UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MATHEUS DE SOUZA MONTEIRO

SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICA DISTRIBUÍDO BASEADO EM VIED DE PROTEÇÃO MULTIFUNÇÃO PADRÃO IEC 61850

FORTALEZA

2025

MATHEUS DE SOUZA MONTEIRO

SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICA DISTRIBUÍDO BASEADO EM VIED DE PROTEÇÃO MULTIFUNÇÃO PADRÃO IEC 61850

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Raimundo Furtado Sampaio.
Coorientadora: Prof. PhD. Ruth Pastôra Saraiva Leão

FORTALEZA

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M778s Monteiro, Matheus de Souza.
Sistema de recomposição automática distribuído baseado em vIED de proteção multifunção padrão IEC 61850 / Matheus de Souza Monteiro. – 2025.
107 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2025.
Orientação: Prof. Dr. Raimundo Furtado Sampaio.
Coorientação: Profa. Dra. Ruth Pastôra Saraiva Leão.

1. Virtualização. 2. Sistema Elétrico de Potência. 3. Recomposição Automática. 4. Sistema de Distribuição. 5. Container Docker. I. Título.

CDD 621.3

MATHEUS DE SOUZA MONTEIRO

SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICA DISTRIBUÍDO BASEADO EM VIED DE PROTEÇÃO MULTIFUNÇÃO PADRÃO IEC 61850

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

_____
Prof. Dr. Raimundo Furtado Sampaio (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. PhD. Ruth Pastôra Saraiva Leão (Coorientadora)
Universidade Federal do Ceará (UFC)

_____
Eng. Amanda Carvalho Alexandre
Universidade Federal do Ceará (UFC)

_____
Eng. Luís Pablo Pereira Saraiva
Universidade Federal do Ceará (UFC)

A Deus, por toda a glória e por sempre me fortalecer todos os dias!

AGRADECIMENTOS

A Deus, por todo o amor e carinho derramado sobre a minha vida, por ter me transformado em um ser humano melhor e me fazer perceber que não existe felicidade que não seja ao lado dele.

A minha namorada, Marislânia da Silva, por todo o apoio, companheirismo, cuidado e amor que ela tem por mim e por toda minha família. A minha mãe, Claúdia Souza, por todo o esforço empregado durante a vida no meu desenvolvimento e no da minha irmã, pela atenção e ensinamentos que sempre teve conosco. Ao meu pai, Francisco Edilson, por todo apoio e esforço empregado a mim durante a vida. A minha irmã, Lorenna de Souza, por sempre me apoiar, me aconselhar, e amizade que desenvolvemos ao longo da vida. A toda a minha família, por sempre me apoiarem e dicarem seu tempo a minha vida

A Distribuidora Cummins Diesel do Nordeste, empresa na qual trabalho atualmente, em especial, ao meus colegas de trabalho, Luciano Maciel, Daniel Rodrigues e Jaime Linhares, por todos os ensinamentos e oportunidades. Ao meu amigo, José Erivan, por todos os ensinamentos, pela atenção e paciência.

Ao Grupo de Redes Elétricas Inteligentes (GREI), em especial, ao meus orientadores, professor Raimundo Furtado Sampaio e professora Ruth Pastôra Saraiva Leão e ao professor Lucas Silveira Melo, pela atenção, conhecimentos repassados a mim por todos esses anos, ensinamentos de vida e pela humanidade em cada um deles, capacitando um ser humano e não apenas um engenheiro. Aos meus amigos, João Pedro e Lucas Mesquita, que somaram seus esforços e tornaram esse trabalho possível. Ao meu amigo, Luís L'Aiglon, por todo o apoio e disponibilidade durante toda a minha graduação, uma amizade que perdurará por toda a vida.

Aos colegas de curso e professores da graduação, em especial, aos meus amigos, Gesiel da Silva, Izaías Façanha e Iago bastos, pelos vários anos de estudo juntos, trabalhos concluídos e companheirismo durante a graduação.

“Não andeis ansiosos com coisa alguma, mas em tudo, pela oração e súplicas, e com ação de graças, apresentem seus pedidos a Deus”

(Filipenses 4:6)

RESUMO

Os sistemas de distribuição de energia elétrica em média tensão tradicionalmente operam com topologias radiais, que apresentam baixa confiabilidade e baixo investimento em tecnologia e automação. Isso cria um fator limitante para a evolução rumo às Redes Elétricas Inteligentes (REI), cuja inteligência depende de sua infraestrutura. A automação da distribuição é um dos principais pilares para essa transição, integrando tecnologias avançadas como SCADA, sensores, chaves e religadores automatizados, redes de comunicação e relés digitais (IEDs). Para garantir a interoperabilidade nesse ambiente com múltiplos fornecedores, a norma IEC 61850 se destaca, oferecendo modelos de informações padronizados e uma estrutura orientada a objetos para todos os componentes, desde a subestação até o consumidor final. Este trabalho avança no conceito de REI e apresenta o desenvolvimento de um Sistema de Recomposição Automático Distribuído (SRAD). O sistema utiliza IEDs de proteção e Merging Units (MU) virtualizados, aplicados a uma rede de distribuição de 13,8 kV do campus universitário do Pici, na Universidade Federal do Ceará. Para a criação dos dispositivos virtuais (vIEDs e vMUs), foram utilizados *containers Docker*, permitindo que um único hardware hospede múltiplas instâncias virtuais, cada uma funcionando como um dispositivo distinto. A flexibilidade do sistema foi demonstrada com a utilização de bibliotecas de modelagem e arquivos de configuração XML da norma IEC 61850, para dispositivos de diferentes fabricantes. A comunicação entre os dispositivos da rede de distribuição é essencial para o funcionamento do SRAD, sendo realizada por meio de protocolos da norma IEC 61850, como as mensagens GOOSE e SV para comunicação entre vMUs e vIEDs, mensagens GOOSE entre vIEDs e mensagens MMS para a interface com o sistema SCADA. Para validar o sistema, foi montada uma plataforma PAC. Essa plataforma incluía notebooks para a criação e monitoramento dos containers, um computador industrial para hospedar o SCADA ELIPSE, um *switch* gerenciável e GPS de sincronização. Através de vLANs no *switch*, foi possível modelar o barramento de processo, bay e estação. Os testes de validação do SRAD foram conduzidos com a simulação de faltas em diversos trechos da rede, e o comportamento do sistema foi analisado em tempo real pelo SCADA, que está totalmente integrado, e pelo software de análise de tráfego de rede *Wireshark*. Como resultado, foi comprovado o pleno funcionamento do SRAD. O trabalho demonstra uma abordagem eficaz para a recomposição automática de um sistema, utilizando um algoritmo inteligente distribuído entre os vIEDs de proteção da rede de distribuição. O sistema desenvolvido também válida o conceito de uso de dispositivos virtualizados em subestações inteligentes, o que representa um avanço na viabilização de testes complexos e

robustos de proteção, automação e controle para sistemas elétricos mais complexos.

Palavras-chave: Virtualização; Sistema Elétrico de Potência; Recomposição Automática; Sistema de Distribuição, Container Docker.

ABSTRACT

Traditional medium-voltage electrical power distribution systems operate with radial topologies, which offer low reliability and low investment in technology and automation. This creates a limiting factor for the evolution towards Smart Grids (SG), whose intelligence depends on their infrastructure. Distribution automation is one of the main pillars for this transition, integrating advanced technologies such as SCADA, sensors, automated switches and reclosers, communication networks, and digital relays (IEDs). To ensure interoperability in this multi-vendor environment, the IEC 61850 standard stands out, offering standardized information models and an object-oriented structure for all components, from the substation to the end consumer. This work advances the concept of REI and presents the development of a Distributed Automatic Reclosing System (SRAD). The system uses virtualized protection IEDs and Merging Units (MU), applied to a 13.8 kV distribution network on the Pici campus of the Federal University of Ceará. Docker containers were used to create the virtual devices (vIEDs and vMUs), allowing a single piece of hardware to host multiple virtual instances, each functioning as a separate device. The flexibility of the system was demonstrated with the use of modeling libraries and XML configuration files from the IEC 61850 standard for devices from different manufacturers. Communication between distribution network devices is essential for the SRAD to function, and is carried out using IEC 61850 protocols, such as GOOSE and SV messages for communication between vMUs and vIEDs, GOOSE messages between vIEDs, and MMS messages for interfacing with the SCADA system. To validate the system, a PAC platform was set up. This platform included notebooks for creating and monitoring containers, an industrial computer to host the ELIPSE SCADA, a manageable switch, and a synchronization GPS. Through vLANs on the switch, it was possible to model the process bus, bay, and station. SRAD validation tests were conducted with fault simulation in various sections of the network, and the system's behavior was analyzed in real time by SCADA, which is fully integrated, and by Wireshark network traffic analysis software. As a result, the full functioning of SRAD was proven. The work demonstrates an effective approach to automatic system restoration using an intelligent algorithm distributed among the distribution network protection vIEDs. The developed system also validates the concept of using virtualized devices in smart substations, which represents an advance in enabling complex and robust protection, automation, and control tests for more complex electrical systems.

Keywords: Virtualization; Electric Power System; Self-Healing; Docker Container; Distribution System.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ANSI	*American National Standards Institute*
GOOSE	*Generic Object Oriented Substation Event*
GREI	Grupo de Redes Elétricas Inteligentes
IEC	*International Electrotechnical Commission*
IID	*IED Capability Description*
IED	*Intelligent Electronic Device*
MMS	*Manufacturing Message Specification*
MU	*Merging Unit*
PAC	Proteção, Automação e Controle
SAS	Sistema de Automação de Subestações
SCADA	*Supervisory Control And Data Acquisition*
SRAD	Sistema de Recomposição Automático Distribuído
SO	Sistema Operacional
SV	*Sampled Values*
vIED	*Virtual Intelligent Electronic Device*
vMU	*Virtual Merging Unit*

SUMÁRIO

1. INTRODUÇÃO

1.1 Justificativa e Motivação

Este trabalho dá continuidade à linha de pesquisa do Grupo de Redes Elétricas Inteligentes (GREI), do Departamento de Engenharia Elétrica da Universidade Federal do Ceará. O objetivo é propor uma nova aplicação para aumentar a confiabilidade e seletividade da rede de distribuição de energia elétrica em média tensão do Campus do Pici.

O ponto de partida para esta proposta é o Sistema de Recomposição Automático (SRA), desenvolvido e implementado por Santos (2015) para a mesma rede de distribuição. Posteriormente, a pesquisa evoluiu com a proposta de um sistema de automação distribuído baseado em multiagentes, conforme Sampaio (2017). Baseado nesses estudos e nos trabalhos de FREITAS (2022a, 2022b) e MARTINS (2023), que deram origem ao Dispositivo Eletrônico Inteligente Virtual (vIED), este projeto busca aprofundar o conceito de automação distribuída.

Com o uso do vIED como objeto de estudo, o presente trabalho propõe uma nova forma de SRA, agora completamente distribuída entre os dispositivos inteligentes da rede, aplicada especificamente à subestação do Campus do Pici. Essa abordagem visa inovar o sistema existente, explorando a potencialidade da virtualização para aprimorar a automação e o controle da rede.

1.2 Objetivos

1.2.1 Objetivo geral

O presente trabalho tem como objetivo principal desenvolver e validar um Sistema de Recomposição Automático (SRA) distribuído, baseado em vIEDs de proteção multifunção, em conformidade com o padrão IEC 61850.

1.2.2 Objetivos específicos

Para atingir o objetivo geral, este trabalho visa:

- Especificar as lógicas de proteção e das funções de relé para desenvolvimento dos vIEDs;
- Desenvolver vIEDs multifunção que operem em containers Docker;
- Implementar um SRA distribuído com a integração dos vIEDs via

mensagens GOOSE da norma IEC 61850;

- Integrar o SRA distribuído a um sistema SCADA, para monitoramento e controle;
- Testar e validar o SRA proposto em uma plataforma de Proteção, Automação e Controle (PAC).

1.3 Estrutura do Trabalho

Este trabalho está organizado nos seguintes capítulos:

- Capítulo 2 – Fundamentação Teórica: Apresenta os conceitos essenciais sobre sistemas de distribuição, proteção, automação e controle. Aprofunda-se na norma IEC 61850, incluindo uma revisão bibliográfica sobre virtualização de plantas elétricas, tecnologia Docker e o conceito de Dispositivo Eletrônico Inteligente Virtual (vIED) compatível com a norma.
- Capítulo 3 – Metodologia: Descreve a metodologia utilizada para o desenvolvimento do Sistema de Recomposição Automático Distribuído (SRAD) com vIEDs multifunção.
- Capítulo 4 – Desenvolvimento do Sistema: Detalha a implementação do SRAD;
- Capítulo 5 – Resultados e Validação: Apresenta os resultados dos testes de validação do SRAD, realizados na plataforma de Proteção, Automação e Controle (PAC);
- Capítulo 6 – Conclusões e Trabalhos Futuros: Finaliza o trabalho com as conclusões obtidas e sugestões para pesquisas futuras.

2. FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho é apresentada neste capítulo, cobrindo os pilares de Sistemas de Distribuição de Energia, bem como os princípios de proteção, automação e controle. Para contextualizar o ambiente tecnológico do projeto, é detalhada a norma IEC 61850 e sua importância. A seguir, uma revisão bibliográfica explora os avanços em Sistemas de Recomposição Automáticos e as inovações em subestações digitais, com foco na utilização de máquinas virtuais e containers Docker para virtualização de dispositivos em Redes Elétricas Inteligentes.

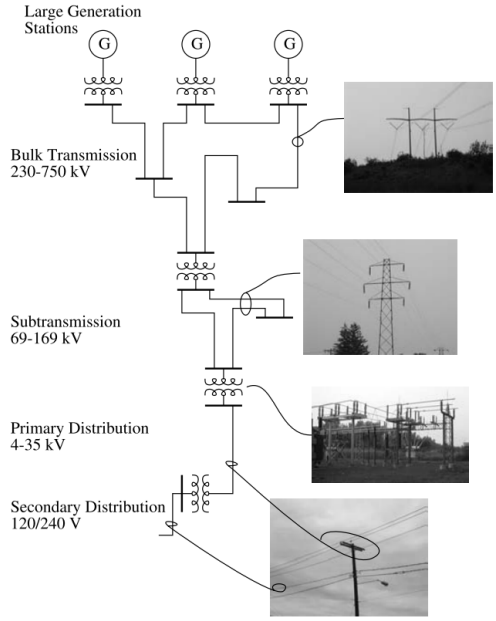
2.1 Sistema de Distribuição de Energia

2.1.1 Subestação Distribuidora de Energia

O sistema elétrico de potência é composto por cinco etapas principais: geração, transmissão, subtransmissão, distribuição primária e distribuição secundária. A energia elétrica é produzida na etapa de geração por diversas fontes e, em seguida, tem seu nível de tensão elevado para a etapa de transmissão (230-750 kV), que transporta por longas distâncias.

Na sequência, a subtransmissão recebe a energia e reduz o seu nível de tensão (69-169 kV) para ser entregue às subestações distribuidoras. A partir dessas subestações, a distribuição primária rebaixa a tensão para níveis de 4-35 kV, formando a rede que é comumente encontrada nas áreas urbanas. Por fim, a distribuição secundária entrega a energia ao consumidor final em tensões mais baixas, como 120/240V (SHORT, 2004). Embora os níveis de tensão e frequência possam variar em diferentes países, a estrutura geral do sistema é a mesma. Na Figura 1, é apresentada uma visão geral de infraestrutura de um sistema elétrico de potência.

Figura 1 – Infraestrutura do sistema elétrico de potência



Fonte: (SHORT, 2004)

A subestação distribuidora de energia desempenha um papel crucial na transição entre os sistemas de subtransmissão e distribuição. Ela é responsável por receber a energia em níveis de tensão mais altos (subtransmissão) e, por meio de **transformadores de potência**, rebaixá-la para os níveis de média tensão utilizados na rede de distribuição primária (geralmente entre 4 kV e 35 kV).

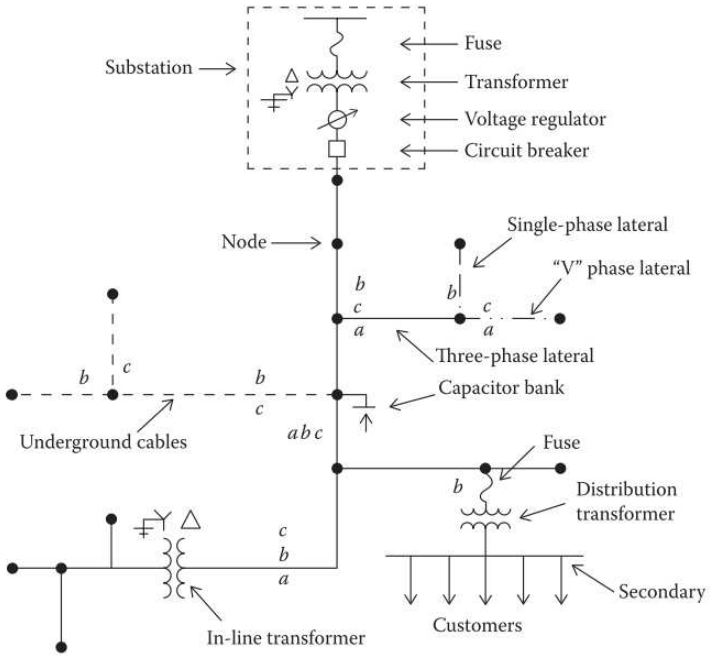
Além de atuar como ponto de conexão e transformação, a subestação distribuidora abriga equipamentos essenciais para o controle, a proteção e a medição do fluxo de energia. Disjuntores, seccionadores, chaves e relés de proteção são utilizados para garantir a segurança e a confiabilidade da rede, isolando falhas e controlando o fluxo de potência. É nessa etapa que se concentram os principais dispositivos de automação e controle que permitem a evolução das redes tradicionais para as Redes Elétricas Inteligentes (REI).

2.1.2 Topologia Radial

A topologia de rede radial é caracterizada por um fluxo de potência unidirecional e segue em direção ao cliente. Por ser mais simples, esse tipo de configuração é amplamente utilizado em redes de distribuição. No entanto, sua principal desvantagem é a baixa confiabilidade. Em um sistema radial, cada ponto da rede é alimentado por uma única fonte. Isso significa que, em caso de falta (curto-circuito, por exemplo), uma grande parte do trecho

afetado é desenergizada e não pode ser reenergizada até que a falha seja completamente reparada. Não há uma alternativa para alimentar o trecho não faltoso, o que causa longos períodos de interrupção no fornecimento de energia (KERSTING, 2018). A Figura 2, ilustra um modelo de sistema com topologia radial, destacando o fluxo único de alimentação.

Figura 2 – Modelo de topologia de rede radial



Fonte: (KERSTING, 2018)

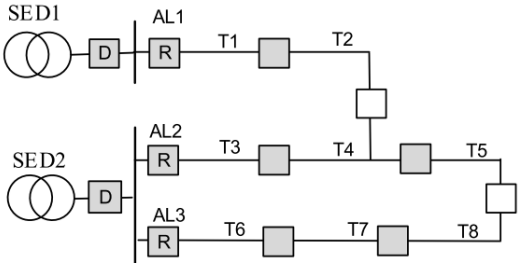
2.1.3 Topologia Radial com Recurso

A rede radial com recurso opera com o mesmo princípio de alimentação do sistema radial simples, mas com uma importante diferença: a existência de chaves de interconexão normalmente abertas, que conectam diferentes alimentadores.

Essa interconexão permite a recomposição do sistema, ou seja, a reenergização de trechos não afetados pela falta. Ao abrir as chaves de um alimentador que sofreu um defeito, a carga pode ser transferida para um alimentador vizinho por meio das chaves de interconexão. Isso aumenta significativamente a confiabilidade do sistema de distribuição, permitindo que a energia seja restaurada para os consumidores mais rapidamente, mesmo durante uma falta (SHORT, 2004).

A Figura 3 demonstra visualmente como essa topologia possibilita a alimentação secundária de trechos desenergizados em caso de faltas.

Figura 3 – Modelo de topologia de rede radial com recurso



Fonte: (SAMPAIO, 2017)

2.2 Sistema de Proteção, Automação e Controle

2.2.1 Dispositivos Eletrônicos Inteligentes

Os Dispositivos Eletrônicos Inteligentes (IEDs), são equipamentos digitais utilizados em subestações e sistemas de distribuição para desempenhar diversas funções. Uma de suas principais aplicações é na proteção de sistemas elétricos, atuando como relés de proteção. No entanto, com o avanço tecnológico, o termo IED engloba também outros equipamentos, como medidores de energia inteligentes, dispositivos responsáveis pela virtualização de grandezas elétricas e controladores. Uma característica fundamental dos IEDs é a sua capacidade de comunicação através de protocolos padronizados, o que permite a troca de informações entre os próprios dispositivos e a integração com sistemas de supervisão, como o SCADA (FÉLIX, 2019).

2.2.2 Estrutura Hierárquica do Sistema de Automação

A automação de subestações (SAS) é organizada em uma estrutura hierárquica, facilitando a subdivisão e o gerenciamento de todas as suas funções.

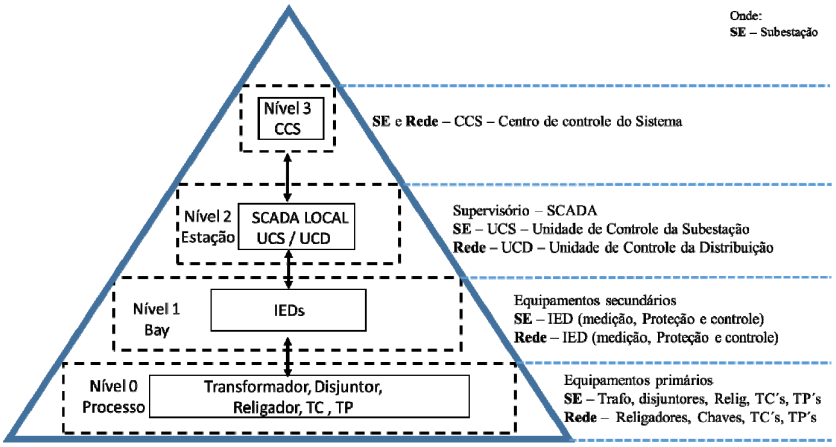
- Nível 0 (Nível de Processo): É o nível mais baixo, onde estão os dados brutos e os equipamentos de campo, como disjuntores, transformadores, seccionadores e transformadores de medição.
- Nível 1 (Nível de Baia): Neste nível, os IEDs de proteção e medição estão localizados. Eles recebem dados do nível de processo e executam as funções de proteção e controle da baia (um conjunto de equipamentos, como um disjuntor e seus seccionadores).
- Nível 2 (Nível de Estação): Responsável por coletar os dados dos IEDs no nível de baia

e controlar os equipamentos. O sistema SCADA da subestação tipicamente opera neste nível.

– Nível 3 (Nível de Operação): Associado ao controle geral de múltiplas subestações ou sistemas de energia. É o nível onde operam associados centros de controle de distribuição ou transmissão (SANTOS, 2015).

Essa estrutura permite uma clara divisão de tarefas, desde a medição no campo até a gestão centralizada do sistema elétrico. A Figura 4 ilustra os níveis hierárquicos de automação de sistemas elétricos.

Figura 4 – Níveis hierárquicos de automação de sistemas de energia



Fonte: (SANTOS, 2015)

2.3 Automação da Distribuição Baseada na Norma IEC 61.850

A norma IEC 61850 vai além da simples definição de protocolos de comunicação, estabelecendo uma estrutura de dados completa, baseada em um modelo orientado a objetos. Essa abordagem padroniza o mapeamento e a identificação de dados para cada componente, garantindo a interoperabilidade em ambientes com múltiplos fornecedores e facilitando a automação de sistemas de distribuição.

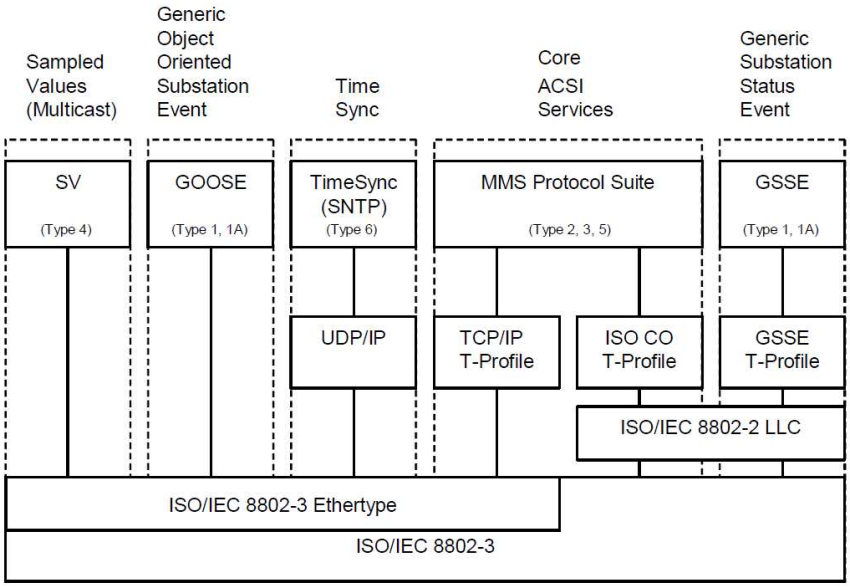
A norma define protocolos de mensagens específicos para os diferentes níveis de automação:

– *Sampled Values (SV)*: Utilizado na comunicação entre o nível de processo e o nível de baia, o protocolo SV é um protocolo de alta taxa envio, ideal para a transmissão eficiente de grandezas elétricas. Com uma amostragem típica de 80 pontos por ciclo senoidal, resulta em 4800 mensagens por segundo, garantindo a precisão necessária para a proteção do sistema.

- *Generic Object Oriented Substation Event* (GOOSE): Este é um protocolo de altíssima prioridade, usado para comunicação entre IEDs, sejam eles de proteção, medição ou de camadas inferiores. As mensagens GOOSE são transmitidas de forma ininterrupta no sistema, garantindo uma resposta rápida e confiável em eventos críticos.
- *Manufacturing Message Specification* (MMS): Para a comunicação entre o nível de baia e a estação, a norma utiliza o protocolo MMS. Este é um protocolo de baixa prioridade, mas que oferece uma comunicação eficaz, pois é mapeado com o protocolo de comunicação TCP/IP, assegurando a entrega da mensagem desde a sua origem.
- *Simple Network Time Protocol* (SNTP): Para o sincronismo de tempo de todos os dados e protocolos de comunicação utilizados em uma subestação. Atualizando assim os dados de estampa de tempo, para cada nó lógico que tenha essa disponibilidade (MELO, 2015).

A Figura 5 ilustra os protocolos definidos na norma IEC 61850 e suas respectivas bases de comunicação.

Figura 5 – Visão geral dos protocolos da norma IEC 61850



Fonte: (IEC 61850-8-1, 2004)

O protocolo NTP é uma alternativa ao SNTP, com uma maior precisão de tempo e com difusão mundial, aplicado a sistemas de subestações e na rede de computadores mundial, trafegando pela internet.

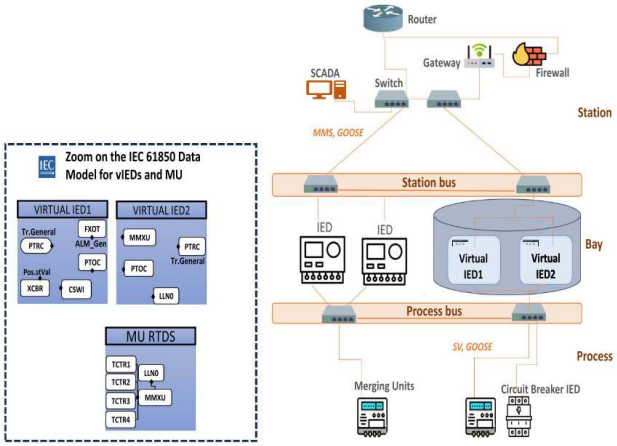
2.4 Revisão Bibliográfica

2.4.1 Virtualização de Subestação

O conceito de virtualização de subestações tem ganhado força com a ampla adoção da norma IEC 61850. Um estudo relevante nessa área é o de Kabbara et al.(2024), que apresenta uma arquitetura para vIEDs (IEDs virtuais) e a implementação de uma rede definida por software para subestações digitais. O artigo utiliza vIEDs em containers e Máquinas Virtuais, empregando a biblioteca LibIEC61850 para modelagem dos protocolos. Para isso, os autores empregaram um *Real-Time Digital Simulator* (RTDS) para simular as variáveis de um sistema de potência de 5 barras (padrão IEEE), bem como MUs e IEDs de proteção.

A Figura 6 ilustra a arquitetura de rede utilizada no estudo, detalhando a comunicação entre IEDs físicos, vIEDs e MUs, e a modelagem IEC 61850 para cada dispositivo. No entanto, o artigo de Kabbara et al. (2024) possui algumas limitações importantes, como a ausência de reprodução de sistemas reais de uma subestação convencional, a falta de comunicação com o sistema SCADA, o uso de apenas uma função de proteção (sobrecorrente instantânea – ANSI 50) e a realização dos testes em um ambiente de simulação, sem uma bancada física.

Figura 6 – Arquitetura de comunicação abordando IEDs, vIEDs e MU



Fonte: (Kabbara et al., 2024)

A viabilidade econômica da virtualização de subestações é tão crucial quanto a sua implementação técnica. Vilaplanaet al. (2024) abordam esse conceito em profundidade, desenvolvendo um modelo dinâmico para avaliar os custos e benefícios associados a novos investimentos em virtualização. O estudo oferece uma estimativa dos benefícios esperados, ao mesmo tempo que discute as limitações da tecnologia e apresenta um panorama de cenários e

incertezas.

Enquanto este trabalho se concentra na aplicação prática da virtualização em um sistema de distribuição, a análise de viabilidade econômica de Vilaplana et al. (2024) serve como um complemento essencial para qualquer estudo sobre a virtualização de plantas elétricas. A tabela 1 apresenta uma adaptação do artigo de Vilaplana et al. (2024), destacando os benefícios e desafios da virtualização em diferentes categorias relevantes para subestações.

Tabela 1 – Benefícios e desafios da virtualização por categoria.

Categoria	**Benefícios da virtualização**	**Desafios da virtualização**
Operacional	- Redução das inspeções de O&M - Mitigação de falhas	- Aumento da especialização dos técnicos - Complexidade do projeto
Financeiro	- Redução do investimento inicial	- Custo inicial de licenciamento - Custo das habilidades digitais
Ambiental	- Redução da pegada de carbono - Redução da energia não fornecida	- Alto consumo unitário de energia dos servidores
Estratégico	- Aumento da flexibilidade	- Caminho de migração da IEC61850 - Bloqueio do fornecedor - Maturidade da tecnologia

Fonte: Adaptado de VILAPLANA et al. (2024).

Existem outras abordagens para a virtualização de subestações que, embora não utilizem a virtualização de IEDs de proteção em containers ou Máquinas Virtuais, ainda exploram o conceito de separação entre hardware e software. Um exemplo disso é o trabalho de Martinezet al. (2024), que desenvolveu um módulo de processamento analógico para sinais de valores amostrados (SV) da IEC 61850. O módulo foi implementado via software em uma plataforma de hardware genérica, demonstrando a separação entre hardware e software em um ambiente de proteção e controle centralizados. O estudo incluiu a realização de testes em laboratório, utilizando uma metodologia específica para comparar os valores gerados por um RTDS com os calculados pelo módulo proposto.

No entanto, o trabalho de Martinez et al. (2024) possui limitações, como a não utilização da técnica de virtualização de IEDs físicos, a ausência de uma implementação em subestação real e o fato de não utilizar uma rede física que se assemelhe a uma aplicação prática em sistemas de distribuição ou transmissão de energia.

2.4.2 Recomposição automática

A recomposição automática é um tema amplamente estudado, com abordagens que podem ser classificadas como centralizadas ou distribuídas. A recomposição centralizada, por exemplo, é detalhada por Santos (2015), aonde um sistema SCADA adquire dados e, por meio

do protocolo OPC, interage com um software de inteligência para realizar a recomposição. Essa é uma metodologia tradicional, amplamente utilizada por concessionárias de energia. No entanto, o foco de pesquisas mais recentes tem se voltado para a recomposição automática distribuída, que se adapta melhor a redes complexas.

Nesse contexto, Hamdeen et al. (2025) apresentam uma revisão abrangente de artigos sobre sistemas multiagentes para recomposição automática distribuída. O estudo analisa soluções propostas com base em critérios como a complexidade e a escalabilidade dos modelos, a velocidade de resposta, a integração com energias renováveis, a detecção de ataques cibernéticos e a resiliência a falhas de comunicação.

Apesar da maioria dos trabalhos revisados focar na recomposição distribuída, nem todos utilizam a norma IEC 61850 de forma completa. Aqueles que o fazem, entanto, demonstram boa velocidade de resposta, baixa complexidade e alta escalabilidade. Contudo, esses trabalhos ainda não aplicam diretamente o uso de mensagens GOOSE para a recomposição, empregando dispositivos de proteção virtuais que se comunicam entre si.

2.4.3 Docker Container

A popularização da virtualização local e em nuvem impulsionou o uso de contêineres, especialmente devido à sua flexibilidade superior em relação às Máquinas Virtuais (VM). Embora o objetivo final seja semelhante, os contêineres se destacam pelo menor consumo de memória e armazenamento, além de uma inicialização mais rápida. Isso é possível porque eles compartilham o *kernel* do sistema operacional, mantendo, no entanto, o isolamento em nível de processo (ZHAO et al., 2021). O Docker é a principal ferramenta de gerenciamento de contêineres, sendo um dos fatores chave para a ampla adoção dessa tecnologia.

2.5 Considerações finais

Este capítulo apresentou a fundamentação teórica essencial para o desenvolvimento do trabalho. Abordou-se a importância da comunicação e da automação em subestações e redes de distribuição, descrevendo as características gerais de diferentes topologias de rede. Em seguida, foram detalhados os conceitos da norma IEC 61850, seus modelos de dados e os principais protocolos de comunicação. A revisão bibliográfica explorou a virtualização de subestações, o funcionamento de sistemas de recomposição automático e o papel dos contêineres na virtualização de dispositivos, estabelecendo a base teórica necessária.

3. METODOLOGIA PARA DESENVOLVIMENTO DO SRAD

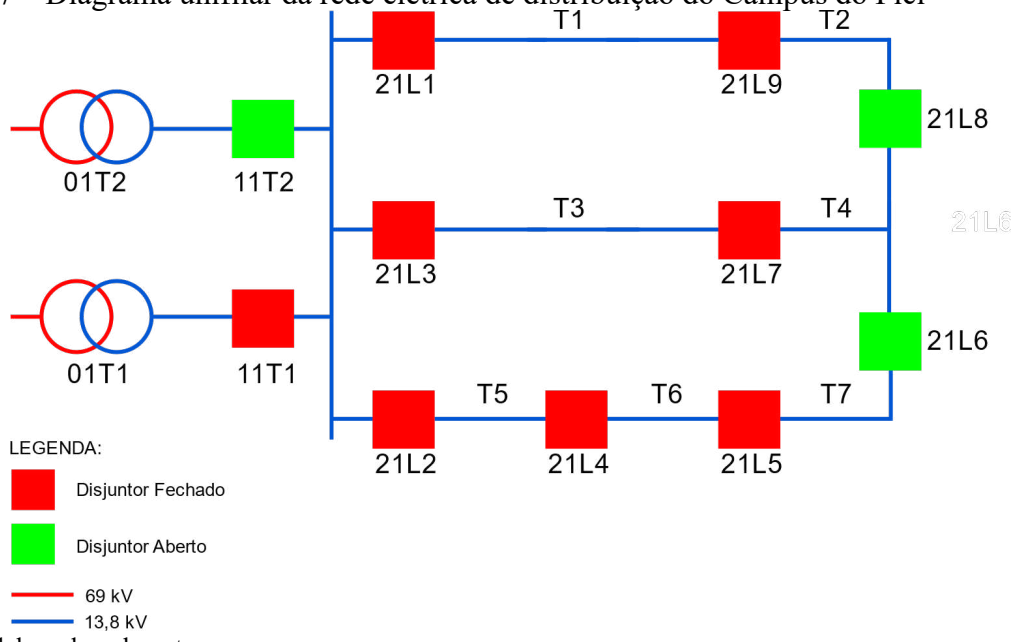
Neste capítulo, são apresentadas as características e métodos necessários para o desenvolvimento do Sistema de Recomposição Automático Distribuído (SRAD). O foco é a integração do sistema de recomposição a um SCADA utilizando vIEDs, com uma descrição detalhada da rede de distribuição, da metodologia do SRAD e das características gerais dos dispositivos virtuais.

3.1 Sistema de Distribuição de Energia do Campus do Pici

3.1.1 Descrição da Rede de Distribuição

A rede elétrica de distribuição do Campus do Pici possui uma topologia radial com recurso, projetada para isolar a menor quantidade possível de trechos em caso de falta. Sua estrutura é composta por 9 religadores, que estão associados aos alimentadores 21L1, 21L2 e 21L3, e aos disjuntores 21L4, 21L5 e 21L9. Os pontos de encontro de alimentadores são designados como 21L6, 21L7 e 21L8. Essa configuração, ilustrada na Figura 7, demanda um sistema inteligente para processar as informações e garantir a continuidade do serviço.

Figura 7 – Diagrama unifilar da rede elétrica de distribuição do Campus do Pici



Fonte: Elaborado pelo autor.

A proteção e o controle de faltas são executados por IEDs de proteção em cada religador, com o objetivo de proteger os trechos a jusante. Embora um sistema de proteção

seletivo e coordenado garanta a segurança da rede, ele não assegura sua recomposição automática após uma falta. Atualmente, a recomposição é frequentemente realizada manualmente por equipes de campo ou remotamente via sala de controle.

Com a crescente inteligência dos IEDs, é possível utilizar esses dispositivos não apenas para proteção, mas também para automatizar processos de controle, como a recomposição da rede. Assim, o SRAD será desenvolvido e aplicado por meio desses dispositivos. A utilização de um rede completa com dispositivos virtuais, é uma virtude desse trabalho, sendo assim, este trabalho utilizará vIEDs para simular completamente a rede, validando a aplicação o SRAD em um ambiente virtual.

3.2 Dispositivo Eletrônico Inteligente Virtual (vIED)

3.2.1 Características Gerais

O desenvolvimento do vIED é fundamentado na biblioteca de código aberto libIEC61850, escrita em linguagem de programação C (ZILGITH). Esta biblioteca oferece uma *Application Programming Interface* (API) que modela e disponibiliza todos os protocolos de comunicação da norma IEC 61850, como SV, GOOSE e MMS, facilitando sua utilização. A API é composta por cliente e servidor, aplicada a todos os protocolos de comunicação, comentados anteriormente (FREITAS, 2022b) . A versão 1.5.1, utilizada neste trabalho, provou ser completamente estável durante a realização dos testes.

Os vIEDs são dispositivos virtuais que operam em conformidade com a norma IEC 61850, integrando-se aos modelos de subestações digitais. deles são capazes de executar todas as funções de proteção ANSI (como 50, 51, 51V, 50N, 51N, 67N), as quais foram testadas e validadas em plataforma SCADA SAGE. A integração com a Merging Unit (MU), responsável pela digitalização de grandezas elétricas e transmissão de dados de processo via protocolo SV, é um ponto crucial. Essa comunicação permite que os vIEDs atuem em equipamentos de campo, viabilizando o controle e a proteção de sistemas reais (MARTINS, 2023).

3.2.2 Novas Funcionalidades

Este trabalho se baseia nas funcionalidades já implementadas em versões anteriores do vIED, aproveitando suas capacidades e aprimorando suas tecnologias. As melhorias e novas funcionalidades incluem:

- Assinatura de múltiplas mensagens GOOSE de múltiplos dispositivos (físicos ou virtuais);
- Utilização de threads na linguagem C para linhas de execuções concorrentes;
- Criação de um vIED multifunção de proteção;
- Implementação de recomposição automática;
- Adoção da conteinerização de vIEDs.

3.2.2.1 Conteinerização de vIEDs

Uma necessidade fundamental deste trabalho é a utilização de múltiplos vIEDs em uma única máquina. Estudos anteriores como o de Freitas et al., (2025), testaram a virtualização por meio de *Hypervisor* Tipo 2, que, embora funcional, demonstrou alto consumo de recursos computacionais. Por essa razão, optou-se pela tecnologia de virtualização via *container Docker*. Ela oferece as mesmas configurações de isolamento de um *Hypervisor*, mas com um consumo computacional significativamente menor, pois compartilha o kernel do sistema operacional hospedeiro.

O funcionamento do vIED é baseado em um sistema Linux, o que é ideal devido às permissões e à compatibilidade com a biblioteca libIEC61850. Para isso, um contêiner *Docker* foi configurado como um modelo para a criação rápida e prática dos vIEDs. A tecnologia de *Container* utiliza uma imagem alocada em um ambiente isolado, que compartilha o kernel do sistema principal, mas mantém suas rotinas e atividades isoladas.

Para que o vIED em contêiner funcione como um dispositivo de rede independente não basta que o ambiente seja isolado. É preciso que ele possa se comunicar de forma autônoma. A solução *Docker Networking* com o *driver Macvlan*, atende a esse requisito. Ela cria uma rede virtual interna, atribuindo endereços MAC únicos a cada *container* e possibilitando a configuração de endereços IP (via protocolo DHCP ou manualmente) (DOCKER, 2013). Além das configurações de rede, o contêiner do vIED foi configurado para ter todas as permissões necessárias para a comunicação e para a utilização da placa de rede física do hardware hospedeiro.

3.2.2.2 Utilização de Threads na Linguagem C

A programação do vIED, desenvolvida em C, utiliza threads para garantir a execução concorrente de suas funções. A principal motivação para o uso dessa tecnologia, é a

necessidade de executar diversas funções de forma paralela, como as funções de proteção e a função de recomposição do sistema. Uma vez que a linguagem C é procedural e naturalmente sequencial, o uso de concorrência é essencial para aumentar a confiabilidade e o desempenho do algoritmo (Tanenbaum; Bos, 2015).

Na API da biblioteca libIEC61850, a utilização de threads já é um padrão para as funções de comunicação como SV, GOOSE, MMS, garantindo que esses processos sejam executados de forma concorrente. Adicionalmente, uma biblioteca local foi desenvolvida para facilitar a criação e o gerenciamento de threads no algoritmo do vIED. Isso possibilitou transformar as lógicas de proteção e de recomposição – que internamente mantêm uma lógica sequencial – em funções concorrentes. Essa abordagem não apenas aumenta a confiabilidade do vIED, mas também otimiza a utilização dos núcleos do processador, mitigando o funcionamento sequencial do algoritmo principal.

3.2.2.3 vIED Multifunção de Proteção

Nos trabalhos de Freitas (2022a), (2022b) e Martins (2023), o vIED operava com um arquivo ou algoritmo separado para cada função de proteção, o que impedia que um único equipamento fosse parametrizado com múltiplas funções simultaneamente. Para mitigar essa limitação, foram implementadas funções concorrentes na linguagem de programação C. Essa abordagem permitiu a criação de um vIED multifunção, onde cada função de proteção é executada por uma thread concorrente. Com essa nova capacidade, o vIED agora pode replicar todas as funções de um dispositivo de proteção real em um único algoritmo, abrindo novos horizontes para testes e aplicações mais complexas.

3.2.2.4 Assinatura de Múltiplas Mensagens GOOSE

Dispositivos físicos de proteção (IEDs) em sistemas elétricos são capazes de assinar múltiplas mensagens GOOSE de diversos dispositivos diferentes. No desenvolvimento inicial abordado no tópico 3.2.2.3, essa funcionalidade não foi implementada, pois os testes se concentravam na comunicação ponto a ponto. No entanto, para aplicação em uma rede complexa como a rede de distribuição do Campus do Pici, a implementação do SRAD exigiu que os vIEDs fossem capazes de assinar múltiplas mensagens GOOSE. O sistema de recomposição proposto neste trabalho depende unicamente da comunicação horizontal entre os vIEDs, ou seja, da troca de mensagens GOOSE, tornando essa funcionalidade essencial para o

seu pleno funcionamento.

3.3 Sistema de Recomposição Automático Distribuído

Um Sistema de Recomposição Automático (SRA) tem como objetivo restaurar a energia de uma rede com topologia radial com recurso ou em anel após a ocorrência de uma falta, visando mitigar o número de consumidores afetados. Tradicionalmente, essa inteligência é centralizada em um software que atua de forma separada do sistema SCADA. A partir da análise de dados da rede elétrica, esse software define cenários e executa as manobras necessárias para isolar o trecho em falta e restaurar a maior parte possível do sistema.

Em contraste, um SRA distribuído não depende de um software unificador. Nesse modelo, a inteligência é descentralizada e reside nos próprios dispositivos inteligentes da rede. Eles se comunicam diretamente entre si para tomar decisões e atuar de forma coordenada na recomposição do sistema em caso de falta. Essa abordagem representa um avanço significativo, aumentando a resiliência e a velocidade de resposta da rede.

3.4 Merging Unit Virtual (vMU)

A Merging Unit (MU) é um dispositivo fundamental que atua como a ponte entre o mundo físico e o virtual em subestações digitais. Sua principal função é digitalizar as grandezas elétricas do campo e transmiti-las por meio de mensagens Sampled Values (SV), além de replicar comandos e estados de dispositivos como disjuntores.

Com a crescente utilização de Dispositivos Eletrônicos Inteligentes Virtuais (vIEDs), surge a necessidade de uma contraparte virtual para simular as grandezas e os estados do campo em cenários de teste complexos. É nesse contexto que se insere a Merging Unit Virtual (vMU). A vMU é um dispositivo virtual que reproduz as características de a uma MU física, com o objetivo de simular as grandezas elétricas e o estado de um disjuntor. Criada a partir de um arquivo .IID de uma MU real, a vMU herda todas as suas características IEC 61850. Além de enviar mensagens SV, a vMU também é capaz de enviar mensagens GOOSE, permitindo que o vIED a controle para simular as operações de um disjuntor real (KABBARA et al., 2024).

3.5 Integração do SRAD e Considerações de Implementação

Embora o SRAD funcione de forma autônoma, sem depender de dados do sistema SCADA para suas decisões, a integração com o supervisório é fundamental. A comunicação do SRAD baseia-se no protocolo horizontal GOOSE entre os vIEDs, garantindo a descentralização da inteligência e a agilidade nas atuações. No entanto, a integração com o SCADA facilita a visualização e o acompanhamento das etapas de recomposição, registrando todos os eventos em um *log* de alarmes. Para isso, os atributos de dados de cada vIED precisam ser mapeados e identificados no sistema supervisório.

Para a simulação completa da rede de distribuição do Campus do Pici, que possui 9 disjuntores controlados por relés de proteção, são necessários 9 vIEDs. A cada vIED, é associada uma vMU, totalizando 18 dispositivos virtuais. As tecnologias de virtualização em *containers Docker* são a solução mais vantajosa para esta implementação, devido ao seu baixo consumo de processamento por contêiner.

3.6 Considerações Finais

Para que a plataforma de testes atenda aos requisitos do SRAD, são necessárias melhorias no vIED desenvolvido em trabalhos anteriores. As modificações incluem o desenvolvimento de um vIED multifunção de proteção, a implementação de funções concorrentes no algoritmo do dispositivo e a capacidade de múltipla assinatura de mensagens GOOSE. Com essas melhorias, a plataforma estará apta a demonstrar o pleno funcionamento de um sistema de recomposição completamente distribuído, onde a inteligência da rede se dissolve entre seus dispositivos inteligentes.

4. DESENVOLVIMENTO DO SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICA DISTRIBUÍDO (SRAD) PARA REDE PICI

Neste capítulo são apresentadas as etapas de desenvolvimento do Sistema de Recomposição Automático Distribuído (SRAD). O foco está nas melhorias implementadas no vIED de proteção, na criação da vMU, na conteinerização dos dispositivos virtuais e na integração do SRAD com o sistema SCADA.

4.1 Melhorias no vIED

4.1.1 Utilização de Threads

Para que o vIED se tornasse multifuncional, foi necessário incorporar o uso de threads em seu algoritmo, superando a execução puramente procedural da linguagem C. As thread permitem que diferentes funções, seja de proteção ou de recomposição, sejam executadas de forma concorrente, aumentando significativamente a velocidade de resposta do dispositivo virtual.

A criação de um thread exige que ele seja vinculado a uma função externa à função *main*. O processo envolve a declaração de uma variável do tipo thread e sua associação à função desejada, conforme ilustrado na Figura 8, que apresenta o trecho de código responsável por essas declarações.

Figura 8 – Declaração de variáveis do tipo Thread

```
Thread thread_50, thread_50N, thread_51, thread_51V, thread_51N, thread_67, thread_67N;
```

Fonte: Elaborado pelo autor.

4.1.2 vIED Multifunção de Proteção

O vIED multifuncional é caracterizado pela execução de funções concorrentes em um único algoritmo. A utilização de threads foi direcionada especificamente para as funções de proteção e recomposição, aumentando a eficiência do processamento. No entanto, é fundamental considerar que o uso excessivo de threads pode impactar negativamente o desempenho do algoritmo e elevar os requisitos mínimos de hardware.

Na Figura 9, é mostrada a criação de threads para cada função de proteção ANSI

disponível no vIED (50, 50N, 51, 51N, 51V, 67, 67N). Para otimizar o uso do hardware, as threads são criadas de forma condicional: apenas as funções de proteção que foram devidamente parametrizadas (com valor de pick-up diferente de zero) têm suas threads iniciadas.

Figura 9 – Criação de threads para funções de proteção no vIED

```
if (pick_up_50 > 0)
{
    thread_50 = Thread_create((ThreadExecutionFunction)funcao_50, (void *) thread_50, false);
    Thread_start(thread_50);
}
if (pick_up_50N > 0)
{
    thread_50N = Thread_create((ThreadExecutionFunction)funcao_50N, (void *) thread_50N, false);
    Thread_start(thread_50N);
}
if (pick_up_51 > 0)
{
    thread_51 = Thread_create((ThreadExecutionFunction)funcao_51, (void *) thread_51, false);
    Thread_start(thread_51);
}
if (pick_up_51V > 0)
{
    thread_51V = Thread_create((ThreadExecutionFunction)funcao_51V, (void *) thread_51V, false);
    Thread_start(thread_51V);
}
if (pick_up_51N > 0)
{
    thread_51N = Thread_create((ThreadExecutionFunction)funcao_51N, (void *) thread_51N, false);
    Thread_start(thread_51N);
}
if (pick_up_67 > 0)
{
    thread_67 = Thread_create((ThreadExecutionFunction)funcao_67, (void *) thread_67, false);
    Thread_start(thread_67);
}
if (pick_up_67N > 0)
{
    thread_67N = Thread_create((ThreadExecutionFunction)funcao_67N, (void *) thread_67N, false);
    Thread_start(thread_67N);
}
```

Fonte: Elaborado pelo autor.

O vIED capta os dados de corrente a partir de mensagens SV, que são atualizadas com a frequência do sistema, 60 Hz, comprovação atestada no trabalho de Martins (2023), onde é validado toda a comunicação entre vIED e MU. Consequentemente, as funções de proteção são atualizadas nessa mesma frequência. A Figura 10 demonstra o algoritmo da função de sobrecorrente instantânea (ANSI 50), com tempo de atualização de 17 milissegundos.

Figura 10 – Algoritmo da função de proteção ANSI 50

```
void funcao_50()
{
    while (1)
    {
        if (corrente_primarioA > pick_up_50)
        {
            printf("-------------------------------------------------------------------------------------------------------------\n");
            printf("                        ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE A                                    \n");
            printf("-------------------------------------------------------------------------------------------------------------\n");
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind04_stVal, true);
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
        }
        if (corrente_primarioB > pick_up_50)
        {
            printf("-------------------------------------------------------------------------------------------------------------\n");
            printf("                        ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE B                                    \n");
            printf("-------------------------------------------------------------------------------------------------------------\n");
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind05_stVal, true);
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
        }
        if (corrente_primarioC > pick_up_50)
        {
            printf("-------------------------------------------------------------------------------------------------------------\n");
            printf("                        ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE C                                    \n");
            printf("-------------------------------------------------------------------------------------------------------------\n");
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind06_stVal, true);
            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
        }
        if (((corrente_primarioA) || (corrente_primarioB) || (corrente_primarioC)) > pick_up_50)
        {
            funcao_50_62BF();
        }
        Thread_sleep(17);
    }
}
```

Fonte: Elaborado pelo autor.

Outro aspecto relevante é a estrutura recursiva das funções alocadas em threads, que permite que elas operem em um loop contínuo até o encerramento do algoritmo do vIED. Esse comportamento é análogo ao de IEDs físicos, que funcionam ininterruptamente na rede.

4.1.3 Assinatura de Múltiplas Mensagens GOOSE

A assinatura de múltiplas mensagens GOOSE é uma funcionalidade essencial para dispositivos virtuais, como o vIED. As ferramentas da biblioteca de modelagem para o padrão IEC 61850 oferecem os recursos necessários, mas o processo exige etapas específicas para ser implementado com sucesso.

A Figura 11 ilustra esse processo, que envolve a identificação de cada mensagem a ser assinada. Contudo, para que a assinatura ocorra, é necessário criar uma função de análise para cada mensagem GOOSE declarada.

Figura 11 – Assinatura de múltiplas de mensagens GOOSE

```
//Recepção e Assinatura de mensagens GOOSE
GooseReceiver receiver = GooseReceiver_create();
GooseReceiver_setInterfaceId(receiver, "eth0");
GooseSubscriber subscriber6 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$GOOSE_STATUS", NULL);
GooseSubscriber subscriber7 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$GOOSE_STATUS", NULL);
GooseSubscriber subscriber  = GooseSubscriber_create("VIED_21L5CFG/LLN0$GO$GOOSE_STATUS", NULL);
GooseSubscriber subscriber1 = GooseSubscriber_create("VIED_21L5CFG/LLN0$GO$GOOSE_POWER", NULL);
GooseSubscriber subscriber2 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$GOOSE_POWER", NULL);
GooseSubscriber subscriber3 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$GOOSE_POWER", NULL);
GooseSubscriber subscriber4 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$CONTROL_21L6", NULL);
GooseSubscriber subscriber8 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$CONTROL_21L6", NULL);
GooseSubscriber subscriber5 = GooseSubscriber_create("MUBinIO_BinaryInputs/LLN0$GO$VMU_06_GOOSE", NULL);
GooseSubscriber_setListener(subscriber, gooseListener, iedServer);
GooseSubscriber_setListener(subscriber1, gooseListener1, iedServer);
GooseSubscriber_setListener(subscriber2, gooseListener2, iedServer);
GooseSubscriber_setListener(subscriber3, gooseListener3, iedServer);
GooseSubscriber_setListener(subscriber4, gooseListener4, iedServer);
GooseSubscriber_setListener(subscriber8, gooseListener8, iedServer);
GooseSubscriber_setListener(subscriber5, gooseListener5, iedServer);
GooseSubscriber_setListener(subscriber6, gooseListener6, iedServer);
GooseSubscriber_setListener(subscriber7, gooseListener7, iedServer);
GooseReceiver_addSubscriber(receiver, subscriber);
GooseReceiver_addSubscriber(receiver, subscriber1);
GooseReceiver_addSubscriber(receiver, subscriber2);
GooseReceiver_addSubscriber(receiver, subscriber3);
GooseReceiver_addSubscriber(receiver, subscriber4);
GooseReceiver_addSubscriber(receiver, subscriber8);
GooseReceiver_addSubscriber(receiver, subscriber5);
GooseReceiver_addSubscriber(receiver, subscriber6);
GooseReceiver_addSubscriber(receiver, subscriber7);
```

Fonte: Elaborado pelo autor.

Essa função, demonstrada na Figura 12, tem como objetivo interpretar a mensagem recebida. O processo inclui a conversão dos dados da mensagem para bytes, seu armazenamento em uma variável de memória e a posterior interpretação. Adicionalmente, os dados são identificados e utilizados em outras partes do algoritmo.

Figura 12 – Análise da mensagem GOOSE assinada

```
static void
gooseListener6(GooseSubscriber subscriber, void* parameter)
{
    MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);

    char buffer[50];

    MmsValue_printToBuffer(values, buffer, 50);

    trip_2117 = buffer[1];
    estado_dj_2117 = atoi(&buffer[7]);

    uint64_t y = Hal_getTimeInMs();
}
```

Fonte: Elaborado pelo autor.

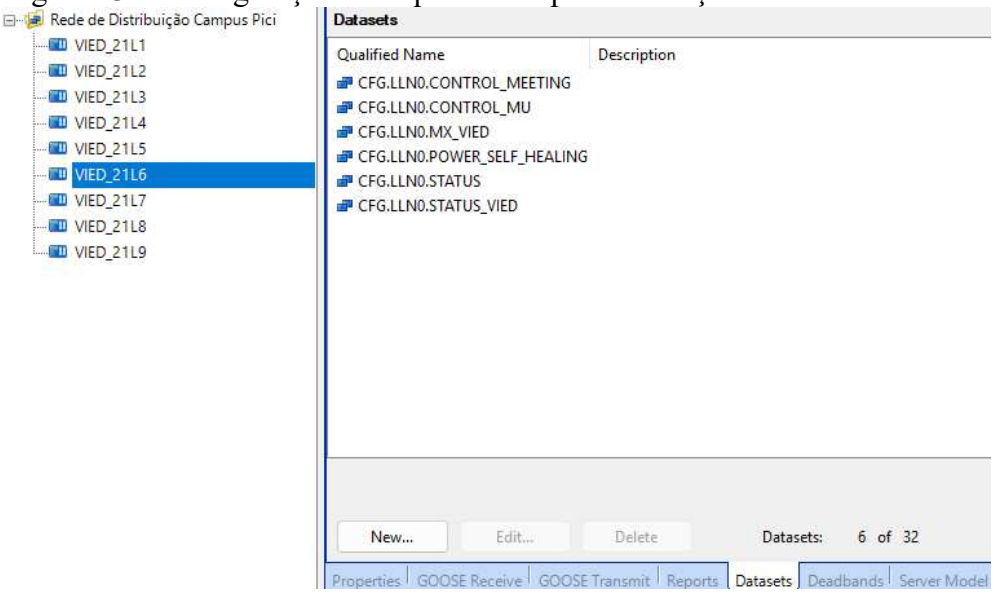
A análise deve levar em conta os diferentes tipos de atributos de dados que compõem a mensagem. Isso é crucial, pois a interpretação e a faixa de bytes ocupada no pacote

de armazenamento variam para cada tipo de dado, como ponto simples, pontos duplos e valores de medição (IEC 61850-7-3, 2003).

4.1.4 Criação de Modelos de Dados para vIEDs

Uma das melhorias necessárias aplicadas ao vIED, é determinada pela criação de nove arquivos lógicos, que atendam a toda rede de distribuição do Campus do Pici. Com auxílio do software AcSELerator Architect, da fabricante Schweitzer Engineering Laboratories (SEL), criou-se arquivos *IED Capability Desciption* (IID), utilizados na criação de cada vIED. A Figura 13 demonstra o processo de criação de Datasets padrão dos vIEDs e quantidade de dispositivos na estação de trabalho IEC 61850.

Figura 13 – Configuração de arquivos IID para utilização em vIEDs



Fonte: Elaborado pelo autor.

4.2 Conteinerização de Dispositivos Virtuais

Os dispositivos virtuais, baseados no sistema operacional Linux, são implementados através de contêineres *Docker*. Para cada dispositivo, a criação de um contêiner utiliza a imagem do sistema Ubuntu Linux, replicando o ambiente operacional necessário. O processo de conteinerização inclui a criação de contêiner a partir da imagem Ubuntu, a instalação das bibliotecas de e, por fim, a instalação do algoritmo do dispositivo virtualizado (vIED ou vMU).

A definição de uma rede virtual é essencial para o funcionamento dos contêineres.

Utilizou-se o driver *Macvlan* para a criação dessa rede no *Docker*, conforme demonstrado na Figura 14. Cada contêiner é associado a uma placa de rede física do hardware central, o que permite a comunicação com dispositivos externos.

Figura 14 – Prompt de configuração de rede e contêineres Docker

```
Network

docker network create -d macvlan --subnet 10.102.76.0/23 --gateway 10.102.76.1 -o parent=enp2s0 rede_vied

Container

docker container run --name VIED --network rede_vied --ip 10.102.76.XX --cap-add=ALL -it vied
```

Fonte: Elaborado pelo autor.

Para a configuração do projeto, foram criados 18 contêineres (9 vIEDs e 9vMUs), cada um com IP e endereço MAC destintos. Cada vIED é responsável por um disjuntor na rede de distribuição estudada, e cada vMU está diretamente associada a um vIED. A criação individual e isolada desses contêineres, apresentada na Figura 15, foi crucial para o desenvolvimento do SRAD na rede de distribuição do campus do Pici.

Figura 15 – Contêineres criado para o teste de recomposição

```
grei@grei-Dell-G15-5530 ~
$ docker container ls -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS                      PORTS     NAMES
6d4e6414f761   vied      "/bin/bash"   4 months ago    Exited (137) 3 weeks ago              VIED_21L6
67be0af49b22   vied      "/bin/bash"   4 months ago    Exited (137) 3 weeks ago              VIED_21L2
83ae9c6b3dba   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L9
6cb7ebb12a9e   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L8
5b86c048cd10   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L7
8078b7e2f721   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L6
11569c2d2b68   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L4
455d505ffbcf   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L3
4960093837ca   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L2
56cb0e5031f4   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L1
45db27badfe3   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L9
312ba6ac546e   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L8
34da3ccaab89   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L7
154ddb03590b   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L4
8473d7befb05   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L3
113d9e92d8e9   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L1
00e1c41a9065   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              MU_21L5
e7c8681bd8c6   vied      "/bin/bash"   6 months ago    Exited (137) 3 weeks ago              VIED_21L5
```

Fonte: Elaborado pelo autor.

4.3 Virtualização de Merging Unit (MU)

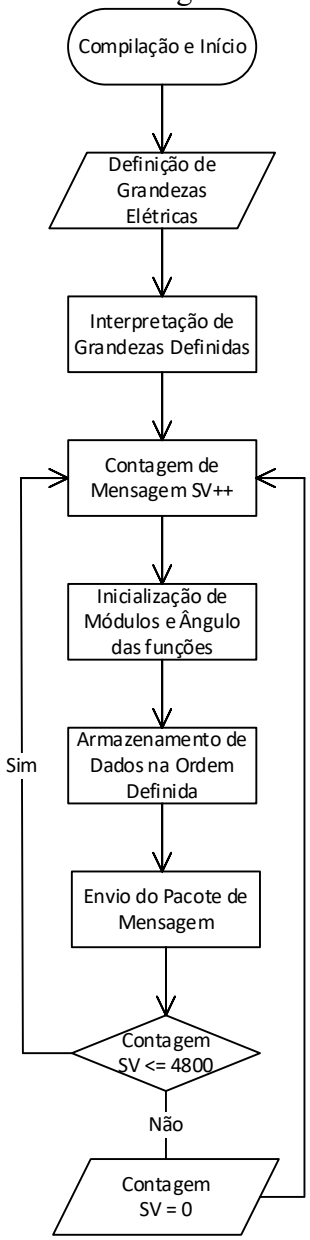
A virtualização de Merging Unit (vMU) foi desenvolvida a partir da interação com MUs físicas. O algoritmo para o envio de mensagens SV, no padrão IEC 61850, foi implementado na linguagem de programação C com auxílio da biblioteca libIEC61850. O arquivo de configuração .IID, que define a estrutura lógica dos dados, foi extraído de uma MU física, da linha de IEDs de proteção Siprotec 5 da Siemens, testada e validada em bancada (Figura 16).

Figura 16 – Linha de IEDs Siprotec 5 da fabricante Siemens

Fonte: Siemens.

O algoritmo da vMU é projetado para transmitir 80 mensagens SV por ciclo de 16,67 milissegundos, totalizando 4800 mensagens por segundo para uma frequência de 60 Hz. A Figura 17 detalha o fluxograma de funcionamento do algoritmo da vMU, descrevendo o processo de envio das mensagens Sampled Values. O algoritmo completo está disponível no Apêndice A.

Figura 17 – Fluxograma de funcionamento do algoritmo vMU



Fonte: Elaborado pelo autor.

4.4 Desenvolvimento do Sistema de Recomposição Automático Distribuído (SRAD)

O desenvolvimento do Sistema de Recomposição Automático Distribuído (SRAD) está dividido em várias etapas: transmissão de grandezas elétricas entre vIEDs, isolamento de trechos em falta, análise do sistema e recomposição.

O SRAD, conforme definido neste trabalho, realiza a recomposição e analisa as variáveis do sistema, por meio de um protocolo de comunicação horizontal, específico para dispositivos de mesma camada de automação. Baseado na norma IEC 61850, o sistema utiliza mensagens GOOSE para a comunicação entre seus componentes. O SRAD é adaptado à rede

de distribuição elétrica em estudo, que é composta por 9 disjuntores, 3 alimentadores (21L1, 21L2 e 21L3), 3 religadores (21L4, 21L5 e 21L9) e 3 encontros de alimentadores (21L6, 21L7 e 21L8), como ilustrado na Figura 7.

A Tabela 2 descreve as três classes de vIEDs presentes na rede de distribuição, com suas funções especificas:

- Alimentador: Responsável pela alimentação geral de saída da subestação e pelo envio de grandezas elétricas.
- Religador: Atua no seccionamento e delimitação de trechos, enviando grandezas elétricas e isolando trechos em falta.
- Encontro de Alimentador: Tem a função de recompor trechos desenergizados que não estejam em falta, mitigando “trechos mortos” na rede, além de enviar grandezas elétricas e isolar falhas.

Tabela 2 – Integrantes do sistema de recomposição

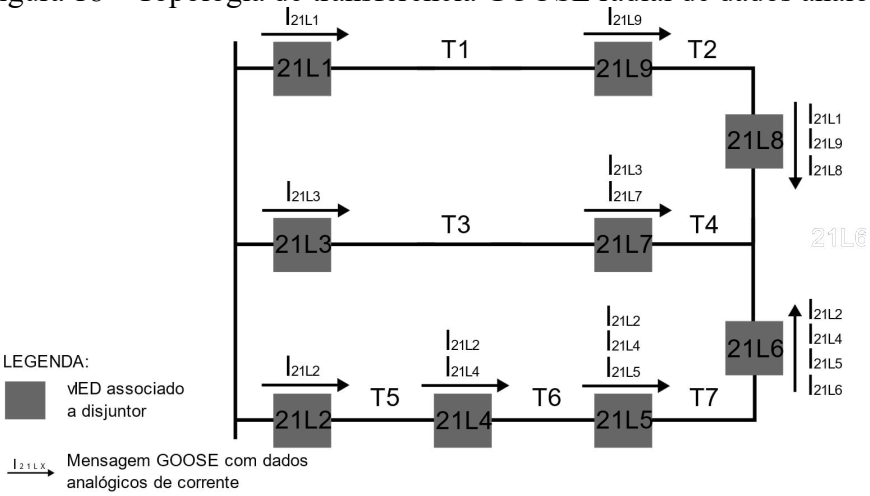
Tipo	**Tarefas**	**Atribuições na Recomposição**
Alimentador	Alimentação geral	Envio de grandezas elétricas
Religador	Delimitação de trechos	Envio de grandezas elétricas Isolação do trecho em falta
Encontro de Alimentador	Recomposição de trechos desenergizados	Envio de grandezas elétricas Isolação do trecho em falta Recomposição automática

Fonte: Elaborado pelo autor.

4.4.1 Envio de Grandezas Elétricas

O envio de grandezas analógicas ocorre de forma radial entre todos os vIEDs da rede de distribuição por meio de mensagens GOOSE. Cada vIED transmite sua medição de corrente e as medições dos vIEDs anteriores. O principal objetivo dessa topologia de transferência (Figura 18) é permitir que os dispositivos do tipo "encontro de alimentadores" construam uma matriz com os valores de corrente de cada trecho.

Figura 18 – Topologia de transferência GOOSE radial de dados analógicos



Fonte: Elaborado pelo autor.

O funcionamento correto da transferência de mensagens GOOSE com grandezas analógicas de corrente possibilita que os encontros de alimentadores construam uma matriz com a identificação da corrente em cada trecho em tempo real. Esses valores são utilizados durante o processo de recomposição para garantir que a energização ocorra de forma mais segura possível, considerando o consumo atual de cada trecho. Além disso, o sistema armazena os valores de pico de corrente para cada trecho, auxiliando na decisão da recomposição.

Para a transmissão dessas mensagens, o dataset POWER_SELF_HEALING, associado à mensagem GOOSE_POWER, armazena as variáveis analógicas de forma específica para cada vIED. A Figura 19 apresenta uma visão geral desse dataset, identificando qual atributo de dado está associado a cada disjuntor ou vIED.

Figura 19 – Dataset de envio de corrente

Mapeamento de Dados IEC 61850 - vIED de Proteção				
Descrição	Dado Padrão IEC	Tipo	Mensagem	Dataset
Corrente no Disjuntor 21L1	ANN.MVGGIO12.AnIn01.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L2	ANN.MVGGIO12.AnIn02.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L3	ANN.MVGGIO12.AnIn03.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L4	ANN.MVGGIO12.AnIn04.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L5	ANN.MVGGIO12.AnIn05.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L6	ANN.MVGGIO12.AnIn06.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L7	ANN.MVGGIO12.AnIn07.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L8	ANN.MVGGIO12.AnIn08.mag.f	MX	GOOSE	POWER_SELF_HEALING
Corrente no Disjuntor 21L9	ANN.MVGGIO12.AnIn09.mag.f	MX	GOOSE	POWER_SELF_HEALING

Fonte: Elaborado pelo autor.

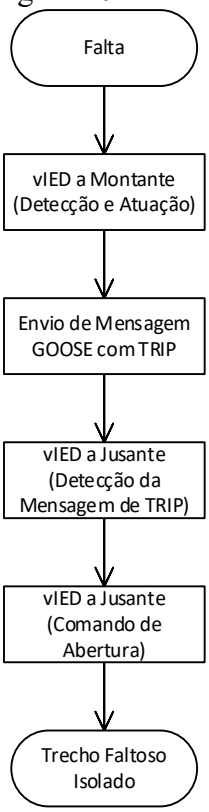
4.4.2 Isolação de Trecho em Falta

O isolamento de trechos em falta é uma das configurações essenciais em redes de distribuição com topologias assimilar à rede de distribuição do Campus do Pici. O procedimento se baseia na comunicação via mensagens GOOSE entre os vIEDs que delimitam um trecho da rede.

Em caso de uma falta, o dispositivo de proteção a montante proteção é o primeiro a atuar. Contudo, devido ao fluxo de potência unidirecional, o dispositivo a jusante não consegue detectar a falta. Para resolver isso, o vIED a montante envia uma mensagem GOOSE ao vIED a jusante, informando sobre sua atuação. Com base nessa informação, o dispositivo a jusante interpreta a ocorrência e isola o trecho, garantindo que a falta seja eliminada do sistema. Esse processo de isolamento é fundamental para que, durante a posterior recomposição, em sistemas elétricos, a falta não seja alimentada novamente.

O fluxograma que ilustra a sequência lógica dessa função está apresentado na Figura 20.

Figura 20 – Fluxograma de isolação de trechos faltosos



Fonte: Elaborado pelo autor.

Para que essa comunicação ocorra, a variável booleana de atuação (TRIP) é armazenada no dataset STATUS e é transmitida através da mensagem GOOSE_STATUS, como mapeado na Figura 21.

Figura 21 – Dataset de envio de status por GOOSE

Mapeamento de Dados IEC 61850 - vIED de Proteção				
Descrição	Dado Padrão IEC	Tipo	Mensagem	Dataset
TRIP	PRO.TRIPPTRC1.Tr.general	ST	GOOSE	STATUS
POSIÇÃO DO DISJUNTOR	PRO.BK1XCBR1.Pos.stVal	ST	GOOSE	STATUS

Fonte: Elaborado pelo autor.

4.4.3 Análise de Recomposição Automática

Antes de iniciar qualquer recomposição, o sistema realiza uma análise detalhada para determinar se a recomposição é viável. Esta etapa, considerada o "cérebro" do sistema, é responsável por tomar decisões sem intervenção humana. A análise considera os seguintes pontos:

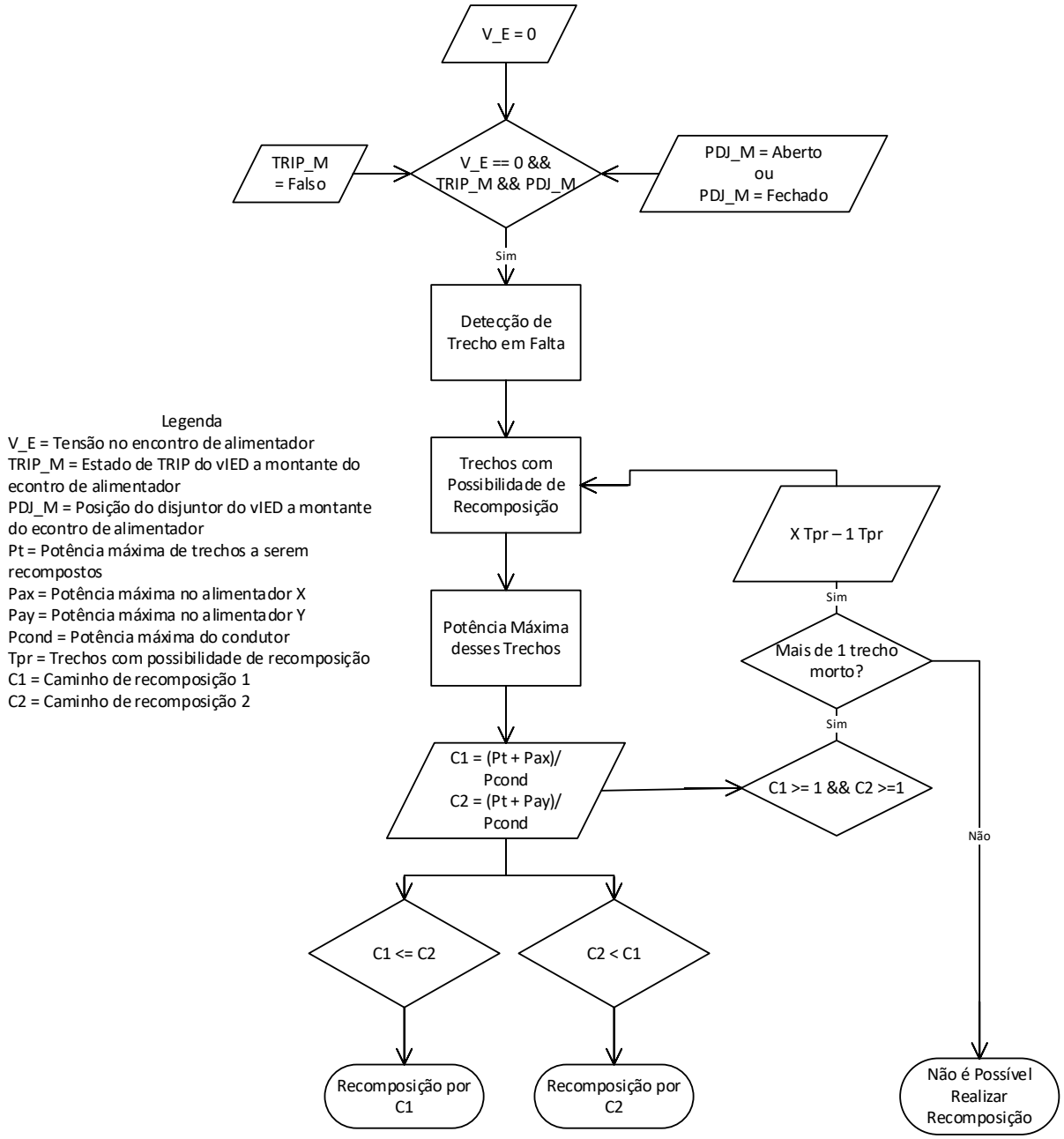
- Identificação de trecho em falta: Confirma se o trecho foi isolado corretamente e se está localizado no alimentador associado ao ponto de encontro;
- Verificação de potência: Avalia se o trecho isolado pode ser recomposto por outros alimentadores;
- Análise de recomposição ótima: Decide qual alimentador deve ser utilizado para a recomposição.

O resultado dessa análise indica qual ação deve ser tomada com base nas variáveis de campo. O fluxograma completo do algoritmo de análise de recomposição está na Figura 22. Para mais detalhes, o Apêndice B apresenta o algoritmo integral do vIED, detalhando a lógica em linguagem C e o uso das ferramentas da libIEC61850.

Esta inteligência está alocada em cada um dos três encontros de alimentadores (21L6, 21L7 e 21L8) O vIED 21L6, em particular, lida com cenários mais complexos por ser capaz de recompor 2 trechos. A versatilidade do algoritmo permite sua adaptação a outras redes distintas com pequenas modificações.

Após a análise, a lógica de recomposição assume total autonomia para atuar, utilizando procedimentos de segurança para garantir uma recomposição rápida e precisa do sistema.

Figura 22 – Fluxograma da análise de possibilidade de recomposição



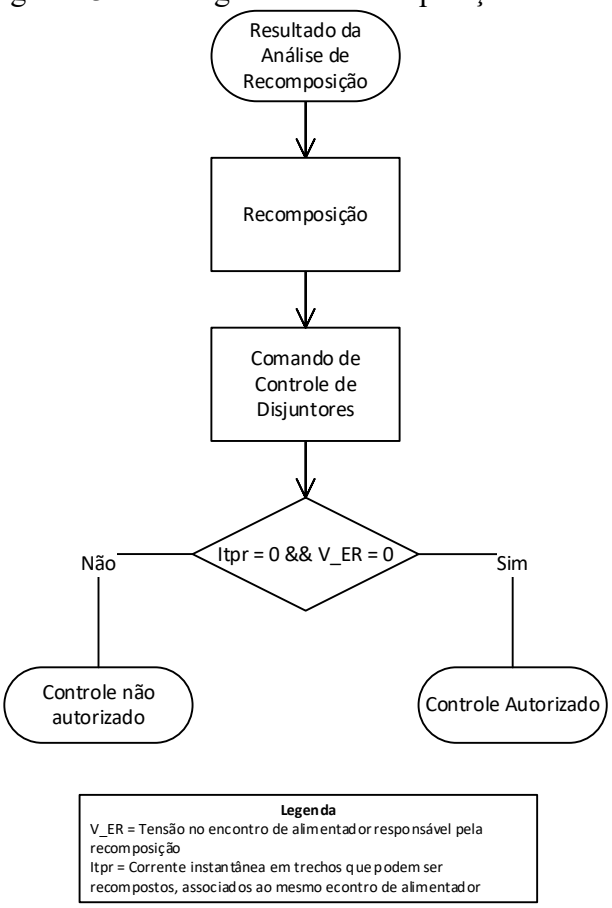
Fonte: Elaborado pelo autor.

4.4.4 Recomposição Automática Distribuída

O processo de recomposição do sistema de distribuição é uma função primordial do SRAD e envolve o controle coordenado entre os encontros de alimentadores. Para garantir a segurança e a seletividade, a recomposição só é acionada sob condições específicas. O encontro de alimentador responsável pela ação, deve confirmar que os trechos a serem recompostos estão desenergizados (corrente e tensão instantânea em zero). Somente após essa verificação, os outros dois encontros de alimentadores liberam seu controle, finalizando o processo de

recomposição. O fluxograma da Figura 23 ilustra a lógica de funcionamento e a etapa final responsável pelo controle e execução da recomposição.

Figura 23 – Fluxograma da recomposição automática



Fonte: Elaborado pelo autor.

Para o controle entre os encontros de alimentadores, o dataset CONTROL_(Encontro que precisa ser controlado) é utilizado. Ele armazena as variáveis booleanas e o dado analógico de tensão necessários para o controle, e está associado à mensagem GOOSE correspondente.

4.4.5 Integração de SRAD ao Sistema SCADA

A plataforma SCADA é utilizada principalmente para o monitoramento e a visualização dos resultados. Para uma integração completa com os vIEDs foram seguidos os passos de configuração descritos a seguir:

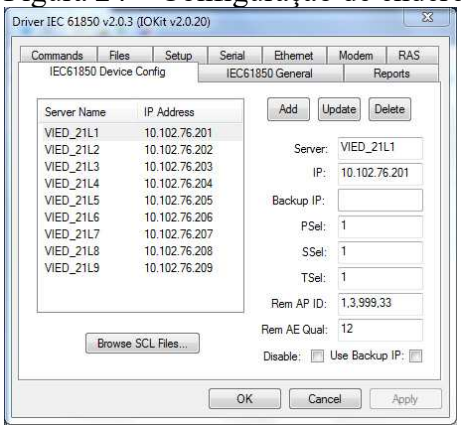
- Configuração de Endereços IP: Necessária devido ao uso do protocolo TCP/IP pela mensagem MMS;

- Mapeamento de Pontos IEC 61850: Realizado conforme uma tabela de dados específica para o SRAD;
- Criação e Configurações de Alarmes: Essenciais para a visualização dos eventos;
- Associação de Pontos de Controle e Status: Vincula os dados dos vIEDs aos elementos visuais na tela do SCADA.

4.4.5.1 Configuração de Endereços IP

A Figura 24 exibe a configuração dos endereços IP no sistema SCADA para os noves vIEDs utilizados na rede de distribuição virtual do Campus do Pici. Esta etapa é fundamental, pois o protocolo MMS, que realiza a comunicação entre o SCADA e os vIEDs, utiliza a camada de transporte TCP/IP.

Figura 24 – Configuração de endereços IP no sistema SCADA



Fonte: Captura de tela do software SCADA ELIPSE.

4.4.5.2 Mapeamento de Dados IEC 61850

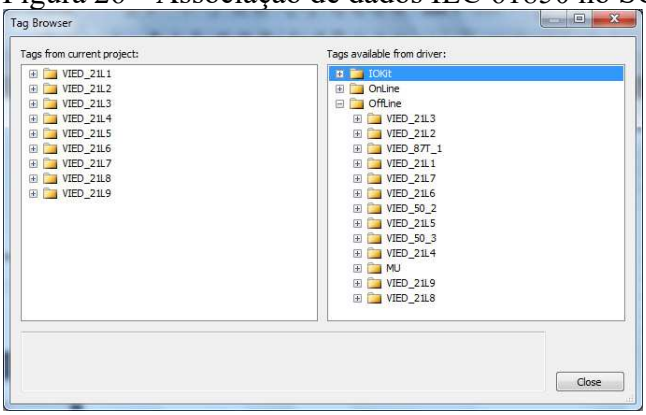
Cada vIED possui datasets criados especificamente para a comunicação com o sistema SCADA. A Figura 25 apresenta o dataset definido para o vIED-21L6, que serve como modelo para os demais dispositivos da rede. Esse mapeamento de dados externos para o banco de dados local do SCADA ELIPSE é demonstrado na Figura 26.

Figura 25 – Dataset para envio de reports MMS

Mapeamento de Dados IEC 61850 - vIED de Proteção				
Descrição	Dado Padrão IEC	Tipo	Mensagem	Dataset
21L6 - Função de Proteção de Falha de Disjuntor	PRO.BFR1RBRF1.OpEx.general	ST	MMS	STATUS_VIED
21L6 - Trip	PRO.TRIPPTRC1.Tr.general	ST	MMS	STATUS_VIED
21L6 - Posição do Disjuntor	PRO.BK1XCBR1.Pos.stVal	ST	MMS	STATUS_VIED
21L6 - Controle de Disjuntor	PRO.BKR1CSWI1.Pos.Oper.ctlVal	CO	MMS	STATUS_VIED
21L6 - Função de Proteção de Sobrecorrente Instantânea Fase A (50)	ANN.SVGGIO3.Ind04.stVal	ST	MMS	STATUS_VIED
21L6 - Função de Proteção de Sobrecorrente Instantânea Fase B (50)	ANN.SVGGIO3.Ind05.stVal	ST	MMS	STATUS_VIED
21L6 - Função de Proteção de Sobrecorrente Instantânea Fase C (50)	ANN.SVGGIO3.Ind06.stVal	ST	MMS	STATUS_VIED
21L6 - Isolação do T7 em Falta	ANN.SVGGIO3.Ind.23.stVal	ST	MMS	STATUS_VIED
21L6 - Comando Recebido de 21L7	ANN.SVGGIO3.Ind.24.stVal	ST	MMS	STATUS_VIED
21L6 - Comando Recebido de 21L8	ANN.SVGGIO3.Ind.25.stVal	ST	MMS	STATUS_VIED
21L6 - Detecção do T5 em Falta	ANN.SVGGIO3.Ind.26.stVal	ST	MMS	STATUS_VIED
21L6 - Análise de Recomposição	ANN.SVGGIO3.Ind.19.stVal	ST	MMS	STATUS_VIED
21L6 - Reeligar por 21L8	ANN.SVGGIO3.Ind15.stVal	ST	MMS	STATUS_VIED
21L6 - Reeligar por 21L7	ANN.SVGGIO3.Ind16.stVal	ST	MMS	STATUS_VIED
21L6 - Recomposição Finalizada	ANN.SVGGIO3.Ind17.stVal	ST	MMS	STATUS_VIED
21L6 - Detecção do T6 em Falta	ANN.SVGGIO3.Ind18.stVal	ST	MMS	STATUS_VIED

Fonte: Elaborado pelo autor.

Figura 26 – Associação de dados IEC 61850 no SCADA ELIPSE

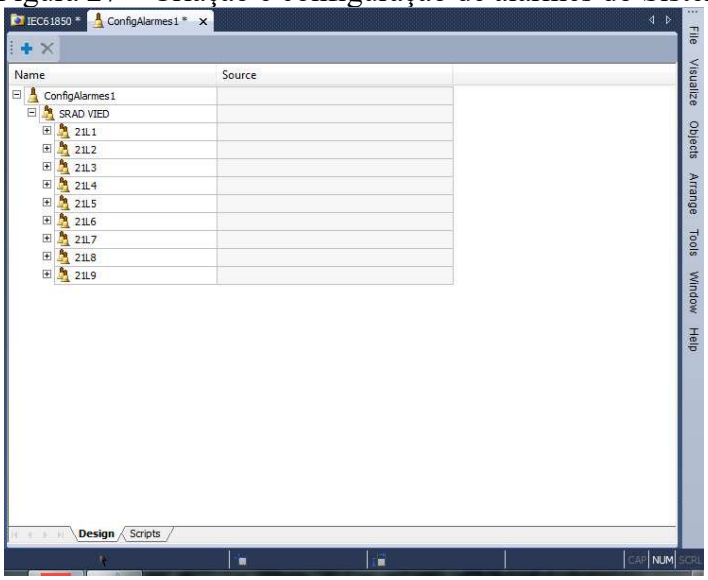


Fonte: Captura de tela do software SCADA ELIPSE.

4.4.5.3 Alarmes

A configuração de alarmes é vital para a visualização dos eventos e da sequência de ocorrências na subestação. A Figura 27 mostra a criação e a configuração de alarmes associados às funções mapeadas, permitindo a correta interpretação dos eventos do sistema.

Figura 27 – Criação e configuração de alarmes do Sistema SCADA

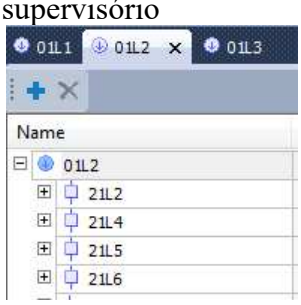


Fonte: Captura de tela do software SCADA ELIPSE.

4.4.5.4 Associação de elementos

A associação de dados do vIEDs a elementos visuais no sistema supervisório é fundamental para a animação da tela, e para o monitoramento em tempo real. Neste trabalho, a animação dos disjuntores foi utilizada como parâmetro de verificação do funcionamento da rede e da comunicação com os nove vIEDs. A Figura 28 apresenta um exemplo de disjuntores associados corretamente, permitindo sua animação de acordo com as atuações no campo.

Figura 28 – Associação de dados a elementos do sistema supervisório



Fonte: Captura de tela do software SCADA ELIPSE.

4.5 Considerações Finais

O desenvolvimento do SRAD, conforme detalhado no capítulo 3, exigiu não apenas a implementação do sistema de recomposição, mas também um aprimoramento da plataforma de simulação, incluindo melhoria dos vIEDs e vMUs. A tecnologia de virtualização por contêiner Docker foi crucial para simular toda a rede de distribuição do Campus do Pici. Cada

contêiner foi criado a partir de uma imagem Ubuntu, com permissões de acesso e associação à placa de rede física do hardware central para virtualização.

Com a finalização dos aprimoramentos nos vIEDs e a criação das vMUs, o desenvolvimento do SRAD foi iniciado. A lógica de recomposição foi concentrada na transferência radial de valores de corrente entre os vIEDs da rede simulada, permitindo que os dispositivos responsáveis pela recomposição compreendessem as cargas em tempo real. O algoritmo do SRAD também foi projetado para traçar o melhor caminho de recomposição, minimizando o estresse na rede e, se necessário, reduzindo os trechos a serem recompostos. A análise contínua é essencial, mas a efetivação da recomposição é determinada pela comunicação segura e coordenada entre os encontros de alimentadores, reforçando a segurança e a confiabilidade do sistema.

5. TESTE E VALIDAÇÃO DO SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICO DISTRIBUÍDO EM PLATAFORMA DE PROTEÇÃO AUTOMAÇÃO E CONTROLE (PAC)

Neste capítulo, são apresentados os resultados dos testes experimentais do Sistema de Recomposição Automático Distribuído (SRAD), desenvolvido no capítulo 4. O SRAD, baseado em vIEDs de proteção multifunção e vMU com tecnologia de virtualização via conteinerização, foi aplicado a uma plataforma PAC para validar seu funcionamento. Os testes consistem na simulação de faltas na rede de distribuição do Campus do Pici, com o objetivo de verificar a capacidade do sistema em isolar o trecho defeituoso e realizar a recomposição automática. O funcionamento do SRAD é totalmente baseado na comunicação por meio de mensagens GOOSE.

5.1 Configuração da Plataforma PAC

A plataforma PAC de testes foi montada com os seguintes componentes:

- PC industrial SEL-3354: um computador industrial da Schweitzer Engineering Laboratories (SEL);
- Switch Gerenciável SEL-2730M: Também da SEL, utilizado para gerenciar o tráfego de rede;
- GPS com link via satélite SEL: Fornece a sincronização de tempo para todos os dispositivos.
- Notebook Dell G15: Responsável por hospedar os contêineres Docker que contêm os vIEDs e vMUs.

As conexões de rede foram realizadas por meio de cabos UTP CAT6, exceto pelo GPS, que está conectado ao SEL-3354, via cabo coaxial usando o protocolo de sincronismo IRIG-B. O SEL-3354, por sua vez, atua como um servidor NTP, provendo sincronização de tempo para todos os dispositivos da bancada. É importante notar que os contêineres Docker herdam a sincronia de tempo do sistema operacional principal do notebook, garantindo que todos os dispositivos virtuais operem com a mesma estampa de tempo.

A rede hospeda um tráfego massivo, pois conta com a utilização de 9 vIEDs e 9 vMUs. Para monitorar esse tráfego e garantir a integridade dos pacotes, foram utilizados os softwares Wireshark e ACMViewer, este último especializado na visualização de dados do

padrão IEC 61850.

A Tabela 3 detalha os endereços IP dos dispositivos na plataforma. A configuração de IP nos vIEDs de proteção necessário é essencial para integração com o sistema SCADA, pois as mensagens MMS utilizam TCP/IP na camada de transporte. Já os vMUs não necessitam de um endereço IP, pois sua comunicação (mensagens GOOSE e SV) é restrita à camada de enlace.

Tabela 3 – Tabela de endereços IP da plataforma

Identificação	Endereço IP	Máscara de rede	Gateway
SEL-3354	10.102.77.81	255.255.255.254	10.102.76.1
Dell G15	10.102.76.77	255.255.255.254	10.102.76.1
vIED_21L1 (Contêiner Docker)	10.102.76.201	255.255.255.254	10.102.76.1
vIED_21L2 (Contêiner Docker)	10.102.76.202	255.255.255.254	10.102.76.1
vIED_21L3 (Contêiner Docker)	10.102.76.203	255.255.255.254	10.102.76.1
vIED_21L4 (Contêiner Docker)	10.102.76.204	255.255.255.254	10.102.76.1
vIED_21L5 (Contêiner Docker)	10.102.76.205	255.255.255.254	10.102.76.1
vIED_21L6 (Contêiner Docker)	10.102.76.206	255.255.255.254	10.102.76.1
vIED_21L7 (Contêiner Docker)	10.102.76.207	255.255.255.254	10.102.76.1
vIED_21L8 (Contêiner Docker)	10.102.76.208	255.255.255.254	10.102.76.1
vIED_21L9 (Contêiner Docker)	10.102.76.209	255.255.255.254	10.102.76.1

Fonte: Elaborado pelo autor.

5.2 Arquitetura SAS da Rede de Distribuição do Campus do Pici

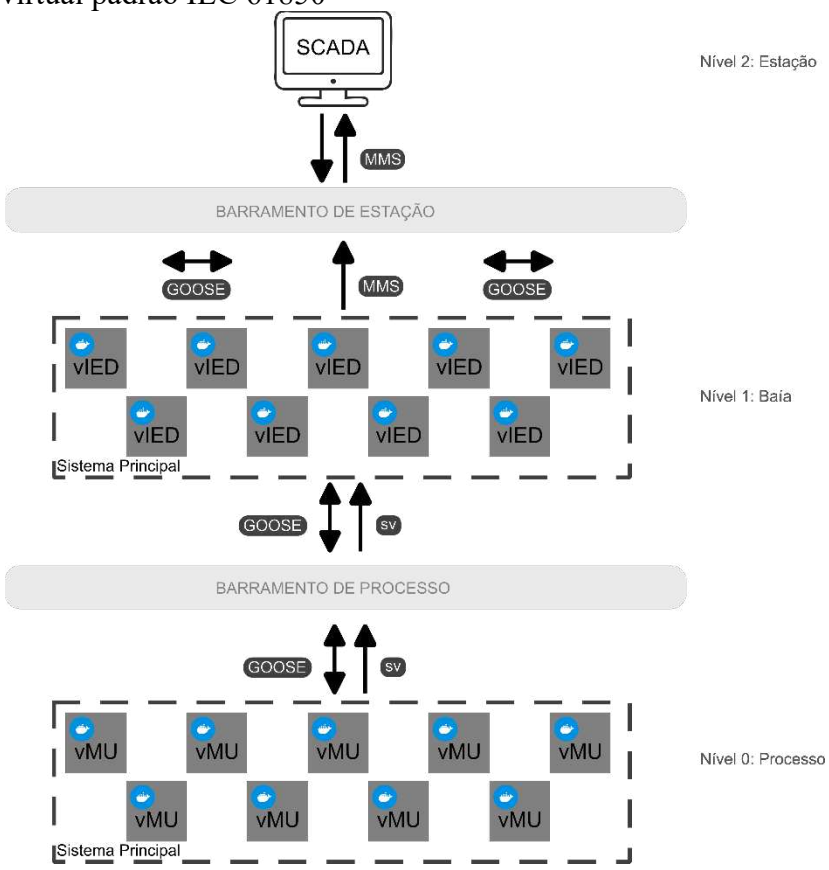
A arquitetura do Sistema de Automação de Subestação (SAS) é de extrema importância para a confiabilidade em sistemas elétricos. Na rede de distribuição do Campus do Pici, a arquitetura virtualizada segue o padrão hierárquico da IEC 61850:

- Nível 0 (Processo): Composto por 9 vMUs, que são responsáveis por enviar dados do processo (mensagens SV e GOOSE) para os vIEDs de proteção, entrada de dados;
- Nível 1 (Baia): Composto por 9 vIEDs, que aquisitam os dados do processo, comunicam-se horizontalmente com outros dispositivos do mesmo nível (via GOOSE) e comunicação verticalmente com o sistema SCADA (via MMS);

- Nível 2 (Estação): Composto pelo sistema SCADA, que adquire as informações dos vIEDs de proteção e realiza o controle do sistema quando necessário.

A Figura 29 ilustra a topologia de automação da rede de distribuição do Campus do Pici, que foi implementada de forma virtual, seguindo o padrão IEC 61850. O sistema utiliza 18 containers Docker no total, com um vIED e uma vMU destinados a cada disjuntor da rede. A comunicação nessa rede de automação é realizada por meio dos três principais protocolos da norma IEC 61850: SV para grandezas elétricas, GOOSE para comunicação horizontal e MMS para comunicação vertical entre os vIEDs e o sistema SCADA.

Figura 29 – Topologia de automação da rede de distribuição do Campus do Pici virtual padrão IEC 61850



Fonte: Elaborado pelo autor.

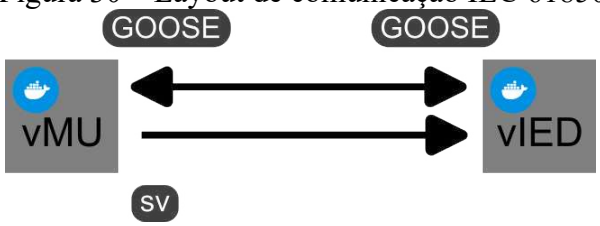
5.3 Integração entre vIED e vMU

A arquitetura da rede de distribuição implementada baseia-se inteiramente em dispositivos virtuais. Para que os vIEDs possam operar, eles precisam se comunicar com um dispositivo que forneça grandezas elétricas e simule as características de um disjuntor. A

integração entre vIED e vMU possível é a solução para essa necessidade. A Merging Unit virtual fornece as grandezas elétricas por meio de mensagens SV e simula o estado do disjuntor (abertura e fechamento) por meio de mensagens GOOSE.

A Figura 30 apresenta a configuração de comunicação padrão IEC 61850 que conecta o vIED e a vMU.

Figura 30 – Layout de comunicação IEC 61850 entre vIED e vMU



Fonte: Elaborado pelo autor.

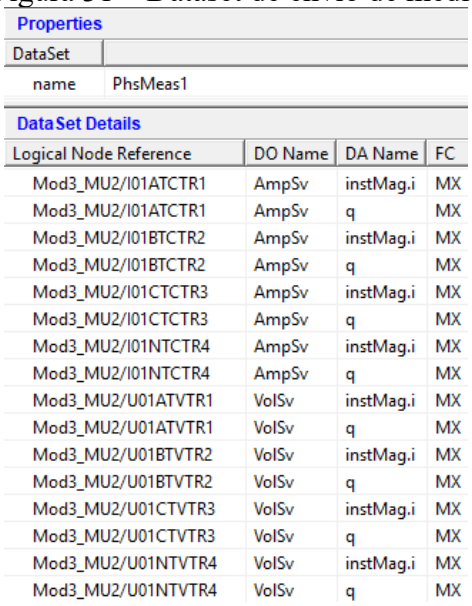
5.3.1 Envio de mensagens

Na Merging Unit virtual, o arquivo .IID é configurado para enviar mensagens GOOSE e SV. Usando o software CET850 da Schneider Eletric para visualização, é possível identificar as configurações:

- A Figura 31 mostra o dataset PhsMeas1, utilizado para envio de medições analógicas via mensagem SV, realizado pela vMU.
- A Figura 32 apresenta o dataset Status_BI, que fornece o estado do disjuntor (atributos de dados booleanos) via mensagem GOOSE, também enviado pela vMU.

Por sua vez, o vIED envia uma mensagem GOOSE específica para controlar o disjuntor virtual simulado pelo vMU. Na Figura 33, o dataset CONTROL_MU demonstra esses atributos de dados booleanos, permitindo que o vIED altere o estado do disjuntor através de uma mensagem GOOSE.

Figura 31 – Dataset de envio de medições analógicas por SV



Properties

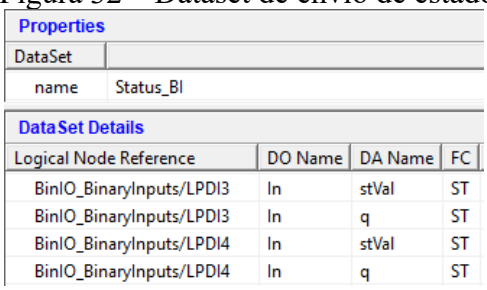
DataSet	
name	PhsMeas1

DataSet Details

Logical Node Reference	DO Name	DA Name	FC
Mod3_MU2/I01ATCTR1	AmpSv	instMag.i	MX
Mod3_MU2/I01ATCTR1	AmpSv	q	MX
Mod3_MU2/I01BTCTR2	AmpSv	instMag.i	MX
Mod3_MU2/I01BTCTR2	AmpSv	q	MX
Mod3_MU2/I01CTCTR3	AmpSv	instMag.i	MX
Mod3_MU2/I01CTCTR3	AmpSv	q	MX
Mod3_MU2/I01NTCTR4	AmpSv	instMag.i	MX
Mod3_MU2/I01NTCTR4	AmpSv	q	MX
Mod3_MU2/U01ATVTR1	VolSv	instMag.i	MX
Mod3_MU2/U01ATVTR1	VolSv	q	MX
Mod3_MU2/U01BTVTR2	VolSv	instMag.i	MX
Mod3_MU2/U01BTVTR2	VolSv	q	MX
Mod3_MU2/U01CTVTR3	VolSv	instMag.i	MX
Mod3_MU2/U01CTVTR3	VolSv	q	MX
Mod3_MU2/U01NTVTR4	VolSv	instMag.i	MX
Mod3_MU2/U01NTVTR4	VolSv	q	MX

Fonte: Captura de tela do software CET850 Schneider Eletric.

Figura 32 – Dataset de envio de estado do disjuntor por GOOSE



Properties

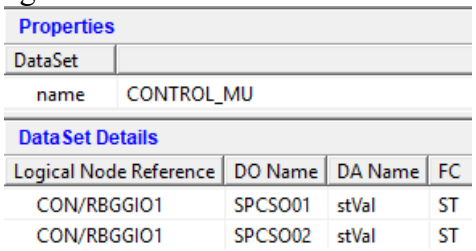
DataSet	
name	Status_BI

DataSet Details

Logical Node Reference	DO Name	DA Name	FC
BinIO_BinaryInputs/LPDI3	In	stVal	ST
BinIO_BinaryInputs/LPDI3	In	q	ST
BinIO_BinaryInputs/LPDI4	In	stVal	ST
BinIO_BinaryInputs/LPDI4	In	q	ST

Fonte: Captura de tela do software CET850 Schneider Eletric.

Figura 33 – Dataset de controle do disjuntor por GOOSE



Properties

DataSet	
name	CONTROL_MU

DataSet Details

Logical Node Reference	DO Name	DA Name	FC
CON/RBGGIO1	SPCSO01	stVal	ST
CON/RBGGIO1	SPCSO02	stVal	ST

Fonte: Captura de tela do software CET850 Schneider Eletric.

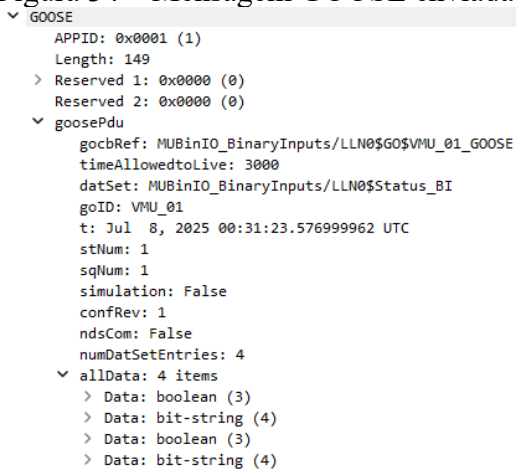
5.3.2 Assinatura de mensagens

A integração entre os dispositivos virtuais (vIED e vMU) é validada pela assinatura mútua de mensagens, um processo essencial para a comunicação e o funcionamento do sistema. . Essa assinatura é realizada diretamente no algoritmo de cada dispositivo virtual. Para verificação, o software de análise de tráfego de rede Wireshark foi utilizado para visualizar as mensagens enviadas e recebidas.

Para exemplificar o processo, o funcionamento foi demonstrado com a integração entre o vIED-21L1 e a vMU-21L1, já que o processo é o mesmo para todos os demais dispositivos virtuais da rede.

A comunicação inicia com a vMU enviando mensagens GOOSE e SV para o vIED. A Figura 34 mostra a mensagem GOOSE enviada pela vMU com as configurações previamente estabelecidas. Na Figura 35, observa-se o trecho do algoritmo do vIED responsável por assinar essa mensagem GOOSE.

Figura 34 – Mensagem GOOSE enviada pela vMU



Fonte: Captura de tela do software Wireshark.

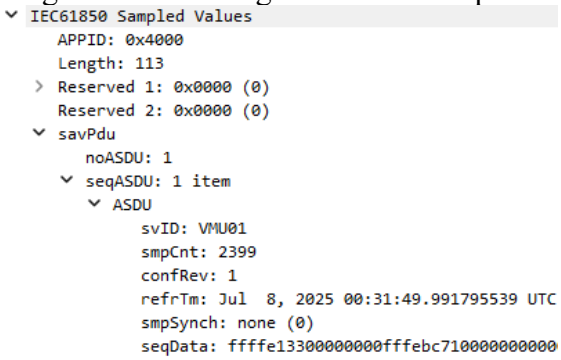
Figura 35 – Assinatura de mensagem GOOSE por parte do vIED

```
//Recepção e Assinatura de mensagens GOOSE
GooseReceiver receiver = GooseReceiver_create();
GooseReceiver_setInterfaceId(receiver, "eth0");
GooseSubscriber subscriber2 = GooseSubscriber_create("MUBinIO_BinaryInputs/LLN0$GO$VMU_01_GOOSE", NULL);
GooseSubscriber_setListener(subscriber2, gooseListener2, iedServer);
GooseReceiver_addSubscriber(receiver, subscriber2);
```

Fonte: Elaborado pelo autor.

A Figura 36 apresenta a mensagem SV, que contém as medições analógicas virtualizadas enviadas pela vMU. Na Figura 37, o trecho do algoritmo do vIED, indicado em vermelho, demonstra o processo necessário para a assinatura dessa mensagem SV.

Figura 36 – Mensagem SV enviada pela vMU



Fonte: Captura de tela do software Wireshark.

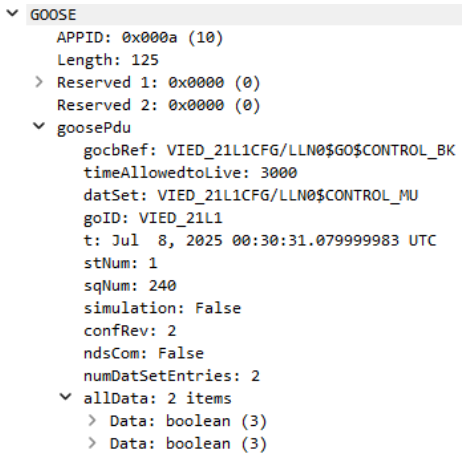
Figura 37 – Assinatura de mensagem SV por parte do vIED

```
/* Callback handler for received SV messages */
static void
svUpdateListener (SVSubscriber subscriber, void* parameter, SVSubscriber_ASDU asdu)
{
    int i;
    const char* svID = SVSubscriber_ASDU_getSvId(asdu);
    if(resposta == true){
        resposta = false;
        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind10_stVal, false);
        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind11_stVal, false);
    }

    if ((strcmp(svID,"VMU01"))== 0){


        SVrms_deltaA = (SVrms_deltaA + pow((SVSubscriber_ASDU_getINT32 (asdu, 0)*0.001),2));
        SVrms_deltaA1 = (SVrms_deltaA1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01),2));
        SVrms_deltaB = (SVrms_deltaB + pow((SVSubscriber_ASDU_getINT32 (asdu, 8)*0.001),2));
        SVrms_deltaB1 = (SVrms_deltaB1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01),2));
        SVrms_deltaC = (SVrms_deltaC + pow((SVSubscriber_ASDU_getINT32 (asdu, 16)*0.001),2));
        SVrms_deltaC1 = (SVrms_deltaC1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01),2));
        SVrms_deltaN = (SVrms_deltaN + pow((SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001),2));
        SVrms_deltaN1 = (SVrms_deltaN1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01),2));
```

Fonte: Elaborado pelo autor.

Em contrapartida, o vIED também envia mensagens GOOSE para a vMU. A Figura 38 exibe uma dessas mensagens, utilizada para o controle da vMU. A Figura 39 detalha o processo no algoritmo da vMU para assinatura da mensagem GOOSE recebida do vIED, completando o ciclo de comunicação bidirecional entre os dispositivos.

Figura 38 – Mensagem GOOSE enviada pelo vIED para controle da vMU



Fonte: Captura de tela do software Wireshark.

Figura 39 – Assinatura de mensagem GOOSE por parte da vMU

```
IedServer_enableGoosePublishing(iedServer);
GooseReceiver receiver = GooseReceiver_create();
GooseReceiver_setInterfaceId(receiver, "eth0");
GooseSubscriber subscriber = GooseSubscriber_create("VIED_21L1CFG/LLN0$GO$CONTROL_BK", NULL);
GooseSubscriber_setListener(subscriber, gooseListener, iedServer);
GooseReceiver_addSubscriber(receiver, subscriber);
GooseReceiver_start(receiver);
```

Fonte: Elaborado pelo autor.

5.4 Testes e Validação do SRAD

A validação do Sistema de Recomposição Automático Distribuído (SRAD) foi realizada em uma rede de distribuição virtual do Campus do Pici, composta por 9 vIEDs e 9 vMUs, conforme o diagrama unifilar apresentado na Figura 2. Para facilitar a visualização do funcionamento sistema, o SRAD foi integrado ao sistema SCADA. O projeto, por ser distribuído não concentra a inteligência da rede em um único ponto, mas a distribui entre os dispositivos que a compõem.

Os testes foram planejados para demonstrar o máximo desempenho do SRAD, com aplicação de faltas estratégicas em trechos específicos. Para isso, foi utilizada a função de proteção de sobrecorrente instantânea (ANSI 50) dos vIEDs, com todas as outras funções desabilitadas. O principal objetivo foi analisar o desempenho do sistema em situações que permitam a recomposição, comparando o resultado teórico esperado com o resultado obtido nos testes. A Tabela 4 apresenta os valores de corrente utilizados para cada trecho, considerando uma demanda de 30%.

Tabela 4 – Valores de corrente para cada trecho com demanda de 30%

Alimentador	Religador	Trecho	S [kVA]	I max [A]	S [kVA] 30%	I [A] 30%
01L1	21L1	T1 + T2	5662,5	236,90	1698,8	71,07
	21L9	T2	2137,5	89,43	641,3	26,83
01L3	21L3	T3 + T4	3862,5	161,60	1158,8	48,48
	21L7	T4	2212,5	92,56	663,8	27,77
01L2	21L2	T5+T6+T7	7325,0	306,46	2197,5	91,94
	21L4	T6+T7	4287,5	179,38	1286,3	53,81
	21L5	T7	2712,5	113,48	813,8	34,04

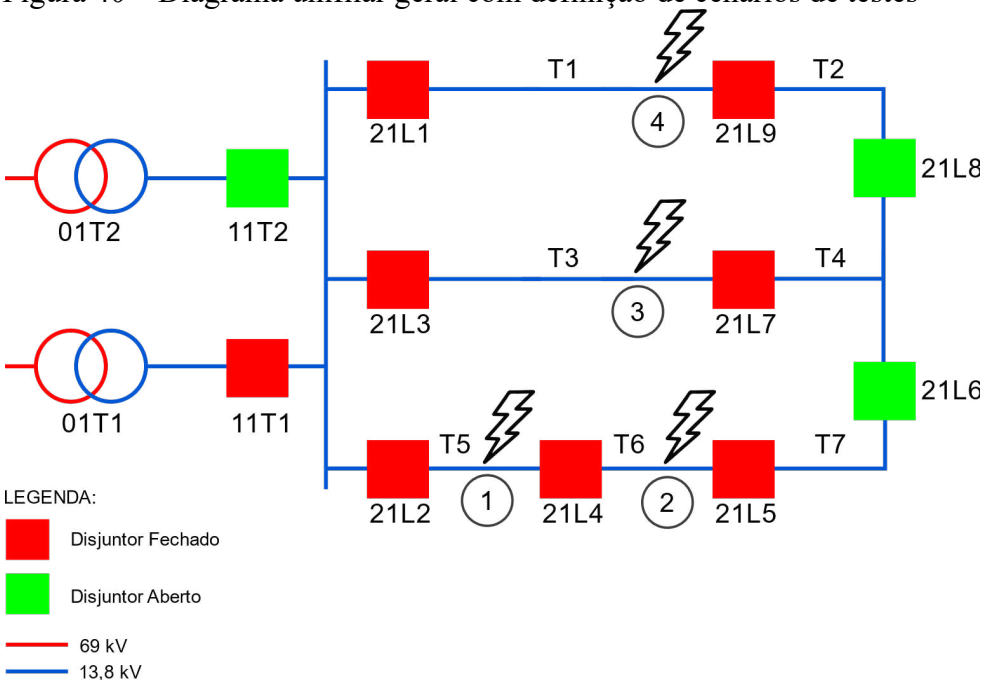
Fonte: Adaptado de (SANTOS, 2015).

5.4.1 Cenários de Testes

A Figura 40 define os cinco cenários de testes distintos na rede de distribuição do Campus do Pici, com o objetivo de validar o funcionamento do SRAD em diferentes contextos. Os cenários de falta de sobrecorrente instantânea foram definidos da seguinte forma:

- 1° Cenário: Falta no trecho 5, entre os disjuntores 21L2 e 21L4;
- 2° Cenário: Falta no trecho 6, entre os disjuntores 21L4 e 21L5;
- 3° Cenário: Falta no trecho 3, entres os disjuntores 21L3 e 21L7;
- 4° Cenário: Falta no trecho 1, entre os disjuntores 21L1 e 21L9.

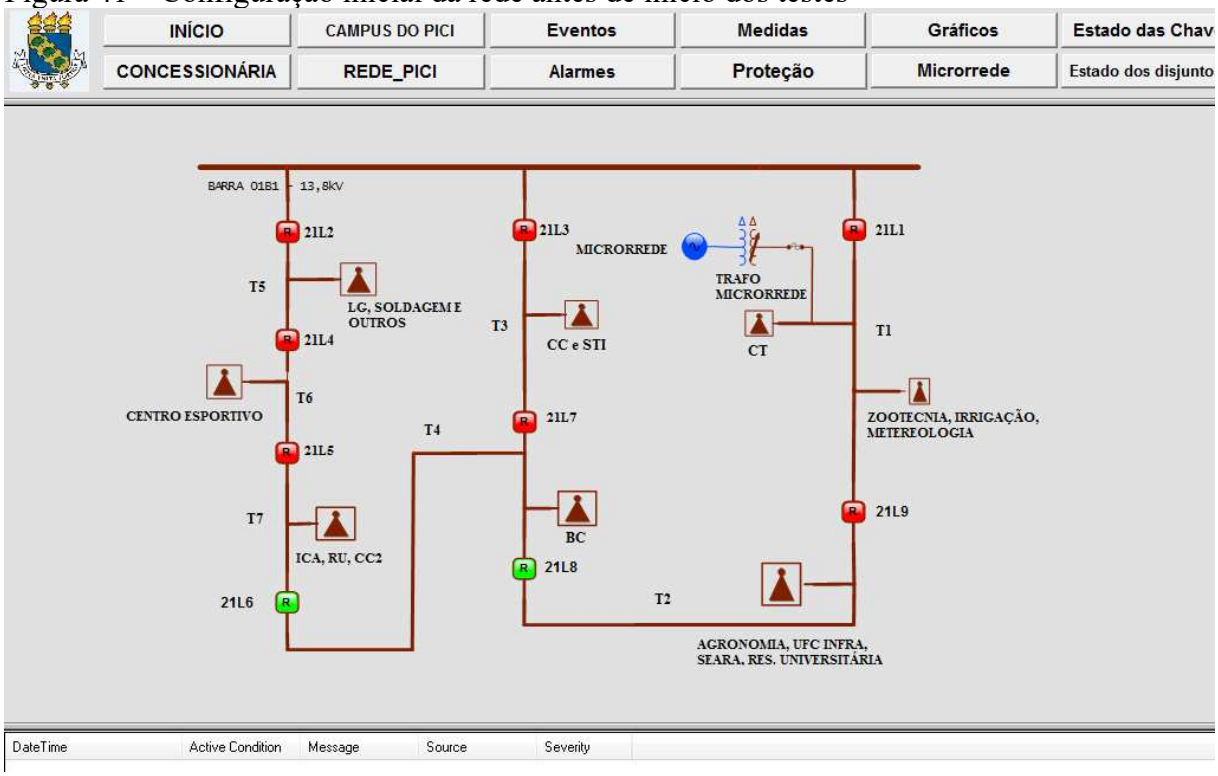
Figura 40 – Diagrama unifilar geral com definição de cenários de testes



Fonte: Elaborado pelo autor.

Antes de cada teste, a plataforma foi reiniciada e configurada conforme a Figura 40. A inicialização do sistema inclui o acionamento de todos os 18 contêineres (9 vIEDs e 9 vMUs) e a verificação da conexão dos vIEDs com o sistema SCADA, como mostrado na Figura 41.

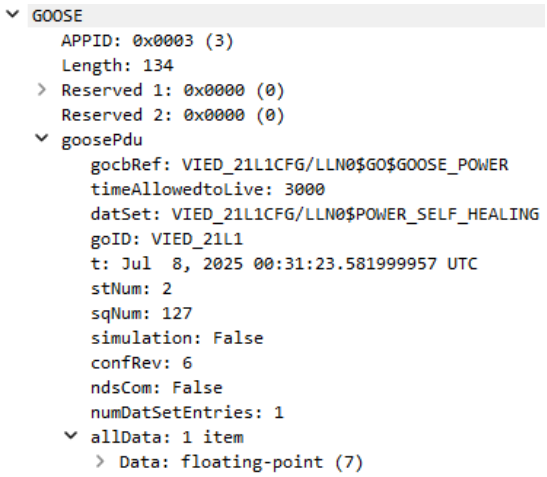
Figura 41 – Configuração inicial da rede antes de início dos testes



Fonte: Captura de tela do software Elipse.

O funcionamento do SRAD é garantido pela comunicação via mensagens GOOSE entre os dispositivos. As mensagens GOOSE_POWER, apresentada na Figura 42, são responsáveis pela transferência radial dos valores de corrente. Já a mensagem GOOSE_STATUS, na Figura 43, é utilizada para a isolação de trechos em falta. Para o controle de múltiplos encontros de alimentadores, a mensagem GOOSE_CONTROL é empregada, como ilustrado na Figura 44.

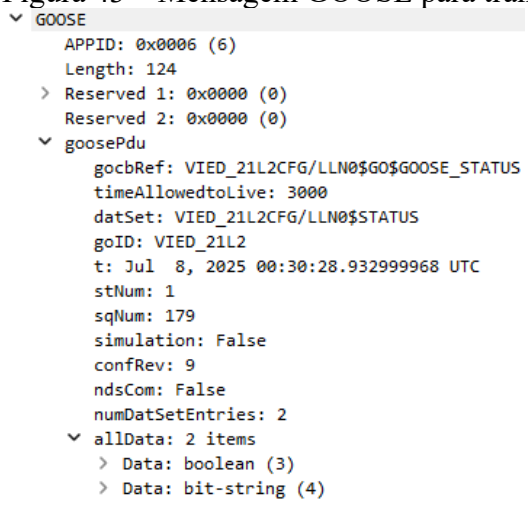
Figura 42 – Mensagem GOOSE para transferência radial de valores de corrente



Fonte: Captura de tela do software Wireshark.

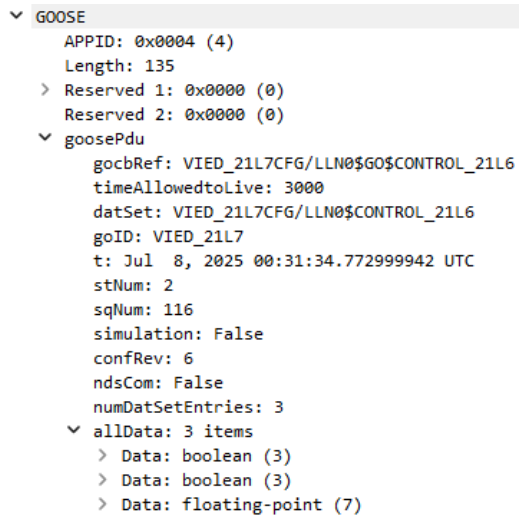
Na Figura 43, está apresentado a mensagem GOOSE_STATUS, responsável pelo processo de isolação de trechos faltosos.

Figura 43 – Mensagem GOOSE para transferência de status



Fonte: Captura de tela do software Wireshark.

Figura 44 – Mensagens GOOSE para controle de outros encontros de alimentadores



Fonte: Captura de tela do software Wireshark.

5.4.1.1 Cenário 1

Neste cenário, uma falta foi simulada no trecho 5, sensibilizando o vIED-21L2. A partir desse evento, o SRAD iniciou o processo de recomposição. A sequência esperada consistia em recompor os trechos T6 e T7, desenergizados pela falta. O algoritmo de recomposição, conforme o cálculo da Figura 45, identificou o encontro de alimentador 21L7 como o melhor caminho para a recomposição, pois exigia menor esforço. O resultado da recomposição no sistema SCADA, mostrado na Figura 46, confirmou o pleno funcionamento do sistema, com a sequência de alarmes registrada na Figura 47 descrevendo de forma cronológica a resposta do SRAD. Para a elaboração do algoritmo, foi considerado um tempo mínimo de 1 segundo para cada atribuição do sistema.

Figura 45 – Cálculo para decisão de melhor caminho para recomposição do cenário 1

$$T1\_T2 := 71.07 \quad [A] \qquad T3\_T4 := 48.48 \quad [A] \qquad T6\_T7 := 53.81 \quad [A]$$

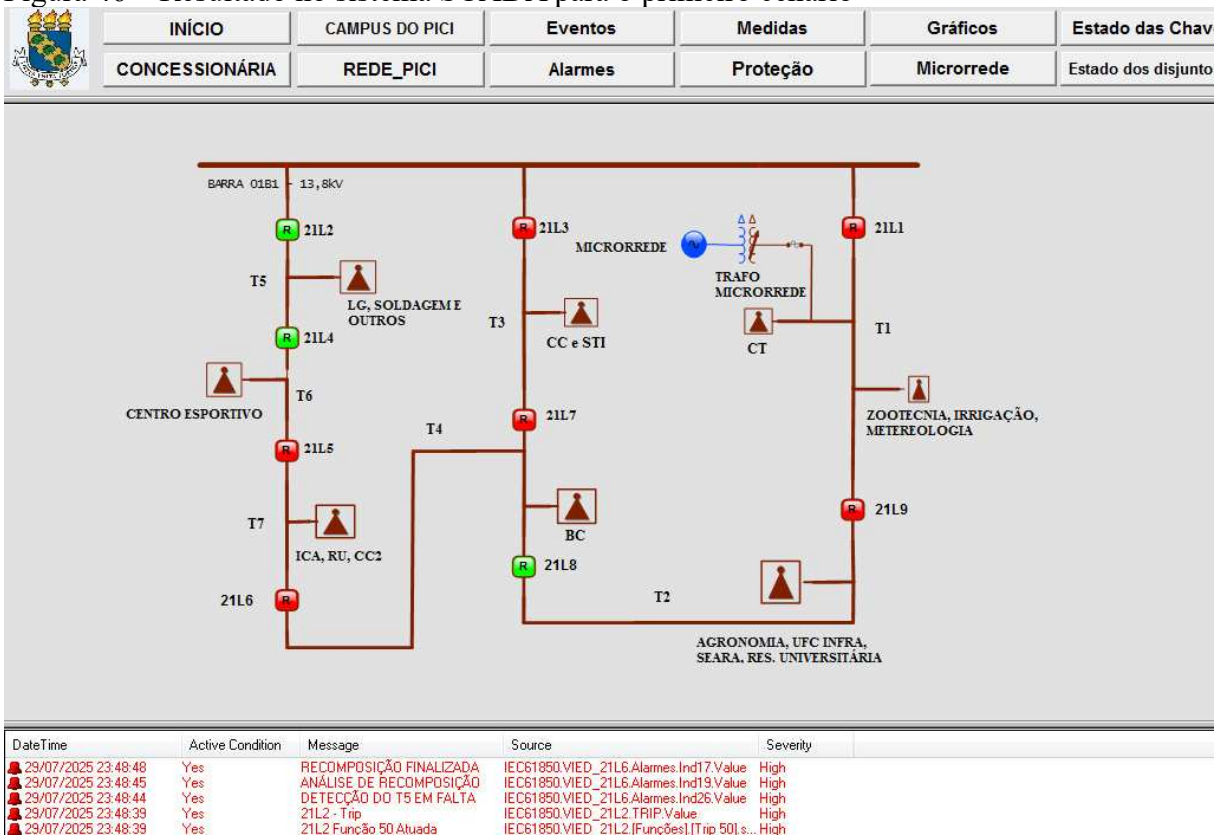
$$Icond := 215 \quad [A]$$

$$R\_21L8 := \frac{(T1\_T2 + T6\_T7)}{Icond} = 0.581$$

$$R\_21L7 := \frac{(T3\_T4 + T6\_T7)}{Icond} = 0.476$$

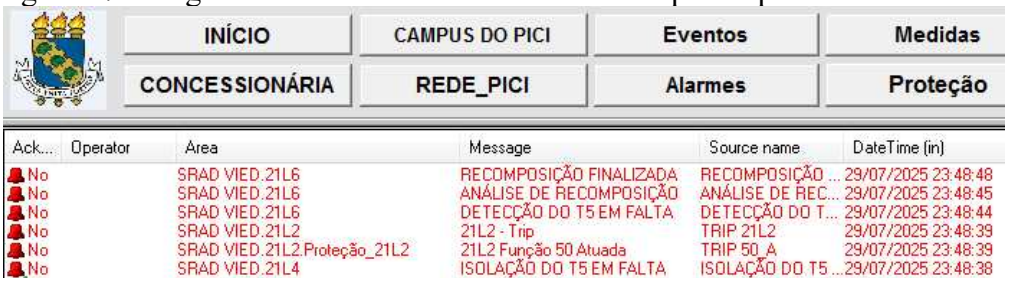
Fonte: Elaborado pelo autor.

Figura 46 – Resultado no sistema SCADA para o primeiro cenário



Fonte: Captura de tela do software Elipse.

Figura 47 – Registro de alarmes no sistema SCADA para o primeiro cenário



Fonte: Captura de tela do software Elipse.

5.4.1.2 Cenário 2

Uma falta foi simulada no trecho 6, ativando a proteção do vIED-21L4. Esperava-se que o sistema iniciasse o processo de recomposição para reenergizar o trecho T7, que foi desenergizado no momento da falta. Conforme o cálculo de decisão de caminho da Figura 48, o encontro de alimentador 21L7 foi novamente o escolhido por exigir menor esforço. O resultado no sistema SCADA, retratado na Figura 49, validou a atuação correta da recomposição. A sequência de alarmes na Figura 50 detalhou a resposta cronológica do sistema a este cenário.

Figura 48 – Cálculo para decisão de melhor caminho para recomposição do cenário 2

$T1\_T2 := 71.07$ [A] $T3\_T4 := 48.48$ [A] $T7 := 34.04$ [A]

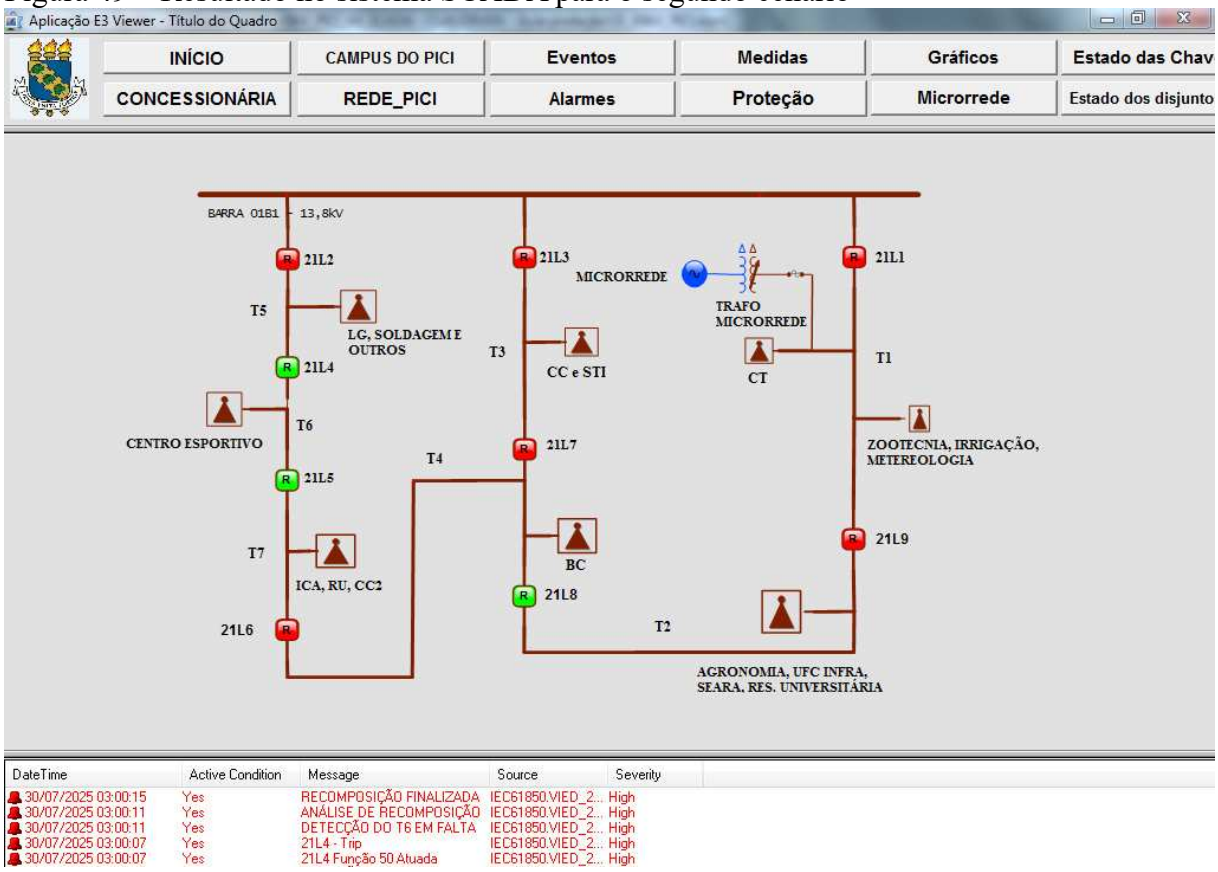
$Icond := 215$ [A]

$$R\_21L8 := \frac{(T1\_T2 + T7)}{Icond} = 0.489$$

$$R\_21L7 := \frac{(T3\_T4 + T7)}{Icond} = 0.384$$

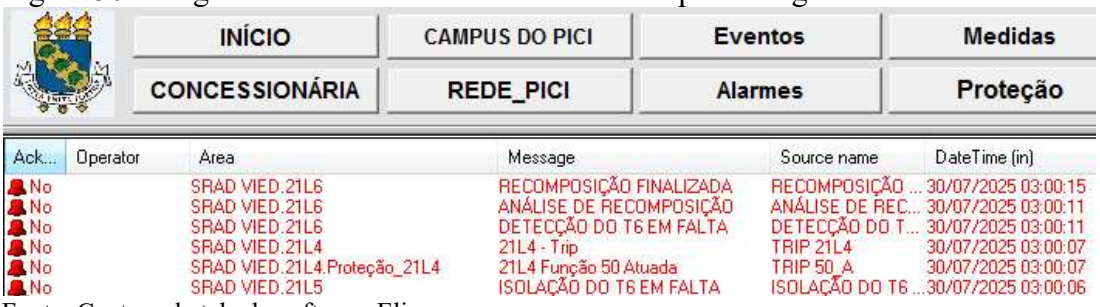
Fonte: Elaborado pelo autor.

Figura 49 – Resultado no sistema SCADA para o segundo cenário



Fonte: Captura de tela do software Elipse.

Figura 50 – Registro de alarmes no sistema SCADA para o segundo cenário



Fonte: Captura de tela do software Elipse.

5.4.1.3 Cenário 3

Neste cenário, a falta foi aplicada no trecho 3, sensibilizando o vIED-21L3. O sistema de recomposição foi então acionado para restaurar o trecho T4, que ficou desenergizado. O cálculo de decisão de caminho, apresentado na Figura 51, indicou que o encontro de alimentador 21L8 era o caminho ideal. O resultado da recomposição no SCADA, exibido na Figura 52, confirmou a atuação correta do SRAD. A resposta cronológica do sistema a este evento foi registrada na sequência de alarmes na Figura 53.

Figura 51 – Cálculo para decisão de melhor caminho para recomposição do cenário 3

$T1\_T2 := 71.07 \quad [A] \qquad T5\_T6\_T7 := 91.94 \quad [A] \qquad T4 := 27.77 \quad [A]$

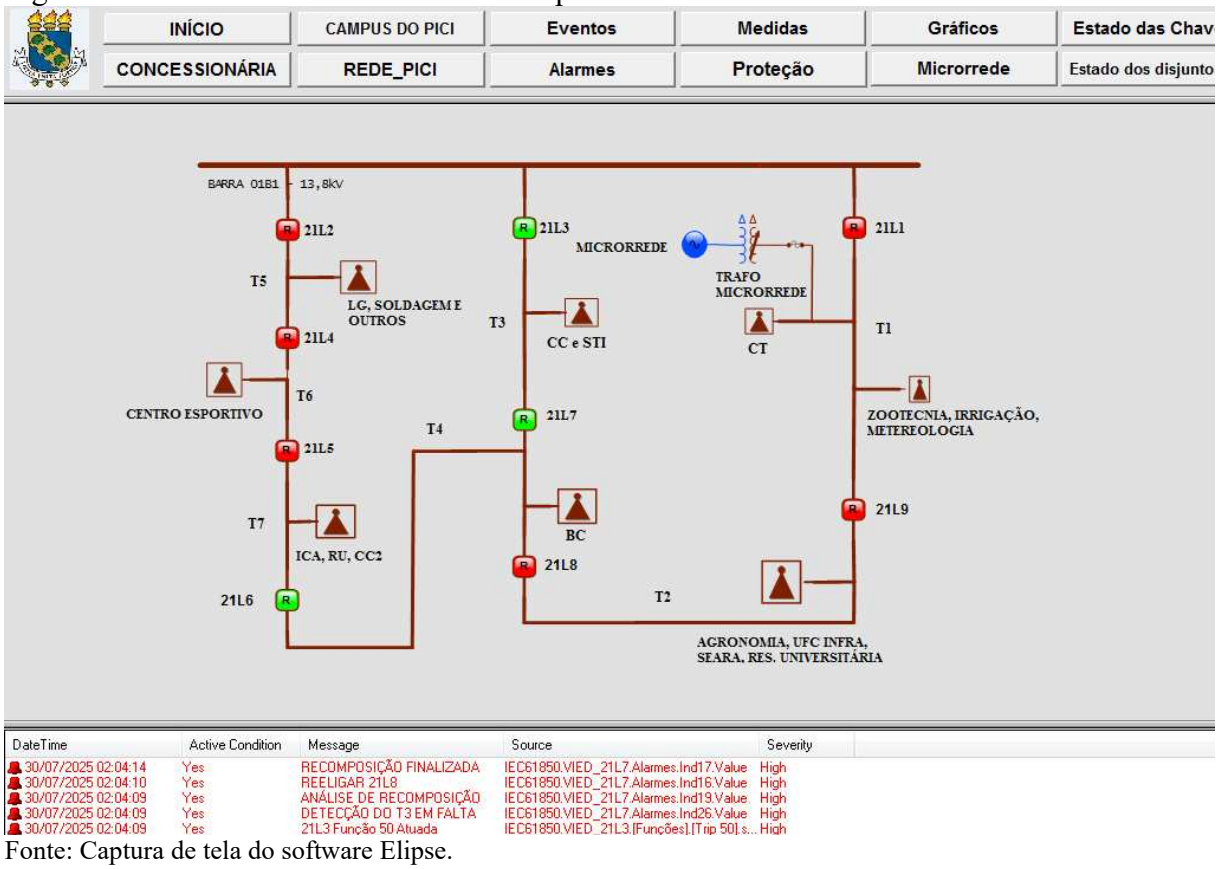
$Icond := 215 \quad [A]$

$$R\_21L6 := \frac{(T5\_T6\_T7 + T4)}{Icond} = 0.557$$

$$R\_21L8 := \frac{(T1\_T2 + T4)}{Icond} = 0.46$$

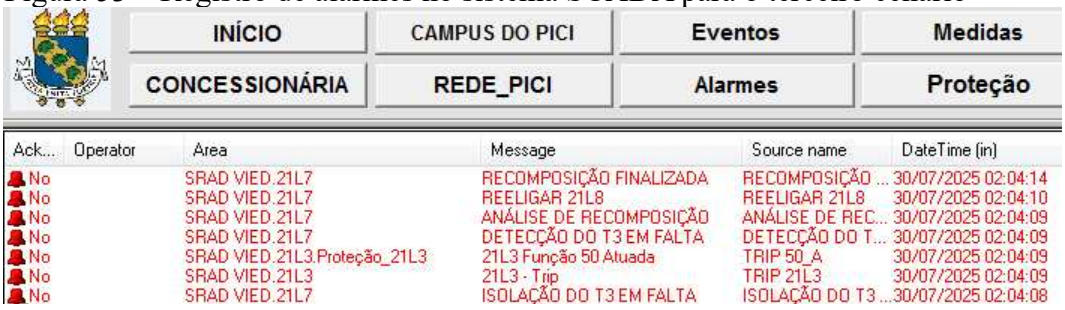
Fonte: Elaborado pelo autor.

Figura 52 – Resultado no sistema SCADA para o terceiro cenário



Fonte: Captura de tela do software Elipse.

Figura 53 – Registro de alarmes no sistema SCADA para o terceiro cenário



Fonte: Captura de tela do software Elipse.

5.4.1.4 Cenário 4

Neste cenário de teste, uma falta foi simulada no trecho 1, sensibilizando o vIED-21L1 da rede de distribuição virtual. Após essa única interferência humana, o SRAD iniciou automaticamente o processo de recomposição.

A sequência esperada visava recompor o trecho T2, desenergizado no momento da falta. Conforme o cálculo de decisão de caminho da Figura 54, o encontro de alimentador 21L7 foi identificado como o melhor caminho, por exigir menor esforço. Embora os dois caminhos fossem válidos, o caminho via 21L7 era o mais eficiente.

O resultado da recomposição, retratado na Figura 55, demonstrou que o sistema atuou corretamente, executando todos os passos de forma adequada, como evidenciado pela sequência de alarmes. No desenvolvimento do algoritmo, foi estabelecido um tempo mínimo de 1 segundo para cada atribuição do SRAD. A Figura 56 apresenta o registro cronológico de alarmes no sistema SCADA, detalhando a resposta do sistema.

Figura 54 – Cálculo para decisão de melhor caminho para recomposição do cenário 4

$T5\_T6\_T7 := 91.94 \quad [A] \qquad T3\_T4 := 48.48 \quad [A] \qquad T2 := 26.83 \quad [A]$

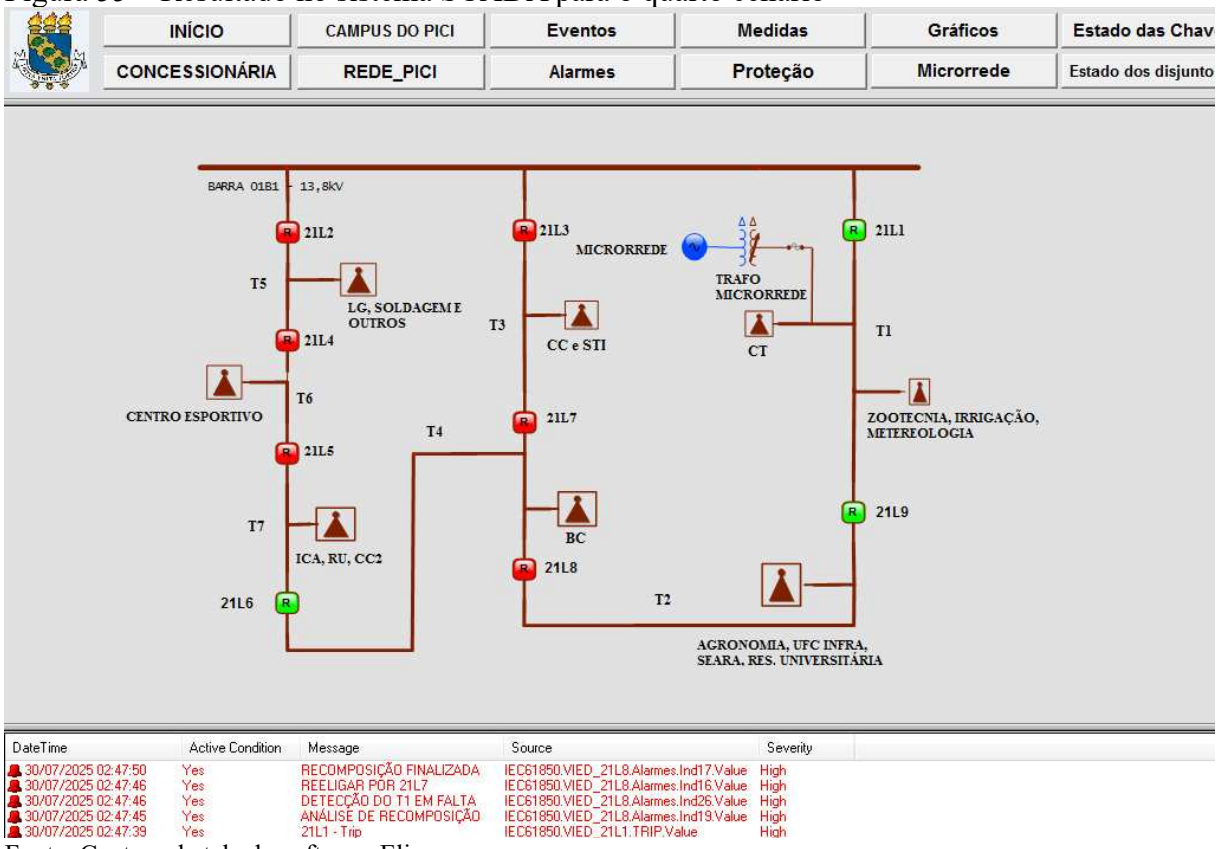
$Icond := 215 \quad [A]$

$$R\_21L6 := \frac{(T5\_T6\_T7 + T2)}{Icond} = 0.552$$

$$R\_21L7 := \frac{(T3\_T4 + T2)}{Icond} = 0.35$$

Fonte: Elaborado pelo autor.

Figura 55 – Resultado no sistema SCADA para o quarto cenário



Fonte: Captura de tela do software Elipse.

Figura 56 – Registro de alarmes no sistema SCADA para o quarto cenário

Fonte: Captura de tela do software Elipse.

5.5 Considerações finais

Nesse capítulo foi apresentada a composição da plataforma de testes, detalhando a comunicação e a integração necessárias para a validação do SRAD. Os testes foram realizados em cenários que abrangeram e validaram o funcionamento máximo do sistema de recomposição.

Foi validada a comunicação GOOSE, que é responsável pela recomposição e a comunicação MMS, que monitora o SRAD, descreve as etapas executadas e registra o tempo.

Os testes elaborados validaram o funcionamento do SRAD para a rede em questão.

Os cenários de testes abordaram os principais casos que podem ocorrer em uma rede dessa magnitude, garantindo a robustez da plataforma utilizada. totalizam plataforma lidou com uma massividade de dados, com um total de 18 dispositivos virtuais conteinerizados, enviando múltiplas mensagens SV, GOOSE e MMS.

6. CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um Sistema de Recomposição Automático Distribuído (SRAD) baseado em vIEDs multifunção de proteção que seguem padrão IEC 61850. A plataforma utilizada para o desenvolvimento é composta de um notebook, um PC industrial, um switch gerenciável IEC 61850 e um GPS com link via satélite. O sistema faz uso de mensagens SV, GOOSE e MMS, com ênfase nas mensagens GOOSE, que são essenciais para a recomposição automática distribuída. Para que SRAD funcionasse, foi necessário aprimorar os dispositivos virtuais que compõem o sistema.

A tecnologia de conteinerização foi um ponto central, permitindo a utilização de vIED e vMUs com baixo consumo de processamento. Graças a essa tecnologia, foi possível simular completamente a rede de distribuição do campus do Pici, utilizando nove vIEDs e nove vMUs. Outra melhoria importante foi a implementação de threads em linguagem de programação C, o que permitiu que as funções internas do vIED compartilhassem o processamento de forma mais eficiente. No entanto, essa abordagem aumenta o uso de processamento do vIED, um fator que deve ser analisado em estudos futuros que envolvam o uso de múltiplas funções.

A implementação da assinatura múltipla de mensagens GOOSE foi o que permitiu o pleno funcionamento do SRAD. O sistema utiliza as tecnologias mencionadas para analisar a rede e executar suas funções. Apesar de a comunicação por mensagens GOOSE ser suficiente para o funcionamento do SRAD, a ausência de uma interface gráfica dificultou a visualização. Por isso, a integração com o sistema SCADA foi realizada. Essa integração permite monitorar todas as etapas do SRAD sem interferir em seu funcionamento, servindo apenas para visualização.

Em suma, o SRAD funcionou corretamente, demonstrando uma ótima eficiência no tempo de resposta, que pode ser aprimorado em futuras versões. O SRAD foi projetado especificamente para a rede de distribuição do Campus do Pici. Se for utilizado em outras redes, deve-se realizar uma reavaliação e fazer as alterações necessárias, embora a base possa ser mantida, variando apenas a quantidade de dispositivos integrados.

6.1 Trabalhos Futuros

Visando dar continuidade a esta linha de pesquisa, os seguintes tópicos são propostos para trabalhos futuros:

- Desenvolver proteção adaptativa com vIED multifunção na rede de distribuição do Campus do Pici. Este seria um trabalho complementar ao SRAD, pois após a recomposição, os parâmetros da rede mudam, exigindo o ajuste automático da proteção;
- Implementação do SRAD em plantas mais complexas, com múltiplos alimentadores e encontros de alimentadores e em redes em anel, para testar o SRAD em situações de maior estresse;
- Desenvolvimento de novas funções de proteção para o vIED multifunção, com as funções 21, 81L, 81B;
- Melhoria no algoritmo existente da vMU, inclusão de sincronismo no envio de cada pacote, com utilização de protocolo PTP ou outros tipos;
- Realização de testes robustos para validação do vIED multifunção com uma MU física, comparando os resultados das múltiplas funções de proteção com os de IEDs físicos.

REFERÊNCIAS

DE FREITAS, Claudivan Domingos *et al.* Testbed for assessing protection, automation, and control functions of digital substations based on the IEC 61850 standard. **Electrical Engineering**, 1 jun. 2025.

DOCKER. **Networking overview**. Disponível em: <https://docs.docker.com/engine/network/>. Acesso em: 18 jul. 2025.

FÉLIX, LUCAS. **PLATAFORMA DE TESTES E VIRTUALIZAÇÃO DE DISPOSITIVOS ELETRÔNICOS INTELIGENTES BASEADOS NA NORMA IEC 61850**. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2019.

FREITAS, Alice. **PLATAFORMA ABERTA PARA ENSINO DE PROTEÇÃO, AUTOMAÇÃO E CONTROLE DE SISTEMAS ELÉTRICOS DE POTÊNCIA COM IEDS FÍSICOS E VIRTUAIS PADRÃO IEC 61850**. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2022a.

FREITAS, Claudivan. **PLATAFORMA DE PROTEÇÃO, AUTOMAÇÃO E CONTROLE DE SISTEMAS ELÉTRICOS COM IEDS FÍSICOS E VIRTUAIS PADRÃO IEC 61850 ORIENTADA À APRENDIZAGEM BASEADA EM PROJETOS**. Dissertação (Mestrado em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2022b.

HAMDEEN, Ibrahim *et al.* **Self-healing multi-agent techniques in electric power distribution systems: A review**. **Renewable and Sustainable Energy Reviews**Elsevier Ltd, , 1 dez. 2025.

IEC 61850-7-3. **Communication networks and systems in substations-Part 7-3: Basic communication structure for substation and feeder equipment-Common data classes**. *[S.l.: S.n.]*. Disponível em: <www.iec.ch>.

IEC 61850-8-1. **Communication networks and systems in substations-Part 8-1:**

Specific Communication Service Mapping (SCSM)-Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3. *[S.l.: S.n.]*. Disponível em: <www.iec.ch/searchpub>.

KABBARA, Nadine *et al.* A Real-Time Implementation and Testing of Virtualized Controllers for Software-Defined IEC 61850 Digital Substations. **IEEE Open Journal of Industry Applications**, 2024.

KERSTING, William H. **Distribution System Modeling and Analysis**. Fourth Edition ed. Boca Raton: CRC Press, 2018.

MARTINEZ, Maria Teresa Villen *et al.* Software-Defined Analog Processing Based on IEC 61850 Implemented in an Edge Hardware Platform to be Used in Digital Substations. **IEEE Access**, v. 12, p. 11549–11560, 2024.

MARTINS, Luís. **VIRTUALIZAÇÃO DE SISTEMAS DE PROTEÇÃO E CONTROLE DE SUBESTAÇÃO COM UTILIZAÇÃO DE MERGING UNIT E IED VIRTUAL INTEGRADO AO SAGE**. Dissertação (Mestrado em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2023.

MELO, LUCAS. **DESENVOLVIMENTO DE UMA PLATAFORMA PARA IMPLANTAÇÃO DE SISTEMAS MULTIAGENTES COM APLICAÇÃO PARA RECOMPOSIÇÃO AUTOMÁTICA DE SISTEMAS DE DISTRIBUIÇÃO DE ENERGIA**. Dissertação (Mestrado em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2015.

SAMPAIO, RAIMUNDO. **SISTEMA DE AUTOMAÇÃO DISTRIBUÍDO: UMA ABORDAGEM BASEADA EM MULTIAGENTE APLICADA A SISTEMAS DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA EM MÉDIA TENSÃO**. Tese (Doutorado em Engenharia Elétrica)—Fortaleza: Universidade Federal do Ceará, 2017.

SANTOS, LUCÉLIA. **INTEGRAÇÃO DE UM SISTEMA DE RECOMPOSIÇÃO AUTOMÁTICO VIA OPC PARA AUTOMAÇÃO DA REDE ELÉTRICA DE DISTRIBUIÇÃO EM MÉDIA TENSÃO DO CAMPUS DO PICI DA UNIVERSIDADE FEDERAL DO CEARÁ**. Dissertação (Mestrado em Engenharia

Elétrica)—Fortaleza: Universidade Federal do Ceará, 2015.

SHORT, T. A. **Electric power distribution handbook**. Boca Raton: CRC Press, 2004.

TANENBAUM, Andrew S.; BOS, Herbert. **MODERN OPERATING SYSTEMS**. 4th ed. ed. Upper Saddle River: Pearson Education, 2015.

VILAPLANA, Jose Angel Leiva *et al.* Virtualized Protection, Automation, and Control in Electrical Substations: An Open-Source Dynamic Cost-Benefit Assessment Model. **IEEE Access**, v. 12, p. 107488–107504, 2024.

ZHAO, Nannan *et al.* Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems. **IEEE Transactions on Parallel and Distributed Systems**, v. 32, n. 4, p. 918–930, 1 abr. 2021.

APÊNDICE A – CÓDIGO FONTE vMU

```
1.  /*
2.   *  iec61850_9_2_LE_example.c
3.   *
4.   *  Copyright 2016 Michael Zillgith
5.   *
6.   *  This file is part of libIEC61850.
7.   *
8.   *  libIEC61850 is free software: you can redistribute it and/or modify
9.   *  it under the terms of the GNU General Public License as published by
10.  *  the Free Software Foundation, either version 3 of the License, or
11.  *  (at your option) any later version.
12.  *
13.  *  libIEC61850 is distributed in the hope that it will be useful,
14.  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
15.  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
16.  *  GNU General Public License for more details.
17.  *
18.  *  You should have received a copy of the GNU General Public License
19.  *  along with libIEC61850.  If not, see <http://www.gnu.org/licenses/>.
20.  *
21.  *  See COPYING file for the complete license text.
22.  */
23.
24. #include "goose_receiver.h"
25. #include "goose_subscriber.h"
26. #include "iec61850_server.h"
27. #include "sv_publisher.h"
28. #include "hal_thread.h"
29. #include <signal.h>
30. #include <stdlib.h>
31. #include <stdio.h>
32.
33. #define _USE_MATH_DEFINES
34. #include <math.h>
35.
36. /* Include the generated header with the model access handles */
37. #include "static_model.h"
38.
39. /* import IEC 61850 device model created from SCL-File */
40. extern IedModel iedModel;
41. static IedServer iedServer = NULL;
42.
43. static int running = 0;
44. static int svcbEnabled = 1;
45.
46. void sigint_handler(int signalId)
47. {
48.     running = 0;
49. }
50.
51. static int amp1;
52. static int amp2;
53. static int amp3;
54. static int amp4;
55.
56. static int amp1q;
```

```
57.  static int amp2q;
58.  static int amp3q;
59.  static int amp4q;
60.
61.  static int vol1;
62.  static int vol2;
63.  static int vol3;
64.  static int vol4;
65.
66.  static int vol1q;
67.  static int vol2q;
68.  static int vol3q;
69.  static int vol4q;
70.
71.  static SVPublisher svPublisher;
72.  static SVPublisher_ASDU asdu;
73.
74.  static float vol10,vol20,vol30;
75.  static float amp10,amp20,amp30;
76.  static float an[6];
77.
78.  static void
79.  setupSVPublisher(const char* svInterface)
80.  {
81.      svPublisher = SVPublisher_create(NULL, svInterface);
82.
83.      if (svPublisher) {
84.
85.          asdu = SVPublisher_addASDU(svPublisher, "VMU01", NULL, 1);
86.
87.          amp1 = SVPublisher_ASDU_addINT32(asdu);
88.          amp1q = SVPublisher_ASDU_addQuality(asdu);
89.          amp2 = SVPublisher_ASDU_addINT32(asdu);
90.          amp2q = SVPublisher_ASDU_addQuality(asdu);
91.          amp3 = SVPublisher_ASDU_addINT32(asdu);
92.          amp3q = SVPublisher_ASDU_addQuality(asdu);
93.          amp4 = SVPublisher_ASDU_addINT32(asdu);
94.          amp4q = SVPublisher_ASDU_addQuality(asdu);
95.
96.          vol1 = SVPublisher_ASDU_addINT32(asdu);
97.          vol1q = SVPublisher_ASDU_addQuality(asdu);
98.          vol2 = SVPublisher_ASDU_addINT32(asdu);
99.          vol2q = SVPublisher_ASDU_addQuality(asdu);
100.         vol3 = SVPublisher_ASDU_addINT32(asdu);
101.         vol3q = SVPublisher_ASDU_addQuality(asdu);
102.         vol4 = SVPublisher_ASDU_addINT32(asdu);
103.         vol4q = SVPublisher_ASDU_addQuality(asdu);
104.
105.         SVPublisher_ASDU_setSmpCntWrap(asdu, 4800);
106.         SVPublisher_ASDU_setRefrTm(asdu, 0);
107.
108.         SVPublisher_setupComplete(svPublisher);
109.     }
110. }
111.
112. static void sVCBEventHandler (SVControlBlock* svcb, int event, void* parameter)
113. {
114.     if (event == IEC61850_SVCB_EVENT_ENABLE)
115.         svcbEnabled = 1;
116.     else if (event == IEC61850_SVCB_EVENT_DISABLE)
117.         svcbEnabled = 0;
118. }
119.
120. static void
121. goCbEventHandler(MmsGooseControlBlock goCb, int event, void* parameter)
122. {
123.     printf("Access to GoCB: %s\n", MmsGooseControlBlock_getName(goCb));
```

```
124.        printf("            GoEna: %i\n", MmsGooseControlBlock_getGoEna(goCb));
125.    }
126.
127.    static void
128.    gooseListener(GooseSubscriber subscriber, void* parameter)
129.    {
130.        MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
131.
132.        char buffer[50];
133.
134.        MmsValue_printToBuffer(values, buffer, 50);
135.
136.        char b; char c; char d;
137.
138.        b = buffer[1];
139.        c = buffer[7];
140.        d = buffer[11];
141.        uint64_t y = Hal_getTimeInMs();
142.
143.        //printf("\n%d\n%d\n", b, c);
144.
145.        if(b == 116){
146.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_BinIO_BinaryInputs_LPDI3_In_stVal, true);
147.            FILE *file;
148.            file = fopen("AjustesMU.txt","r");
149.            fscanf(file,"%f\n",&amp10);
150.            fscanf(file,"%f\n",&an[0]);
151.            fscanf(file,"%f\n",&amp20);
152.            fscanf(file,"%f\n",&an[1]);
153.            fscanf(file,"%f\n",&amp30);
154.            fscanf(file,"%f\n",&an[2]);
155.            fscanf(file,"%f\n",&vol10);
156.            fscanf(file,"%f\n",&an[3]);
157.            fscanf(file,"%f\n",&vol20);
158.            fscanf(file,"%f\n",&an[4]);
159.            fscanf(file,"%f\n",&vol30);
160.            fscanf(file,"%f\n",&an[5]);
161.            fclose(file);
162.        }
163.
164.        if(c == 116){
165.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_BinIO_BinaryInputs_LPDI3_In_stVal, false);
166.            amp10 = 0;
167.            amp20 = 0;
168.            amp30 = 0;
169.            vol10 = 0;
170.            vol20 = 0;
171.            vol30 = 0;
172.            an[0] = 0;
173.            an[1] = 0;
174.            an[2] = 0;
175.            an[3] = 0;
176.            an[4] = 0;
177.            an[5] = 0;
178.        }
179.
180.        /*printf("-----------------------------------------------------------------------------------------------\n");
181.        printf("                                PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 1                                \n");
182.        printf("-----------------------------------------------------------------------------------------------\n");*/
183.
184.    }
185.
186.    int
187.    main(int argc, char** argv)
188.    {
189.        char* svInterface;
190.
```

```
191.        IedServerConfig config = IedServerConfig_create();
192.
193.        iedServer = IedServer_createWithConfig(&iedModel, NULL, config);
194.
195.        IedServerConfig_destroy(config);
196.
197.        //IedServer iedServer = IedServer_create(&iedModel);
198.
199.        if (argc > 1){
200.            svInterface = argv[1];
201.            char* ethernetIfcID = argv[1];
202.            printf("Using GOOSE interface: %s\n", ethernetIfcID);
203.            IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);
204.        }
205.
206.        if (argc > 2){
207.            svInterface = "eth0";
208.            char* ethernetIfcID = argv[2];
209.            printf("Using GOOSE interface for GenericIO/LLN0.gcbAnalogValues: %s\n", ethernetIfcID);
210.            //IedServer_setGooseInterfaceIdEx(iedServer, IEDMODEL_CFG_LLN0, "BRep0201", ethernetIfcID);
211.        }
212.
213.        IedServer_enableGoosePublishing(iedServer);
214.        GooseReceiver receiver = GooseReceiver_create();
215.        GooseReceiver_setInterfaceId(receiver, "eth0");
216.        GooseSubscriber subscriber = GooseSubscriber_create("VIED_21L1CFG/LLN0$GO$CONTROL_BK", NULL); //Especificação de quem o ied irá receber as
    mensagens goose
217.        GooseSubscriber_setListener(subscriber, gooseListener, iedServer);
218.        GooseReceiver_addSubscriber(receiver, subscriber);
219.        GooseReceiver_start(receiver);
220.
221.        IedServer_setGoCBHandler(iedServer, goCbEventHandler, NULL);
222.
223.        /* MMS server will be instructed to start listening to client connections. */
224.        IedServer_start(iedServer, 120);
225.
226.        if (!IedServer_isRunning(iedServer)) {
227.            printf("Starting server failed! Exit.\n");
228.            IedServer_destroy(iedServer);
229.            exit(-1);
230.        }
231.
232.        running = 1;
233.
234.        signal(SIGINT, sigint_handler);
235.
236.        setupSVPublisher(svInterface);
237.
238.        if (svPublisher) {
239.
240.            IedServer_enableGoosePublishing(iedServer);
241.
242.            SVControlBlock* svcb = IedModel_getSVControlBlock(&iedModel, IEDMODEL_Mod3_MU2_LLN0, "MSVCB01");
243.
244.            if (svcb == NULL) {
245.                printf("Lookup svcb failed!\n");
246.                exit(1);
247.            }
248.
249.            IedServer_setSVCBHandler(iedServer, svcb, sVCBEventHandler, NULL);
250.
251.            Quality q = QUALITY_VALIDITY_GOOD;
252.
253.            //int vol = (int) (6350.f * sqrt(2));
254.            //int amp = (int) (350.f);
255.            static float phaseAngle = 0.f;
256.
```

```
257.        FILE *file;
258.        file = fopen("AjustesMU.txt","r");
259.        fscanf(file,"%f\n",&amp10);
260.        fscanf(file,"%f\n",&an[0]);
261.        fscanf(file,"%f\n",&amp20);
262.        fscanf(file,"%f\n",&an[1]);
263.        fscanf(file,"%f\n",&amp30);
264.        fscanf(file,"%f\n",&an[2]);
265.        fscanf(file,"%f\n",&vol10);
266.        fscanf(file,"%f\n",&an[3]);
267.        fscanf(file,"%f\n",&vol20);
268.        fscanf(file,"%f\n",&an[4]);
269.        fscanf(file,"%f\n",&vol30);
270.        fscanf(file,"%f\n",&an[5]);
271.        fclose(file);
272.
273.        int voltageA;
274.        int voltageB;
275.        int voltageC;
276.        int voltageN;
277.        int currentA;
278.        int currentB;
279.        int currentC;
280.        int currentN;
281.
282.        int sampleCount = 0;
283.
284.        uint64_t nextCycleStart = Hal_getTimeInMs() + 1000;
285.
286.        while (running) {
287.
288.            int samplePoint = sampleCount % 80;
289.
290.            /*double angleA = (2 * M_PI / 80) * samplePoint;
291.            double angleB = (2 * M_PI / 80) * samplePoint - ( 2 * M_PI / 3);
292.            double angleC = (2 * M_PI / 80) * samplePoint - ( 4 * M_PI / 3);*/
293.
294.            double angleA1 = (2 * M_PI / 80) * samplePoint + (an[3] * M_PI / 180);
295.            double angleB1 = (2 * M_PI / 80) * samplePoint + (an[4] * M_PI / 180);
296.            double angleC1 = (2 * M_PI / 80) * samplePoint + (an[5] * M_PI / 180);
297.
298.            double angleA2 = (2 * M_PI / 80) * samplePoint + (an[0] * M_PI / 180);
299.            double angleB2 = (2 * M_PI / 80) * samplePoint + (an[1] * M_PI / 180);
300.            double angleC2 = (2 * M_PI / 80) * samplePoint + (an[2] * M_PI / 180);
301.
302.            voltageA = (vol10 * sqrt(2) * sin(angleA1)) * 100;
303.            voltageB = (vol20 * sqrt(2) * sin(angleB1)) * 100;
304.            voltageC = (vol30 * sqrt(2) * sin(angleC1)) * 100;
305.            voltageN = voltageA + voltageB + voltageC;
306.
307.            currentA = (amp10 * sqrt(2) * sin(angleA2 - phaseAngle)) * 1000;
308.            currentB = (amp20 * sqrt(2) * sin(angleB2 - phaseAngle)) * 1000;
309.            currentC = (amp30 * sqrt(2) * sin(angleC2 - phaseAngle)) * 1000;
310.            currentN = currentA + currentB + currentC;
311.
312.            IedServer_lockDataModel(iedServer);
313.
314.            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_I01ATCTR1_AmpSv_instMag_i, currentA);
315.            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_I01ATCTR1_AmpSv_q, q);
316.            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_I01BTCTR2_AmpSv_instMag_i, currentA);
317.            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_I01BTCTR2_AmpSv_q, q);
318.            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_I01CTCTR3_AmpSv_instMag_i, currentA);
319.            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_I01CTCTR3_AmpSv_q, q);
320.            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_I01NTCTR4_AmpSv_instMag_i, currentA);
321.            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_I01NTCTR4_AmpSv_q, q);
322.
323.            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_U01ATVTR1_VolSv_instMag_i, voltageA);
```

```
            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_U01ATVTR1_VolSv_q, q);
            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_U01BTVTR2_VolSv_instMag_i, voltageB);
            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_U01BTVTR2_VolSv_q, q);
            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_U01CTVTR3_VolSv_instMag_i, voltageC);
            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_U01CTVTR3_VolSv_q, q);
            IedServer_updateInt32AttributeValue(iedServer, IEDMODEL_Mod3_MU2_U01NTVTR4_VolSv_instMag_i, voltageN);
            IedServer_updateQuality(iedServer, IEDMODEL_Mod3_MU2_U01NTVTR4_VolSv_q, q);

            IedServer_unlockDataModel(iedServer);

            if (svcbEnabled) {

                SVPublisher_ASDU_setINT32(asdu, amp1, currentA);
                SVPublisher_ASDU_setQuality(asdu, amp1q, q);
                SVPublisher_ASDU_setINT32(asdu, amp2, currentB);
                SVPublisher_ASDU_setQuality(asdu, amp2q, q);
                SVPublisher_ASDU_setINT32(asdu, amp3, currentC);
                SVPublisher_ASDU_setQuality(asdu, amp3q, q);
                SVPublisher_ASDU_setINT32(asdu, amp4, currentN);
                SVPublisher_ASDU_setQuality(asdu, amp4q, q);

                SVPublisher_ASDU_setINT32(asdu, vol1, voltageA);
                SVPublisher_ASDU_setQuality(asdu, vol1q, q);
                SVPublisher_ASDU_setINT32(asdu, vol2, voltageB);
                SVPublisher_ASDU_setQuality(asdu, vol2q, q);
                SVPublisher_ASDU_setINT32(asdu, vol3, voltageC);
                SVPublisher_ASDU_setQuality(asdu, vol3q, q);
                SVPublisher_ASDU_setINT32(asdu, vol4, voltageN);
                SVPublisher_ASDU_setQuality(asdu, vol4q, q);

                SVPublisher_ASDU_setRefrTmNs(asdu, Hal_getTimeInNs());

                SVPublisher_ASDU_setSmpCnt(asdu, (uint16_t) sampleCount);

                SVPublisher_publish(svPublisher);
            }

            sampleCount = ((sampleCount + 1) % 4800);

            if ((sampleCount % 480) == 0) {
                uint64_t timeval = Hal_getTimeInMs();

                while (timeval < nextCycleStart + 100) {
                    Thread_sleep(1);

                    timeval = Hal_getTimeInMs();
                }

                nextCycleStart = nextCycleStart + 100;//teste
            }
        }
    }
    else {
        printf("Cannot start SV publisher!\n");
    }

    /* stop MMS server - close TCP server socket and all client sockets */
    IedServer_stop(iedServer);

    /* Cleanup - free all resources */
    SVPublisher_destroy(svPublisher);
    IedServer_destroy(iedServer);

    return 0;
} /* main() */

```

APÊNDICE B – CÓDIGO FONTE vIED MULTIFUNÇÃO DE PROTEÇÃO COM FUNÇÃO SRAD

```
/*
 *  server_example_goose.c
 *
 *  This example demonstrates how to use GOOSE publishing, Reporting and the
 *  control model.
 *
 */

#include "iec61850_client.h"
#include "goose_receiver.h"
#include "goose_subscriber.h"
#include "iec61850_server.h"
#include "hal_thread.h" /* for Thread_sleep() */
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mms_value.h"
#include "goose_publisher.h"
#include "sv_subscriber.h"
#include <math.h>
#include "static_model.h"
#include <sys/timeb.h>
#include <sys/time.h>
#include <complex.h>
#define MICRO_PER_SECOND 1000000
#define pi 3.14

Thread thread_50, thread_50N, thread_51, thread_51V, thread_51N, thread_67, thread_67N;
Thread self_healing;


/* import IEC 61850 device model created from SCL-File */
extern IedModel iedModel;

bool contador = true;
bool contador10 = true;
bool contador9 = true;
bool contador11 = true;
bool contador7 = true;
bool contador8 = true;
bool contador1 = true;
bool contador2 = true;
bool contador3 = true;
bool contador4 = true;
bool contador5 = true;
bool contador6 = true;
bool contador12 = true;
bool contador13 = true;
bool resposta = true;
bool comando = false;
bool maximo1 = true;
bool maximo2 = true;
int curva = 0;
int funcao = 0;

char* svID2;
```

```
58.  static int running = 0;
59.  static IedServer iedServer = NULL;
60.  int contadorSV1 = 0;
61.  int contadorSV2 = 0;
62.  int contadorSV3 = 0;
63.  float j = 0;
64.  float SVrms_deltaA = 0;
65.  float SVrms_deltaB = 0;
66.  float SVrms_deltaC = 0;
67.  float SVrms_deltaN = 0;
68.  float SVrms_deltaA1 = 0;
69.  float SVrms_deltaB1 = 0;
70.  float SVrms_deltaC1 = 0;
71.  float SVrms_deltaN1 = 0;
72.  float max_corrente_a = 0;
73.  float max_corrente_b = 0;
74.  float max_corrente_c = 0;
75.  float max_corrente_n = 0;
76.  float max_tensao_a = 0;
77.  float max_tensao_b = 0;
78.  float max_tensao_c = 0;
79.  float max_tensao_n = 0;
80.  static float corrente_primarioA = 0;
81.  static float corrente_primarioB = 0;
82.  static float corrente_primarioC = 0;
83.  static float corrente_primarioN = 0;
84.  static float tensao_primarioA = 0;
85.  static float tensao_primarioB = 0;
86.  static float tensao_primarioC = 0;
87.  static float tensao_primarioN = 0;
88.  float teste[80];
89.  static float K_51, K_51N;
90.  static float K_51V;
91.  static float K_67, K_67N;
92.  static float alfa_51, alfa_51N;
93.  static float alfa_51V;
94.  static float alfa_67, alfa_67N;
95.  static float M1_51A, M2_51B, M3_51C, a, t, t1, t2, B = 1;
96.  static float M1_51V_A, M2_51V_B, M3_51V_C;
97.  static float M1_67A, M2_67B, M3_67C;
98.  static float M1_50N, M1_51N, M1_67N;
99.  static float act_51A, act_51B, act_51C, act_51N;
100. static float act_51V_A, act_51V_B, act_51V_C;
101. static float act_67A, act_67B, act_67C, act_67N;
102. static bool enable_51A = true;
103. static bool enable_51B = true;
104. static bool enable_51C = true;
105. static bool enable_51V_A = true;
106. static bool enable_51V_B = true;
107. static bool enable_51V_C = true;
108. static bool enable_51N = true;
109. static bool enable_67A = true;
110. static bool enable_67B = true;
111. static bool enable_67C = true;
112. static bool enable_67N = true;
113. struct timeval start_51A, start_51B, start_51C, start_51N;
114. struct timeval stop_51A, stop_51B, stop_51C, stop_51N;
115. struct timeval start_51V_A, start_51V_B, start_51V_C, start_51V_N;
116. struct timeval stop_51V_A, stop_51V_B, stop_51V_C, stop_51V_N;
117. struct timeval start_67A, start_67B, start_67C, start_67N;
118. struct timeval stop_67A, stop_67B, stop_67C, stop_67N;
119. struct timeval start_50_62BF;
120. struct timeval stop_50_62BF;
121. static float ang1, ang2, ang3, ang4, ang5, ang6, ang7, ang8;
122. static float time_51A, time_51B, time_51C, time_51N, time_50_62BF;
123. static float time_51V_A, time_51V_B, time_51V_C, time_51V_N;
124. static float time_67A, time_67B, time_67C, time_67N;
```

```
125.  static float ime_diff = 0, ime_diff1 = 0, ime_diff2 = 0;
126.  double an[8], ar[6], br[6], teta, teta1, teta2, torque, torque1, torque2, tetaN, torqueN;
127.  double complex yr[6], Vbc, aVbc, mVbc,Vca, aVca, mVca, Vab, aVab, mVab;
128.
129.  static int funcoes, curva_51, curva_51V, curva_51N, curva_67, curva_67N;
130.  static float pick_up_50, pick_up_50N, pick_up_51, pick_up_51V, pick_up_51N, pick_up_67, pick_up_67N, atm_67, atm_67N;
131.  static float dial_51, dial_51V, dial_51N, dial_67, dial_67N, tensao_51V;
132.
133.  static float a, b, c, d, e, f, g, h, i;
134.  static float a1, b1, c1, d1, e1, f1, g1, h1, i1;
135.  static float a2, b2, c2, d2, e2, f2, g2, h2, i2;
136.  static float pMax_21l4, pMax_21l5, pMax_21l7;
137.
138.  static float pMaxS21l1 = 215, pMaxS21l2 = 215, pMaxS21l3 = 215;
139.
140.  static char sh, trip_21l5, trip_21l7, trip_21l8;
141.
142.  static int estado_dj_21l5, estado_dj_21l7, estado_dj_21l8;
143.
144.  static char comando_received_21l7_A, comando_received_21l8_A, comando_received_21l7_F, comando_received_21l8_F;
145.
146.  struct timeval delay1, delay2, delay3, delay4, delay5;
147.  struct timeval delay6, delay7, delay8, delay9, delay10;
148.  struct timeval delay1p, delay2p, delay3p, delay4p, delay5p;
149.  struct timeval delay6p, delay7p, delay8p, delay9p, delay10p;
150.  static float time_delay1, time_delay2, time_delay3, time_delay4, time_delay5;
151.  static float time_delay6, time_delay7, time_delay8, time_delay9, time_delay10;
152.
153.  bool delay_1 = true;
154.  bool delay_2 = true;
155.  bool delay_3 = true;
156.  bool delay_4 = true;
157.  bool delay_5 = true;
158.  bool delay_6 = true;
159.  bool delay_7 = true;
160.  bool delay_8 = true;
161.  bool delay_9 = true;
162.  bool delay_10 = true;
163.
164.  void sigint_handler(int signalId)
165.  {
166.          running = 0;
167.  }
168.
169.  void funcao_50_62BF()
170.  {
171.      if (contador9 == true)
172.      {
173.          gettimeofday(&start_50_62BF, NULL);
174.          contador9 = false;
175.      }
176.      gettimeofday(&stop_50_62BF, NULL);
177.      time_50_62BF = (float)(stop_50_62BF.tv_sec - start_50_62BF.tv_sec);
178.      time_50_62BF += (stop_50_62BF.tv_usec - start_50_62BF.tv_usec) / (float)MICRO_PER_SECOND;
179.      if ((time_50_62BF) >= 0.250)
180.      {
181.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_BFR1RBRF1_OpEx_general, true);
182.      }
183.  }
184.
185.  void funcao_50()
186.  {
187.      while (1)
188.      {
189.
190.          if (corrente_primarioA > pick_up_50)
191.          {
```

```
192.                printf("-----------------------------------------------------------------------------------------------------------\n");
193.                printf("                                                ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE A
    \n");
194.                printf("-----------------------------------------------------------------------------------------------------------\n");
195.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind04_stVal, true);
196.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
197.            }
198.            if (corrente_primarioB > pick_up_50)
199.            {
200.                printf("-----------------------------------------------------------------------------------------------------------\n");
201.                printf("                                                ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE B
    \n");
202.                printf("-----------------------------------------------------------------------------------------------------------\n");
203.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind05_stVal, true);
204.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
205.            }
206.            if (corrente_primarioC > pick_up_50)
207.            {
208.                printf("-----------------------------------------------------------------------------------------------------------\n");
209.                printf("                                                ATUAR FUNÇÃO 50: SOBRECORRENTE INSTANTÂNEA FASE C
    \n");
210.                printf("-----------------------------------------------------------------------------------------------------------\n");
211.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind06_stVal, true);
212.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
213.            }
214.            if (((corrente_primarioA) || (corrente_primarioB) || (corrente_primarioC)) > pick_up_50)
215.            {
216.                funcao_50_62BF();
217.            }
218.            Thread_sleep(17);
219.        }
220.    }
221.
222.    void funcao_50N()
223.    {
224.        while(1){
225.        if ((corrente_primarioN) >= pick_up_50N)
226.        {
227.            printf("-----------------------------------------------------------------------------------------------------------\n");
228.            printf("                                                ATUAR FUNÇÃO 50N: SOBRECORRENTE TEMPORIZADA DE NEUTRO
    \n");
229.            printf("-----------------------------------------------------------------------------------------------------------\n");
230.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind14_stVal, true);
231.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
232.            funcao_50_62BF();
233.        }
234.        Thread_sleep(17);
235.        }
236.    }
237.
238.    void funcao_51()
239.    {
240.        while(1){
241.        M1_51A = corrente_primarioA / pick_up_51;
242.        act_51A = (dial_51 * (K_51 / ((pow(M1_51A, alfa_51)) - B)));
243.
244.        M2_51B = corrente_primarioB / pick_up_51;
245.        act_51B = (dial_51 * (K_51 / ((pow(M2_51B, alfa_51)) - B)));
246.
247.        M3_51C = corrente_primarioC / pick_up_51;
248.        act_51C = (dial_51 * (K_51 / ((pow(M3_51C, alfa_51)) - B)));
249.
250.        if (act_51A > 0)
251.        {
252.            if (enable_51A == true)
253.            {
254.                gettimeofday(&start_51A, NULL);
```

```
255.                enable_51A = false;
256.            }
257.            gettimeofday(&stop_51A, NULL);
258.            time_51A = (float)(stop_51A.tv_sec - start_51A.tv_sec);
259.            time_51A += (stop_51A.tv_usec - start_51A.tv_usec) / (float)MICRO_PER_SECOND;
260.            if ((time_51A) >= act_51A)
261.            {
262.                if (funcao == 1)
263.                {
264.                    // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_ATPTOC20_Op_general, true);
265.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind20_stVal, true);
266.                    printf("-----------------------------------------------------------------------------------------------------\n");
267.                    printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE A
    \n");
268.                    printf("-----------------------------------------------------------------------------------------------------\n");
269.                }
270.                else
271.                {
272.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind01_stVal, true);
273.                    printf("-----------------------------------------------------------------------------------------------------\n");
274.                    printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE A
    \n");
275.                    printf("-----------------------------------------------------------------------------------------------------\n");
276.                }
277.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
278.            }
279.        }
280.        if (act_51B > 0)
281.        {
282.            if (enable_51B == true)
283.            {
284.                gettimeofday(&start_51B, NULL);
285.                enable_51B = false;
286.            }
287.            gettimeofday(&stop_51B, NULL);
288.            time_51B = (float)(stop_51B.tv_sec - start_51B.tv_sec);
289.            time_51B += (stop_51B.tv_usec - start_51B.tv_usec) / (float)MICRO_PER_SECOND;
290.            if ((time_51B) >= act_51B)
291.            {
292.                if (funcao == 1)
293.                {
294.                    // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_BTPTOC21_Op_general, true);
295.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind21_stVal, true);
296.                    printf("-----------------------------------------------------------------------------------------------------\n");
297.                    printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE B
    \n");
298.                    printf("-----------------------------------------------------------------------------------------------------\n");
299.                }
300.                else
301.                {
302.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind02_stVal, true);
303.                    printf("-----------------------------------------------------------------------------------------------------\n");
304.                    printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE B
    \n");
305.                    printf("-----------------------------------------------------------------------------------------------------\n");
306.                }
307.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
308.            }
309.        }
310.        if (act_51C > 0)
311.        {
312.            if (enable_51C == true)
313.            {
314.                gettimeofday(&start_51C, NULL);
315.                enable_51C = false;
316.            }
317.            gettimeofday(&stop_51C, NULL);
```

```
318.            time_51C = (float)(stop_51C.tv_sec - start_51C.tv_sec);
319.            time_51C += (stop_51C.tv_usec - start_51C.tv_usec) / (float)MICRO_PER_SECOND;
320.            if ((time_51C) >= act_51C)
321.            {
322.                if (funcao == 1)
323.                {
324.                    // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_CTPTOC22_Op_general, true);
325.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind22_stVal, true);
326.                    printf("--------------------------------------------------------------------------------------------------------\n");
327.                    printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE C
    \n");
328.                    printf("--------------------------------------------------------------------------------------------------------\n");
329.                }
330.                else
331.                {
332.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind03_stVal, true);
333.                    printf("--------------------------------------------------------------------------------------------------------\n");
334.                    printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE C
    \n");
335.                    printf("--------------------------------------------------------------------------------------------------------\n");
336.                }
337.                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
338.            }
339.        }Thread_sleep(17);}
340.    }
341.
342.    void funcao_51V()
343.    {
344.        while(1){
345.        j = tensao_primarioA / tensao_51V;
346.
347.        M1_51V_A = corrente_primarioA / (j * pick_up_51V);
348.        act_51V_A = (dial_51V * (K_51V / ((pow(M1_51V_A, alfa_51V)) - B)));
349.
350.        M2_51V_B = corrente_primarioB / (j * pick_up_51V);
351.        act_51V_B = (dial_51V * (K_51V / ((pow(M2_51V_B, alfa_51V)) - B)));
352.
353.        M3_51V_C = corrente_primarioC / (j * pick_up_51V);
354.        act_51V_C = (dial_51V * (K_51V / ((pow(M3_51V_C, alfa_51V)) - B)));
355.
356.        if (act_51V_A > 0)
357.        {
358.            if (enable_51V_A == true)
359.            {
360.                gettimeofday(&start_51V_A, NULL);
361.                enable_51V_A = false;
362.            }
363.            gettimeofday(&stop_51V_A, NULL);
364.            time_51V_A = (float)(stop_51V_A.tv_sec - start_51V_A.tv_sec);
365.            time_51V_A += (stop_51V_A.tv_usec - start_51V_A.tv_usec) / (float)MICRO_PER_SECOND;
366.            if ((time_51V_A) >= act_51V_A)
367.            {
368.                if (funcao == 1)
369.                {
370.                    // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_ATPTOC20_Op_general, true);
371.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind20_stVal, true);
372.                    printf("--------------------------------------------------------------------------------------------------------\n");
373.                    printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE A
    \n");
374.                    printf("--------------------------------------------------------------------------------------------------------\n");
375.                }
376.                else
377.                {
378.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind01_stVal, true);
379.                    printf("--------------------------------------------------------------------------------------------------------\n");
380.                    printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE A
    \n");
```

```
381.                         printf("-----------------------------------------------------------------------------------------------------\n");
382.                     }
383.                     IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
384.                 }
385.             }
386.             if (act_51V_B > 0)
387.             {
388.                 if (enable_51V_B == true)
389.                 {
390.                     gettimeofday(&start_51V_B, NULL);
391.                     enable_51V_B = false;
392.                 }
393.                 gettimeofday(&stop_51V_B, NULL);
394.                 time_51V_B = (float)(stop_51V_B.tv_sec - start_51V_B.tv_sec);
395.                 time_51V_B += (stop_51V_B.tv_usec - start_51V_B.tv_usec) / (float)MICRO_PER_SECOND;
396.                 if ((time_51V_B) >= act_51V_B)
397.                 {
398.                     if (funcao == 1)
399.                     {
400.                         // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_BTPTOC21_Op_general, true);
401.                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind21_stVal, true);
402.                         printf("-----------------------------------------------------------------------------------------------------\n");
403.                         printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE B
    \n");
404.                         printf("-----------------------------------------------------------------------------------------------------\n");
405.                     }
406.                     else
407.                     {
408.                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind02_stVal, true);
409.                         printf("-----------------------------------------------------------------------------------------------------\n");
410.                         printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE B
    \n");
411.                         printf("-----------------------------------------------------------------------------------------------------\n");
412.                     }
413.                     IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
414.                 }
415.             }
416.             if (act_51V_C > 0)
417.             {
418.                 if (enable_51V_C == true)
419.                 {
420.                     gettimeofday(&start_51V_C, NULL);
421.                     enable_51V_C = false;
422.                 }
423.                 gettimeofday(&stop_51V_C, NULL);
424.                 time_51V_C = (float)(stop_51V_C.tv_sec - start_51V_C.tv_sec);
425.                 time_51V_C += (stop_51V_C.tv_usec - start_51V_C.tv_usec) / (float)MICRO_PER_SECOND;
426.                 if ((time_51V_C) >= act_51V_C)
427.                 {
428.                     if (funcao == 1)
429.                     {
430.                         // IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_CTPTOC22_Op_general, true);
431.                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind22_stVal, true);
432.                         printf("-----------------------------------------------------------------------------------------------------\n");
433.                         printf("                                                    ATUAR FUNÇÃO 51: SOBRECORRENTE TEMPORIZADA FASE C
    \n");
434.                         printf("-----------------------------------------------------------------------------------------------------\n");
435.                     }
436.                     else
437.                     {
438.                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind03_stVal, true);
439.                         printf("-----------------------------------------------------------------------------------------------------\n");
440.                         printf("                              ATUAR FUNÇÃO 51V: SOBRECORRENTE TEMPORIZADA COM RESTRIÇÃO DE TENSÃO FASE C
    \n");
441.                         printf("-----------------------------------------------------------------------------------------------------\n");
442.                     }
443.                     IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
```

```
444.                }
445.          }Thread_sleep(17);}
446.    }
447.
448.    void funcao_51N()
449.    {
450.          while(1){
451.          M1_51N = corrente_primarioN / (pick_up_51N);
452.          t = (dial_51N * (K_51N / ((pow(M1_51N, a)) - B)));
453.
454.          if (act_51N > 0)
455.          {
456.                if (enable_51N == true)
457.                {
458.                      gettimeofday(&start_51N, NULL);
459.                      enable_51N = false;
460.                }
461.                gettimeofday(&stop_51N, NULL);
462.                time_51N = (float)(stop_51N.tv_sec - start_51N.tv_sec);
463.                time_51N += (stop_51N.tv_usec - start_51N.tv_usec) / (float)MICRO_PER_SECOND;
464.                if ((time_51N) >= act_51N)
465.                {
466.                      printf("----------------------------------------------------------------------------------------------------\n");
467.                      printf("                                                ATUAR FUNÇÃO 51N: SOBRECORRENTE TEMPORIZADA DE NEUTRO
      \n");
468.                      printf("----------------------------------------------------------------------------------------------------\n");
469.                      IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind13_stVal, true);
470.                      IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
471.                }
472.          }Thread_sleep(17);}
473.    }
474.
475.    void funcao_67()
476.    {
477.          while(1){
478.          ar[0] = max_tensao_a * cos(an[0]);
479.          br[0] = max_tensao_a * sin(an[0]);
480.          ar[1] = max_tensao_b * cos(an[1]);
481.          br[1] = max_tensao_b * sin(an[1]);
482.          ar[2] = max_tensao_c * cos(an[2]);
483.          br[2] = max_tensao_c * sin(an[2]);
484.          ar[3] = max_corrente_a * cos(an[3]);
485.          br[3] = max_corrente_a * sin(an[3]);
486.          ar[4] = max_corrente_b * cos(an[4]);
487.          br[4] = max_corrente_b * sin(an[4]);
488.          ar[5] = max_corrente_c * cos(an[5]);
489.          br[5] = max_corrente_c * sin(an[5]);
490.          yr[0] = ar[0] + br[0] * I;
491.          yr[1] = ar[1] + br[1] * I;
492.          yr[2] = ar[2] + br[2] * I;
493.          yr[3] = ar[3] + br[3] * I;
494.          yr[4] = ar[4] + br[4] * I;
495.          yr[5] = ar[5] + br[5] * I;
496.
497.          Vbc = yr[1] - yr[2];
498.          Vca = yr[2] - yr[0];
499.          Vab = yr[0] - yr[1];
500.          aVbc = atan(cimag(Vbc) / creal(Vbc));
501.          aVca = atan(cimag(Vca) / creal(Vca));
502.          aVab = atan(cimag(Vab) / creal(Vab));
503.          mVbc = sqrt(pow(creal(Vbc), 2) + pow(cimag(Vbc), 2));
504.          mVca = sqrt(pow(creal(Vca), 2) + pow(cimag(Vca), 2));
505.          mVab = sqrt(pow(creal(Vab), 2) + pow(cimag(Vab), 2));
506.          teta = an[3] - aVbc;
507.          teta1 = an[4] - aVca;
508.          teta2 = an[5] - aVab;
509.          torque = max_corrente_a * mVbc * cos((atm_67 - teta) * M_PI / 180);
```

```
510.        torque1 = max_corrente_b * mVca * cos((atm_67 - teta1) * M_PI / 180);
511.        torque2 = max_corrente_c * mVab * cos((atm_67 - teta2) * M_PI / 180);
512.
513.        if (torque > 0)
514.        {
515.            M1_67A = corrente_primarioA / (pick_up_67N);
516.            act_67A = (dial_67 * (K_67 / ((pow(M1_67A, alfa_67)) - B)));
517.            if (act_67A > 0)
518.            {
519.                if (enable_67A == true)
520.                {
521.                    gettimeofday(&start_67A, NULL);
522.                    enable_67A = false;
523.                }
524.                gettimeofday(&stop_67A, NULL);
525.                time_67A = (float)(stop_67A.tv_sec - start_67A.tv_sec);
526.                time_67A += (stop_67A.tv_usec - start_67A.tv_usec) / (float)MICRO_PER_SECOND;
527.                if ((time_67A) >= act_67A)
528.                {
529.                    printf("----------------------------------------------------------------------------------------------------\n");
530.                    printf("                                                ATUAR FUNÇÃO 67: SOBRECORRENTE DIRECIONAL TEMPORIZADA FASE A
    \n");
531.                    printf("----------------------------------------------------------------------------------------------------\n");
532.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind07_stVal, true);
533.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
534.                }
535.            }
536.        }
537.
538.        if (torque1 > 0)
539.        {
540.            M2_67B = corrente_primarioB / (pick_up_67);
541.            act_67B = (dial_67 * (K_67 / ((pow(M2_67B, alfa_67)) - B)));
542.            if (act_67B > 0)
543.            {
544.                if (enable_67B == true)
545.                {
546.                    gettimeofday(&start_67B, NULL);
547.                    enable_67B = false;
548.                }
549.                gettimeofday(&stop_67B, NULL);
550.                time_67B = (float)(stop_67B.tv_sec - start_67B.tv_sec);
551.                time_67B += (stop_67B.tv_usec - start_67B.tv_usec) / (float)MICRO_PER_SECOND;
552.                if ((time_67B) >= act_67B)
553.                {
554.                    printf("----------------------------------------------------------------------------------------------------\n");
555.                    printf("                                                ATUAR FUNÇÃO 67: SOBRECORRENTE DIRECIONAL TEMPORIZADA FASE B
    \n");
556.                    printf("----------------------------------------------------------------------------------------------------\n");
557.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind08_stVal, true);
558.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
559.                }
560.            }
561.        }
562.
563.        if (torque2 > 0)
564.        {
565.            M3_67C = corrente_primarioC / (pick_up_67);
566.            act_67C = (dial_67 * (K_67 / ((pow(M3_67C, alfa_67)) - B)));
567.            if (act_67C > 0)
568.            {
569.                if (enable_67C == true)
570.                {
571.                    gettimeofday(&start_67C, NULL);
572.                    enable_67C = false;
573.                }
574.                gettimeofday(&stop_67C, NULL);
```

```
575.                    time_67C = (float)(stop_67C.tv_sec - start_67C.tv_sec);
576.                    time_67C += (stop_67C.tv_usec - start_67C.tv_usec) / (float)MICRO_PER_SECOND;
577.                    if ((time_67C) >= act_67C)
578.                    {
579.                        printf("-----------------------------------------------------------------------------------------------------------\n");
580.                        printf("                                                ATUAR FUNÇÃO 67: SOBRECORRENTE DIRECIONAL TEMPORIZADA FASE C
    \n");
581.                        printf("-----------------------------------------------------------------------------------------------------------\n");
582.                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind09_stVal, true);
583.                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
584.                    }
585.                }
586.            }Thread_sleep(17);}
587.    }
588.
589.    void funcao_67N()
590.    {
591.        while(1){
592.        if (tensao_primarioN > (tensao_51V / 10))
593.        {
594.            tetaN = an[7] - (an[6]);
595.            torqueN = max_tensao_n * max_corrente_n * cos((atm_67N - tetaN) * pi / 180);
596.        }
597.
598.        if (torqueN > 0)
599.        {
600.            M1_67N = corrente_primarioN / (pick_up_67N);
601.            act_67N = (dial_67N * (K_67N / ((pow(M1_67N, alfa_67N)) - B)));
602.            if (act_67N > 0)
603.            {
604.                if (enable_67N == true)
605.                {
606.                    gettimeofday(&start_67N, NULL);
607.                    enable_67N = false;
608.                }
609.                gettimeofday(&stop_67N, NULL);
610.                time_67N = (float)(stop_67N.tv_sec - start_67N.tv_sec);
611.                time_67N += (stop_67N.tv_usec - start_67N.tv_usec) / (float)MICRO_PER_SECOND;
612.                if ((time_67N) >= act_67N)
613.                {
614.                    printf("-----------------------------------------------------------------------------------------------------------\n");
615.                    printf("                                            ATUAR FUNÇÃO 67N: DIRECIONAL DE SOBRECORRENTE TEMPORIZADA DE NEUTRO
    \n");
616.                    printf("-----------------------------------------------------------------------------------------------------------\n");
617.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind12_stVal, true);
618.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_TRIPPTRC1_Tr_general, true);
619.                }
620.            }
621.        }Thread_sleep(17);}
622.    }
623.
624.
625.    /* Callback handler for received SV messages */
626.    static void
627.    svUpdateListener (SVSubscriber subscriber, void* parameter, SVSubscriber_ASDU asdu)
628.    {
629.        int i;
630.        const char* svID = SVSubscriber_ASDU_getSvId(asdu);
631.        if(resposta == true){
632.            resposta = false;
633.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind10_stVal, false);
634.            IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind11_stVal, false);
635.        }
636.
637.        if ((strcmp(svID,"VMU06"))== 0){
638.
639.
```

```
640.            SVrms_deltaA = (SVrms_deltaA + pow((SVSubscriber_ASDU_getINT32 (asdu, 0)*0.001),2));
641.            SVrms_deltaA1 = (SVrms_deltaA1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01),2));
642.            SVrms_deltaB = (SVrms_deltaB + pow((SVSubscriber_ASDU_getINT32 (asdu, 8)*0.001),2));
643.            SVrms_deltaB1 = (SVrms_deltaB1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01),2));
644.            SVrms_deltaC = (SVrms_deltaC + pow((SVSubscriber_ASDU_getINT32 (asdu, 16)*0.001),2));
645.            SVrms_deltaC1 = (SVrms_deltaC1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01),2));
646.            SVrms_deltaN = (SVrms_deltaN + pow((SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001),2));
647.            SVrms_deltaN1 = (SVrms_deltaN1 + pow((SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01),2));
648.
649.
650.            if((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01)>=0){
651.                teste[contadorSV1] = (SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01);
652.            }
653.
654.            if ((contador1 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01)>=0)){
655.                max_tensao_a = (SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01);
656.                contador1 = false;
657.            }
658.            if((contador2 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01)>=0)){
659.                max_tensao_b = (SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01);
660.                contador2 = false;
661.            }
662.
663.            if((contador3 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01)>=0)){
664.                max_tensao_c = (SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01);
665.                contador3 = false;
666.            }
667.            if ((contador4 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 0)*0.01)>=0)){
668.                max_corrente_a = (SVSubscriber_ASDU_getINT32 (asdu, 0)*0.01);
669.                contador4 = false;
670.            }
671.            if((contador5 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 8)*0.01)>=0)){
672.                max_corrente_b = (SVSubscriber_ASDU_getINT32 (asdu, 8)*0.01);
673.                contador5 = false;
674.            }
675.
676.            if((contador6 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 16)*0.01)>=0)){
677.                max_corrente_c = (SVSubscriber_ASDU_getINT32 (asdu, 16)*0.01);
678.                contador6 = false;
679.            }
680.
681.            if((contador12 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01)>=0)){
682.                max_tensao_n = (SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01);
683.                contador12 = false;
684.            }
685.
686.            if((contador13 == true)&&((SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001)>=0)){
687.                max_corrente_n = (SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001);
688.                contador13 = false;
689.            }
690.
691.            if(((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01)>=0)){
692.                if(teste[contadorSV1]>teste[contadorSV1-1]){
693.                    if((SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01)>max_tensao_a){
694.                        max_tensao_a = (SVSubscriber_ASDU_getINT32 (asdu, 32)*0.01);
695.                        ang1 = SVSubscriber_ASDU_getSmpCnt(asdu);
696.                    }
697.                }
698.            }
699.
700.            if(((SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01)>=0)){
701.                    if((SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01)>max_tensao_b){
702.                        max_tensao_b = (SVSubscriber_ASDU_getINT32 (asdu, 40)*0.01);
703.                        ang2 = SVSubscriber_ASDU_getSmpCnt(asdu);
704.                    }
705.            }
706.
```

```
707.            if(((SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01)>=0)){
708.                    if((SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01)>max_tensao_c){
709.                        max_tensao_c = (SVSubscriber_ASDU_getINT32 (asdu, 48)*0.01);
710.                        ang3 = SVSubscriber_ASDU_getSmpCnt(asdu);
711.                    }
712.            }
713.            if(((SVSubscriber_ASDU_getINT32 (asdu, 0)*0.01)>=0)){
714.                    if((SVSubscriber_ASDU_getINT32 (asdu, 0)*0.01)>max_corrente_a){
715.                        max_corrente_a = (SVSubscriber_ASDU_getINT32 (asdu, 0)*0.01);
716.                        ang4 = SVSubscriber_ASDU_getSmpCnt(asdu);
717.                    }
718.            }
719.
720.            if(((SVSubscriber_ASDU_getINT32 (asdu, 8)*0.01)>=0)){
721.                    if((SVSubscriber_ASDU_getINT32 (asdu, 8)*0.01)>max_corrente_b){
722.                        max_corrente_b = (SVSubscriber_ASDU_getINT32 (asdu, 8)*0.01);
723.                        ang5 = SVSubscriber_ASDU_getSmpCnt(asdu);
724.                    }
725.            }
726.
727.            if(((SVSubscriber_ASDU_getINT32 (asdu, 16)*0.01)>=0)){
728.                    if((SVSubscriber_ASDU_getINT32 (asdu, 16)*0.01)>max_corrente_c){
729.                        max_corrente_c = (SVSubscriber_ASDU_getINT32 (asdu, 16)*0.01);
730.                        ang6 = SVSubscriber_ASDU_getSmpCnt(asdu);
731.                    }
732.            }
733.
734.            if(((SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01)>=0)){
735.                    if((SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01)>max_tensao_n){
736.                        max_tensao_n = (SVSubscriber_ASDU_getINT32 (asdu, 56)*0.01);
737.                        ang8 = SVSubscriber_ASDU_getSmpCnt(asdu);
738.                    }
739.            }
740.
741.            if(((SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001)>=0)){
742.                    if((SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001)>max_corrente_n){
743.                        max_corrente_n = (SVSubscriber_ASDU_getINT32 (asdu, 24)*0.001);
744.                        ang7 = SVSubscriber_ASDU_getSmpCnt(asdu);
745.                    }
746.            }
747.            contadorSV1 += 1;
748.        }
749.
750.        if (contadorSV1==80)
751.        {
752.            corrente_primarioA = sqrt(SVrms_deltaA / 80);
753.            corrente_primarioB = sqrt(SVrms_deltaB / 80);
754.            corrente_primarioC = sqrt(SVrms_deltaC / 80);
755.            tensao_primarioA = sqrt(SVrms_deltaA1 / 80);
756.            tensao_primarioB = sqrt(SVrms_deltaB1 / 80);
757.            tensao_primarioC = sqrt(SVrms_deltaC1 / 80);
758.
759.
760.            an[1] = (ang1 - ang2)*4.5;
761.            an[2] = (ang1 - ang3)*4.5;
762.            an[3] = (ang1 - ang4)*4.5;
763.            an[4] = (ang1 - ang5)*4.5;
764.            an[5] = (ang1 - ang6)*4.5;
765.
766.            if(an[1]<0){
767.                an[1] = 360 + an[1];
768.            }if(an[2]<0){
769.                an[2] = 360 + an[2];
770.            }if(an[3]<0){
771.                an[3] = 360 + an[3];
772.            }if(an[4]<0){
773.                an[4] = 360 + an[4];
```

```
774.            }if(an[5]<0){
775.                an[5] = 360 + an[5];
776.            }
777.
778.            /*funcao_50();
779.            funcao_50N();
780.            funcao_51();
781.            funcao_51V();
782.            funcao_51N();
783.            funcao_67();
784.            funcao_67N();*/
785.
786.            contadorSV3 ++;
787.
788.            if(contadorSV3 == 60){
789.                system ("clear");
790.                printf("    svID=(%s)\n", svID);
791.                printf("    smpCnt: %i\n", SVSubscriber_ASDU_getSmpCnt(asdu));
792.                printf("-------------------------------------------------------\n");
793.                printf("     A tensão RMS da fase A no primário é: %.2f [V] %.1f°\n", tensao_primarioA, an[0] );
794.                printf("     A tensão RMS da fase B no primário é: %.2f [V] %.2f°\n", tensao_primarioB, an[1] );
795.                printf("     A tensão RMS da fase C no primário é: %.2f [V] %.1f°\n", tensao_primarioC, an[2] );
796.                printf("-------------------------------------------------------\n");
797.                printf("     A corrente RMS da fase A no primário é: %.2f [A] %.1f°\n", corrente_primarioA, an[3] );
798.                printf("     A corrente RMS da fase B no primário é: %.2f [A] %.1f°\n", corrente_primarioB, an[4] );
799.                printf("     A corrente RMS da fase C no primário é: %.2f [A] %.1f°\n", corrente_primarioC, an[5] );
800.                printf("-------------------------------------------------------\n");
801.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsA_cVal_mag_f, corrente_primarioA);
802.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsB_cVal_mag_f, corrente_primarioB);
803.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsC_cVal_mag_f, corrente_primarioC);
804.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsAB_cVal_mag_f, tensao_primarioA);
805.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsBC_cVal_mag_f, tensao_primarioB);
806.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsCA_cVal_mag_f, tensao_primarioC);
807.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsA_cVal_ang_f, an[3]);
808.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsB_cVal_ang_f, an[4]);
809.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_A_phsC_cVal_ang_f, an[5]);
810.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsAB_cVal_ang_f, an[0]);
811.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsBC_cVal_ang_f, an[1]);
812.                IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsCA_cVal_ang_f, an[2]);
813.                contadorSV3 = 0;
814.            }
815.
816.
817.            contadorSV1=0;
818.            SVrms_deltaA=0;
819.            SVrms_deltaB=0;
820.            SVrms_deltaC=0;
821.            SVrms_deltaA1=0;
822.            SVrms_deltaB1=0;
823.            SVrms_deltaC1=0;
824.            contador1 = true;
825.            contador2 = true;
826.            contador3 = true;
827.            contador4 = true;
828.            contador5 = true;
829.            contador6 = true;
830.        }
831.    }
832.
833.    void
834.    controlHandlerForBinaryOutput(ControlAction action, void* parameter, MmsValue* value, bool test)
835.    {
836.
837.        if (MmsValue_getType(value) == MMS_BOOLEAN) {
838.            printf("received binary control command: ");
839.
840.            if (MmsValue_getBoolean(value)){
```

```
841.                    printf("on\n");
842.
843.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
844.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
845.             }
846.             else{
847.                    printf("off\n");
848.
849.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
850.                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
851.             }
852.
853.      }
854.      uint64_t timestamp = Hal_getTimeInMs();
855.
856.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO01) {
857.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_t, timestamp);
858.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, value);
859.      }
860.
861.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO02) {
862.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_t, timestamp);
863.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, value);
864.      }
865.
866.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO03) {
867.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO03_t, timestamp);
868.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO03_stVal, value);
869.      }
870.
871.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO04) {
872.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO04_t, timestamp);
873.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO04_stVal, value);
874.      }
875.
876.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO05) {
877.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO05_t, timestamp);
878.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO05_stVal, value);
879.      }
880.
881.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO06) {
882.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO06_t, timestamp);
883.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO06_stVal, value);
884.      }
885.
886.      if (parameter == IEDMODEL_CON_RBGGIO1_SPCSO07) {
887.             IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO07_t, timestamp);
888.             IedServer_updateAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO07_stVal, value);
889.      }
890.
891.      if (parameter == IEDMODEL_PRO_BKR1CSWI1_Pos) {
892.             //IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_BKR1CSWI1_Pos_Oper_ctlVal);
893.
894.      }
895.
896.  }
897.
898.  static void
899.  goCbEventHandler(MmsGooseControlBlock goCb, int event, void* parameter)
900.  {
901.      printf("Access to GoCB: %s\n", MmsGooseControlBlock_getName(goCb));
902.      printf("               GoEna: %i\n", MmsGooseControlBlock_getGoEna(goCb));
903.  }
904.
905.  //Função Listener
906.
907.  static void
```

```
908.  gooseListener6(GooseSubscriber subscriber, void* parameter)
909.  {
910.      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
911.
912.      char buffer[50];
913.
914.      MmsValue_printToBuffer(values, buffer, 50);
915.
916.      trip_2117 = buffer[1];
917.      estado_dj_2117 = atoi(&buffer[7]);
918.
919.      uint64_t y = Hal_getTimeInMs();
920.
921.      /*printf("-------------------------------------------------------------------------------------------------------\n");
922.      printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 1                                    \n");
923.      printf("-------------------------------------------------------------------------------------------------------\n");*/
924.
925.  }
926.
927.  static void
928.  gooseListener7(GooseSubscriber subscriber, void* parameter)
929.  {
930.      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
931.
932.      char buffer[50];
933.
934.      MmsValue_printToBuffer(values, buffer, 50);
935.
936.      trip_2118 = buffer[1];
937.      estado_dj_2118 = atoi(&buffer[7]);
938.
939.      uint64_t y = Hal_getTimeInMs();
940.
941.      /*printf("-------------------------------------------------------------------------------------------------------\n");
942.      printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 1                                    \n");
943.      printf("-------------------------------------------------------------------------------------------------------\n");*/
944.
945.  }
946.
947.  static void
948.  gooseListener(GooseSubscriber subscriber, void* parameter)
949.  {
950.      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
951.
952.      char buffer[50];
953.
954.      MmsValue_printToBuffer(values, buffer, 50);
955.
956.      trip_2115 = buffer[1];
957.
958.      if (trip_2115 == 116){
959.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
960.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
961.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind23_stVal, true);
962.      }
963.
964.      estado_dj_2115 = atoi(&buffer[7]);
965.
966.      uint64_t y = Hal_getTimeInMs();
967.
968.      /*printf("-------------------------------------------------------------------------------------------------------\n");
969.      printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 1                                    \n");
970.      printf("-------------------------------------------------------------------------------------------------------\n");*/
971.
972.  }
973.
974.  static void
```

```
975.  gooseListener1(GooseSubscriber subscriber, void* parameter)
976.  {
977.      MmsValue* values1 = GooseSubscriber_getDataSetValues(subscriber);
978.
979.      char buffer1[100];
980.
981.      MmsValue_printToBuffer(values1, buffer1, 100);
982.
983.      b = atof(&buffer1[1]);//21L2
984.      d = atof(&buffer1[11]);//21L4
985.      e = atof(&buffer1[21]);//21L5
986.
987.      if (maximo1 == true){
988.          pMax_21l4 = d;
989.          maximo1 = false;
990.      }
991.      if (maximo2 == true){
992.          pMax_21l5 = e;
993.          maximo2 = false;
994.      }
995.
996.      if (d > pMax_21l4){
997.          pMax_21l4 = d;
998.      }
999.      if (e > pMax_21l5){
1000.         pMax_21l5 = e;
1001.     }
1002.
1003.     uint64_t y = Hal_getTimeInMs();
1004.
1005.     /*printf("-------------------------------------------------------------------------------------------------------\n");
1006.     printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 2                                    \n");
1007.     printf("-------------------------------------------------------------------------------------------------------\n");*/
1008.
1009. }
1010.
1011. static void
1012. gooseListener2(GooseSubscriber subscriber, void* parameter)
1013. {
1014.     MmsValue* values2 = GooseSubscriber_getDataSetValues(subscriber);
1015.
1016.     char buffer2[100];
1017.
1018.     MmsValue_printToBuffer(values2, buffer2, 100);
1019.
1020.     c1 = atof(&buffer2[1]);//21L3
1021.     g1 = atof(&buffer2[11]);//21L7
1022.     uint64_t y = Hal_getTimeInMs();
1023.
1024.     /*printf("-------------------------------------------------------------------------------------------------------\n");
1025.     printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 3                                    \n");
1026.     printf("-------------------------------------------------------------------------------------------------------\n");*/
1027.
1028. }
1029.
1030. static void
1031. gooseListener3(GooseSubscriber subscriber, void* parameter)
1032. {
1033.     MmsValue* values3 = GooseSubscriber_getDataSetValues(subscriber);
1034.
1035.     char buffer3[100];
1036.
1037.     MmsValue_printToBuffer(values3, buffer3, 100);
1038.
1039.     a2 = atof(&buffer3[1]);//21L1
1040.     h2 = atof(&buffer3[11]);//21L8
1041.     i2 = atof(&buffer3[20]);//21L9
```

```
1042.       uint64_t y = Hal_getTimeInMs();
1043.
1044.       /*printf("-----------------------------------------------------------------------------------------------------\n");
1045.       printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 4                                    \n");
1046.       printf("-----------------------------------------------------------------------------------------------------\n");*/
1047.
1048. }
1049.
1050. static void
1051. gooseListener4(GooseSubscriber subscriber, void* parameter)
1052. {
1053.       MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
1054.
1055.       char buffer[50];
1056.
1057.       MmsValue_printToBuffer(values, buffer, 50);
1058.
1059.       comando_received_2117_A = buffer[1];
1060.       comando_received_2117_F = buffer[7];
1061.
1062.       float tensao;
1063.       tensao = atof(&buffer[13]);
1064.
1065.       if ((g1 == 0)&&(tensao==0))
1066.       {
1067.             if (comando_received_2117_A == 116)
1068.             {
1069.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1070.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
1071.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind24_stVal, true);
1072.                   /*
1073.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1074.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1075.                   */
1076.             }
1077.             if (comando_received_2117_F == 116)
1078.             {
1079.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1080.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1081.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind24_stVal, true);
1082.                   /*
1083.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1084.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
1085.                   */
1086.             }
1087.       }
1088.       /*system("clear");
1089.       printf("\n%d\n%d\n", buffer[1], buffer[6]);*/
1090.
1091.       uint64_t y = Hal_getTimeInMs();
1092.
1093.       /*printf("-----------------------------------------------------------------------------------------------------\n");
1094.       printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 2                                    \n");
1095.       printf("-----------------------------------------------------------------------------------------------------\n");*/
1096.
1097. }
1098.
1099. static void
1100. gooseListener8(GooseSubscriber subscriber, void* parameter)
1101. {
1102.       MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
1103.
1104.       char buffer[100];
1105.
1106.       MmsValue_printToBuffer(values, buffer, 100);
1107.
1108.       comando_received_2118_A = buffer[1];
```

```
1109.       comando_received_21l8_F = buffer[7];
1110.
1111.       float tensao;
1112.       tensao = atof(&buffer[13]);
1113.
1114.       if ((i2 == 0)&&(tensao==0))
1115.       {
1116.           if (comando_received_21l8_A == 116)
1117.           {
1118.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1119.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
1120.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind25_stVal, true);
1121.               /*
1122.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1123.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1124.               */
1125.           }
1126.           if (comando_received_21l8_F == 116)
1127.           {
1128.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1129.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1130.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind25_stVal, false);
1131.               /*
1132.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1133.               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
1134.               */
1135.           }
1136.       }
1137.
1138.       uint64_t y = Hal_getTimeInMs();
1139.
1140.       /*printf("-------------------------------------------------------------------------------------------------------\n");
1141.       printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 2                                    \n");
1142.       printf("-------------------------------------------------------------------------------------------------------\n");*/
1143.
1144.   }
1145.
1146.   static void
1147.   gooseListener5(GooseSubscriber subscriber, void* parameter)
1148.   {
1149.       MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
1150.
1151.       char buffer[50];
1152.
1153.       MmsValue_printToBuffer(values, buffer, 50);
1154.
1155.
1156.       char b; char c; char d;
1157.
1158.       b = buffer[1];
1159.       c = buffer[6];
1160.       d = buffer[11];
1161.       uint64_t y = Hal_getTimeInMs();
1162.
1163.       if(b == 116){
1164.           IedServer_updateDbposValue(iedServer, IEDMODEL_PRO_BK1XCBR1_Pos_stVal, DBPOS_ON);
1165.           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1166.       }else{
1167.           IedServer_updateDbposValue(iedServer, IEDMODEL_PRO_BK1XCBR1_Pos_stVal, DBPOS_INTERMEDIATE_STATE);
1168.           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1169.       }
1170.
1171.       /*printf("-------------------------------------------------------------------------------------------------------\n");
1172.       printf("                                    PRIMEIRA MENSAGEM GOOSE ASSINADA VIED 3                                    \n");
1173.       printf("-------------------------------------------------------------------------------------------------------\n");*/
1174.
1175.   }
```

```
1176.
1177.  void self_h()
1178.  {
1179.      while (1)
1180.      {
1181.          float x, y1;
1182.
1183.          /*system("clear");
1184.          printf("\n%f\n",a2);//2111
1185.          printf("%f\n",b);//2112
1186.          printf("%f\n",c1);//2113
1187.          printf("%f\n",d);//2114
1188.          printf("%f\n",e);//2115
1189.          printf("%f\n",corrente_primarioA);//2116
1190.          printf("%f\n",g1);//2117
1191.          printf("%f\n",h2);//2118
1192.          printf("%f\n",i2);//2119*/
1193.
1194.          /*IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_stVal, true);
1195.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_stVal, true);
1196.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind29_stVal, false);
1197.          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind30_stVal, true);
1198.          IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_MET_METMMXU1_PPV_phsAB_cVal_mag_f, 0);*/
1199.
1200.          uint64_t y = Hal_getTimeInMs();
1201.
1202.
1203.          // DETECÇÃO DO TRECHO EM FALTA PARA ENNCONTRO 21L6
1204.
1205.          // TRIP FALSO E DISJUNTOR FECHADO E TENSÃO EM ZERO
1206.
1207.          system("clear");
1208.
1209.          //printf("\n%f\n%d\n%d\n%d\n%d\n",tensao_primarioA,trip_2115,estado_dj_2115, estado_dj_2117, estado_dj_2118);
1210.
1211.          if ((tensao_primarioA == 0) && (trip_2115 != 116) && (estado_dj_2115 == 10))
1212.          {
1213.              if (delay_1 == true)
1214.              {
1215.                  gettimeofday(&delay1p, NULL);
1216.                  delay_1 = false;
1217.              }
1218.              gettimeofday(&delay1, NULL);
1219.              time_delay1 = (float)(delay1.tv_sec - delay1p.tv_sec);
1220.              time_delay1 += (delay1.tv_usec - delay1p.tv_usec) / (float)MICRO_PER_SECOND;
1221.              if ((time_delay1) >= 1)
1222.              {
1223.                  //system("clear");
1224.                  IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind26_stVal, true);
1225.                  IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind26_t, y);
1226.                  printf("\n----------");
1227.                  printf("\n----T5 em Falta------");
1228.                  printf("\n----------");
1229.                  // printf("\n%f\n%d\n%d\n",tensao_primarioA,trip_2115,estado_dj_2115);
1230.                  x = (a2 + pMax_2114 + g1) / pMaxS2111;
1231.                  y1 = (c1 + pMax_2114) / pMaxS2113;
1232.                  IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_stVal, true);
1233.                  IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_t, y);
1234.                  if ((x >= 1) && (y1 >= 1))
1235.                  {
1236.                      IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_stVal, true);
1237.                      IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_t, y);
1238.                      x = (a2 + pMax_2115 + g1) / pMaxS2111;
1239.                      y1 = (c1 + pMax_2115) / pMaxS2113;
1240.                  }
1241.                  if (delay_2 == true)
1242.                  {
```

```
1243.                     gettimeofday(&delay2p, NULL);
1244.                     delay_2 = false;
1245.                 }
1246.                 gettimeofday(&delay2, NULL);
1247.                 time_delay2 = (float)(delay2.tv_sec - delay2p.tv_sec);
1248.                 time_delay2 += (delay2.tv_usec - delay2p.tv_usec) / (float)MICRO_PER_SECOND;
1249.                 if ((time_delay2) >= 1)
1250.                 {
1251.                     if (x < y1)
1252.                     {
1253.                         printf("\n--------------------------");
1254.                         printf("\n----Reeligar por 21L8------");
1255.                         printf("\n--------------------------");
1256.                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind15_stVal, true);
1257.                         IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind15_t, y);
1258.                         if (delay_3 == true)
1259.                         {
1260.                             gettimeofday(&delay3p, NULL);
1261.                             delay_3 = false;
1262.                         }
1263.                         gettimeofday(&delay3, NULL);
1264.                         time_delay3 = (float)(delay3.tv_sec - delay3p.tv_sec);
1265.                         time_delay3 += (delay3.tv_usec - delay3p.tv_usec) / (float)MICRO_PER_SECOND;
1266.                         if ((time_delay3) >= 1)
1267.                         {
1268.                             if ((estado_dj_21l7 == 10) && (estado_dj_21l8 == 0))
1269.                             {
1270.                                 printf("\n----Abrir 21L7------");
1271.                                 // ABRIR 21L7
1272.                                 IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_stVal, true);
1273.                                 IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_t, y);
1274.                             }
1275.                             if (delay_4 == true)
1276.                             {
1277.                                 gettimeofday(&delay4p, NULL);
1278.                                 delay_4 = false;
1279.                             }
1280.                             gettimeofday(&delay4, NULL);
1281.                             time_delay4 = (float)(delay4.tv_sec - delay4p.tv_sec);
1282.                             time_delay4 += (delay4.tv_usec - delay4p.tv_usec) / (float)MICRO_PER_SECOND;
1283.                             if ((time_delay4) >= 1)
1284.                             {
1285.                                 if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 0))
1286.                                 {
1287.                                     printf("\n----Fechar 21L8------");
1288.                                     // FECHAR 21L8
1289.                                     IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind30_stVal, true);
1290.                                     IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_t, y);
1291.                                 }
1292.                                 if (delay_5 == true)
1293.                                 {
1294.                                     gettimeofday(&delay5p, NULL);
1295.                                     delay_5 = false;
1296.                                 }
1297.                                 gettimeofday(&delay5, NULL);
1298.                                 time_delay5 = (float)(delay5.tv_sec - delay5p.tv_sec);
1299.                                 time_delay5 += (delay5.tv_usec - delay5p.tv_usec) / (float)MICRO_PER_SECOND;
1300.                                 if ((time_delay5) >= 1)
1301.                                 {
1302.                                     if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 10))
1303.                                     {
1304.                                         printf("\n----Fechar 21L6------");
1305.                                         // FECHAR 21L6
1306.                                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1307.                                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1308.                                         IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_stVal, true);
1309.                                         IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_t, y);
```

```
                                        /*
                                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
                                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
                                        */
                                    }
                                }
                            }
                        }
                    }

                    if (y1 < x)
                    {
                        printf("\n--------------------------");
                        printf("\n----Reeligar por 21L7------");
                        printf("\n--------------------------");
                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind16_stVal, false);
                        IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind16_t, y);
                        if (delay_6 == true)
                        {
                            gettimeofday(&delay6p, NULL);
                            delay_6 = false;
                        }
                        gettimeofday(&delay6, NULL);
                        time_delay6 = (float)(delay6.tv_sec - delay6p.tv_sec);
                        time_delay6 += (delay6.tv_usec - delay6p.tv_usec) / (float)MICRO_PER_SECOND;
                        if ((time_delay6) >= 1)
                        {
                            if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 10))
                            {
                                printf("\n----Abrir 21L8------");
                                // ABRIR 21L8
                                IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind29_stVal, true);
                                IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_t, y);
                            }
                            if (delay_7 == true)
                            {
                                gettimeofday(&delay7p, NULL);
                                delay_7 = false;
                            }
                            gettimeofday(&delay7, NULL);
                            time_delay7 = (float)(delay7.tv_sec - delay7p.tv_sec);
                            time_delay7 += (delay7.tv_usec - delay7p.tv_usec) / (float)MICRO_PER_SECOND;
                            if ((time_delay7) >= 1)
                            {
                                if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 0))
                                {
                                    printf("\n----Fechar 21L7------");
                                    // FECHAR 21L7
                                    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_stVal, true);
                                    IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_t, y);
                                }
                                if (delay_8 == true)
                                {
                                    gettimeofday(&delay8p, NULL);
                                    delay_8 = false;
                                }
                                gettimeofday(&delay8, NULL);
                                time_delay8 = (float)(delay8.tv_sec - delay8p.tv_sec);
                                time_delay8 += (delay8.tv_usec - delay8p.tv_usec) / (float)MICRO_PER_SECOND;
                                if ((time_delay8) >= 1)
                                {
                                    if ((estado_dj_21l7 == 10) && (estado_dj_21l8 == 0))
                                    {
                                        printf("\n----Fechar 21L6------");
                                        // FECHAR 21L6
                                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
                                        IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
```

```
1377.                                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_stVal, true);
1378.                                           IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_t, y);
1379.                                           /*
1380.                                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, false);
1381.                                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, true);
1382.                                           */
1383.                                       }
1384.                                   }
1385.                               }
1386.                           }
1387.                       }
1388.                   }
1389.               }
1390.           }
1391.
1392.           //CONCLUIR DELAY DE TEMPO PARA TRECHO 6 (T6)
1393.           if ((tensao_primarioA == 0) && (trip_21l5 != 116) && (estado_dj_21l5 == 0))
1394.           {
1395.               if (delay_1 == true)
1396.               {
1397.                   gettimeofday(&delay1p, NULL);
1398.                   delay_1 = false;
1399.               }
1400.               gettimeofday(&delay1, NULL);
1401.               time_delay1 = (float)(delay1.tv_sec - delay1p.tv_sec);
1402.               time_delay1 += (delay1.tv_usec - delay1p.tv_usec) / (float)MICRO_PER_SECOND;
1403.               if ((time_delay1) >= 1)
1404.               {
1405.                   //system("clear");
1406.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind18_stVal, true);
1407.                   IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind18_t, y);
1408.                   /*printf("\n----------------------");
1409.                   printf("\n----T6 em Falta------");
1410.                   printf("\n----------------------");*/
1411.                   //printf("\n%f\n%d\n%d\n%d\n%d\n",tensao_primarioA,trip_21l5,estado_dj_21l5, estado_dj_21l7, estado_dj_21l8);
1412.                   x = (a2 + pMax_21l5 + g1) / pMaxS21l1;
1413.                   y1 = (c1 + pMax_21l5) / pMaxS21l3;
1414.                   IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_stVal, true);
1415.                   IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind19_t, y);
1416.                   if (delay_2 == true)
1417.                   {
1418.                       gettimeofday(&delay2p, NULL);
1419.                       delay_2 = false;
1420.                   }
1421.                   gettimeofday(&delay2, NULL);
1422.                   time_delay2 = (float)(delay2.tv_sec - delay2p.tv_sec);
1423.                   time_delay2 += (delay2.tv_usec - delay2p.tv_usec) / (float)MICRO_PER_SECOND;
1424.                   if ((time_delay2) >= 1)
1425.                   {
1426.                       if (x < y1)
1427.                       {
1428.                           printf("\n--------------------------");
1429.                           printf("\n----Reeligar por 21L8------");
1430.                           printf("\n--------------------------");
1431.                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind15_stVal, true);
1432.                           IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind15_t, y);
1433.                           if (delay_3 == true)
1434.                           {
1435.                               gettimeofday(&delay3p, NULL);
1436.                               delay_3 = false;
1437.                           }
1438.                           gettimeofday(&delay3, NULL);
1439.                           time_delay3 = (float)(delay3.tv_sec - delay3p.tv_sec);
1440.                           time_delay3 += (delay3.tv_usec - delay3p.tv_usec) / (float)MICRO_PER_SECOND;
1441.                           if ((time_delay3) >= 1)
1442.                           {
1443.                               if ((estado_dj_21l7 == 10) && (estado_dj_21l8 == 0))
```

```
1444.                                   {
1445.                                       printf("\n----Abrir 21L7------");
1446.                                       // ABRIR 21L7
1447.                                       IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_stVal, true);
1448.                                       IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_t, y);
1449.                                   }
1450.                                   if (delay_4 == true)
1451.                                   {
1452.                                       gettimeofday(&delay4p, NULL);
1453.                                       delay_4 = false;
1454.                                   }
1455.                                   gettimeofday(&delay4, NULL);
1456.                                   time_delay4 = (float)(delay4.tv_sec - delay4p.tv_sec);
1457.                                   time_delay4 += (delay4.tv_usec - delay4p.tv_usec) / (float)MICRO_PER_SECOND;
1458.                                   if ((time_delay4) >= 1)
1459.                                   {
1460.                                       if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 0))
1461.                                       {
1462.                                           printf("\n----Fechar 21L8------");
1463.                                           // FECHAR 21L8
1464.                                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind30_stVal, true);
1465.                                           IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_t, y);
1466.                                       }
1467.                                       if (delay_5 == true)
1468.                                       {
1469.                                           gettimeofday(&delay5p, NULL);
1470.                                           delay_5 = false;
1471.                                       }
1472.                                       gettimeofday(&delay5, NULL);
1473.                                       time_delay5 = (float)(delay5.tv_sec - delay5p.tv_sec);
1474.                                       time_delay5 += (delay5.tv_usec - delay5p.tv_usec) / (float)MICRO_PER_SECOND;
1475.                                       if ((time_delay5) >= 1)
1476.                                       {
1477.                                           if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 10))
1478.                                           {
1479.                                               printf("\n----Fechar 21L6------");
1480.                                               // FECHAR 21L6
1481.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1482.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1483.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_stVal, true);
1484.                                               IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_t, y);
1485.                                           }
1486.                                       }
1487.                                   }
1488.                               }
1489.                           }
1490.
1491.                           if (y1 < x)
1492.                           {
1493.                               printf("\n---------------------------");
1494.                               printf("\n----Reeligar por 21L7------");
1495.                               printf("\n---------------------------");
1496.                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind16_stVal, false);
1497.                               IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind16_t, y);
1498.                               if (delay_6 == true)
1499.                               {
1500.                                   gettimeofday(&delay6p, NULL);
1501.                                   delay_6 = false;
1502.                               }
1503.                               gettimeofday(&delay6, NULL);
1504.                               time_delay6 = (float)(delay6.tv_sec - delay6p.tv_sec);
1505.                               time_delay6 += (delay6.tv_usec - delay6p.tv_usec) / (float)MICRO_PER_SECOND;
1506.                               if ((time_delay6) >= 1)
1507.                               {
1508.                                   if ((estado_dj_21l7 == 0) && (estado_dj_21l8 == 10))
1509.                                   {
1510.                                       printf("\n----Abrir 21L8------");
```

```
1511.                                       // ABRIR 21L8
1512.                                       IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind29_stVal, true);
1513.                                       IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_t, y);
1514.                                   }
1515.                                   if (delay_7 == true)
1516.                                   {
1517.                                       gettimeofday(&delay7p, NULL);
1518.                                       delay_7 = false;
1519.                                   }
1520.                                   gettimeofday(&delay7, NULL);
1521.                                   time_delay7 = (float)(delay7.tv_sec - delay7p.tv_sec);
1522.                                   time_delay7 += (delay7.tv_usec - delay7p.tv_usec) / (float)MICRO_PER_SECOND;
1523.                                   if ((time_delay7) >= 1)
1524.                                   {
1525.                                       if ((estado_dj_2117 == 0) && (estado_dj_2118 == 0))
1526.                                       {
1527.                                           printf("\n----Fechar 21L7------");
1528.                                           // FECHAR 21L7
1529.                                           IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind28_stVal, true);
1530.                                           IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind27_t, y);
1531.                                       }
1532.                                       if (delay_8 == true)
1533.                                       {
1534.                                           gettimeofday(&delay8p, NULL);
1535.                                           delay_8 = false;
1536.                                       }
1537.                                       gettimeofday(&delay8, NULL);
1538.                                       time_delay8 = (float)(delay8.tv_sec - delay8p.tv_sec);
1539.                                       time_delay8 += (delay8.tv_usec - delay8p.tv_usec) / (float)MICRO_PER_SECOND;
1540.                                       if ((time_delay8) >= 1)
1541.                                       {
1542.                                           if ((estado_dj_2117 == 10) && (estado_dj_2118 == 0))
1543.                                           {
1544.                                               printf("\n----Fechar 21L6------");
1545.                                               // FECHAR 21L6
1546.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01_stVal, true);
1547.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02_stVal, false);
1548.                                               IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_stVal, true);
1549.                                               IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_SVGGIO3_Ind17_t, y);
1550.                                           }
1551.                                       }
1552.                                   }
1553.                               }
1554.                           }
1555.                       }
1556.                   }
1557.               }
1558.               Thread_sleep(17);
1559.           }
1560.   }
1561.
1562.   int
1563.   main(int argc, char** argv)
1564.   {
1565.       IedServerConfig config = IedServerConfig_create();
1566.
1567.       iedServer = IedServer_createWithConfig(&iedModel, NULL, config);
1568.
1569.       IedServerConfig_destroy(config);
1570.
1571.       SVReceiver receiverSV = SVReceiver_create();
1572.
1573.
1574.       if (argc > 1) {
1575.           char* ethernetIfcID = argv[1];
1576.
1577.           printf("Using GOOSE interface: %s\n", ethernetIfcID);
```

```
1578.
1579.           /* set GOOSE interface for all GOOSE publishers (GCBs) */
1580.           IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);
1581.           SVReceiver_setInterfaceId(receiverSV, ethernetIfcID);
1582.       }
1583.
1584.       if (argc > 2) {
1585.           char* ethernetIfcID = argv[2];
1586.
1587.           printf("Using GOOSE interface for GenericIO/LLN0.gcbAnalogValues: %s\n", ethernetIfcID);
1588.
1589.           /* set GOOSE interface for a particular GOOSE publisher (GCB) */
1590.           IedServer_setGooseInterfaceIdEx(iedServer, IEDMODEL_CFG_LLN0, "BRep0201", ethernetIfcID);
1591.       }
1592.
1593.       FILE *file;
1594.       file = fopen("Ajustes_50.txt", "r");
1595.       fscanf(file, "%f\n", &pick_up_50);
1596.       fclose(file);
1597.
1598.       FILE *file1;
1599.       file1 = fopen("Ajustes_50N.txt", "r");
1600.       fscanf(file1, "%f\n", &pick_up_50N);
1601.       fclose(file1);
1602.
1603.       FILE *file2;
1604.       file2 = fopen("Ajustes_51.txt", "r");
1605.       fscanf(file2, "%f\n", &pick_up_51);
1606.       fscanf(file2, "%f\n", &K_51);
1607.       fscanf(file2, "%f\n", &alfa_51);
1608.       fscanf(file2, "%f\n", &dial_51);
1609.       fclose(file2);
1610.
1611.       FILE *file3;
1612.       file3 = fopen("Ajustes_51V.txt", "r");
1613.       fscanf(file3, "%f\n", &pick_up_51V);
1614.       fscanf(file3,"%f\n", &tensao_51V);
1615.       fscanf(file3, "%f\n", &K_51V);
1616.       fscanf(file3, "%f\n", &alfa_51V);
1617.       fscanf(file3, "%f\n", &dial_51V);
1618.       fclose(file3);
1619.
1620.       FILE *file4;
1621.       file4 = fopen("Ajustes_51N.txt", "r");
1622.       fscanf(file4, "%f\n", &pick_up_51N);
1623.       fscanf(file4, "%f\n", &K_51N);
1624.       fscanf(file4, "%f\n", &alfa_51N);
1625.       fscanf(file4, "%f\n", &dial_51N);
1626.       fclose(file4);
1627.
1628.       FILE *file5;
1629.       file5 = fopen("Ajustes_67.txt", "r");
1630.       fscanf(file5, "%f\n", &pick_up_67);
1631.       fscanf(file5, "%f\n", &K_67);
1632.       fscanf(file5, "%f\n", &alfa_67);
1633.       fscanf(file5, "%f\n", &dial_67);
1634.       fscanf(file5, "%f\n", &atm_67);
1635.       fclose(file5);
1636.
1637.       FILE *file6;
1638.       file6 = fopen("Ajustes_67N.txt", "r");
1639.       fscanf(file6, "%f\n", &pick_up_67N);
1640.       fscanf(file6, "%f\n", &K_67N);
1641.       fscanf(file6, "%f\n", &alfa_67N);
1642.       fscanf(file6, "%f\n", &dial_67N);
1643.       fscanf(file6, "%f\n", &atm_67N);
1644.       fclose(file6);
```

```
1645.
1646.       /*Preparando o código para receber mensagens SV*/
1647.
1648.       SVSubscriber subscriberSV = SVSubscriber_create(NULL, 0x4000);
1649.       SVSubscriber_setListener(subscriberSV, svUpdateListener, NULL);
1650.       SVReceiver_addSubscriber(receiverSV, subscriberSV);
1651.
1652.
1653.       //Habilitando a publicação de mensagens GOOSE
1654.       IedServer_enableGoosePublishing(iedServer);
1655.
1656.       //VIED_21L6CFG/LLN0$GO$CONTROL_21L7
1657.
1658.       //Recepção e Assinatura de mensagens GOOSE
1659.       GooseReceiver receiver = GooseReceiver_create();
1660.       GooseReceiver_setInterfaceId(receiver, "eth0");
1661.       GooseSubscriber subscriber6 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$GOOSE_STATUS", NULL);
1662.       GooseSubscriber subscriber7 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$GOOSE_STATUS", NULL);
1663.       GooseSubscriber subscriber   = GooseSubscriber_create("VIED_21L5CFG/LLN0$GO$GOOSE_STATUS", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1664.       GooseSubscriber subscriber1 = GooseSubscriber_create("VIED_21L5CFG/LLN0$GO$GOOSE_POWER", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1665.       GooseSubscriber subscriber2 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$GOOSE_POWER", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1666.       GooseSubscriber subscriber3 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$GOOSE_POWER", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1667.       GooseSubscriber subscriber4 = GooseSubscriber_create("VIED_21L7CFG/LLN0$GO$CONTROL_21L6", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1668.       GooseSubscriber subscriber8 = GooseSubscriber_create("VIED_21L8CFG/LLN0$GO$CONTROL_21L6", NULL); //Especificação de quem o ied irá receber as
      mensagens goose
1669.       GooseSubscriber subscriber5 = GooseSubscriber_create("MUBinIO_BinaryInputs/LLN0$GO$VMU_06_GOOSE", NULL); //Especificação de quem o ied irá
      receber as mensagens goose
1670.       GooseSubscriber_setListener(subscriber, gooseListener, iedServer);
1671.       GooseSubscriber_setListener(subscriber1, gooseListener1, iedServer);
1672.       GooseSubscriber_setListener(subscriber2, gooseListener2, iedServer);
1673.       GooseSubscriber_setListener(subscriber3, gooseListener3, iedServer);
1674.       GooseSubscriber_setListener(subscriber4, gooseListener4, iedServer);
1675.       GooseSubscriber_setListener(subscriber8, gooseListener8, iedServer);
1676.       GooseSubscriber_setListener(subscriber5, gooseListener5, iedServer);
1677.       GooseSubscriber_setListener(subscriber6, gooseListener6, iedServer);
1678.       GooseSubscriber_setListener(subscriber7, gooseListener7, iedServer);
1679.       GooseReceiver_addSubscriber(receiver, subscriber);
1680.       GooseReceiver_addSubscriber(receiver, subscriber1);
1681.       GooseReceiver_addSubscriber(receiver, subscriber2);
1682.       GooseReceiver_addSubscriber(receiver, subscriber3);
1683.       GooseReceiver_addSubscriber(receiver, subscriber4);
1684.       GooseReceiver_addSubscriber(receiver, subscriber8);
1685.       GooseReceiver_addSubscriber(receiver, subscriber5);
1686.       GooseReceiver_addSubscriber(receiver, subscriber6);
1687.       GooseReceiver_addSubscriber(receiver, subscriber7);
1688.
1689.       //Ligação dos Servidores
1690.       GooseReceiver_start(receiver);
1691.       SVReceiver_start(receiverSV);
1692.
1693.       //Ligação dos Servidores para cada função de proteção
1694.       if (pick_up_50 > 0)
1695.       {
1696.           thread_50 = Thread_create((ThreadExecutionFunction)funcao_50, (void *) thread_50, false);
1697.           Thread_start(thread_50);
1698.       }
1699.       if (pick_up_50N > 0)
1700.       {
1701.           thread_50N = Thread_create((ThreadExecutionFunction)funcao_50N, (void *) thread_50N, false);
1702.           Thread_start(thread_50N);
1703.       }
1704.       if (pick_up_51 > 0)
```

```
1705.     {
1706.         thread_51 = Thread_create((ThreadExecutionFunction)funcao_51, (void *) thread_51, false);
1707.         Thread_start(thread_51);
1708.     }
1709.     if (pick_up_51V > 0)
1710.     {
1711.         thread_51V = Thread_create((ThreadExecutionFunction)funcao_51V, (void *) thread_51V, false);
1712.         Thread_start(thread_51V);
1713.     }
1714.     if (pick_up_51N > 0)
1715.     {
1716.         thread_51N = Thread_create((ThreadExecutionFunction)funcao_51N, (void *) thread_51N, false);
1717.         Thread_start(thread_51N);
1718.     }
1719.     if (pick_up_67 > 0)
1720.     {
1721.         thread_67 = Thread_create((ThreadExecutionFunction)funcao_67, (void *) thread_67, false);
1722.         Thread_start(thread_67);
1723.     }
1724.     if (pick_up_67N > 0)
1725.     {
1726.         thread_67N = Thread_create((ThreadExecutionFunction)funcao_67N, (void *) thread_67N, false);
1727.         Thread_start(thread_67N);
1728.     }
1729.
1730.     self_healing = Thread_create((ThreadExecutionFunction)self_h, (void *) self_healing, false);
1731.     Thread_start(self_healing);
1732.
1733.     IedServer_setGoCBHandler(iedServer, goCbEventHandler, NULL);
1734.
1735.     /* MMS server will be instructed to start listening to client connections. */
1736.     IedServer_start(iedServer, 102);
1737.
1738.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO01, (ControlHandler) controlHandlerForBinaryOutput,
1739.     IEDMODEL_CON_RBGGIO1_SPCSO01);
1740.
1741.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO02, (ControlHandler) controlHandlerForBinaryOutput,
1742.     IEDMODEL_CON_RBGGIO1_SPCSO02);
1743.
1744.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO03, (ControlHandler) controlHandlerForBinaryOutput,
1745.     IEDMODEL_CON_RBGGIO1_SPCSO03);
1746.
1747.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO04, (ControlHandler) controlHandlerForBinaryOutput,
1748.     IEDMODEL_CON_RBGGIO1_SPCSO04);
1749.
1750.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO05, (ControlHandler) controlHandlerForBinaryOutput,
1751.     IEDMODEL_CON_RBGGIO1_SPCSO05);
1752.
1753.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO06, (ControlHandler) controlHandlerForBinaryOutput,
1754.     IEDMODEL_CON_RBGGIO1_SPCSO06);
1755.
1756.     IedServer_setControlHandler(iedServer, IEDMODEL_CON_RBGGIO1_SPCSO07, (ControlHandler) controlHandlerForBinaryOutput,
1757.     IEDMODEL_CON_RBGGIO1_SPCSO07);
1758.
1759.     IedServer_setControlHandler(iedServer, IEDMODEL_PRO_BKR1CSWI1_Pos, (ControlHandler) controlHandlerForBinaryOutput,
      IEDMODEL_PRO_BKR1CSWI1_Pos);
1760.
1761.     if (!IedServer_isRunning(iedServer)) {
1762.         printf("Starting server failed! Exit.\n");
1763.         IedServer_destroy(iedServer);
1764.         exit(-1);
1765.     }
1766.
1767.     running = 1;
1768.
1769.     signal(SIGINT, sigint_handler);
1770.
```

```
1771.         while (running) {
1772.
1773.             IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_ANN_MVGGIO12_AnIn02_mag_f, b);
1774.             IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_ANN_MVGGIO12_AnIn04_mag_f, d);
1775.             IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_ANN_MVGGIO12_AnIn05_mag_f, e);
1776.             IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_ANN_MVGGIO12_AnIn06_mag_f, corrente_primarioA);
1777.
1778.
1779.
1780.             /*
1781.             system("clear");
1782.             printf("\n%f\n",a2);//2111
1783.             printf("%f\n",b);//2112
1784.             printf("%f\n",c1);//2113
1785.             printf("%f\n",d);//2114
1786.             printf("%f\n",e);//2115
1787.             printf("%f\n",corrente_primarioA);//2116
1788.             printf("%f\n",g1);//2117
1789.             printf("%f\n",h2);//2118
1790.             printf("%f\n",i2);//2119
1791.             */
1792.
1793.             IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_ANN_INAGGIO1_Ind01_stVal, true);
1794.             Thread_sleep(17);
1795.
1796.         }
1797.
1798.         /* stop MMS server - close TCP server socket and all client sockets */
1799.             IedServer_stop(iedServer);
1800.
1801.             /* Cleanup - free all resources */
1802.             SVReceiver_destroy(receiverSV);
1803.             IedServer_destroy(iedServer);
1804.
1805.
1806.         return 0;
1807.     } /* main() */
```