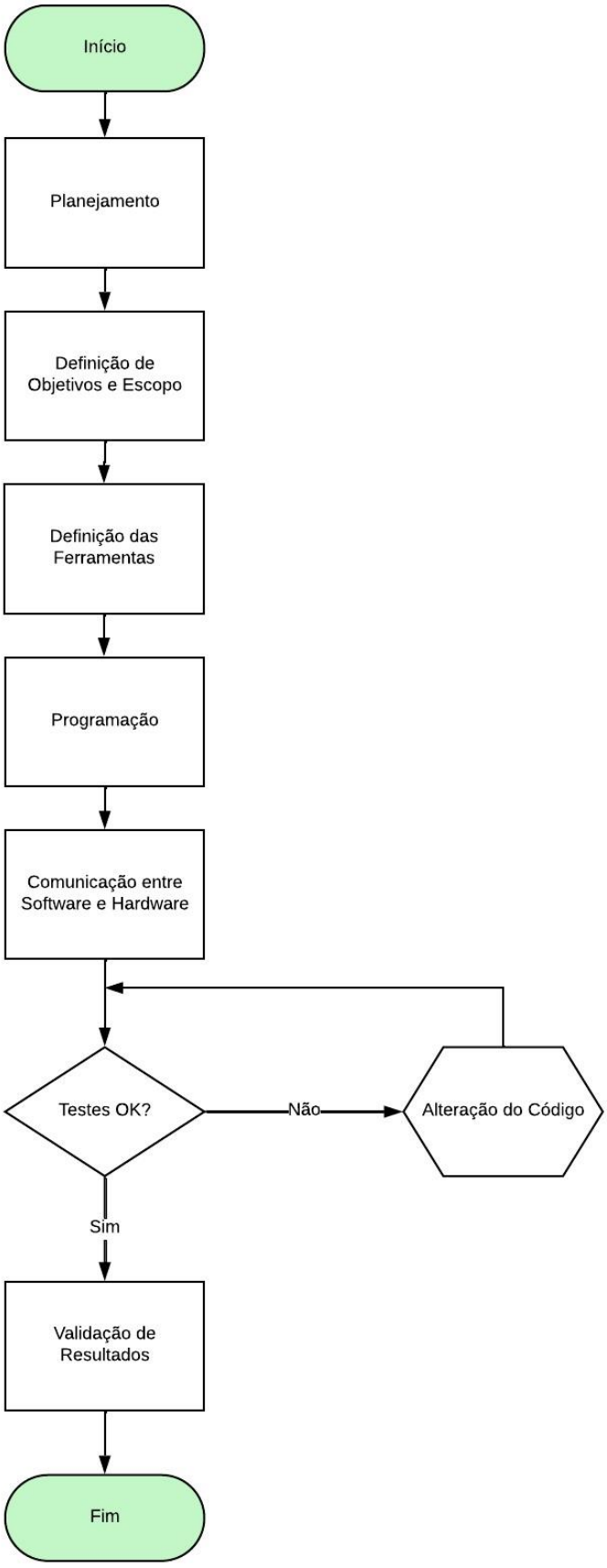
A metodologia aplicada no desenvolvimento deste *software* inclui várias etapas fundamentais, delineadas no fluxograma pertinente a esta seção, exposto na figura 16.

O projeto iniciou com a fase de planejamento, onde foram definidas as principais metas, prazos e recursos necessários. A partir desse ponto, realizou-se uma análise detalhada das necessidades do projeto, que permitiu identificar os requisitos funcionais e técnicos, bem como os desafios específicos a serem superados. Esse levantamento foi essencial para estabelecer os objetivos e o escopo, assegurando que todas as ações subsequentes estivessem alinhadas às demandas da organização.

Com os objetivos e escopo definidos, passou-se à seleção das ferramentas e da linguagem de programação mais adequadas para o desenvolvimento do *software*.

Na fase seguinte, o foco foi na execução do código e no *design* da interface gráfica (*front-end*), aspectos essenciais para garantir tanto a funcionalidade do *software* quanto a experiência do usuário final. Posteriormente, foi iniciada a etapa de testes e validação. Essa fase foi crucial para garantir que o *software* atendesse aos requisitos definidos, fosse estável e funcionasse corretamente.

Figura 16: Fluxograma das etapas gerais.



Fonte: Autoria própria.

3.2.1 Planejamento

Nesta etapa, foi realizada uma reunião estratégica com o objetivo de definir o planejamento geral do projeto. Durante o encontro, foram discutidas e organizadas as principais atividades a serem desenvolvidas, estabelecendo prazos específicos para cada uma delas. Além disso, foram levantados possíveis imprevistos e desafios que poderiam surgir ao longo do desenvolvimento, o que nos permitiu adotar uma abordagem preventiva e flexível para lidar com eventuais problemas. Com base nessas discussões, elaboramos um cronograma inicial, que serve como um guia para o progresso do projeto, garantindo o cumprimento das metas dentro dos prazos estipulados.

O cronograma pode ser visualizado na tabela 2, detalhando as etapas-chave, proporcionando uma visão clara e estruturada do andamento das atividades.

Tabela 2: Cronograma.

CRONOGRAMA								
Plano de Trabalho 2024	abr.	mai.	jun.	jul.	ago.	set.	out.	nov.
Ferramenta e linguagem de programação.								
Aprendizado da linguagem e estruturação do código.								
Pesquisa de hardware adequado.								
Desenvolvimento do artigo.								
Testes funcionais, de desempenho e de integração.								
Reunião com o orientador e coorientador.								
Submissão do Artigo SEAC e definição da banca.								
Entrega para a banca.								
Defesa.								
Encadernação.								

Fonte: Autoria própria.

A maioria das etapas previstas foi concluída dentro do prazo estipulado, demonstrando o progresso eficiente do projeto até o momento. No entanto, algumas atividades precisaram ser modificadas e reestruturadas de acordo com as necessidades que surgiram durante o desenvolvimento, garantindo que o planejamento permanecesse adaptável às novas circunstâncias. Essa flexibilidade foi fundamental para manter o projeto em andamento, apesar de ajustes pontuais que se fizeram necessários ao longo do processo.

3.2.2 Objetivos e Escopo

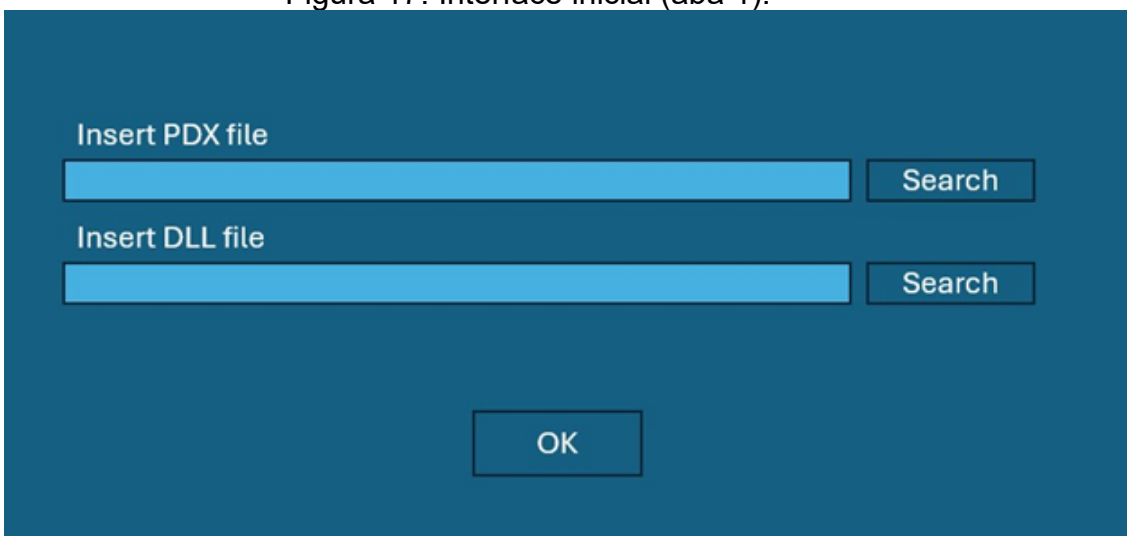
Para definir os objetivos do projeto, foi conduzida uma análise dos problemas enfrentados pela organização, com o intuito de identificar as necessidades específicas e as oportunidades de melhoria nos processos operacionais. Essa análise permitiu mapear os principais desafios, fornecendo uma base sólida para a definição de objetivos estratégicos e mensuráveis. Os objetivos estabelecidos focam tanto na resolução desses desafios quanto na implementação de soluções que agreguem valor, melhorem a eficiência e promovam inovações nos procedimentos internos.

Além disso, foi elaborado um escopo detalhado para o desenvolvimento do *front-end*, que incluiu uma descrição precisa dos requisitos funcionais e visuais. Esse escopo foi fundamental para guiar a escolha da ferramenta de desenvolvimento, uma vez que delineou as exigências técnicas e de design necessárias para garantir uma interface de usuário funcional. A partir desse escopo, foi possível identificar as limitações de algumas plataformas.

Esse processo de definição de objetivos e escopo não apenas direcionou a escolha das tecnologias adequadas, mas também garantiu que todas as etapas do desenvolvimento fossem alinhadas às expectativas, resultando em uma solução eficiente e bem estruturada.

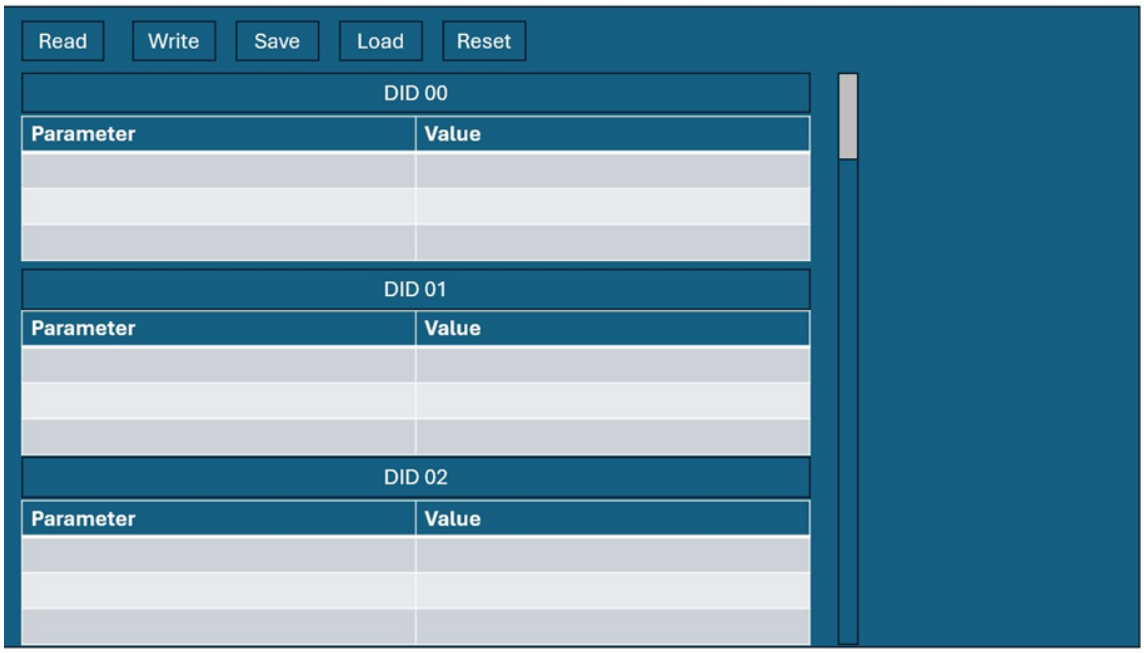
Nessa etapa, também foi desenvolvida uma interface preliminar (figuras 17 e 18) como um protótipo visual, com o objetivo de servir como parâmetro inicial para o projeto. Essa interface foi essencial para proporcionar uma visão clara e tangível de como o *software* poderia ser estruturado em termos de usabilidade, *layout* e funcionalidades.

Figura 17: Interface inicial (aba 1).



Fonte: Autoria própria.

Figura 18: Interface inicial (aba 2).



Fonte: Autoria própria.

3.2.3 Definição das Ferramentas

Inicialmente, foi realizada uma análise da viabilidade de utilizar a plataforma CodeBlocks em conjunto com a linguagem de programação C++ para o desenvolvimento do *software*. Essa escolha considerava a robustez e a eficiência do C++ em termos de desempenho e a familiaridade da equipe com a plataforma. No entanto, ao aprofundar a análise, foi identificado que a interface gráfica do CodeBlocks apresenta limitações significativas, especialmente no que se refere à implementação de elementos visuais interativos e avançados, essenciais para a criação da interface. Essa restrição comprometeria a experiência do usuário final e a flexibilidade do *design*, levando à decisão de optar por uma plataforma que ofereça maior suporte a componentes gráficos e uma interface de desenvolvimento mais rica e adaptada às necessidades do projeto.

Diante das limitações identificadas, foram realizadas novas pesquisas para encontrar uma plataforma que melhor atendesse às necessidades específicas do projeto. Após uma análise criteriosa das opções disponíveis, o Visual Studio foi escolhido como a ferramenta ideal.

A configuração do ambiente de desenvolvimento envolveu a instalação e configuração do Visual Studio 2022, além de outras ferramentas essenciais.

3.2.4 Programação

A programação é uma das etapas centrais no desenvolvimento deste *software*, responsável por traduzir as necessidades do projeto em um conjunto de instruções lógicas e funcionais.

Ao longo desta etapa, será apresentado um fluxograma nas figuras 19, 21, 24 e 25, contendo os principais estágios do código, destacando a lógica implementada para realizar a comunicação entre o *software* e os módulos automotivos, bem como a criação das interfaces gráficas que facilitam a interação do usuário com o sistema. Essa comunicação é essencial para a integração dos dados e controle dos dispositivos, permitindo a troca de informações entre o sistema e os componentes veiculares.

Ao longo do desenvolvimento, diversos desafios foram enfrentados e solucionados. Um dos principais obstáculos esteve relacionado às limitações comportamentais de certos elementos gráficos usados inicialmente, alguns controles da biblioteca WPF não ofereciam a flexibilidade necessária para atender às exigências específicas do projeto. Como solução, foi necessário buscar alternativas dentro da própria biblioteca WPF, substituindo esses elementos por outros que atendessem melhor às expectativas em termos de funcionalidade e usabilidade.

Na fase de *design*, foram criados modelos das interfaces gráficas para definir a arquitetura do *software*, assegurando uma base sólida para o desenvolvimento subsequente. Após o *design*, iniciou-se a construção do código, em que cada componente foi desenvolvido conforme os modelos previamente estabelecidos. Durante esta fase, foram realizados testes unitários para garantir que cada componente do *software* funcionasse corretamente de forma isolada, minimizando o risco de falhas ou inconsistências no comportamento geral do *software*.

Outro desafio encontrado ocorreu na comunicação com a interface CANCase, usada como curso de comunicabilidade com os módulos automotivos. A complexidade dessa integração exigiu uma busca por soluções mais especializadas. Para resolver o problema, foi necessário consultar o site da Vector, uma das principais fornecedoras de ferramentas para o desenvolvimento e testes de redes CAN, a fim de estudar a documentação técnica disponível. Isso possibilitou não apenas a integração funcional, mas também um melhor entendimento do comportamento esperado durante a comunicação com a rede automotiva, garantindo uma operação estável e eficiente do sistema.

Figura 19: Fluxograma programação parte 1.

Fonte: Autoria própria.

Conforme figura 19 acima, no início do desenvolvimento do *software*, foi definido quais os componentes gráficos utilizados (como botões, caixa de texto, *expander* etc.). Isso inclui configurar a aparência e funcionalidade de cada elemento com os quais o usuário poderá visualizar e interagir.

Na etapa seguinte a plataforma foi organizada em diferentes seções ou “abas”. Essa separação representa diferentes aplicações, que permite uma navegação fácil entre os elementos.

Na inicialização do programa, foi utilizado o *Thread*, uma execução de processos que permite que partes do código sejam executadas simultaneamente, no carregamento do código, conforme figura 20. Isso é útil para melhorar o desempenho

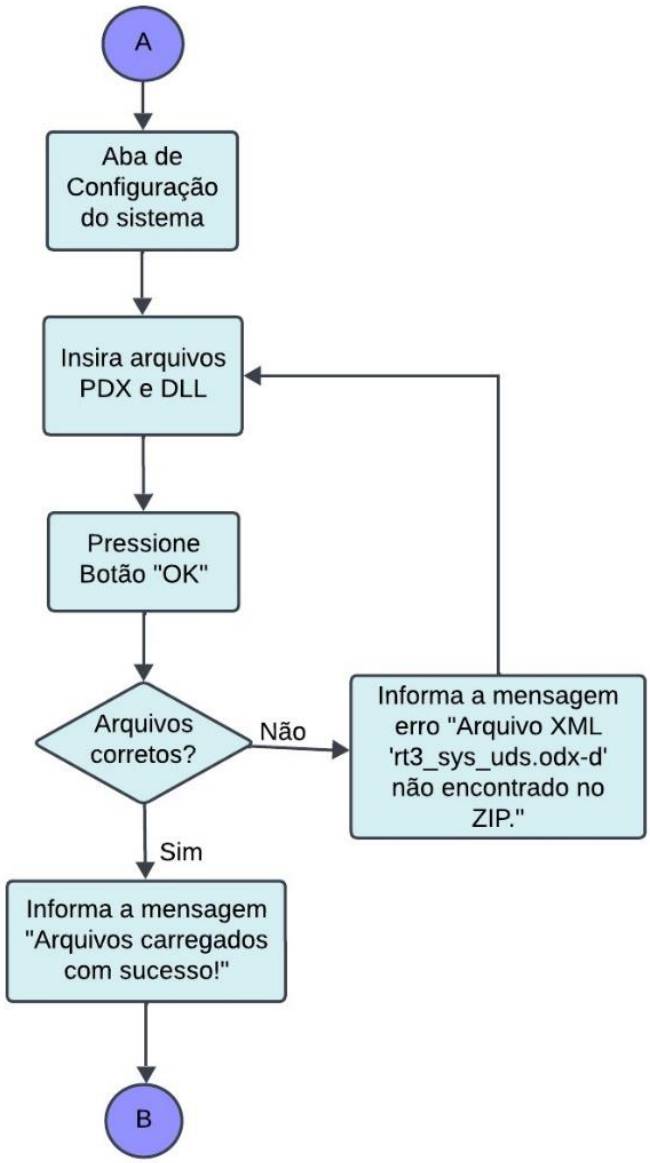
da aplicação, especialmente em operações que podem levar tempo, como leitura de dados do módulo CAN, sem bloquear a interface do usuário.

Figura 20: Unidade de execução Thread.

```
public MainWindow()
{
    InitializeComponent();

    ThreadReadingCAN.DoWork += new DoWorkEventHandler(ThreadReadingCAN_DoWork);
    ThreadReadingCAN.RunWorkerCompleted += new RunWorkerCompletedEventHandler(ThreadReadingCAN_RunWorkerCompleted);
    ThreadReadingCAN.ProgressChanged += new ProgressChangedEventHandler(ThreadReadingCAN_ProgressChanged);
}
```

Fonte: Autoria própria.

Figura 21: Fluxograma programação parte 2.



Fonte: Autoria própria.

A aba "Configuração" é o primeiro *TabItem* do projeto e permite ao usuário carregar os arquivos essenciais para o funcionamento do sistema. Nesta seção, o usuário encontra botões "*Search*", que, ao serem acionados, abrem uma janela de navegação nos arquivos do computador, limitando a seleção a arquivos com as extensões .pdx e .dll, conforme ilustrado no código da figura 22. Ao escolher um arquivo, seu caminho completo será exibido na tela, facilitando a verificação visual.

Figura 22: Seleção de arquivos.

```
private void Button_Searchpdx(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "PDX Files (*.pdx)|*.pdx";
    if (openFileDialog.ShowDialog() == true)
    {
        TextBox_pdxFilePath.Text = openFileDialog.FileName;
    }
}

0 referências
private void Button_Searchdll(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "DLL Files (*.dll)|*.dll";
    if (openFileDialog.ShowDialog() == true)
    {
        TextBox_dllFilePath.Text = openFileDialog.FileName;
    }
}
```

Fonte: Autoria própria.

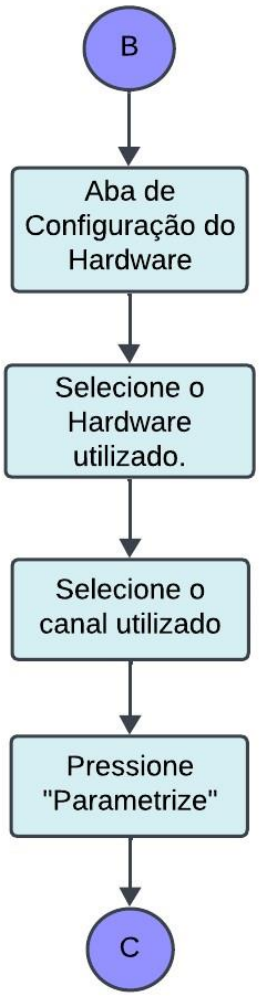
Após selecionar os arquivos e confirmar com o botão "OK", o sistema inicia o processo de validação dos arquivos carregados. Caso algum dos arquivos não tenha sido selecionado, será exibida a mensagem "Por favor, selecione ambos os arquivos", conforme ilustrado no código da figura 23. Se ambos os arquivos atenderem aos requisitos, a mensagem "Arquivos carregados com sucesso" será exibida, e o fluxo de processamento avançará automaticamente até o ponto B (figura 24). Caso algum arquivo esteja incorreto ou o sistema não consiga localizar o arquivo XML esperado no pacote .zip, uma mensagem de erro será exibida: "Erro: Arquivo XML não encontrado no zip". Nesse caso, o usuário retornará ao processo inicial, devendo inserir os arquivos novamente.

Figura 23 : Mensagem indicando que os dois arquivos devem ser selecionados.

```
private void Button_OK(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(TextBox_pdxFilePath.Text) || string.IsNullOrEmpty(TextBox_dllFilePath.Text))
    {
        MessageBox.Show("Por favor, selecione ambos os arquivos.");
        return;
    }

    MainTabControl.SelectedIndex = 1;
    MessageBox.Show("Arquivos carregados com sucesso!");
    LoadZipAndProcessXml(TextBox_pdxFilePath.Text);
}
```

Fonte: Autoria própria.

Figura 24: Fluxograma programação parte 3.



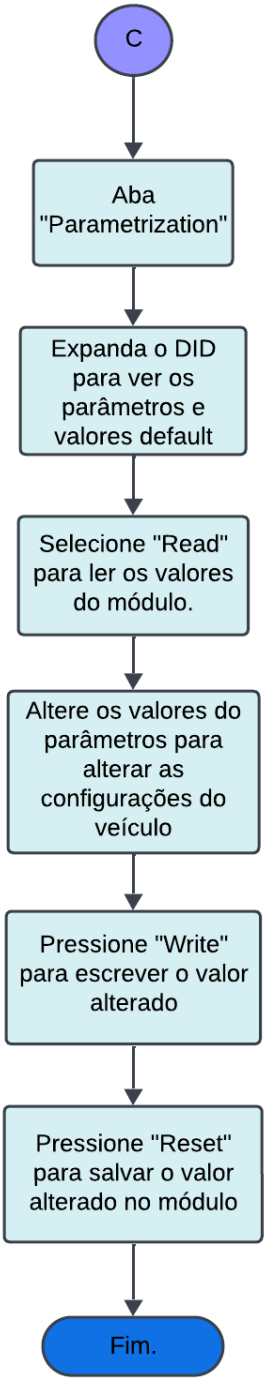
Fonte: Autoria própria.

A aba "CANCaseConfig" permite configurar o *hardware* utilizado para comunicação com a rede CAN. Nessa aba, ao clicar nos botões específicos, o usuário poderá selecionar o modelo do dispositivo CANcase e definir os canais que serão utilizados para a comunicação. Essas opções garantem que o sistema esteja

corretamente configurado para a interação com a rede CAN, de acordo com o modelo e as necessidades de comunicação do projeto.

Após concluir as configurações e selecionar a opção "Parametrize", o sistema direcionará automaticamente o usuário para a aba "*Parametrization*" (C), onde será possível prosseguir com a definição detalhada dos parâmetros. A estrutura do código desta etapa é ilustrada na figura 25.

Figura 25: Fluxograma programação parte 4.



Fonte: Autoria própria.

Nesta etapa, o usuário poderá visualizar os parâmetros específicos do módulo desejado. Ao expandir um determinado DID (*Data Identifier*), uma lista detalhada dos parâmetros do módulo será exibida, incluindo os valores *default* definidos pelo *software*. Essa visualização facilita o entendimento dos valores iniciais atribuídos a cada parâmetro e permite ao usuário uma análise detalhada antes de prosseguir para as etapas seguintes.

O código ilustrado abaixo na figura 26 demonstra a lógica implementada para exibir os parâmetros e seus valores iniciais, detalhando como a estrutura de dados foi organizada para facilitar a expansão dos DIDs e a exibição de suas informações correspondentes.

Figura 26: Código para visualização dos parâmetros.

```
RequestObject requestSaved = new()
{
    ShortName = requestShortName
};

Expander expander = new()
{
    Header = requestShortName,
    IsExpanded = false
};

ListView listView = new();
GridView gridView = new();

gridView.Columns.Add(new GridViewColumn { Header = "Short Name", DisplayMemberBinding = new System.Windows.Data.Binding("ShortName") });
gridView.Columns.Add(new GridViewColumn
{
    Header = "Physical Default Value",
    CellTemplate = CreateEditableTemplate("PhysicalDefaultValue")
});

listView.View = gridView;
listView.PreviewMouseWheel += ListView_PreviewMouseWheel;
```

Fonte: Autoria própria.

Ao selecionar a opção "*Read*", o sistema realiza uma leitura dos valores atualmente configurados no módulo e exibe essas informações para o usuário. Esse processo permite visualizar os parâmetros ativos e verificar as configurações atuais do módulo, servindo como referência para ajustes.

Com os valores carregados na interface, o usuário pode modificar os parâmetros conforme as necessidades específicas do veículo, ajustando-os de forma precisa para atender às especificações desejadas. Após realizar todas as alterações necessárias, o usuário deve pressionar o botão "*Write*" para enviar os novos valores ao módulo. Esse comando transmite as atualizações diretamente ao *hardware*, aplicando as mudanças em tempo real.

Por fim, o botão "*Reset*" permite confirmar e salvar permanentemente os valores ajustados no módulo, garantindo que as novas configurações sejam mantidas após o ciclo de energia. Ao pressionar "*Reset*", o processo de parametrização é concluído, e

o módulo estará totalmente configurado conforme as preferências definidas pelo usuário.

Além disso, para facilitar a navegação na aba, foi implementada uma programação de eventos para a barra de rolagem, conforme figura 27. Essa funcionalidade permite ao usuário manipular a barra de rolagem com o mouse, proporcionando uma experiência de interação mais fluida e intuitiva. O sistema mantém a configuração da barra de rolagem mesmo após a expansão dos DIDs, garantindo que o usuário possa visualizar facilmente todos os parâmetros.

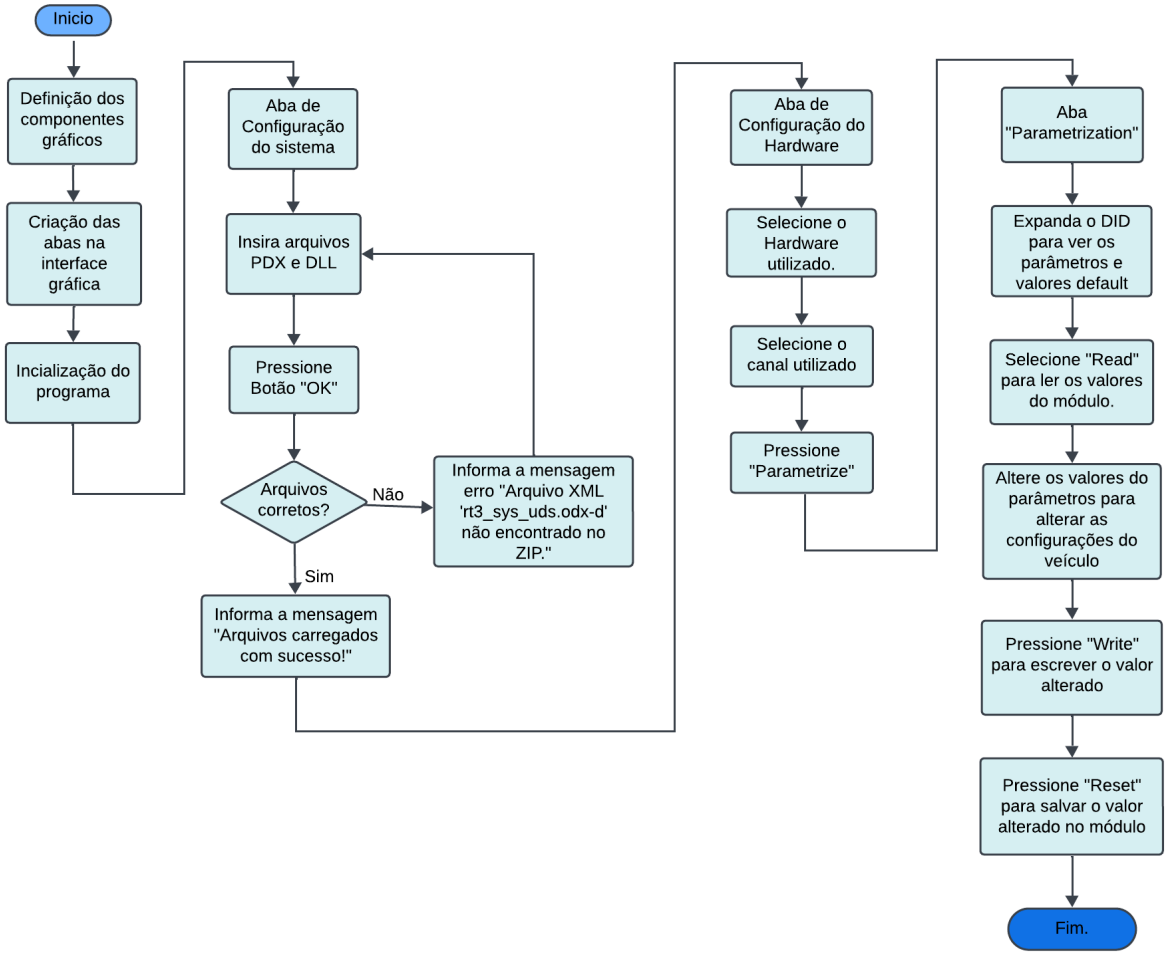
Figura 27: Eventos para a barra de rolagem.

```
// Manipulador para permitir que o ScrollViewer role quando o mouse estiver sobre o ListView
1 referência
private void ListView_PreviewMouseWheel(object sender, System.Windows.Input.MouseWheelEventArgs e)
{
    var listView = (ListView)sender;
    var scrollViewer = FindParent<ScrollViewer>(listView);

    if (scrollViewer != null)
    {
        scrollViewer.ScrollToVerticalOffset(scrollViewer.VerticalOffset - e.Delta / 3);
        e.Handled = true; // Marcar como tratado para evitar o comportamento padrão
    }
}


// Função auxiliar para encontrar o ScrollViewer pai
2 referências
private static T FindParent<T>(DependencyObject child) where T : DependencyObject
{
    DependencyObject parentObject = VisualTreeHelper.GetParent(child);

    if (parentObject == null) return null;

    if (parentObject is T parent)
        return parent;
    else
        return FindParent<T>(parentObject);
}
```

Fonte: Autoria própria.

Por fim, na figura 27 apresenta o fluxograma geral da programação, detalhando as etapas principais e suas interconexões. Esse fluxograma ilustra de forma clara e objetiva a sequência lógica das operações, desde a inicialização do sistema até a finalização do processo. Ele serve como uma ferramenta essencial para compreender o funcionamento do programa, permitindo uma visão abrangente de como os módulos se comunicam, as condições de decisão e os resultados esperados em cada etapa.

Figura 28: Fluxograma geral da programação.



Fonte: Autoria própria.

3.2.5 Comunicação entre Software e Hardware

Nesta seção, foram iniciados os testes de integração, nos quais verificou-se a comunicação entre os diferentes componentes do *software* e a interação com a CANcase e o módulo automotivo. Esses testes tiveram como objetivo assegurar que as diversas partes do sistema, desenvolvidas de forma modular, fossem capazes de funcionar de maneira coordenada e eficiente quando integradas em um ambiente unificado. A partir do desenvolvimento do código integrado à biblioteca Driver Vector, foi possível obter uma comunicação eficiente entre o *software* e a CANcase, responsável pela interface de comunicação com o barramento CAN. Assim, esse equipamento desempenhou um papel central, garantindo que os dados fossem transmitidos e recebidos corretamente entre o *software* e o módulo automotivo.

Durante essa etapa, foi verificado não apenas a troca de informações, mas também o sincronismo entre os comandos enviados pelo *software* e as respostas do

hardware automotivo. Foram analisados falhas de comunicação ou incompatibilidades, permitindo a identificação e resolução de possíveis falhas antes da fase de testes finais.

Esse processo envolveu ajustes no código e na arquitetura do *software*, assegurando que a funcionalidade do sistema fosse mantida mesmo em situações complexas de troca de dados em tempo real. Dessa forma, essa fase se mostrou crucial para validar a robustez e a estabilidade da solução desenvolvida.

3.2.6 Testes

Na etapa final do processo de desenvolvimento, foram realizados testes de desempenho com o intuito de garantir que o *software* operasse de maneira eficiente em uma variedade de condições de uso. Essa fase envolveu a simulação de cenários reais, projetados para replicar as situações que o *software* enfrentaria em um ambiente industrial. Durante esses testes, foram avaliadas diferentes métricas de desempenho, incluindo tempo de resposta, uso de recursos do sistema e capacidade de processamento sob carga.

Os testes foram estruturados para identificar e corrigir quaisquer gargalos de performance que pudessem impactar negativamente a experiência do usuário. Para isso, diferentes níveis de estresse foram aplicados ao *software*, simulando condições como alta demanda de dados e múltiplas interações simultâneas. Essa abordagem abrangente permitiu não apenas a detecção de problemas, mas também a avaliação da robustez do sistema frente a condições desafiadoras.

Para a simulação do módulo, utilizou-se o *software* CANoe, onde foi elaborado um modelo que replicasse com fidelidade o funcionamento do *hardware* real. Através dessa simulação, desenvolveu-se a lógica de comunicação utilizando a linguagem CAPL, que permitiu a programação detalhada dos comandos do protocolo UDS *(Unified Diagnostic Services*). As funções de requisição *(request*), leitura (*read*) e escrita (*write*) foram implementadas para simular todas as operações de diagnóstico e controle esperadas do módulo. Essa abordagem tornou possível validar o comportamento e a resposta do *software*, garantindo uma maior confiabilidade e eficiência no desenvolvimento e testes.

Além disso, a equipe analisou os resultados obtidos e implementou ajustes refinados no código, otimizações de algoritmos e melhorias na gestão de recursos. Essa ênfase na performance assegurou que o *software* não apenas atendesse aos

requisitos técnicos estabelecidos, mas também proporcionasse uma experiência de usuário de alta qualidade.

3.2.7 Validação de Resultados

A fase de validação de resultados foi essencial no desenvolvimento deste *software*, pois não apenas validou sua eficácia, mas também reforçou o compromisso da equipe em entregar um produto alinhado às melhores práticas da indústria automotiva. Resultando em uma solução que não só atende às expectativas dos usuários, mas também contribui qualidade no ambiente automotivo. Além disso, estabeleceu um padrão de excelência que pode ser replicado em futuros projetos.

4 RESULTADOS E DISCUSSÃO

A partir da implementação do código de programação na IDE Visual Studio, foi possível estruturar um *software* específico para a parametrização de módulos automotivos, atendendo às necessidades técnicas do projeto.

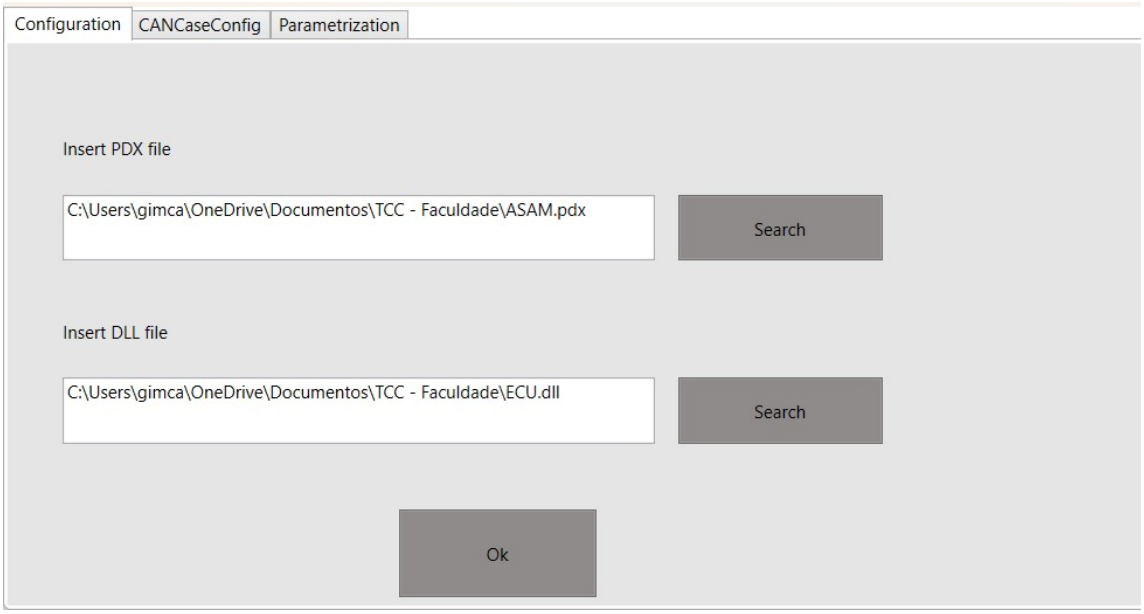
O *software* é composto por três abas principais, cada uma com uma função específica.

Na primeira aba (figura 28), o usuário encontra dois campos para anexar arquivos essenciais com extensões distintas: .dll e .pdx. O arquivo .pdx contém informações detalhadas sobre a versão de *software* em uso, incluindo configurações específicas, mapeamento de parâmetros e valores padrão para o módulo. Esse arquivo armazena todos os parâmetros configuráveis e os dados técnicos essenciais para a parametrização e operação do módulo. Ele serve como um repositório de informações, garantindo que o sistema opere com as configurações adequadas e que o usuário possa acessar, visualizar e ajustar os parâmetros do módulo conforme necessário.

O arquivo com extensão .dll, por sua vez, atua como um componente de segurança e autenticação, funcionando como uma espécie de chave de acesso. Sua função principal é validar o processo, autorizando o acesso ao módulo e permitindo a manipulação e alteração dos parâmetros internos do sistema. Sem a presença do arquivo .dll, qualquer tentativa de modificação do conteúdo do arquivo .pdx é bloqueada, assegurando um nível de segurança adicional e mantendo o controle de acesso ao sistema de forma rigorosa.

Após a inserção dos dois arquivos, o usuário deve selecionar o botão “OK” para que o sistema valide ambos os arquivos e, em seguida, avance automaticamente para a segunda aba, onde o processo de parametrização poderá ser continuado.

Figura 29: Aba *Configuration*.

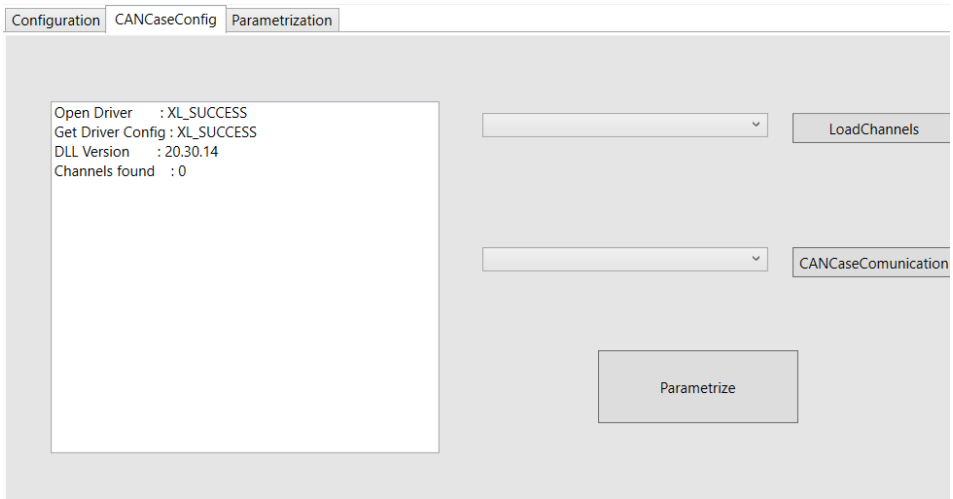


Fonte: Autoria própria.

Na segunda aba (figura 29), é estabelecida a conexão com a CANcase para permitir a comunicação com o módulo automotivo. O usuário deve selecionar o modelo da CANcase de acordo com o dispositivo em uso, garantindo que a interface de comunicação esteja configurada para compatibilidade com o módulo. Além disso, a escolha da porta correta é necessária para que o sistema reconheça a conexão física com o módulo, estabelecendo um canal de comunicação robusto e confiável entre o *software* e o *hardware*.

Após essas definições, o sistema realiza automaticamente uma verificação da integridade da conexão com a CANcase e exibe o status da comunicação em um campo lateral.

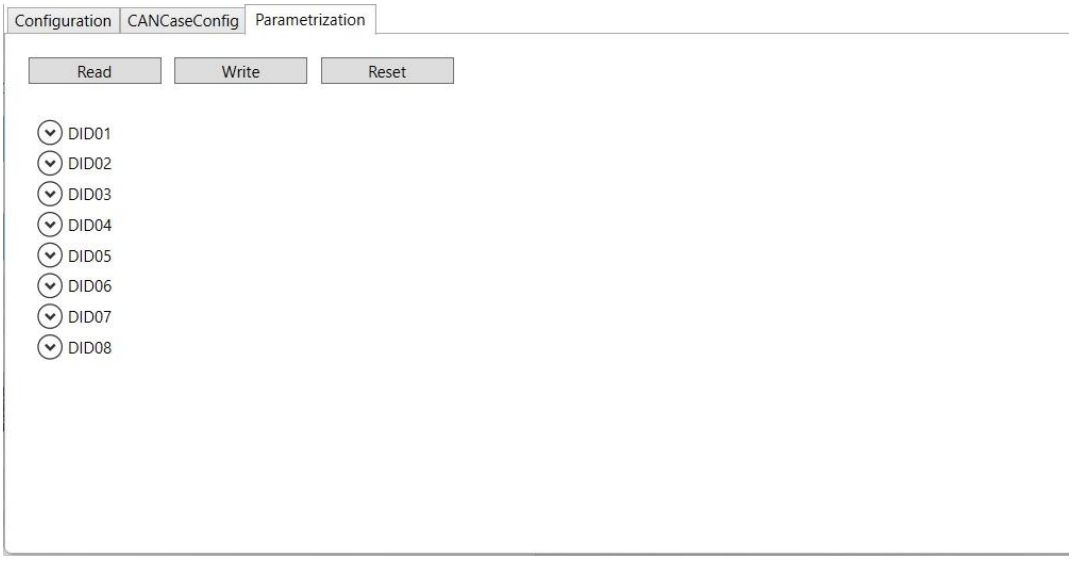
Figura 30: Aba *CanCaseConfig*.



Fonte: Autoria própria.

Na última aba (figura 30), inicia-se o processo de parametrização propriamente dito, que permite ao usuário selecionar e modificar os parâmetros de configuração de acordo com suas necessidades específicas.

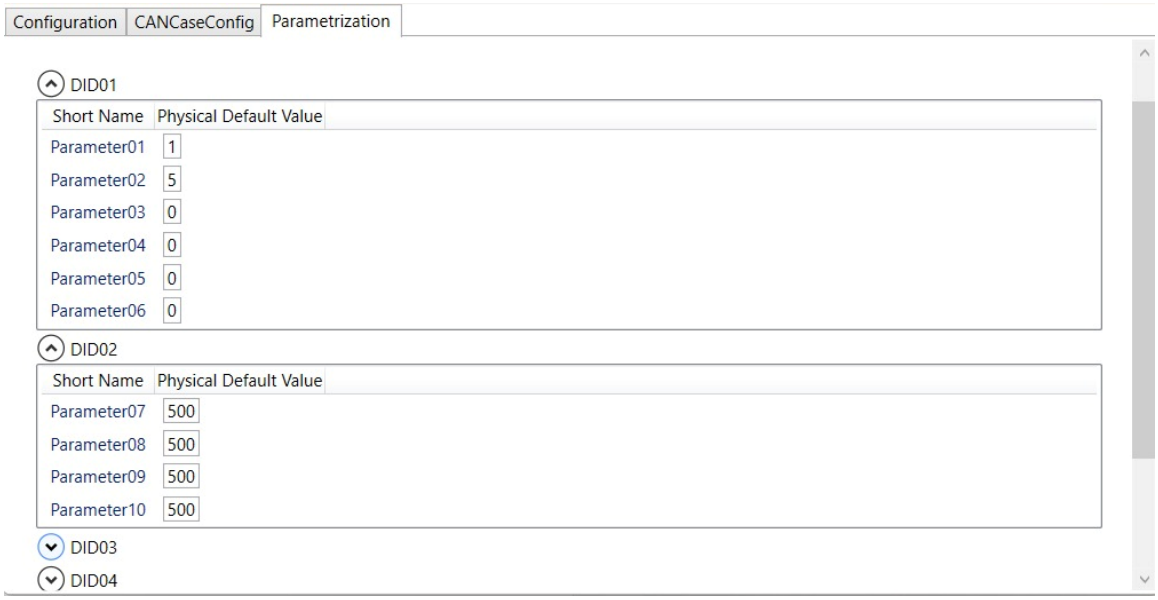
Figura 31: Aba *Parametrization*.



Fonte: Autoria própria.

Nesta etapa, os parâmetros são organizados através de DIDs, que ao serem expandidos revela uma lista detalhada de opções disponíveis. Ao clicar para expandir um DID, o usuário pode visualizar todos os parâmetros associados a ele, conforme ilustrado na figura 31.

Figura 32: DIDs.



Fonte: Autoria própria.

O sistema possibilita que o usuário insira valores específicos para cada parâmetro, garantindo a personalização das configurações do módulo conforme os requisitos do projeto. Além disso, o usuário pode habilitar ou desabilitar determinadas funções, permitindo um controle sobre o comportamento do veículo. Essa flexibilidade é essencial para otimizar o desempenho do veículo e assegurar que todas as especificações sejam atendidas.

Após os parâmetros ajustados, o usuário salva as configurações e, realiza testes para validar as alterações feitas. Essa abordagem assegura que o módulo operará de maneira eficaz, e alinhado com as expectativas e demandas do projeto automotivo.

Posteriormente, foi realizada uma comparação detalhada entre o tempo de configuração do *software* disponibilizado por um fornecedor e o *software* desenvolvido ao longo deste projeto. O objetivo dessa análise era avaliar a eficiência e a eficácia de ambos os sistemas em um cenário prático. Os resultados foram satisfatórios: o *software* desenvolvido no projeto demonstrou uma eficiência superior no tempo de configuração em comparação ao adquirido.

Além de reduzir o tempo total de configuração, o *software* desenvolvido também proporcionou uma interface mais intuitiva, essas melhorias não apenas otimizaram o processo de configuração, mas também contribuíram para uma experiência do usuário mais satisfatória.

5 CONCLUSÃO

Concluindo o projeto, é póssivel afirmar que o objetivo principal foi plenamente alcançado: desenvolver um *software* intuitivo para a parametrização de módulos automotivos. O resultado não apenas atende às expectativas iniciais, mas também se destaca pela sua eficácia e facilidade de uso.

Além de alcançar o objetivo principal, todos os objetivos específicos foram atingidos com sucesso. Foi desenvolvido uma interface clara e amigável, projetada com o usuário final em mente, garantindo acessibilidade e eficiência em cada etapa do uso do sistema. A simplicidade da interface permite que mesmo usuários com pouca experiência em tecnologia consigam navegar pelo *software* com facilidade, reduzindo a curva de aprendizado e aumentando a produtividade.

Os testes realizados durante o desenvolvimento foram abrangentes, validando o funcionamento do *software* em diversos cenários operacionais. Esses testes não apenas confirmaram a eficácia das funcionalidades implementadas, mas também garantiram a confiabilidade do sistema.

Além disso, foi elaborado um manual de instruções detalhado, que serve como um guia completo para o usuário em todas as etapas de uso. Ao fornecer esse suporte é assegurado que os usuários tenham todos os recursos necessários para tirar o máximo proveito da aplicação, facilitando uma transição suave e bem-sucedida para a nova ferramenta.

Em resumo, o projeto não apenas atingiu seus objetivos, mas também criou um produto que se destaca pela sua usabilidade e eficácia, colocando-se como uma solução valiosa para a parametrização de módulos automotivos.

6 INDICAÇÕES PARA TRABALHOS FUTUROS

Para trabalhos futuros, diversas vertentes podem ser exploradas no aprimoramento do *software* de parametrização de módulos automotivos. Uma linha de pesquisa promissora é a incorporação da Ethernet Automotiva, tecnologia que oferece alta taxa de transferência de dados em comparação com protocolos tradicionais, como o CAN, e que vem se tornando fundamental para sistemas automotivos modernos que demandam alta largura de banda, como câmeras de monitoramento, sensores avançados e sistemas de *infotainment*. Ao integrar a Ethernet Automotiva ao *software*, ampliam-se as possibilidades de comunicação com módulos de alta velocidade e otimiza-se a atualização e transmissão de parâmetros em tempo real. A Ethernet Automotiva também facilita a interoperabilidade com sistemas baseados em IP, criando uma plataforma robusta para futuras expansões, como a conexão com infraestruturas externas e serviços em nuvem.

Além da Ethernet Automotiva, o *software* poderia ser adaptado para outros protocolos de comunicação amplamente utilizados na indústria, como o FlexRay e o LIN. A inclusão desses protocolos aumentaria a flexibilidade do sistema, permitindo atender a uma gama mais ampla de módulos e sistemas de controle, o que amplia suas aplicações e torna o *software* mais versátil em ambientes variados. A capacidade de suporte a múltiplos protocolos é um diferencial que se alinha com as necessidades da indústria automotiva, que exige soluções adaptáveis a diversas plataformas de comunicação.

Outra possibilidade de desenvolvimento reside na integração de recursos de segurança avançados para proteger os dados que circulam na rede automotiva. Com a crescente conectividade dos veículos, surgem ameaças de segurança digital que podem comprometer a integridade do sistema. Trabalhos futuros poderiam incluir a implementação de mecanismos de criptografia e autenticação, garantindo que apenas dispositivos autorizados tenham acesso ao *software* de parametrização e evitando possíveis invasões.

Além disso, a implementação de uma biblioteca de scripts configuráveis pelo usuário também representaria um avanço significativo. Essa biblioteca permitiria aos técnicos e engenheiros personalizar e automatizar determinadas parametrizações, otimizando o tempo necessário para configurar os módulos de acordo com as especificações do projeto. Esse recurso traria um diferencial ao *software*, tornando-o uma ferramenta adaptável a diferentes demandas de parametrização.

Por fim, um trabalho futuro poderia explorar o desenvolvimento de um sistema de log detalhado, que registre todas as operações de parametrização realizadas, permitindo um histórico completo para fins de auditoria e controle de qualidade. Esse registro seria útil para rastrear alterações, avaliar o desempenho das configurações implementadas e diagnosticar eventuais problemas com maior precisão.

REFERÊNCIAS

TAFFAREL, V. S. **Mobilidade Urbana: Análise dos fatores que causam o contínuo crescimento do uso do automóvel nas cidades brasileiras.** Universidade Federal do Rio Grande do Sul, Escola de Engenharia – Departamento de Engenharia Civil, Porto Alegre, junho de 2015.

CAVALCANTE, L. H. **Sistema de Monitoramento Automotivo via Rede Can.** Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Câmpus Florianópolis – Departamento Acadêmico de Metal-Mecânica, Bacharelado em Engenharia Mecatrônica. Florianópolis, julho de 2018.

TIRONI, P. O. et al. **Rede Can Automotiva – Perspectivas Gerais e Vulnerabilidades.** XVICEEL - ISSN2178-8308, Universidade Federal de Uberlândia. Uberlândia, novembro de 2018.

SILVA, R. R. **Projeto de controladores para um sistema de direção elétrica utilizando a metodologia de projeto baseado em modelos.** Dissertação de Mestrado. Universidade de Brasília - Faculdade de Tecnologia. Brasília, agosto de 2017.

GIRARDI, A. **Comportamento Térmico de Capacitores Eletrolíticos de Alumínio para aplicações Automotivas**. Universidade Federal do Rio Grande do Sul, Escola de Engenharia – Departamento de Engenharia Elétrica. Porto Alegre, 2015.

MACEDO; LOPES. **Uma revisão sobre Transmissões Automáticas com Sistemas de Engrenagens Planetárias.** Rio de Janeiro, 2020.

NAUNHEIMER, et. al. **Controle de Transmissão Eletrônica**. Transmissões automotivas, 2ª edição, 2011 p.2.

SANTOS, E. D. **Utilização de Redes de Comunicação Automotiva na racionalização de chicotes elétricos automotivos**. Curso de Especialização em Engenharia Automotiva. Faculdade de Tecnologia, SENAI CIMATEC. Salvador, 2012.

BOCCI, E. D. **Projeto de um sistema embarcado de aquisição de dados com implementação e testes de sistema de telemetria**. Universidade Federal de São Carlos - Departamento de Engenharia Elétrica. São Paulo, 2020.

LIMA, A. F. et. al. **Comparação do Desempenho de Frenagem em Veículo com o Sistema ABS Ativo e Inativo**. Trabalho de Conclusão de Curso II, Centro Universitário de Anápolis – UniEvangélica. Anápolis, junho de 2018.

DUTRA, P. R. G. et al**. Avaliação dinâmica de Frenagem de um Veículo envolvido num acidente considerando a atuação de um sistema de freio equipado com Sistema Antibloqueio de Freio – ABS**. IV SIMPÓSITO NACIONAL DE CIÊNCIAS E ENGENHARIAS – SINAGEM. Anápolis, maio de 2020.

GTI RACING PARTS. **MÓDULO DE DISTRIBUIÇÃO DE POTÊNCIA – MILLIATI**. Disponível em: <https://www.gtiracingparts.com.br/>. Acesso em setembro de 2024.

SOUSA, P. M. F. **Desenvolvimento de Sistemas Eletrónicos para um Veículo de Competição Formula *Student***. Dissertação Mestrado em Engenharia Automóvel. Leiria, setembro de 2022.
VITORINO, G. J. de A. F. et. al. **Desenvolvimento de um Gateway Baseado no Protocolo CAN para Integração e Diagnósticos de Periféricos**. Universidade Facens Sorocaba. São Paulo, 2022.

FERREIRA, B. F. et. al. **Desenvolvimento de Checklist para Análise de Falhas em Aplicações Automotiva Eletrônica Embarcada**. 9ª Edição da ISTI (*International Symposium on Technological Innovation*). Aracajú, setembro de 2018.

BORTH, T. F. **Analisando os Impactos do Uso Do Protocolo CAN FD em Aplicações Automotivas – Estudo de Caso.** Programa de Pós-Graduação em Engenharia Elétrica. Universidade Federal Do Rio Grande Do Sul. Escola de Engenharia. Porto Alegre, 2016.

CARDOSO, S. L. **Desenvolvimento de um Gêmeo Digital para um Manipulador Robótico utilizando Padrões Abertos de Automação**. Universidade Federal de Campina Grande – UFCG
Centro de Engenharia Elétrica e Informática - Departamento de Engenharia Elétrica. Campina Grande, 2024.

OLIVEIRA, E. M. **Disseminação de Conceitos de Propriedade Intelectual através de um Role-Playing Game para Estudantes do Instituto Federal da Paraíba.** Programa de Pós-Graduação Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Em propriedade intelectual e transferência de tecnologia para inovação. Campina Grande, 2024.

BRANDÃO, S. R. O. **Sistema para uso da Tecnologia CAN em Veículos Automotivos**. Universidade Federal de Campina Grande. Centro de Engenharia Elétrica e Informática, Departamento de Engenharia Elétrica. Campina Grande, agosto de 2022.

Visual Studio IDE 2022 – **Ferramenta de Programação.** Disponível em:< https://visualstudio.microsoft.com/pt-br/vs/ >. Acesso em 15 de agosto de 2024.

FILHO, V. P. **Sistemas Supervisórios utilizando Linguagem de Programação C#**. Centro Bacharelado em Sistemas de Informação. Universitário do Sul de Minas - UNIS-MG. Varginha, 2015.

OLIVEIRA, L. C. et al. **Realidade Virtual aplicada no desenvolvimento de um Serious Game para Reabilitação de Cadeirantes utilizando Kinec**t. XV SBGames, São Paulo, 2016.

GOMES; OLIVEIRA. **Tecnologia Assistiva aplicada no desenvolvimento de um Jogo para Reabilitação de Indivíduos com Deficiência** – XIII CEEL - Universidade Federal de Uberlândia. Outubro, 2015.

XL DRIVER LIBRARY MANUAL. Version 20.30 - English. Acesso em 20 de setembro de 2024. Disponível em <

file:///C:/Users/ms94947/Downloads/XL_Driver_Library_Manual_EN.pdf>.

Pereira, F. R. S. **Testes Automáticos utilizando o Vector CANoe num Sensor Inercial.** Mestrado em Engenharia Eletrotécnica e de Computadores. Novembro de 2020.

ESPORS, Nils. **CANoe - Integração do Simulink do Modelo de Veículo no Ambiente de Teste Existente.** Divisão de Engenharia Elétrica Industrial e Automação, Faculdade de Engenharia, Universidade de Lund, 2018.

PENDRILL, Richard. **Automação de atualização baseada em UDS para fins de testes de software no CANoe**. Lund: Universidade de Lund, Divisão de Engenharia Elétrica Industrial e Automação,2016.

BIBLIOGRAFIA CONSULTADA

BRANDÃO, R. L. **O automóvel no Brasil entre 1955 e 1961: a invenção de novos imaginários na era JK. 2011.** Dissertação (Mestrado em História) – Universidade Federal de Juiz de Fora, Programa de Pós-Graduação em História. Juiz de Fora, 2011.

CARVALHO, S. M. T.; CAMPOS, G. L. **Transmissão de mensagens e gerenciamento de erros em uma rede CAN automotiva**. ForScience, v. 6, n. 1, 2018.

GODOY, E. P. et al. **Modelagem e simulação de redes de comunicação baseadas no protocolo CAN-Controller Area Network**. Sba: Controle & Automação Sociedade Brasileira de Automatica. Agosto de 2010.

MEDINA, H. V. de. **Inovação em materiais na indústria automobilística**. 2001.

Schmidt, K. et al. (2020). **Sobre o Desenvolvimento de Redes Veiculares Orientadas a Serviços Baseadas no CANoe**. Relatórios sobre Mobilidade Energética Eficiente, 2020.

SILVA, B. S. de S. da Silva. **Desenvolvimento de Software Embarcado Automotivo Aderente Aderente ao Padrão Autosar**. Universidade de Brasília - FGA Engenharia Eletrônica. Brasília, DF 2014.

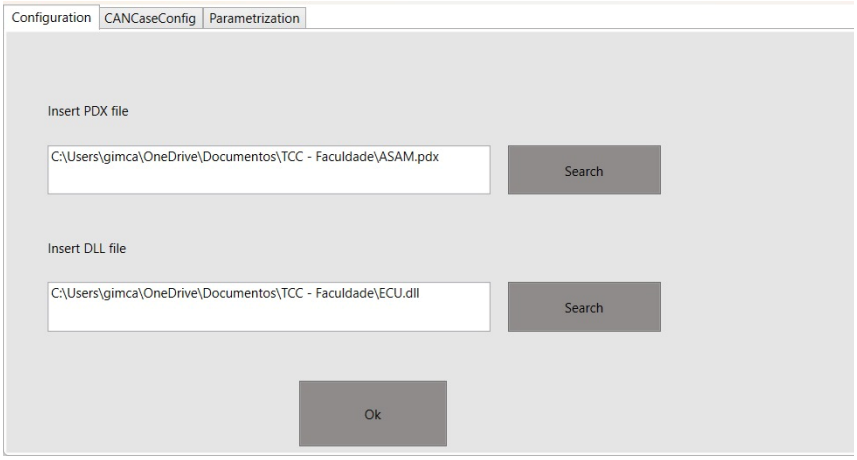
APÊNDICE A – DOCUMENTAÇÃO DE USO

Introdução

Este manual guia o usuário no uso do software para parametrização de módulos automotivos. O software possui três abas principais: **Configuração**, **Configuração da CANcase** e **Parametrização**. Abaixo, detalhamos os passos e funções de cada aba para que o usuário possa realizar uma configuração segura e eficiente do módulo.

---

1. **Primeira Aba: Configuração**



Objetivo:

A aba **Configuração** permite que o usuário carregue os arquivos essenciais para iniciar o processo de parametrização. Esses arquivos garantem que o sistema tenha os dados técnicos necessários e a segurança exigida.

Instruções:

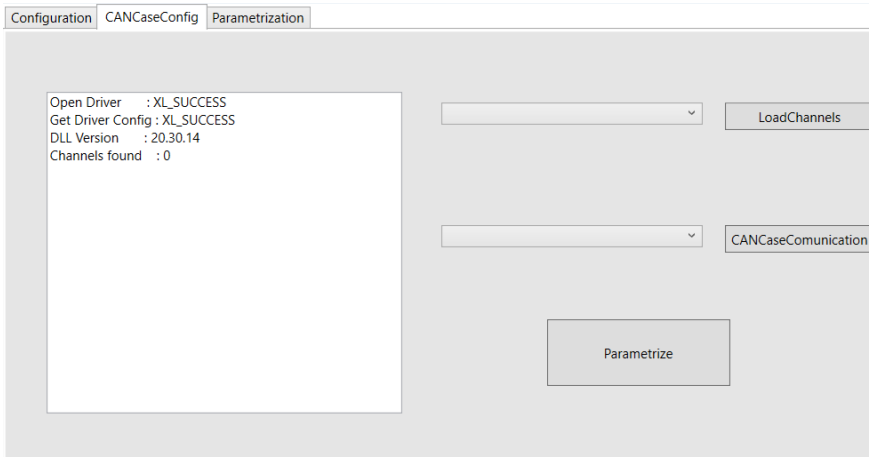
1. **Anexe o Arquivo .pdx**:
    - Na aba **Configuração**, localize o campo destinado ao arquivo com a extensão .pdx.
    - Clique no botão "Search" e selecione o arquivo .pdx no seu computador.
    - **Função do .pdx**: Este arquivo contém as configurações de software do módulo, com informações detalhadas sobre parâmetros, mapeamento e valores padrão. Ele serve como um repositório de dados para garantir que o sistema opere conforme o esperado.
2. **Anexe o Arquivo .dll**:
    - No mesmo local, encontre o campo de anexo do arquivo .dll.
    - Clique em "Search" e selecione o arquivo .dll.

- **Função do .dll**: Este arquivo atua como uma chave de segurança. Sem ele, o acesso aos parâmetros do .pdx é bloqueado, garantindo controle rigoroso de acesso ao sistema.

3. **Validação**:
   - Após anexar ambos os arquivos, clique no botão "OK".
   - O sistema verificará os arquivos e, ao validar a integridade e autenticidade deles, avançará automaticamente para a próxima aba.
   - Se ocorrer um erro, o sistema exibirá uma mensagem. Verifique se os arquivos estão corretos e tente novamente.

---

2. Segunda Aba: Configuração da CANcase



Objetivo:

A aba **Configuração da CANcase** é utilizada para estabelecer a comunicação entre o software e o módulo automotivo, permitindo que o sistema identifique e se conecte ao hardware correto.

Instruções:

1. **Selecionar Modelo da CANcase**:
   - Escolha o modelo de **CANcase** utilizado no projeto a partir de um menu suspenso.
   - Esta seleção é essencial para a compatibilidade da interface de comunicação com o módulo.
2. **Escolher Porta de Conexão**:
   - Selecione a porta correta para o dispositivo CANcase. Geralmente, ela é exibida automaticamente, mas se necessário, consulte o manual do dispositivo para verificar a porta correta.

- A seleção adequada da porta garante que o sistema identifique a conexão física com o módulo.

3. **Verificar Conexão**:
   - Após configurar o modelo e a porta, o sistema realizará automaticamente uma verificação da integridade da conexão.
   - O status da comunicação aparecerá em um campo lateral. Certifique-se de que a conexão está ativa antes de prosseguir para a próxima aba.

---

3. Terceira Aba: Parametrização

Objetivo:

Na aba **Parametrização**, o usuário pode visualizar e modificar parâmetros específicos do módulo, adaptando-os conforme as necessidades do projeto.

Instruções:

1. **Navegar pelos DIDs**:
   - Os parâmetros do módulo estão organizados por **DIDs** (Diagnose Identifier), que representam grupos de parâmetros específicos.
   - Clique em um DID para expandi-lo e visualizar a lista de parâmetros associados.
2. **Modificar Parâmetros**:
   - Dentro do DID expandido, cada parâmetro estará listado com um campo de entrada.
   - Insira valores nos campos correspondentes aos parâmetros que deseja modificar.
   - Para habilitar ou desabilitar uma função, marque ou desmarque as opções apropriadas.

3. **Salvar Configurações**:
    - Após ajustar todos os parâmetros necessários, clique em **"**Write".
    - O sistema salvará as configurações, permitindo que o módulo opere com as novas especificações.
    - **Observação**: Após salvar, é recomendável testar as configurações para garantir que estão funcionando conforme esperado.
4. **Realizar Testes de Validação**:
    - Para validar as alterações feitas, execute testes no módulo conforme as especificações do projeto.
    - Caso algum parâmetro precise ser ajustado, volte à aba **Parametrização**, faça os ajustes necessários e salve novamente.

Considerações Finais

Este manual cobre as instruções básicas para o uso do software de parametrização de módulos automotivos. Para dúvidas adicionais, consulte o suporte técnico ou a equipe de desenvolvimento.
