UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GUILHERME LAWRENCE REBOUÇAS OLIVEIRA

DESENVOLVIMENTO DE PLATAFORMA DE SIMULAÇÃO DE SISTEMA DE AUTOMAÇÃO BASEADO EM TECNOLOGIAS LIVRES

FORTALEZA

2020

GUILHERME LAWRENCE REBOUÇAS OLIVEIRA

DESENVOLVIMENTO DE PLATAFORMA DE SIMULAÇÃO DE SISTEMA DE AUTOMAÇÃO BASEADO EM TECNOLOGIAS LIVRES

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

Coorientador: Prof Me. Felipe Carvalho Sampaio

FORTALEZA

2020

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

O47d Oliveira, Guilherme Lawrence Rebouças.
Desenvolvimento de plataforma de simulação de sistema de automação baseado em tecnologias livres / Guilherme Lawrence Rebouças Oliveira. – 2020.
73 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2020.
Orientação: Prof. Dr. Raimundo Furtado Sampaio.
Coorientação: Prof. Me. Felipe Carvalho Sampaio.

1. Automação. 2. SCADA. 3. ScadaBR. 4. Protocolo de comunicação. 5. Código aberto. I. Título.
CDD 621.3

GUILHERME LAWRENCE REBOUÇAS OLIVEIRA

DESENVOLVIMENTO DE PLATAFORMA DE SIMULAÇÃO DE SISTEMA DE AUTOMAÇÃO BASEADO EM TECNOLOGIAS LIVRES

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 Me. Felipe Carvalho Sampaio (Coorientador)
Universidade de Fortaleza (UNIFOR)

Profa. Ph.D. Ruth Pastôra Saraiva Leão
Universidade Federal do Ceará (UFC)

Eng. Claudivan Domingos Freitas
Universidade Federal do Ceará (UFC)

Aos meus pais, Gustavo e Lawrentinna.

Ao meu irmão, Luís Gustavo (Bubu).

À minha tia-avó, Maria das Dores.

À minha babá, Gracilene (Nena).

AGRADECIMENTOS

Aos meus pais, Gustavo e Lawrentinna, por toda dedicação à minha formação em todas as esferas.

Ao meu irmão, Luís, por me forçar, todos os dias, a ser melhor exemplo de pessoa.

Ao Programa de Educação Tutorial (PET), por todas as experiências, aprendizados e por todas as pessoas excelentes que me levou a conhecer. Em especial, aos membros com quem tive o prazer de trabalhar com mais proximidade, dentre os quais posso citar Bruno Luíz Faustino e Henrique César Teixeira Hermógenes;

Aos professores cuja humildade e interesse genuíno, certamente, marcam, inspiram e dão ânimo para muitos discentes a cada semestre. Em especial, à Natália Cordeiro Barroso, minha professora de cálculo; ao René Pastor Torrico Bascopé, meu tutor do PET; Raimundo Furtado Sampaio e Felipe Carvalho Sampaio, meus orientadores do TCC. E aos demais que, ainda que por meios nem sempre agradáveis, me tornaram uma pessoa mais dedicada, resiliente e curiosa.

Por fim, aos amigos que conheci na graduação cujo suporte e companheirismo não somente foram essenciais para possibilitar essa conclusão, como também tornaram essa jornada muito prazerosa e memorável. Em especial, Valessa Valentim Viana, Rodrigo Tornisiello e Giovanni de Oliveira Confalonieri.

RESUMO

A rápida disseminação da automação a torna uma ferramenta cada vez mais interessante para as mais diversas entidades, desde grandes indústrias e até mesmo escritórios que buscam melhor eficiência energética e conforto para seus colaboradores. Contudo, o desenvolvimento de projetos de automação e supervisão é, muitas vezes, limitado pela falta de alternativas ao uso de *softwares* proprietários, que podem representar custos que inviabilizam a iniciativa além de terem compatibilidade com outros dispositivos limitada. Este trabalho visa apresentar a implementação de um sistema de automação utilizando apenas ferramentas livres, como módulos da linguagem Python e o protocolo Modbus. Isso é realizado a partir do desenvolvimento de um programa que simula um sistema elétrico e seus componentes em uma rede de comunicação e passa a se comunicar com o *software* livre ScadaBR. A utilização proposta e os códigos desenvolvidos podem ser adaptados a diversos contextos, o que pode possibilitar o desenvolvimento de aplicações de baixo custo em diversas áreas.

Palavras-chave: Automação. Sistema SCADA. ScadaBR. Código Aberto. Modbus. Protocolos de Comunicação.

ABSTRACT

The rapid dissemination of automation makes it an increasingly interesting tool diverse entities, from large industries to offices that seek Efficient energy use and comfort for their employees. However, the development of automation and supervision projects is often limited by the lack of alternatives to proprietary software, which can represent high costs that can make them nonviable, in addition to having limited compatibility with other devices. This work aims to propose the implementation of an automation system using open technologies only, such as python modules and Modbus protocol. This is done through the development of a program that simulates an electrical system and its components in a communication network and communicates with the free software ScadaBR. The proposed use and the developed codes can be adapted to different contexts which can enable the development of low cost applications in several areas.

Keywords: Automation. SCADA Systems. ScadaBR. Open Source. Comunication Protocols. Modbus

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

CSV	Arquivo de valores separados por vírgula
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
SCADA	Sistema de Supervisão, Controle e Aquisição de Dados

SUMÁRIO

1 INTRODUÇÃO

Em uma sociedade na qual a utilização de sistemas com capacidade de conectar múltiplos e variados componentes tem um papel cada vez mais essencial, tem-se uma indústria de tecnologias *open source* ainda pouco popular. Em geral, sistemas de controle e automação utilizam tecnologias proprietárias, que podem representar altos custos. Tal fato pode inviabilizar o desenvolvimento de estudos e projetos utilizando esse tipo desse tipo de sistema, tanto em universidades como em empresas de pequeno e médio porte. Contudo, existe uma tendência de mudança dessa realidade nos próximos anos. Isso se dá pela ascensão de padrões de comunicação abertos e dos dispositivos de *hardware* livre como Arduino, BeagleBoard e Raspberry Pi (KAZALA; STRACZYNSKI, 2019a).

Nesse contexto, o presente trabalho busca explorar a utilização de um Sistema de Supervisão, Controle e Aquisição de Dados Sistema de Supervisão, Controle e Aquisição de Dados (SCADA) de uma forma facilmente replicável em diferentes *softwares* e *hardwares* e para aplicações diversas utilizando apenas ferramentas de código aberto. Para esse fim, foi desenvolvido um *software* de simulação de um sistema elétrico que se comunica continuamente com o sistema ScadaBR, onde se pode monitorar e controlar a variáveis desejadas em uma interface gráfica.

1.1 Objetivos

1.1.1 Objetivos Gerais

Desenvolver uma plataforma de simulação de sistema de automação baseando em um contexto do sistema elétrico, utilizando-se do *software* ScadaBR e outras tecnologias livres.

1.1.2 Objetivos Específicos

Entre os objetivos específicos, pode-se citar:

- Realizar revisão bibliográfica acerca do ScadaBR e das tecnologias livres com potencial de aplicação em sistemas de automação;
- Desenvolver um sistema de simulação de proteção e operação de subestação para teste e validação do sistema SCADA;
- Desenvolver uma aplicação de supervisão e aquisição de dados para subestação na plata-

forma *ScadaBR*;

- Desenvolver interface de comunicação entre o sistema simulador de subestação e o ScadaBR utilizando python e protocolo de comunicação aberto.

1.2 Estrutura do trabalho

Este trabalho está dividido em 5 capítulos, discriminados da seguinte forma:

- O capítulo 1 introduz o assunto e os objetivos do trabalho;
- O capítulo 2 aborda os conhecimentos teóricos que permeiam o trabalho, abrangendo os conceitos relacionados aos sistemas elétricos, aos sistemas supervisórios e às tecnologias utilizadas;
- O capítulo 3 relata a implementação dos conceitos apresentados anteriormente no desenvolvimento do sistema proposto;
- O capítulo 4 contém os resultados, ou seja, retrata o sistema com seu funcionamento final e o comportamento observado nos testes;
- O capítulo 5 contém as conclusões e as sugestões para trabalhos futuros, que podem ser desenvolvidos como continuação deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas Scada

Os sistemas SCADA (Supervisory Control and Data Aquisition) são caracterizados por fornecer uma conexão de um agente operador a um processo sendo executado. Esses *softwares* possuem as funções de supervisão, aquisição de dados e controle. Essas são possibilitadas pela utilização de dispositivos diversos que se encarregam de colher informações e transmiti-las por meio de um protocolo de comunicação. O fato de terem a capacidade de se comunicar com diversos protocolos vinculados a distintos meios de transmissão, como rede sem fio, cabos coaxiais e cabos de fibra ótica, é uma das grandes vantagens da integração desse tipo de software a um processo. Por esse motivo, os sistemas Scada abrangem mercados bastante amplos, sendo adequados para aplicações industriais, aeroviários e até residenciais (PINHEIRO, 2006).

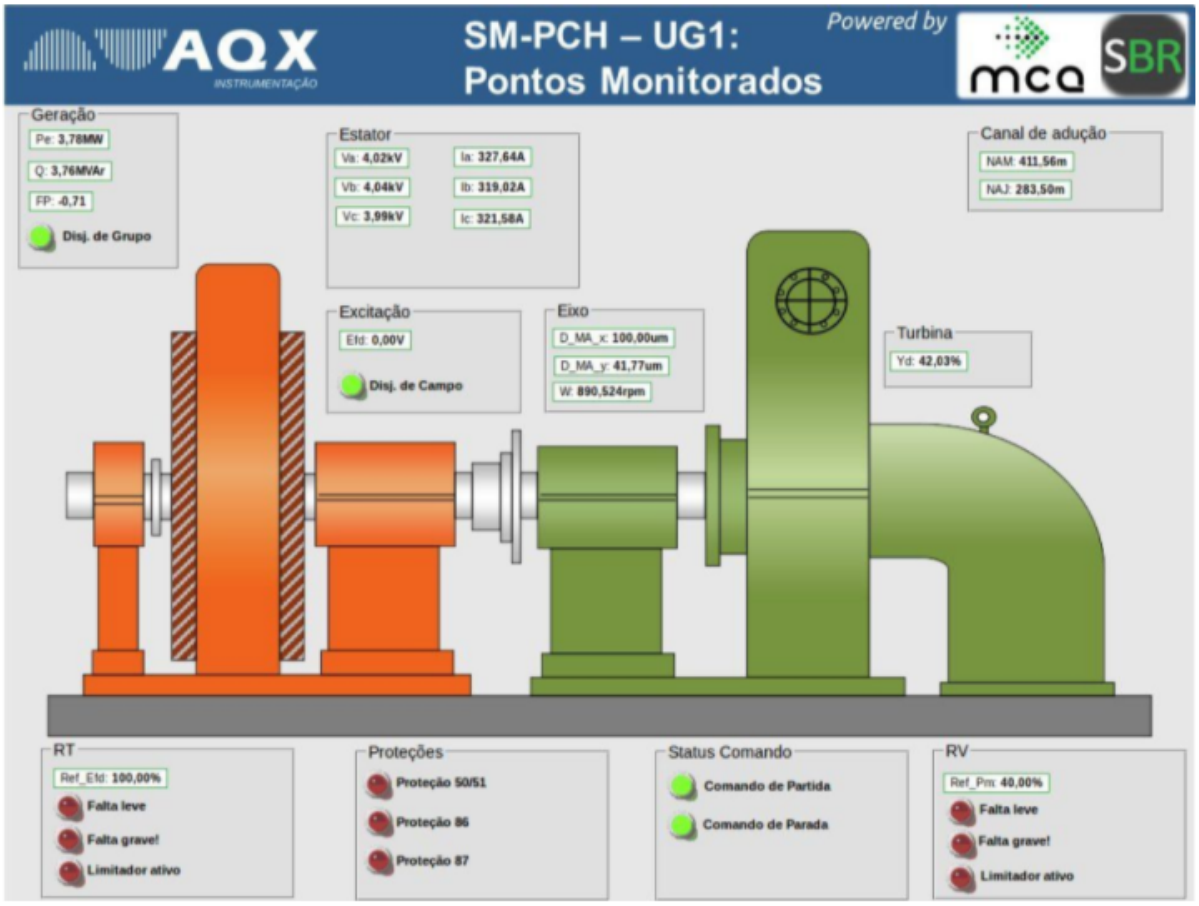
Além da aquisição de dados, existem outras funções primordiais que são abrangidas pelos sistemas SCADA, das quais podemos citar a visualização de dados e o processamento de alarmes. A visualização de dados é crucial, pois uma das finalidades do sistema é fornecer satisfatoriamente as informações necessárias para o agente responsável pelo processo monitorado. Para garantir que isso ocorra de forma satisfatória, é necessária a presença de uma Interface homem-máquina clara e completa a fim de garantir o melhor entendimento do processo pelos atores envolvidos. O processamento de alarmes permite que, automaticamente, sejam reportados alertas geralmente relacionados a questões de segurança. Esses podem ser classificados de acordo com o nível de criticidade dos acontecimentos que representam. Por exemplo, a presença de uma carga um pouco maior que a nominal em um transformador de uma subestação pode indicar um alarme de prioridade baixa, enquanto a falha na comunicação com os transformadores de corrente pode acionar um alarme de prioridade alta. Em determinadas situações, a ocorrência de alarmes pode, inclusive, ocasionar uma atuação automática do sistema (PINHEIRO, 2006).

2.1.1 ScadaBR

Em 2006, surgiu o projeto ScadaBR por iniciativa da MCA Sistemas (atualmente Sensorweb), que foi desenvolvido em parceria com as empresas Unis Sistemas, Conetec, Fundação CERTI e com a Universidade Federal de Santa Cataria. Essa iniciativa também teve apoio financeiro da FINEP, SEBRAE e CNPq. O ScadaBR é um software de código aberto para desenvolvimento mantido e atualizado por sua comunidade, formada por usuários, programadores e

profissionais liberais. Sua principal finalidade é possibilitar o desenvolvimento de aplicações de Automação, Aquisição de dados e Controle Supervisório para pequenas automatizações, profissionais autônomos, micro-empresas e utilização acadêmica em geral. A presença dessa iniciativa *open source* é de grande valor, uma vez que Softwares Scada tradicionais geralmente são inacessíveis a esses públicos devido ao seu alto custo. Com ScadaBR é possível desenvolver sistemas completos até construção de interface homem-máquina (IHM) para aplicações profissionais. Um exemplo disponibilizado no site oficial do ScadaBR pode ser observado na Figura 1.

Figura 1 – Interface Homem-Máquina desenvolvida no ScadaBR



Fonte: Sensorweb (2017a)

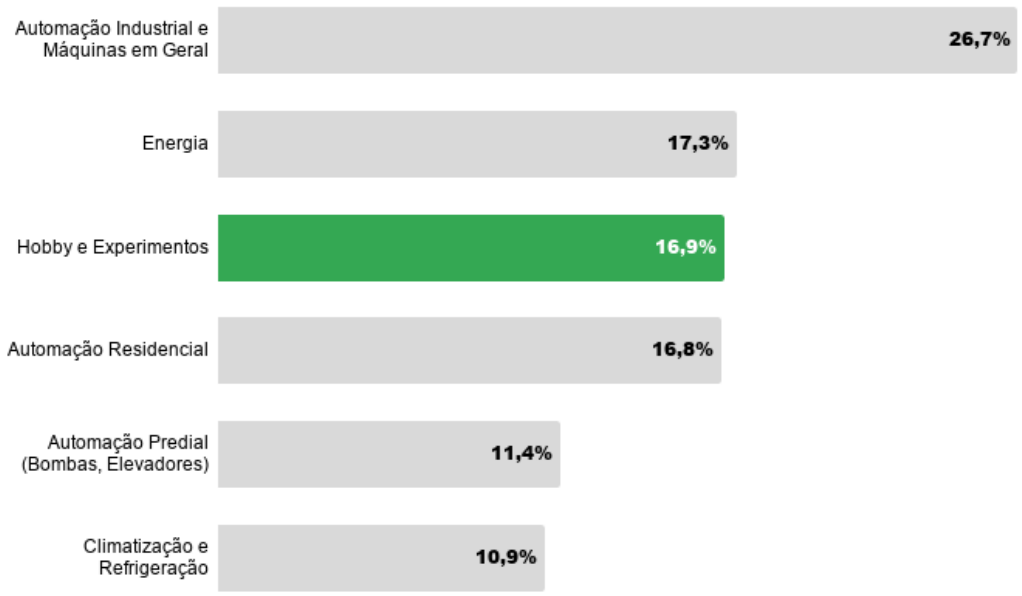
O ScadaBR foi desenvolvido a partir do software Mango, por já possuir uma interface considerada bastante intuitiva além de já suportar a comunicação com protocolo Modbus entre outras funções típicas para esse tipo de programa. O fato de que esse programa possui uma quantidade estimada de 10 mil usuários e mais de 6 mil membros em sua comunidade, afirma que o sistema cumpre seu papel em ser uma tecnologia acessível e útil para seu público alvo (SENSORWEB, 2017b).

Pode-se destacar alguns dos principais pontos responsáveis por esse sucesso:

- É uma tecnologia gratuita;
- Incorpora uma grande variedade de protocolos de comunicação;
- Possui interface simples e intuitiva;
- Utiliza a língua portuguesa;
- Pode ser utilizado em diversos sistemas operacionais (Windows, Linux, Mac entre outros);
- Pode ser integrado a qualquer linguagem de programação por meio de uma API em Web Services.

Existem também outras vantagens percebidas em relação à ergonomia de software relacionados com o esforço mínimo do usuário, frustração mínima, notificação imediata de problemas entre outros. Todos esses fatores corroboram para que parte relevante da utilização dessa ferramenta ocorra não somente em contextos profissionais como automação industrial, como também largamente para hobbies e experimentos, conforme pode ser observado na Figura 2. Esses dados indicam que o projeto é bem sucedido no objetivo de estimular o desenvolvimento de soluções a partir das universidades e de iniciativas individuais de entusiastas do assunto (SILVA *et al.*, 2013).

Figura 2 – Utilização do ScadaBR por finalidade



Fonte: Adaptado de Pinheiro (2006)

2.2 Protocolos de Comunicação Abertos

Atualmente, a maioria dos sistemas de automação utiliza protocolos proprietários das mesmas empresas que fornecem os equipamentos. Tal realidade pode caracterizar um problema, mesmo para as empresas que buscam utilizar os equipamentos de uma mesma empresa fornecedora, principalmente, no que tange à expansão dos sistemas. Esse fato, além de dificultar o uso de equipamentos de fabricantes diferentes em um mesmo sistema, acarreta a obsolescência de um equipamento de acordo com o prazo em que a empresa continua a comercializar dispositivos compatíveis. Existe a possibilidade de se utilizar *gateways* para integrar diferentes protocolos, mas é uma solução que aumenta significativamente o custo e a complexidade do sistema além de acarretar perda de performance (LOHIA *et al.*, 2019).

Nesse contexto, a ascensão dos protocolos livres possibilita a construção e manutenção de sistema com dispositivos diversos de forma muito fácil tanto por fatores técnicos como também por fatores econômicos. Considerando que as plataformas de hardware aberto são de fácil compatibilidade com sensores e atuadores, esse processo possibilita que sejam realizadas implementações mesmo em sistemas críticos e que utilizam tecnologias proprietárias com um custo muito baixo, como a implementação de um sensor de temperatura de sistema de geração fotovoltaica em operação. Nessa implementação, pode-se ter uma placa Arduíno de baixo custo se comunicando com o mesmo sistema SCADA que os elementos nativos de um sistema industrial prioritário (GONZALEZ; CALDERON, 2019).

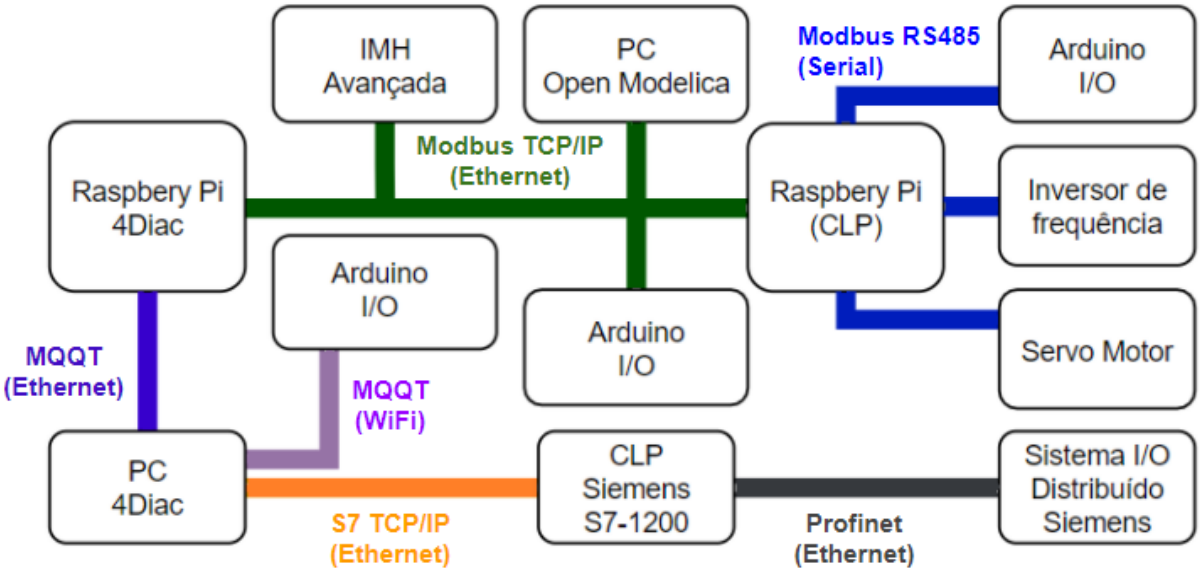
Dada a diversidade e o nível de maturidade desse tipo de tecnologia, já é possível projetar um sistema de controle completo utilizando em conjunto de diversos tipos de hardwares livres ou não, comunicando-se apenas por meio de protocolos livres. A Figura 3 ilustra como um seria um sistema dessa forma.

2.2.1 Modbus

O Modbus é um protocolo da camada de aplicação que foi originalmente introduzido em 1979. Este protocolo utiliza uma comunicação Cliente/Servidor (Mestre/Escravo) onde cada mensagem contém o endereço do destinatário, o comando a ser executado (leitura ou escrita) e a informação de interesse (FOVINO *et al.*, 2009).

Desde 2004, o protocolo é mantido e controlado pela comunidade Modbus-IDA, formada por usuários e fornecedores de equipamentos de automação. Diferentes interfaces

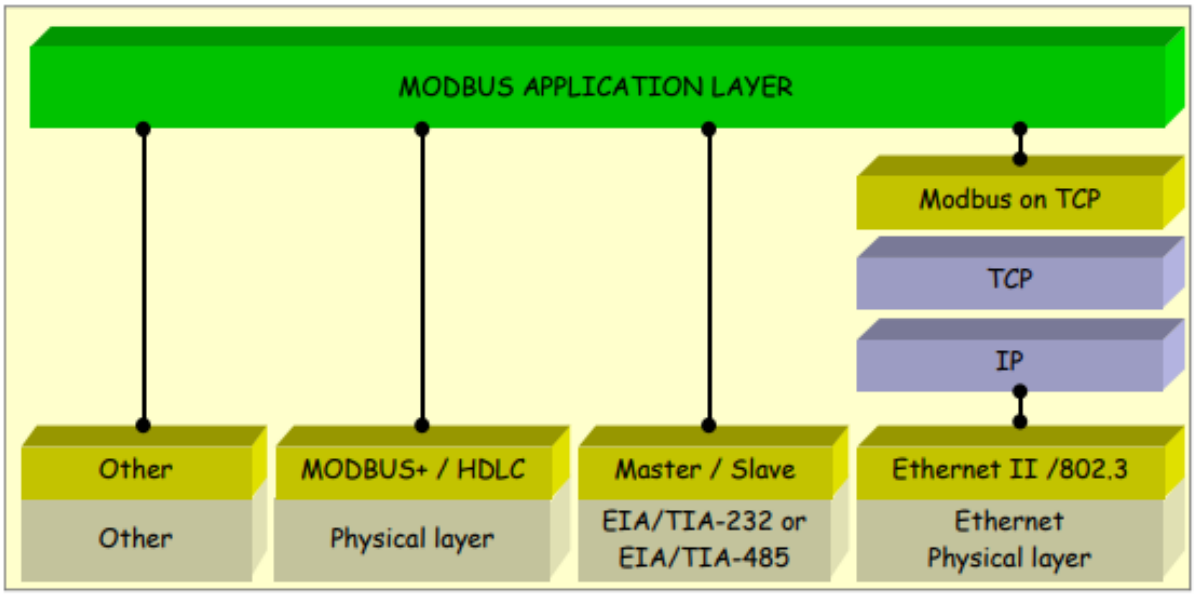
Figura 3 – Comunicação *open souce* em sistema de controle



Fonte: Adapatado de Kazala e Straczynski (2019b)

de comunicação são possibilitadas, das quais as mais populares são a transmissão por porta serial, RS-232, RS-485, ou por TCP/IP, usando protocolo *ethernet* (IEEE 802.3).Na Figura 4 é apresentada a estrutura do protocolo Modbus e suas várias interfaces (Guarese *et al.*, 2012).

Figura 4 – Estrutura das camadas do protocolo Modbus

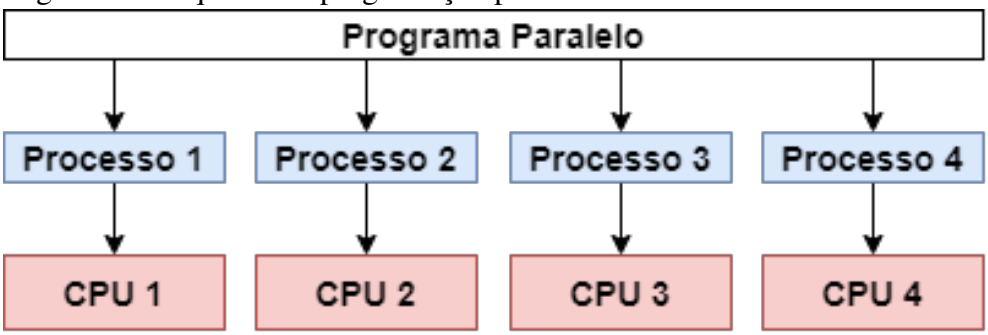


Fonte: Modbus-IDA (2006)

2.3 Multiprocessamento

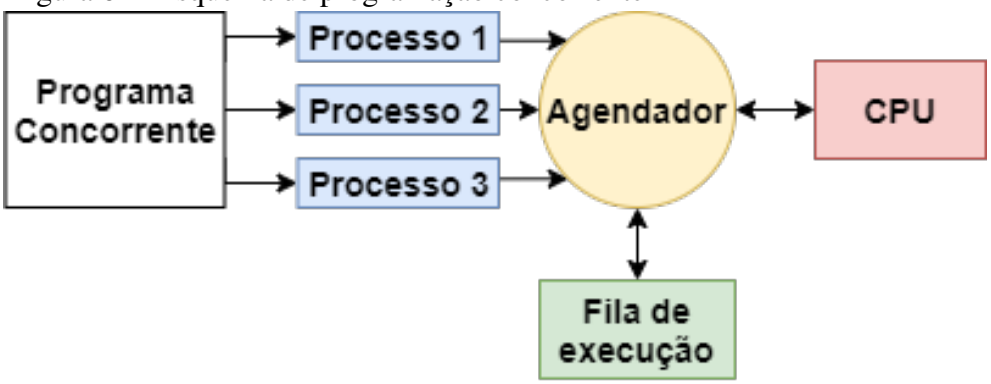
Programação paralela é um conceito relacionado com o desenvolvimento de programas que executam mais de uma instrução simultaneamente. Tal conceito possibilita que sejam utilizados múltiplos processadores em uma máquina, o que é uma alternativa à que permite um aumento na performance sem a necessidade de aumentar a frequência de trabalho de cada um deles. A limitação na utilização de frequências muito grandes é relacionada com as limitações físicas dos dispositivos, que passam a consumir mais energia e gerar mais calor para a máquina. Em uma execução paralela, existem funções específicas definidas no programa que são executadas por diferentes unidades de processamento. Sistemas em que isso não ocorre são chamados de concorrentes e existe uma ordem na qual cada programa é executado pelo processador. Os dois tipos são exemplificados, respectivamente, nas Figuras 5 e 6.

Figura 5 – Esquema de programação paralela



Fonte: Adaptado de Palach (2014)

Figura 6 – Esquema de programação concorrente



Fonte: Adaptado de Palach (2014)

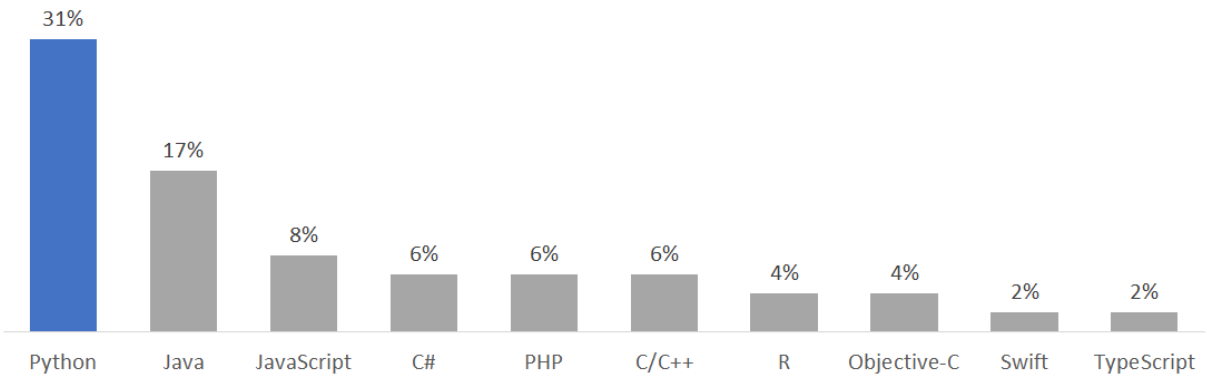
2.4 Python

Python é uma linguagem de programação de alto nível criada pelo programador holandês Guido van Rossum na década de 1990. Sua filosofia é descrita, entre outros, por princípios como (PETERS, 2010):

- Bonito é melhor do que feio;
- Explícito é melhor que implícito;
- Simples é melhor do que complexo;
- Erros nunca devem passar sem aviso (a menos que explicitamente silenciados);
- Deve haver uma, preferencialmente apenas uma, forma óbvia de utilizar;
- Esparso é melhor que denso;
- Em caso de ambiguidade, recuse a tentação de adivinhar;
- Se a implementação é difícil de explicar, ela deve ser uma má ideia.

Python pode ser considerada a linguagem de programação mais popular da atualidade. Na Figura 7 é apresentado um resultado que indica a popularidade das principais linguagens de programação de acordo com a quantidade de buscas por tutoriais realizadas no Google. Nesse quesito, Python é responsável por uma parcela equivalente a quase o dobro do segundo colocado. Apesar da facilidade de aprendizado e entendimento da linguagem, que se pode inferir a partir dos princípios citados, o fator mais determinante para que a linguagem se mantenha tão destacada é a enorme quantidade de bibliotecas disponíveis. Até novembro de 2019, o repositório oficial da linguagem já possuía mais de 200 mil pacotes relacionados a uma grande quantidade de aplicações como automação, análise de dados, aprendizado de máquina, processamento de imagem e desenvolvimento para plataformas móveis (ROSSUM *et al.*, 2007).

Figura 7 – Percentual das buscas no Google por tutoriais de programação por linguagem

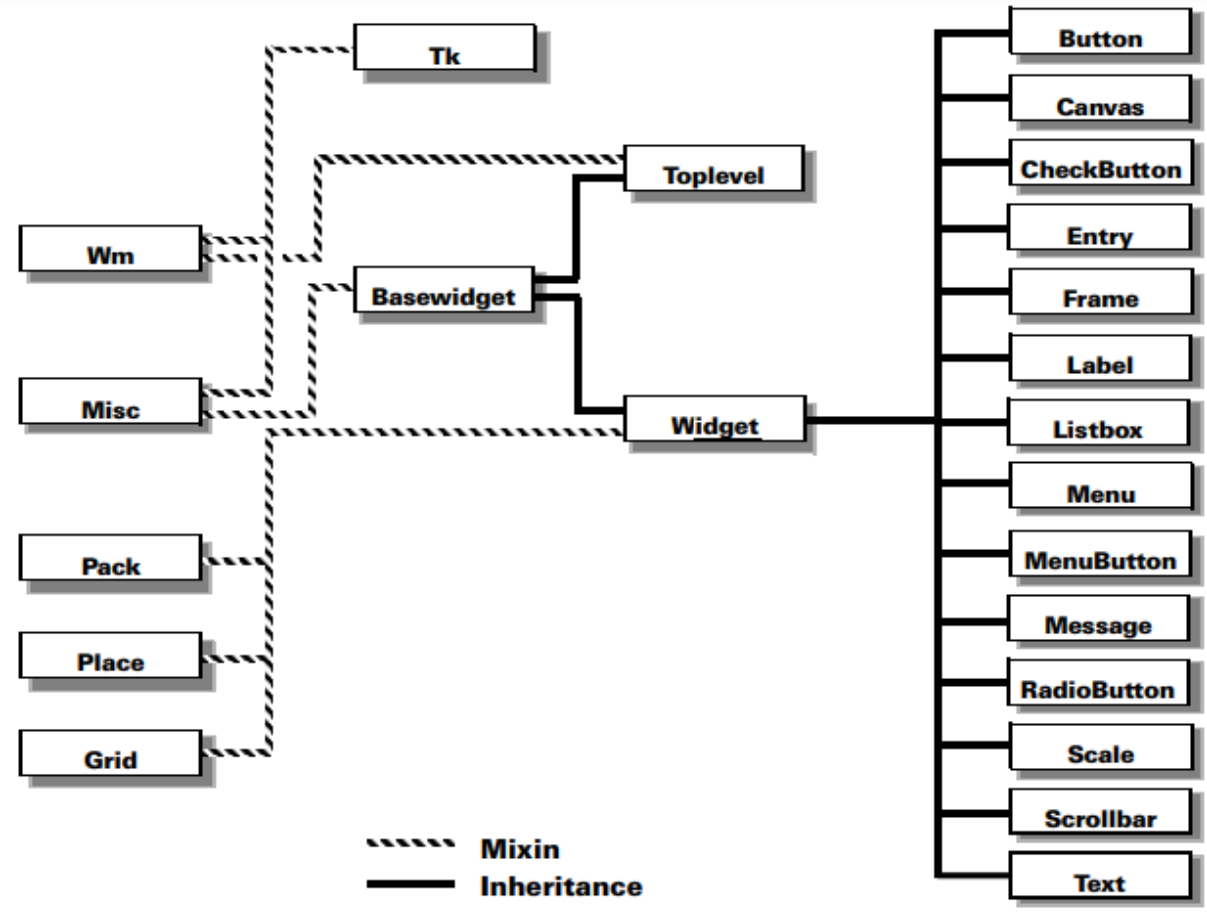


Fonte: Adaptado de Carbonnelle (2018)

2.4.1 Tkinter

Tkinter é um módulo da linguagem Python que propicia o desenvolvimento de interfaces gráficas a partir de programação simplificada e de forma rápida. Desde 1999 é um dos módulos que faz parte da instalação padrão da linguagem, dessa forma, é apenas necessária a importação a partir do comando *import Tkinter* em qualquer sistema que utilize Python. O desenvolvimento é baseado em orientação a objeto, de forma que cada *Widget* é um objeto com seus respectivos métodos e atributos. A Figura 8 representa a simplicidade da hierarquia relacionada a um desses widgets. Para a maioria das aplicações, programadores apenas precisam saber sobre os dois últimos níveis representados (GRAYSON, 2000).

Figura 8 – Hierarquia do Widget Tkinter



Fonte: Grayson (2000)

Para os fins deste trabalho, foram utilizados *Widgets* principais: Label (Rótulo), Entry (Caixa de Entrada) e Button (Botão). Os principais atributos e métodos utilizados de cada uma dessas classes são representadas na Figura 9.

Figura 9 – Classes Tkinter mais utilizadas

Label	Entry	Button
bg (Cor de fundo)	bg (Cor de fundo)	bg (Cor de fundo)
bd (Espessura da Borda)	bd (Espessura da Borda)	bd (Espessura da Borda)
fg (Cor da fonte)	fg (Cor da fonte)	fg (Cor da fonte)
image (Imagem de fundo)	width (Largura)	state (Estado)
text (Texto)		
config (Configurações)	get (Extrai valor)	invoke (Função associada)
	insert (Insere valor)	config (Configurações)

Fonte: O Autor (2020)

2.4.2 Pandas

Pandas é uma biblioteca que tem o objetivo de fornecer bases que possibilitam o desenvolvimento de análises e manipulação de dados em Python. É uma das bibliotecas mais populares da linguagem, o que se dá pelo fato de trabalhar bem com conjunto de dados estruturados, geralmente em formato de tabela. Pandas suporta e pode integrar dados de diferentes fontes, como arquivos json, excel, texto separado por vírgulas, Arquivo de valores separados por vírgula (CSV) ou bancos de dados. O fato de se trabalhar com objetos chamados dataframes, cujas informações possuem forma tabular, conceito bem explorado em tecnologias muito populares como o Microsoft Excel, torna a biblioteca bastante acessível mesmo para além do público programador experiente. A Figura 10 contém um exemplo de dataframe importado a partir de um arquivo CSV (MCKINNEY *et al.*, 2011).

Figura 10 – Importação de dados a partir de arquivo CSV

```
[1] import pandas as pd
    confirmed_cases = pd.read_csv('/content/sample_data/california_housing_test.csv')
    confirmed_cases.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.05	37.37	27.0	3885.0	661.0	1537.0
1	-118.30	34.26	43.0	1510.0	310.0	809.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0
3	-118.36	33.82	28.0	67.0	15.0	49.0
4	-119.67	36.33	19.0	1241.0	244.0	850.0

Fonte: Autor

2.4.3 PyModbus

Pymodbus é uma implementação do protocolo mobdus para utilização a partir de um código em Python. Pode ser utilizado para a implementação de um cliente ou servidor, utilizando TCP, UDP ou conexão serial. Uma de suas vantagens é a possibilidade de testar e integrar grande quantidade de dispositivos, limitada apenas pela quantidade de IPs virtuais suportados pelo sistema operacional utilizado (COLLINS, 2013).

2.4.4 Multiprocessing

Multiprocessing é um pacote que possibilita o programador fazer uso da multiplicidade de processadores em uma máquina. O exemplo padrão das funcionalidades da biblioteca é a utilização do objeto *Pool* que gera execuções paralelas de uma função para cada valor em seu vetor de argumento. A Figura 11 indica um exemplo de utilização presente na documentação oficial da linguagem (PYTHON, 2020).

Figura 11 – Exemplo de utilização de função *Pool*

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

will print to standard output

```
[1, 4, 9]
```

Fonte: Python (2020)

2.5 Proteção de Sobrecorrente

O ocorrência de altos níveis de corrente é, em geral, um indicativo de falha no sistema. Para detectar e proteger os sistemas contra essas falhas, diversos dispositivos são empregados, como disjuntores termomagnéticos, fusíveis e relés de proteção. Por ser a forma de proteção mais comum nos sistemas de potencia, os relés de sobrecorrente são os explorados nessa seção (GERS; HOLMES, 2004).

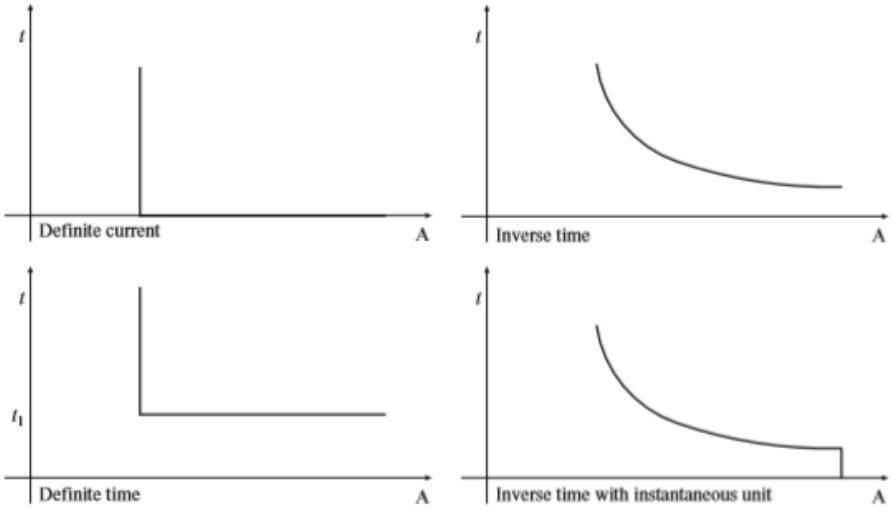
2.5.1 Tipo de relé de sobrecorrente

Os relés de sobrecorrentes podem ser classificados em três grupos de acordo com seu comportamento:

- Corrente definida: Atua instantaneamente perante um nível predeterminado de corrente. Possui coordenação limitada para altos nível de corrente;
- Tempo definido: Pode ser ajustado para que, para um mesmo nível de corrente, diferentes dispositivos atuem em tempos distintos. Sua maior deficiência reside no fato de que faltas próximas à fonte podem ter tempos de resposta relativamente longo;
- Tempo inverso: Possuem tempo de atuação inversamente proporcional à magnitude da corrente de falta, o que é propício para a coordenação. São geralmente classificados de acordo com o a característica da curva de tempo de atuação em função da corrente de percebida.

A Figura 12 contém as curvas características de cada um dos tipos indicados e de um relé de tempo inverso com uma componente de corrente definida, que pode possibilitar maior flexibilidade e robustez à proteção de um sistema elétrico.

Figura 12 – Curvas tempo/corrente caracteristicas dos relés de sobrecorrente



Fonte: Adaptado de Gers e Holmes (2004)

2.5.2 Parâmetros dos relé de sobrecorrente

2.5.2.1 Corrente de pick-up

Para Kindermann (1999), a corrente de *pick-up* é a mínima corrente que pode induzir a atuação do relé. Esse ajuste considera um fator de sobrecorrente que varia de acordo com o dispositivo sendo protegido. É definida como indicado na Equação 2.1:

$$IP = \frac{(OLF * IN)}{CTR} \tag{2.1}$$

Onde:

IP = corrente de *pick-up*

OLF = fator de sobrecorrente

IN corrente nominal do sistema

CTR = relação de transformação do transformador de corrente

2.5.2.2 Dial de tempo

O dial (ou múltiplo) de tempo é o intervalo antes da operação do relé uma vez que a corrente de falta atinge o valor de *pick-up*. Nos relés eletromecânicos, o múltiplo de tempo é definido ajustando a distância entre os contatos móveis. Gers e Holmes (2004) definem a seguinte metodologia de 5 passos para a definição do dial de tempo:

1. Determinar o tempo de operação (T1) para o relé mais distante da fonte (R1) utilizando a menor configuração de dial possível e considerando a corrente que gera a atuação instantânea;
2. Determinar o tempo de operação para o relé seguinte (R2) somando a margem de coordenação, tipicamente de 0,2 e 0,4 segundos, ao valor de T1;
3. Considerando a mesma corrente de falta nos dois dispositivos e o valor de *pick-up* de R2, calcular o dial seu dial de tempo;
4. Novamente calcular o tempo de atuação de R2, agora utilizando a máxima corrente antes de sua atuação instantânea.
5. Continuar o processo para os próximos dispositivos.

2.5.2.3 *Função de atuação do relé*

O Instituto de Engenheiros Eletricistas e Eletrônicos Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) define o tempo de operação dos relés de tempo inverso a partir da Equação 2.2:

$$t = \frac{K\beta}{(I/I_s)^{\alpha} - 1} + L \tag{2.2}$$

Onde:

t = Tempo de atuação do relé (s)

k = Multiplicador de tempo escolhido (s)

I = Corrente de Falta

Is = Corrente de *pick-up*

Os valores α, β e L são constantes características de cada tipo de curva e seus valores definidos pela IEEE são indicados na Tabela 2.

Tabela 2 – Valores IEEE para curvas de atuação de relés

Curva	α	β	L
Moderadamente Inversa	0,02	0,0515	0.1140
Muito Inversa	2,0	19,61	0.4910
Extremamente inversa	2,0	28,20	0.1217

Fonte: Gers e Holmes (2004)

2.6 Considerações Finais

Considerando as informações apresentadas neste caítulo decidiu-se desenvolver o projeto utilizando-se ScadaBR, Protocolo Modbus TCP e linguagem Python.

O ScadaBR se apresenta como uma alternativa *open source* muito interessante por ser multiplataforma e apresentar interface simples.

O protocolo Modbus TCP foi escolhido não somente por sua popularidade entre os dispositivos como os utilizados na proteção do sistema elétrico como também por sua facilidade

de interface com tecnologias *open source*, inclusive, por possibilitar a transmissão por diferentes meios físicos.

Considerando a programação, foram utilizadas 3 bibliotecas que merecem destaque, que são *Tkinter*, para construção de interface gráfica; *Pandas* para manuseio de dados e *PyModbus*, para possibilitar a leitra e escrita a partir do protocolo Modbus.

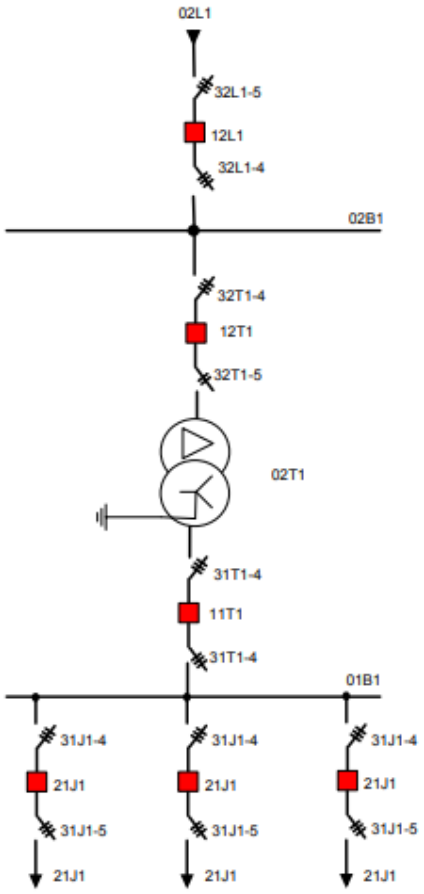
3 METODOLOGIA

3.1 Introdução

Neste capítulo é descrita a implementação de um simulador de sistema elétrico a partir do qual foram definidos equipamentos que se comunicam, bidirecionalmente, com o sistema ScadaBR. Isso foi feito buscando-se explorar o desenvolvimento de interface gráfica, da implementação computacional do comportamento de equipamentos elétricos, da correta configuração do sistema ScadaBR e de uma comunicação satisfatória entre os sistemas a partir do protocolo Modbus.

O sistema escolhido para a simulação é uma subestação cuja topologia é representada na Figura 13. Essa escolha se deu tanto pela simplicidade do sistema, quanto por sua popularidade em sistemas industriais. Também possui múltiplos relés e um transformador cuja modelagem está de acordo com os objetivos desse trabalho.

Figura 13 – Diagrama unifilar.



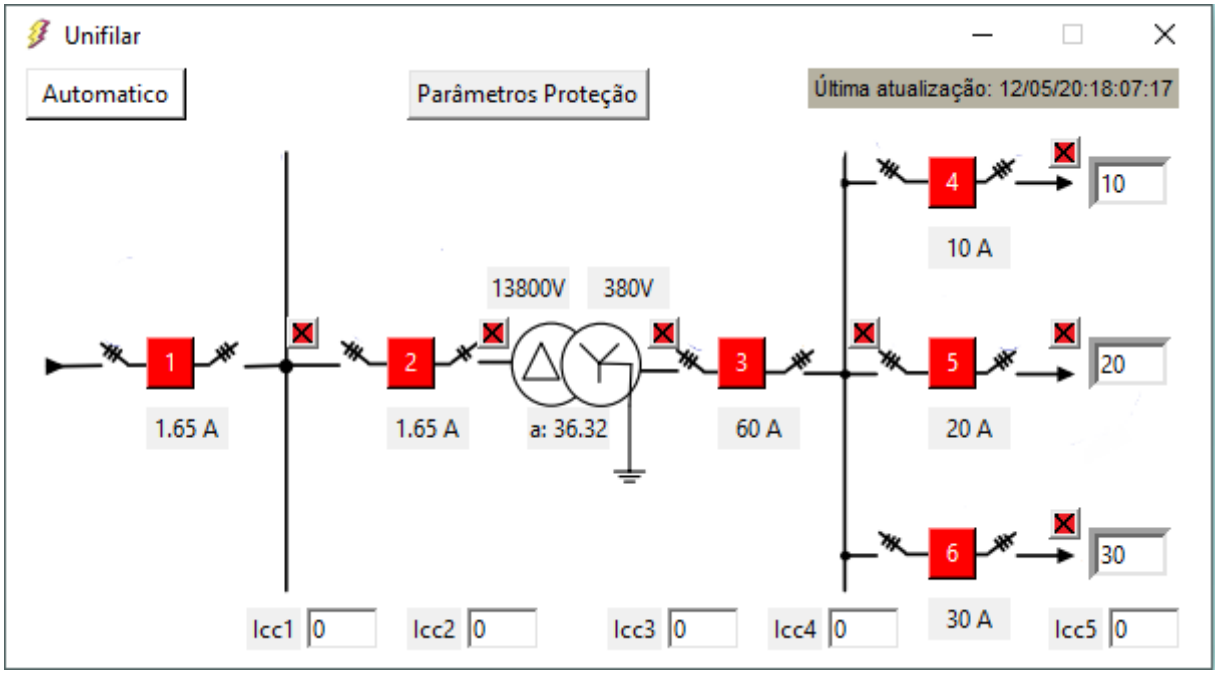
Fonte: O Autor (2020).

3.2 Desenvolvimento do simulador

3.2.1 Interface Gráfica

A fim de desenvolver o sistema proposto, foi definido um esquemático de subestação a partir do qual foi desenvolvida a interface gráfica do simulador de proteção e controle o qual foi integrado ao ScadaBR. A interface representada na Figura 14 foi desenvolvida a partir da biblioteca *tkinter*. Nela são representados Relés e indicadores de falha, com botões; Valores de corrente de carga e de curto em cada parte do sistema, com *entry boxes*; e grandezas não diretamente configuráveis como tensão e corrente, rótulos simples. A atualização é realizada a cada 1 segundo e o estado dos relés pode ser definido de forma manual ou automática, o que é definido a partir da interação com o botão na parte superior esquerda.

Figura 14 – Interface Gráfica do Simulador de Proteção e Controle de Subestação



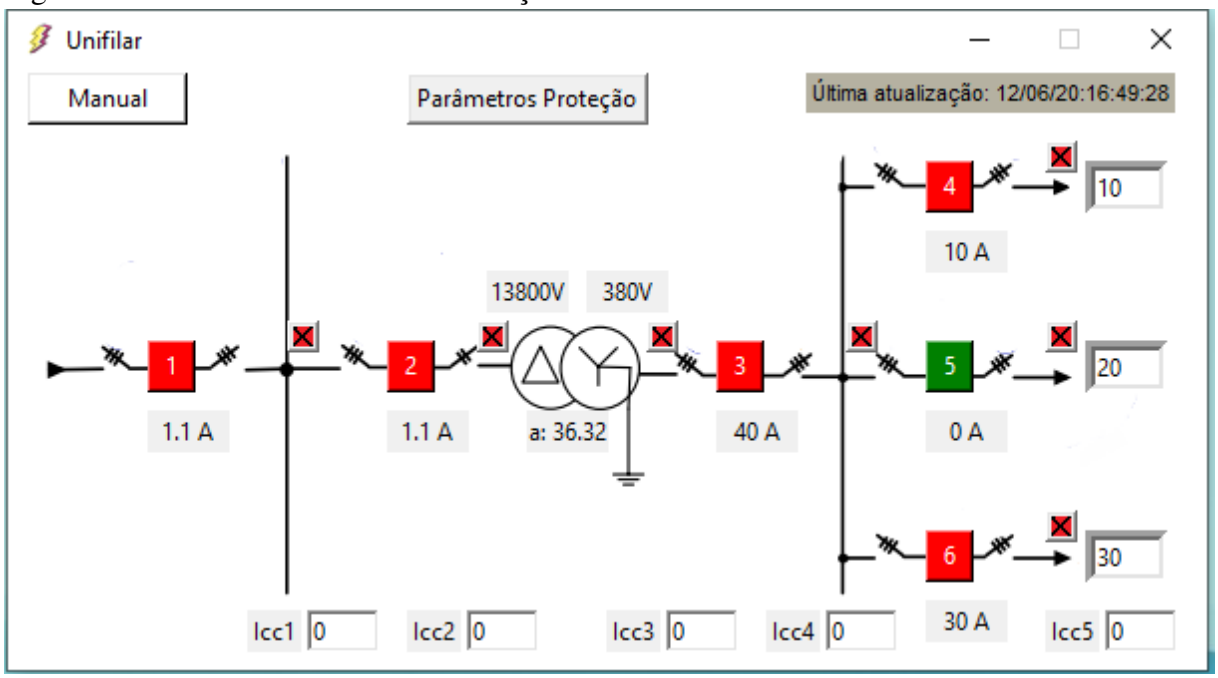
Fonte: O Autor (2020)

A cada atualização os dados das variáveis relacionadas a cada relé são salvos em um arquivo com valores separados por vírgula (CSV). Caso o botão superior esquerdo esteja no estado "Automático" também são lidas e executados os comandos advindos do sistema SCADA, que são armazenados em outro arquivo CSV.

A função primária da simulação é, definidas as correntes de carga, indicar a corrente em cada dispositivo e permitir a abertura ou fechamento dos disjuntores relacionados aos relés de proteção. Na Figura 15 é ilustrada a mudança do estado do disjuntor 5 para aberto (cor

verde) após a simulação de sua abertura manual. Pode-se observar que a abertura do disjuntor interrompe a alimentação da carga 5 (corrente 0 A) e a mudança dos valores das grandezas medidas pelos relés a montante.

Figura 15 – Sistema simulado com atuação manual



Fonte: O Autor (2020)

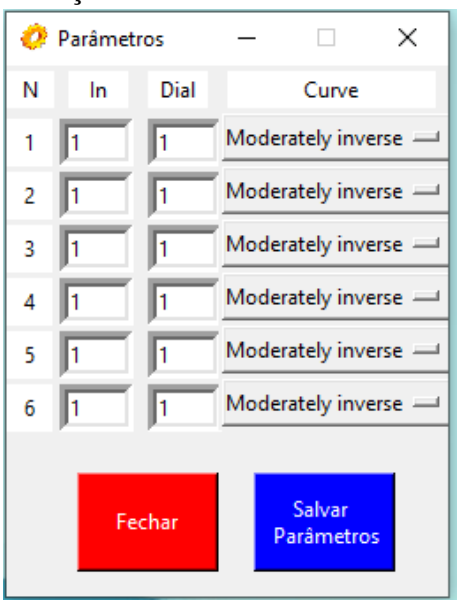
3.2.2 Simulação de falta

Além da abertura manual via simulador e do comando pelo SCADA, os relés podem atuar de acordo com a simulação de um curto-circuito em algum dos pontos indicados com os pequenos botões vermelhos com "X" próximos a cada relé. A corrente de curto em cada ponto deve ser indicada, em amperes, nas caixas inferiores da tela, sendo cada uma referente ao curto nos pontos na mesma linha vertical.

O tempo de atuação e a coordenação da proteção ocorre de acordo com as equações e parâmetros da IEEE indicados na Tabela 2. A janela para definição dos parâmetros do relé é acessada a partir do clique no botão "Parâmetros de Proteção". Como pode ser observado na Figura 16, podem ser definidos o múltiplo de corrente, o dial de tempo e o tipo de curva a ser seguida em cada um dos dispositivos. Após a definição dos valores, deve-se pressionar o botão "Salvar Parâmetros" para que sejam considerados no cálculo dos tempos de atuação.

Uma vez definidas as correntes de curto para cada ponto e parametrizado cada relé, pode-se simular uma situação de falta. Clicando em algum dos botões indicados com "X",

Figura 16 – Janela de parametrização dos relés



Fonte: O Autor (2020)

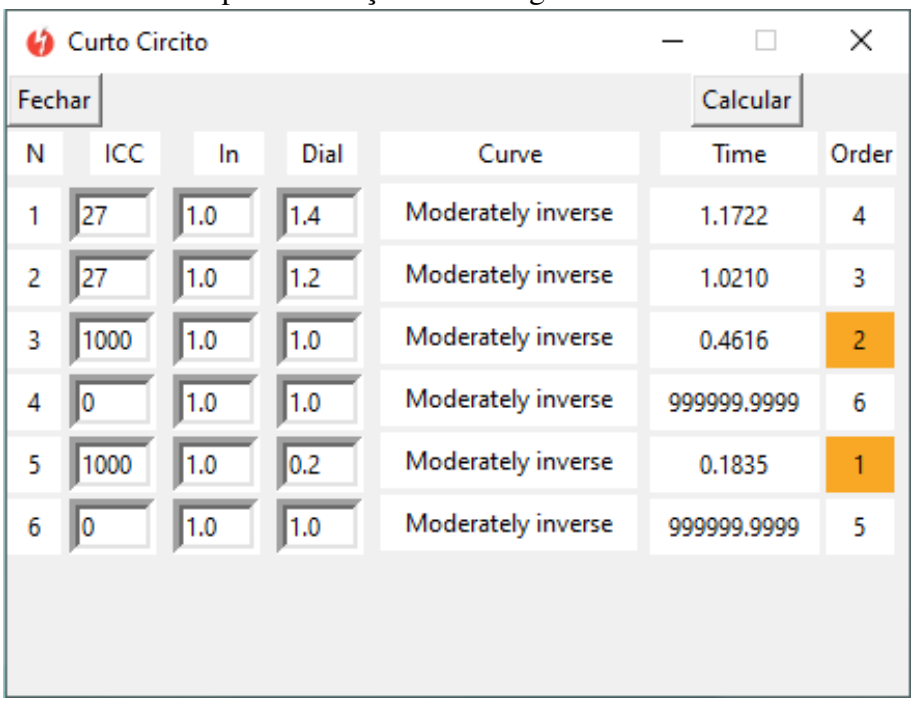
abre-se a janela indicada na Figura 17. Nela são indicados os parâmetros definidos, corrente de curto em cada equipamento, de acordo com a corrente definida para o ponto selecionado, e o tempo de atuação calculado. Caso os tempos de atuação sejam muito próximos, ou seja, com diferença inferior ao intervalo de coordenação definido, nenhuma modificação é realizada nos estados dos dispositivos simulados. Caso a coordenação seja satisfeita, o dispositivo a atuar é destacado com a cor verde, como pode ser observado na Figura 18 e ocorre a mudança do estado no diagrama suimulado.

3.3 Comunicação Modbus

Uma vez definidos e exportados os dados do sistema, é necessário simular a presença dos componentes na rede. Tal tarefa é possibilitada pelo uso da biblioteca *Pymodbus*, permitindo a criação de servidores para leitura e escrita. O protocolo utilizado foi o Modbus TCP/IP. A conexão é realizada a partir de um endereço e uma porta de comunicação definidos para cada dispositivo que se deseja simular.

Foi escolhido o número de 20 para a quantidade de variáveis a serem transmitidas para cada um dos dispositivos. Dessas, 10 são para a escrita e 10 para a recepção de informações vindas do SCADA. Isso significa que o programa de simulação se relaciona com 2 arquivos CSV com 10 linhas de informação para cada um dos dispositivos simulados. Para garantir a disponibilização contínua desses dados para o sistema Scada, são executados 6 programas em paralelo. Foi verificado que dedicar um processo separado para cada via de comunicação aberta

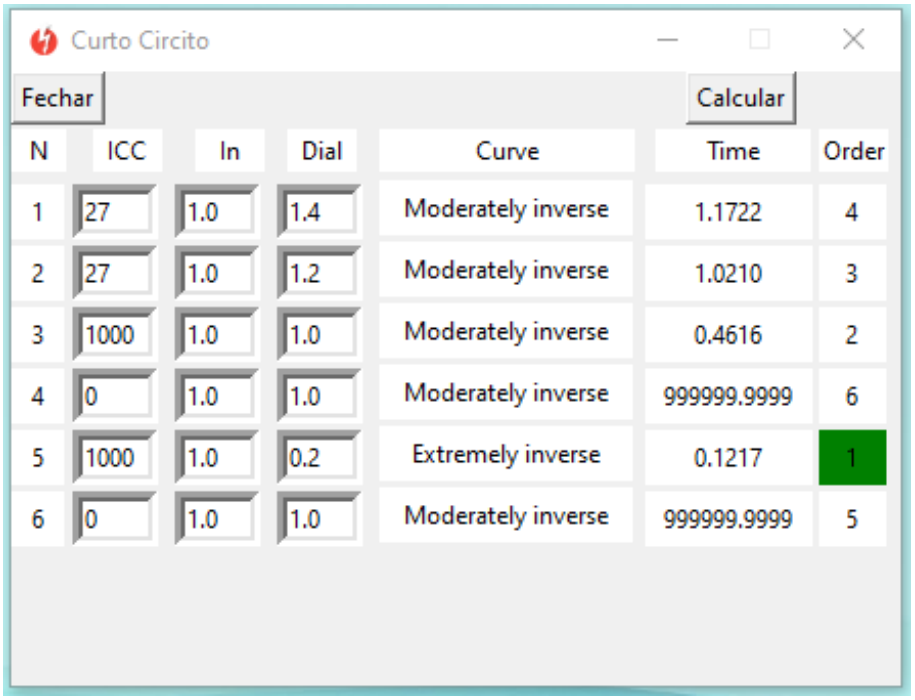
Figura 17 – cálculo dos tempos de atuação com margem insatisfeita



N	ICC	In	Dial	Curve	Time	Order
1	27	1.0	1.4	Moderately inverse	1.1722	4
2	27	1.0	1.2	Moderately inverse	1.0210	3
3	1000	1.0	1.0	Moderately inverse	0.4616	2
4	0	1.0	1.0	Moderately inverse	999999.9999	6
5	1000	1.0	0.2	Moderately inverse	0.1835	1
6	0	1.0	1.0	Moderately inverse	999999.9999	5

Fonte: O Autor (2020)

Figura 18 – cálculo dos tempos de atuação com margem satisfeita



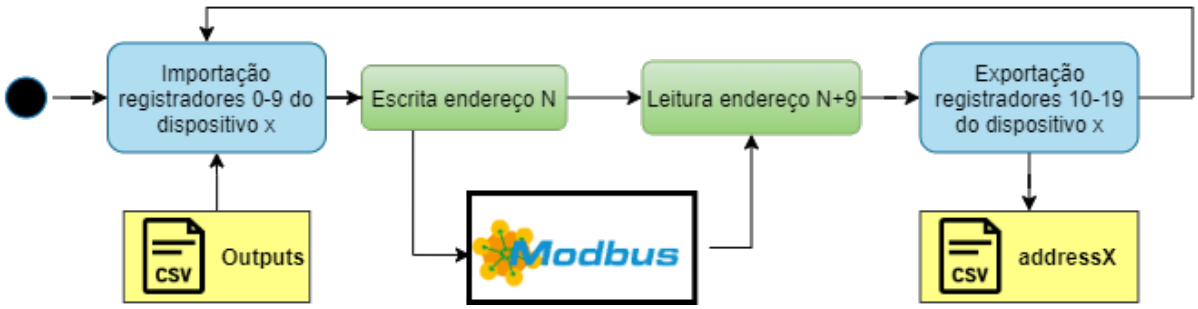
N	ICC	In	Dial	Curve	Time	Order
1	27	1.0	1.4	Moderately inverse	1.1722	4
2	27	1.0	1.2	Moderately inverse	1.0210	3
3	1000	1.0	1.0	Moderately inverse	0.4616	2
4	0	1.0	1.0	Moderately inverse	999999.9999	6
5	1000	1.0	0.2	Extremely inverse	0.1217	1
6	0	1.0	1.0	Moderately inverse	999999.9999	5

Fonte: O Autor (2020)

é crucial para a garantir uma comunicação efetiva no sistema. Essa estratégia de multiprocessamento é possibilitada pelo uso da função *pool* do módulo *pymodbus*. Cada um dos processos se mantém, continuamente, importando os arquivos provenientes da simulação e escrita dos valores nos endereços de 0 a 9 de cada servidor criado para um determinado dispositivo, assim, disponibilizando os dados para serem lidos pelo Scada. Após a escrita, é realizada a leitura dos

espaços destinados aos inputs do Scada, 10 a 19. Esses dois processos acontecem em ciclos de 10 repetições, onde é realizada a escrita no endereço "N" e a leitura no endereço "N+9". O processo é representado na Figura 19.

Figura 19 – Diagrama representativo da comunicação com servidor Modbus



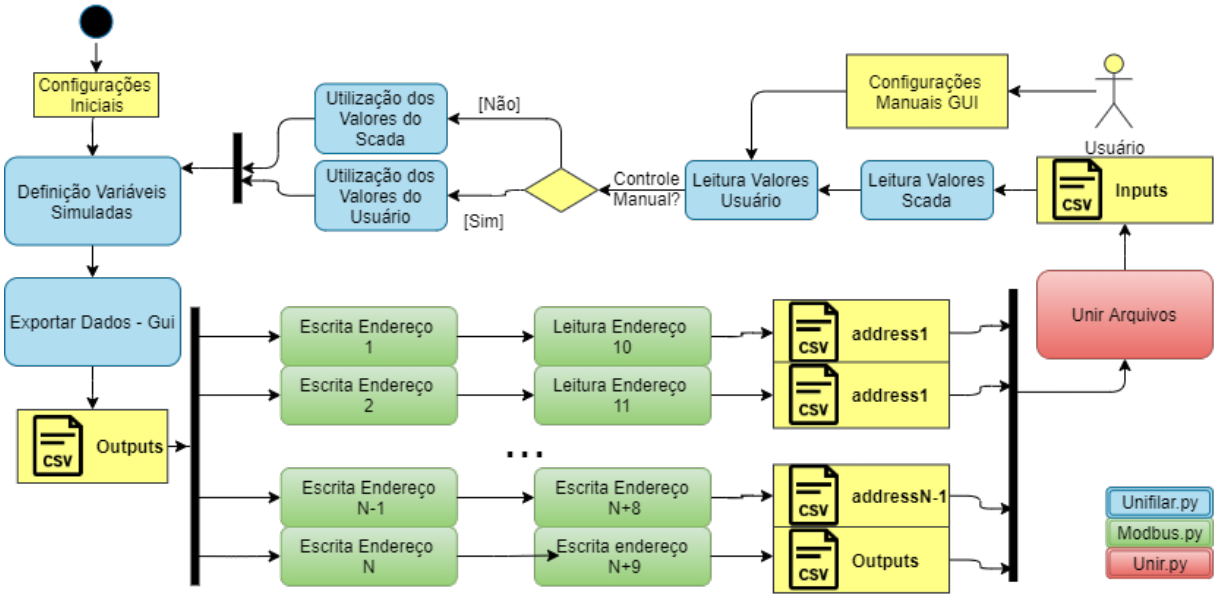
Fonte: O Autor (2020).

3.4 Fluxo de atividades

Os processos já indicados nesta seção ocorrem continuamente, garantido a interação contínua com o sistema SCADA. São utilizados 3 programas distintos, sendo um responsável pela interface e simulação, outro pela comunicação por meio do protocolo modbus e um terceiro atuando apenas como auxiliar na comunicação entre eles. Foram escolhidos arquivos CSV como meio de comunicação entre os programas tanto pela simplicidade de implementação, quanto pela fácil integração com outros softwares, como o Microsoft Excel. Essa integração pode facilitar não somente no monitoramento da integridade das informações durante os testes, como também facilita a introdução de novas aplicações. O diagrama das atividades envolvidas nesse processo pode ser observado na Figura 20, onde cada cor representa o programa responsável por cada atividade. As cores azul, verde e vermelho representam os programas desenvolvidos e a cor amarela representa os elementos não programados. Os códigos fontes desses programas estão disponíveis nos Apêndices A e B.

O primeiro programa, "Unifilar.py" é responsável pela leitura dos valores enviados pelo SCADA, por toda simulação e visualização do sistema, pela interação com o usuário e pela exportação sas variáveis do sistema simulado. O segundo, "Modbus.py", verifica o os dados vistos do primeiro e, de acordo com a quantidade de endereços presentes, inicia a mesma quantidade de programas em paralelo para leitura e escrita de informações pelo protocolo modbus. Após a leitura dos dados vindos do SCADA, cada um dos programas em paralelo gera um arquivo no formato "127.2.2.x.CSV", escolhido arbitrariamente. O terceiro programa,

Figura 20 – Diagrama de atividades do sistema desenvolvido



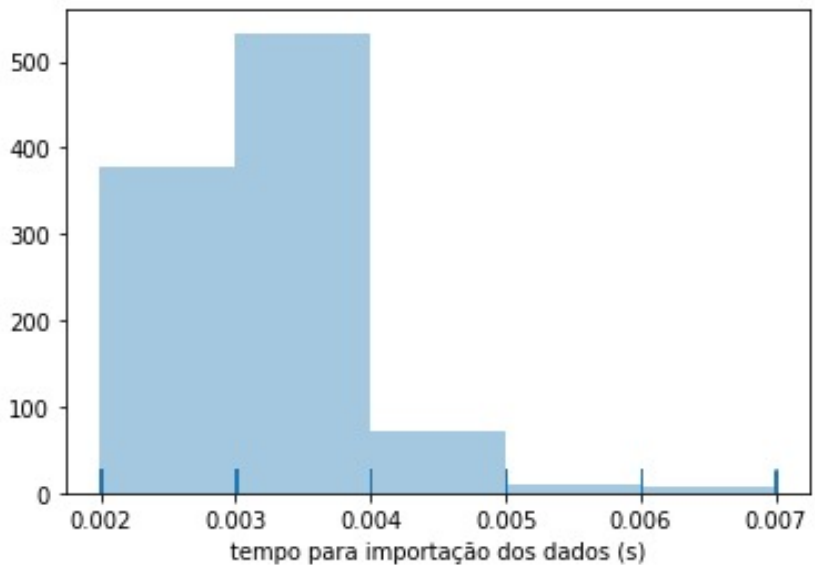
Fonte: O Autor (2020).

"Unir.py" é responsável por buscar os arquivos no formato citado e unificar seus dados, facilitando a importação pelo primeiro programa e deixando as informações acessíveis ao outro. Uma vez com os dados unificados, eles são lidos pelo "Unifilar.py" que, de acordo com a seleção do comando na interface, atualiza o estado dos dispositivos de acordo com eles ou espera a indicação manual do usuário para mudar algum dos estados.

Vale notar que o intervalo definido para a conclusão do ciclo de atividades é, aproximadamente, de 1 segundo para cada um dos programas citados. Esse tempo foi considerado suficiente para gerar um experiencia satisfatória de testes. Contudo, uma vez que os processos intermediários ocorrem em cerca de alguns milésimos de segundo, seria possível realizar uma comunicação mais rápida, se fosse necessário para a aplicação. A Figura 21 retrata os resultados de 1000 testes para a importação dos dados. Cerca de 90% das importações ocorreram em até 4 milésimos de segundo.

O processo citado ocorre continuamente de forma independente da simulação de falta. Caso ela ocorra e acarrete a mudança em algum dos estados, o controle é alterado para manual. Essa mudança ocorre de forma similar à mudança manualmente configurada pelo usuário.

Figura 21 – histograma de tempo de para importação dos dados em 1000 testes.



Fonte: O Autor (2020).

3.5 Desenvolvimento do sistema SCADA

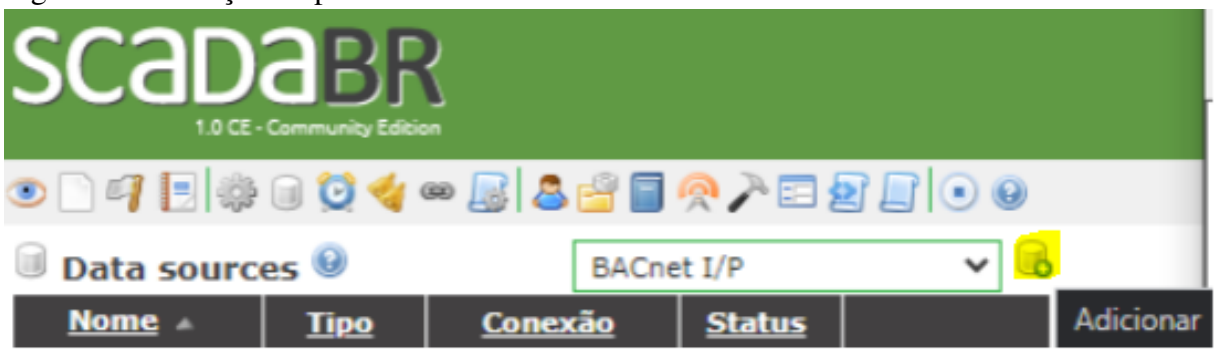
Apesar de se destacar pela simplicidade, o ScadaBR necessita de diversos passos de configuração para se comunicar com cada um dos dispositivos simulados e para, de forma satisfatória, representar e possibilitar a configuração de valores. Esses passos se iniciam na definição de cada fonte de dados, *data source*, e posterior definição dos dados específicos, *data points*, a serem observados até a construção de interface gráfica que possibilite todas as interações necessárias ao projeto.

3.5.1 Data sources

As fontes de dados podem ser qualquer *softwares* que ofereça suporte a algum dos protocolos implementados no ScadaBR. Para definir um novo *data source*, deve-se acessar o menu de mesmo nome e, primeiramente, selecionar o protocolo desejado. Dada sua popularidade, inclusive em relés de proteção, e pelo fato de ser livre, foi utilizado o protocolo Modbus IP. Uma vez escolhido o protocolo na lista, deve-se clicar na opção "adicionar", como indicado na Figura 22.

Após a adição da fonte de dados, ela deve ser configurada em diversos parâmetros, como o seu nome, taxa de atualização e os alarmes para cada tipo de situação adversa. Além desses, o mais importante parâmetro é o endereço, composto por "Host" e "Porta". Esses valores devem coincidir com os valores disponíveis na Tabela 4, especificando cada equipamento. Todas

Figura 22 – Seleção do protocolo da fonte de dados



Fonte: O Autor (2020).

as fontes de dados envolvidas neste trabalho possuem configurações similares às indicadas na Figura 23.

3.5.2 Data Points

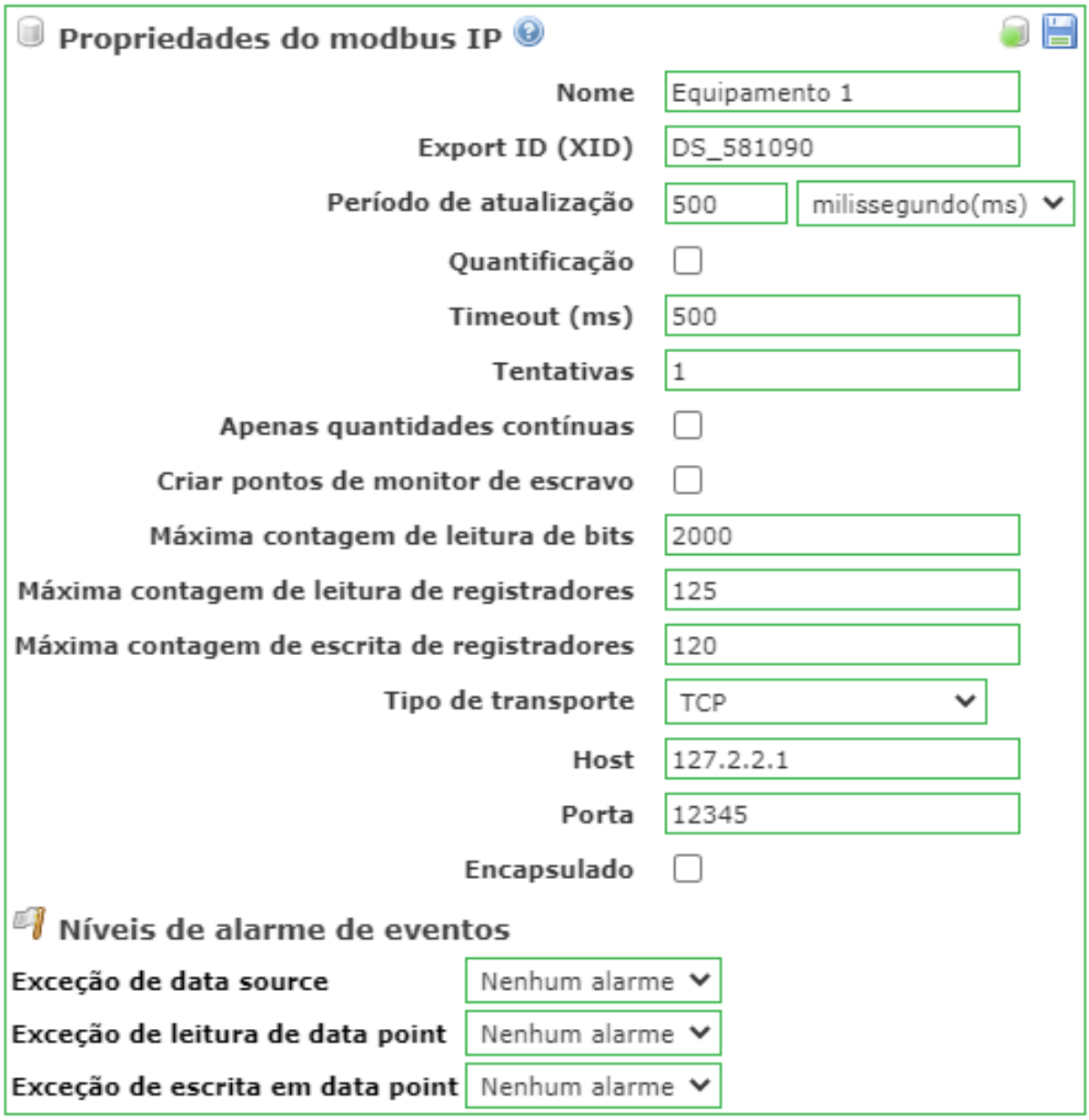
Os *data points* são as variáveis de interesse, que podem representar uma leitura ou uma escrita, dependendo do seu tipo. Eles são adicionados na parte inferior da mesma página que as fontes de dados, a partir do botão destacado na Figura 24.

A identificação de cada dado é definida pelo parâmetro *offset*. Seguindo o padrão definido nos programas desenvolvidos, os *offfsets* de 0 a 9 serão os que apenas representaram leituras de dados enquanto os de 10 a 19 são reservados para dados que podem ser definidos pelo Scada. Dessa forma, a configuração "Faixa de registro" foi definida como "Registador de entrada" para os primeiros e como "Registrador de holding" para os últimos. A escolha dessas Faixas se dá pelo fato de possibilitarem a utilização de dados do tipo binário e inteiro de 2 Bytes, possibilitando a representação de valores digitais e analógicos. A configuração de um desses *data points* de entrada é exemplificada na Figura 25.

3.5.3 Monitoramento e escrita de dados

Uma vez criados e ativados todos os *data sources* e *data points*, a interação com os dados ja é possivel a partir da página de *watch list*. Nela, estão listados todos os *data points* como pode ser observado na Figura 26. Para adicionar as variáveis à visualização, basta clicar na seta ao lado de cada item. Uma vez adicionados, podem ser obsersados os valores e a hora de atualização de cada item. Para os *points* do tipo "Registrador de holding", pode-se definir os valores a partir do clique na imagem de chave de fenda ao lado do nome do *data point*. A Figura 26 mostra a watchlist com alguns dos *data points* utilizados neste trabalho.

Figura 23 – Propriedades da fonte de dados

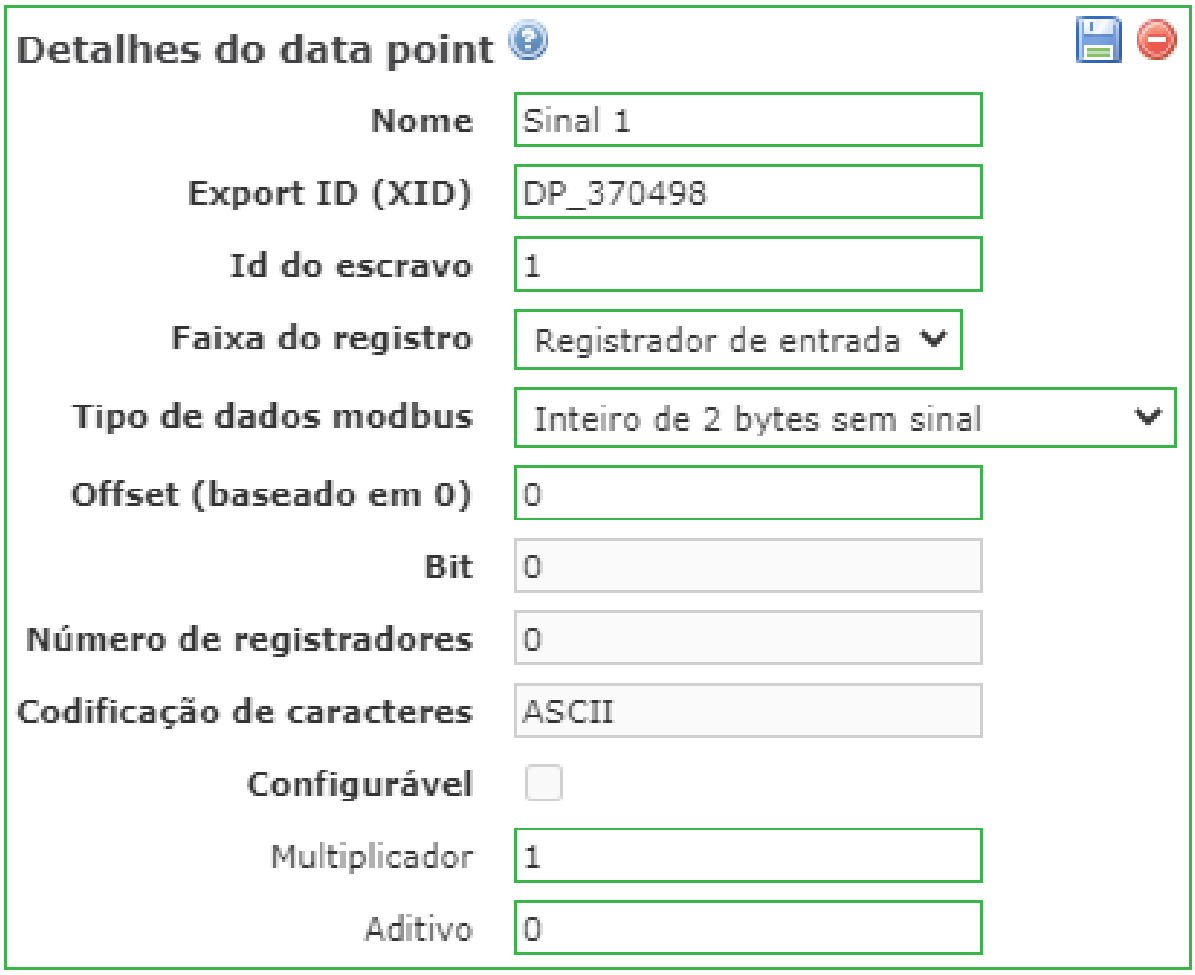


Fonte: O Autor (2020).

Figura 24 – Lista de data points com indicação do botão "adicionar"

Fonte: O Autor (2020).

Figura 25 – Propriedades do *data point*.



Fonte: O Autor (2020).

Figura 26 – Visualização da *Watch List*.

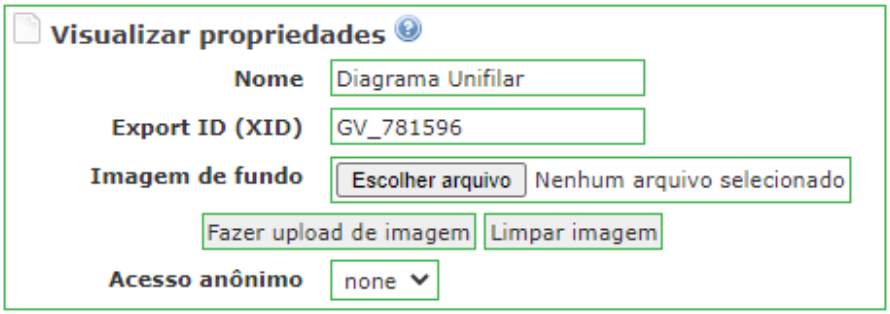
Fonte: O Autor (2020).

3.5.4 Interface gráfica Scada

Além da *Watch List*, o ScadaBR também possibilita a integração dos *data points* com elementos gráficos a partir da a página "Representação Gráfica". Após selecionar a opção "Nova Representação", o usuário visualiza a janela indicada na Figura 27, onde é definido o nome da representação e se pode adicionar o plano de fundo, funcionalidade aproveitada para exibir o unifilar do sistema elétrico simulado. Das 18 opções de objetos gráficos, foram utilizados 3, descritos na Tabela 3, para representar o sistema simulado na visão do SCADA. É necessária a

inclusão e configuração de um objeto por vez. Cada objeto é associado a um *data point* específico. Clicando nos objetos relacionados com grandezas configuráveis é possivel mudar seus valores na própria tela de visualização. A representação final se assemelha à interface gráfica desenvolvida em python e está representada na Figura 28.

Figura 27 – Configurações iniciais da representação gráfica no ScadaBR.



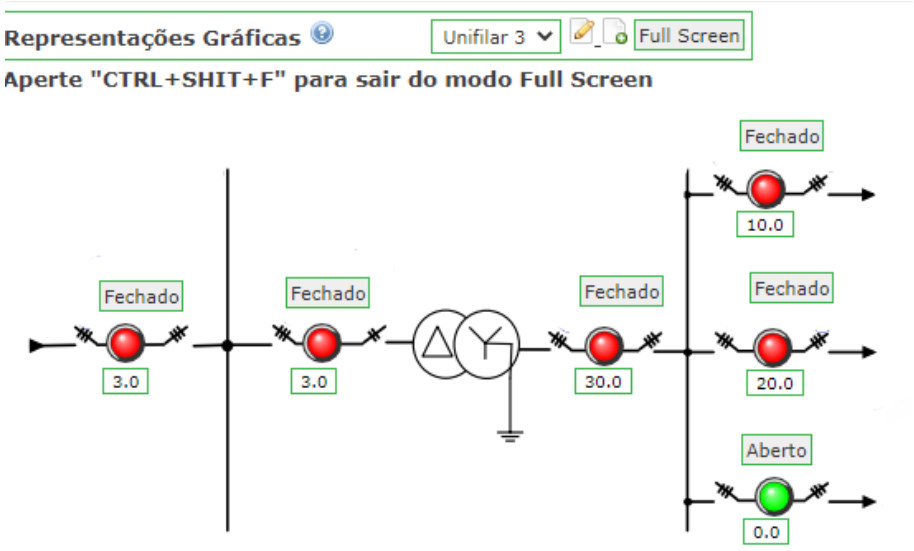
Fonte: O Autor (2020).

Tabela 3 – Componentes Visuais Utilizados no Scada

Componente	Tipo de dado	Representação
Gif Binário	Binário	Estado dos Relés
Data point simples	Inteiro sem sinal	Intensidade das correntes
Botão (Escrita)	Binário	Comando Abertura/Fechamento Relé

Fonte: O Autor (2020).
Nota: Enderços e portas definidas arbitráriamente.

Figura 28 – Configurações iniciais da representação gráfica.



Fonte: O Autor (2020).

4 RESULTADOS

Neste capítulo são apresentados os testes para validação do sistema desenvolvido. São apresentados: os parâmetros definidos na configuração dos *softwares*, o funcionamento da interação entre o simulador e o sistema ScadaBR e um estudo de caso que exemplifica a funcionalidade de simulação da proteção.

4.1 Parâmetros utilizados

Considerando a comunicação, foram definidos 6 endereços para representar cada um dos 6 relés do sistema. Os valores escolhidos podem ser observados na Tabela 4. Eles são utilizados no arquivo de transmissão dos dados como indicado na Tabela 5. Observe-se que apenas duas linhas são utilizadas por dispositivo, a primeira para representar o valor da corrente e a segunda o estado (aberto ou fechado) do relé. Os demais possuem um valor padrão de "999".

Tabela 4 – Dispositivos e endereços

Dispositivo	Endereço	Porta
Relé 1	127.2.2.1	12345
Relé 2	127.2.2.2	12345
Relé 3	127.2.2.3	12345
Relé 4	127.2.2.4	12345
Relé 5	127.2.2.5	12345
Relé 6	127.2.2.6	12345

Fonte: O Autor (2020).

Nota: Enderços e portas definidas arbitráriamente.

Essas informações serão lidas pelo programa "Modbus.py"e disponibilizadas a partir do protocolo Modbus para que sejam captados pelo ScadaBR e, posteriormente, outras serão lidas para que as informações definidas pelo SCADA sejam transmitidas para o simulador. Para que isso ocorra, é necessário que as mesmas informações de endereço e porta sejam configuradas nos *data sources* e *data points* utilizados. As configurações de *data sources* utilizadas podem ser observadas na Figura 29 e a dos *data points* são como as já apresentadas na Figura 24.

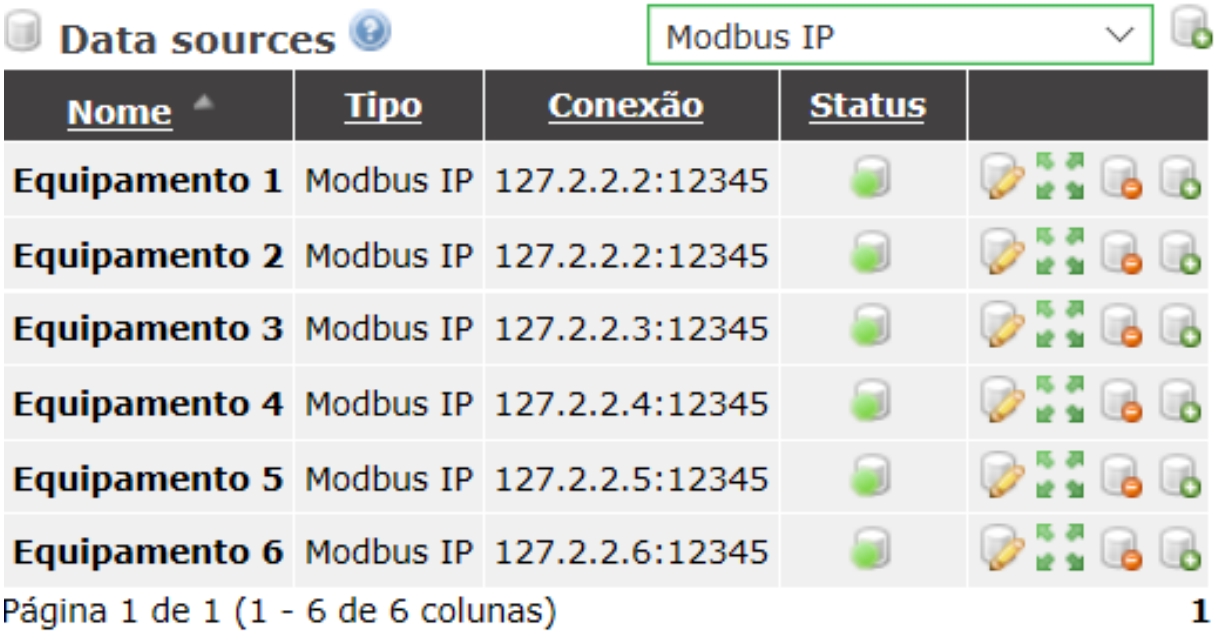
Quanto aos tempos de atualização dos dados, tanto no Scada quanto na interface simulada foram definidos tempos de 1 segundo. No Scada, essa é uma configuração simples, efetuada nas configurações principais do *data point*, já para executar a atualização na interface, é necessário agendar uma interrupção, que pode ser observada nas linhas 98 a 101 do Apêndice A.

Tabela 5 – Dados de saída associados a cada dispositivo.

Address	Base	N	Port	Offset	Value
127.2.2.1	0127.2.2.	1	12345	0	1100
127.2.2.1	0127.2.2.	1	12345	1	0
127.2.2.1	0127.2.2.	1	12345	2	999
127.2.2.1	0127.2.2.	1	12345	3	999
127.2.2.1	0127.2.2.	1	12345	4	999
127.2.2.1	0127.2.2.	1	12345	5	999
127.2.2.1	0127.2.2.	1	12345	6	999
127.2.2.1	0127.2.2.	1	12345	7	999
127.2.2.1	0127.2.2.	1	12345	8	999
127.2.2.1	0127.2.2.	1	12345	9	999
127.2.2.2	0127.2.2.	2	12345	0	1100
127.2.2.2	0127.2.2.	2	12345	1	0
127.2.2.2	0127.2.2.	2	12345	2	999
127.2.2.2	0127.2.2.	2	12345	3	999

Fonte: O Autor (2020).
Nota: Representação parcial.

Figura 29 – *Data sources* utilizados

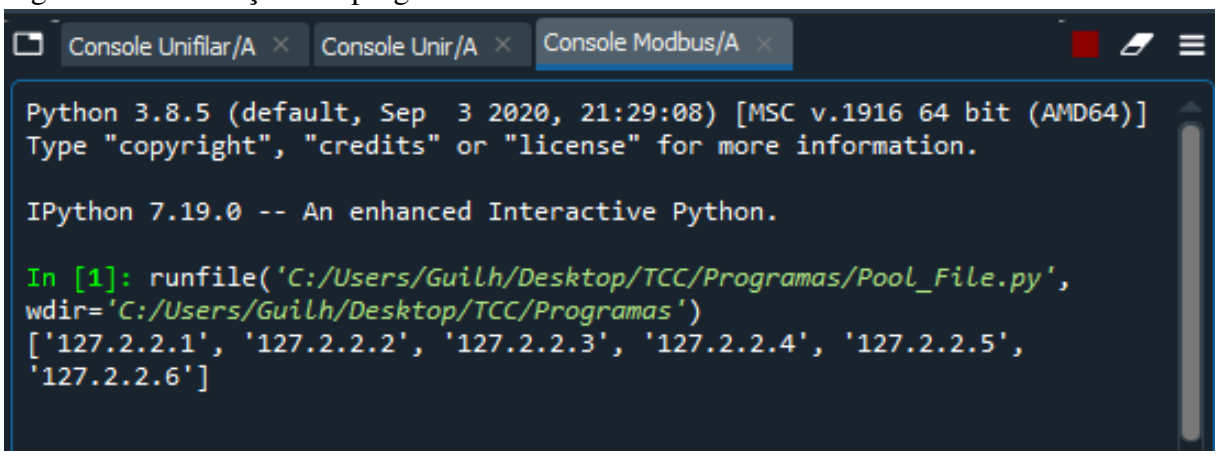


Fonte: O Autor (2020).

4.2 Execução dos programas

Para utilizar o sistema desenvolvido é necesário iniciar os 3 programas a partir de consoles Python e abrir o ScadaBR. Pela simplicidade em se utilizar múltiplos consoles, foi utilizada a plataforma *Spyder* para os testes. A execução deles na plataforma é representada na Figura 30. Contudo, qualquer outro ambiente que permite a utilização de múltiplos consoles poderia ser utilizado.

Figura 30 – Execução dos programas desenvolvidos.



Fonte: O Autor (2020).

Uma vez com os programas em execução nos terminais python, a interface gráfica já está disponível e o SCADA já começa a receber os valores assim que iniciado, sem a necessidade de nenhum passo adicional. Pode-se observar na Figura 31 o funcionamento da transmissão dos dados da interface para o ScadaBR. Os "Sinais 1" representam os valores de corrente. Para manter a comunicação por números inteiros, os valores transmitidos representam miliamperes.

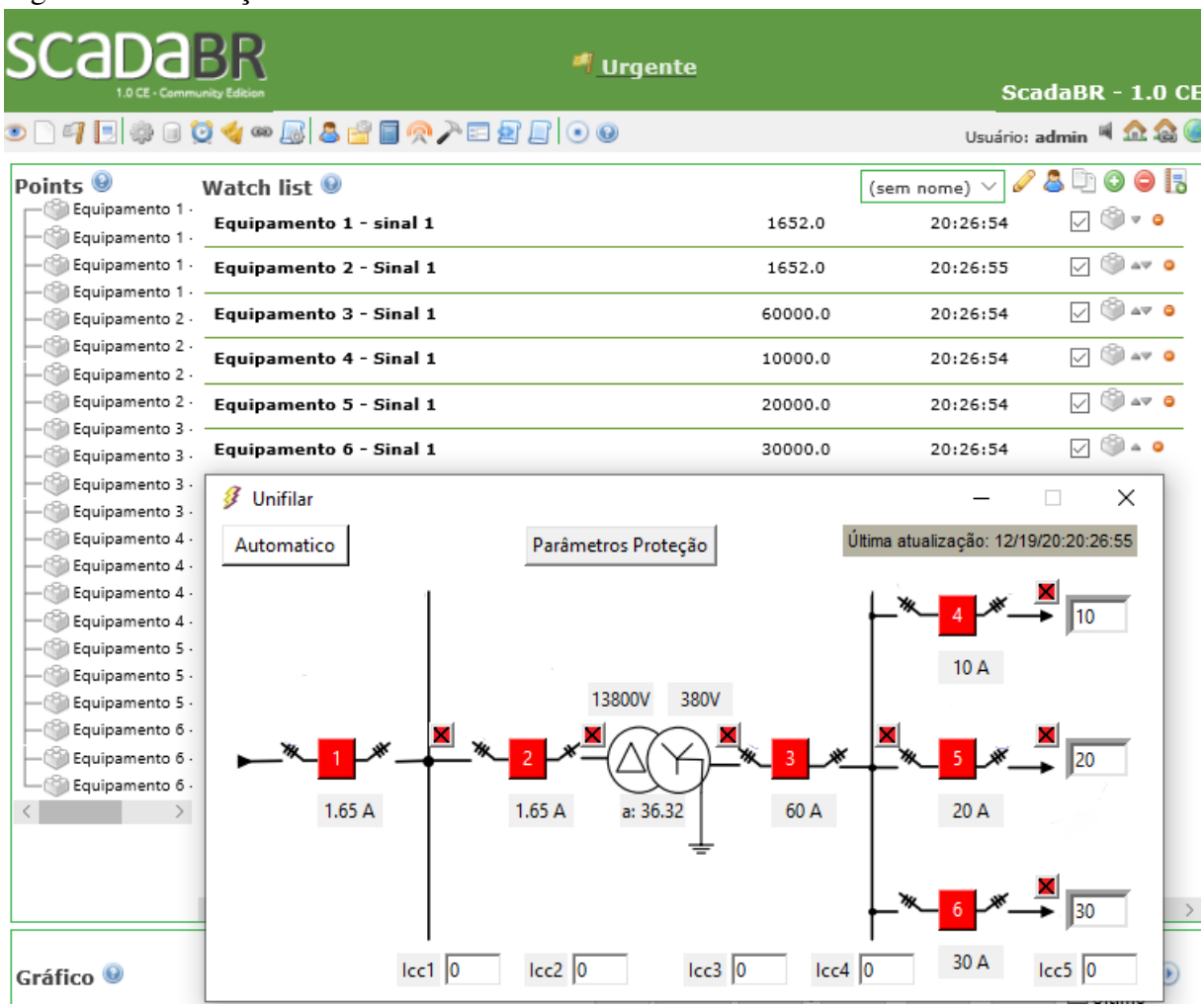
Alterando-se os valores de carga e o estado de um dos relés, os novos parâmetros são calculados e as informações recebidas pelo SCADA. Pode-se observar um exemplo dessas mudanças na comparação entre a Figura 31 e 32. O Sinal 2 representa o estado aberto (1) ou fechado (0) de cada relé. Para que essa mudança de estado ocorra pela interface é necessário mudar o modo de atualização para manual e clicar na representação. No modo "Automático" a alteração deve ser feita pelo SCADA, como indicado na seção 3.4.3.

4.3 Atuação dos Relés

A função de seletividade da proteção deve garantir que o chave a ser aberta é a mais próxima da fonte possível, de forma a garantir o menor impacto possível no sistema. Considerando o sistema escolhido com um transformador e 3 cargas, um exemplo seria o caso de falta na zona de proteção do relé 5. Nesse caso, os parâmetros da proteção devem garantir que o a atuação do relé 5 seja mais rápida do que a dos relés 1,2 e 3 que, apesar de serem atravessados pela corrente de curto, afetariam as outras cargas desnecessariamente.

Para demonstrar o cálculo da atuação dos relés foi realizado um teste para uma corrente de 1800 amperes na zona de proteção do relé 5 e busca-se garantir a coordenação com os relés a montante. Essa corrente é indicada na parte inferior direita da janela "Unifilar", que

Figura 31 – Interação entre as interfaces.



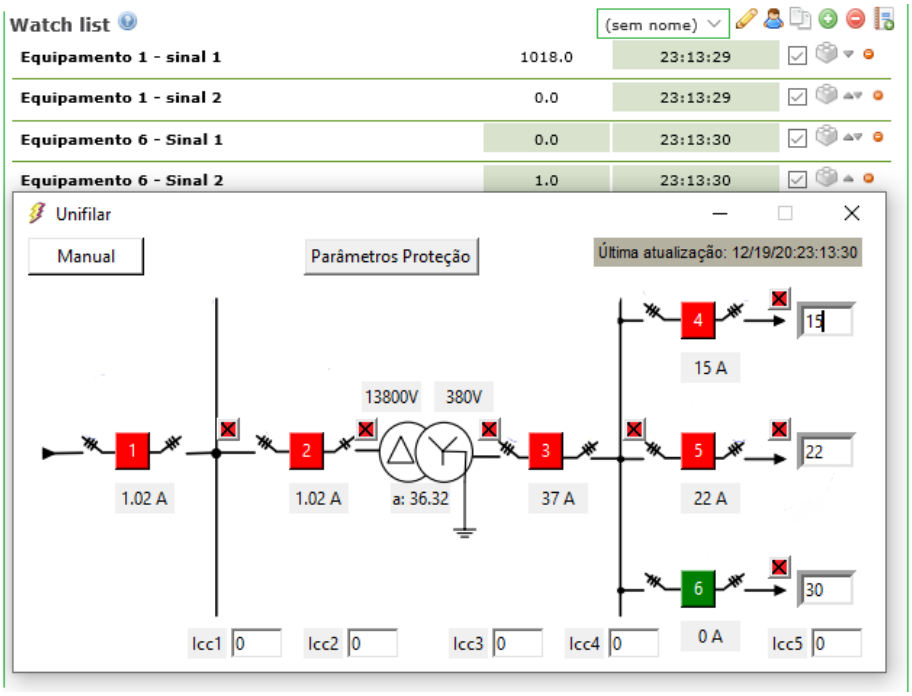
Fonte: O Autor (2020).

pode ser observada na Figura 14.

Para uma primeira simulação serão considerados uma corrente de ajuste de 1 Ampére para os relés a montante do transformador e 2,5 e 6 amperes, respectivamente, para os relés 5 e 3, todos com a curva moderadamente inversa e um dial de tempo de 0,1 segundo. Essas definições são realizadas na janela "Parâmetros". Após a definição e o salvamento deles, inicia-se a simulação a partir do clique no botão referente ao local da falta, que inicia a janela "Curto Circuito". Os botões responsáveis pela abertura das janelas citadas estão destacados na Figura 33, respectivamente, em azul e laranja. Com o destaque em verde, está a definição da corrente de falta no ponto desejado. Como pode ser observado na Figura 34, esses parâmetros resultam em uma ordem de atuação correta, mas com tempos muito próximos. Nesta situação, não ocorre mudança no estado de nenhum dispositivo.

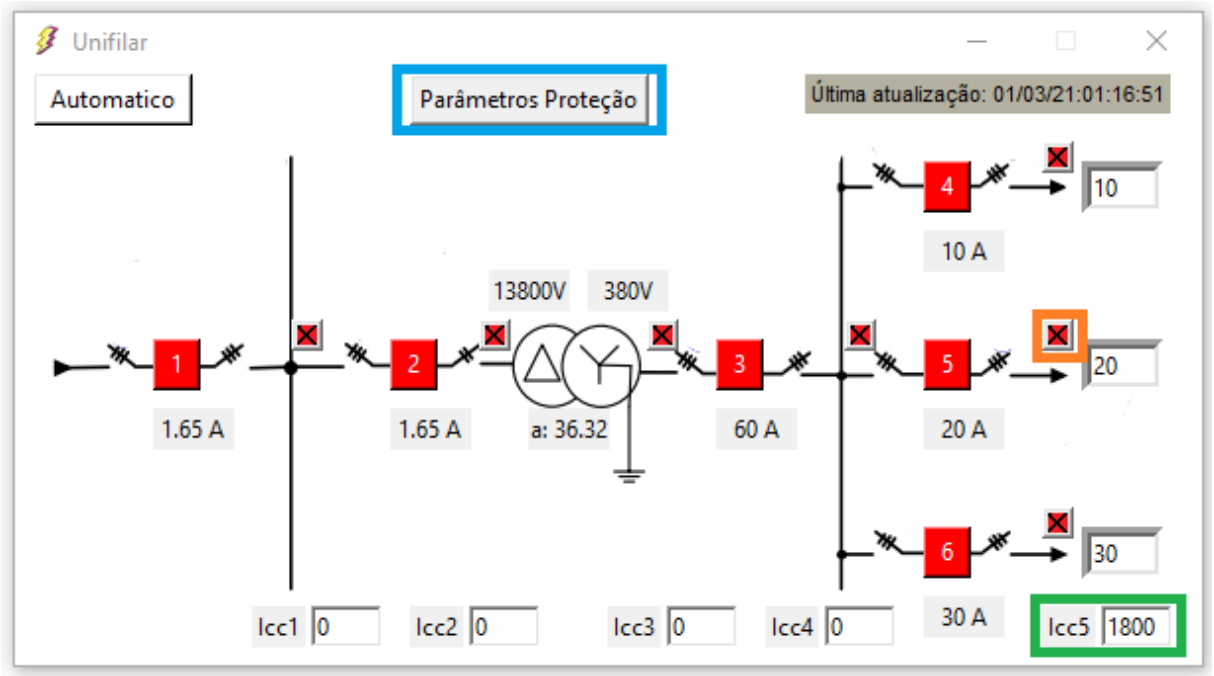
Uma situação na qual o intervalo previsto para a atuação dos relés é menor que a margem de coordenação, definida como 0,3 segundos, é indesejada. Para encontrar ajustes

Figura 32 – Interação entre as interfaces após mudança de parâmetros.



Fonte: O Autor (2020).

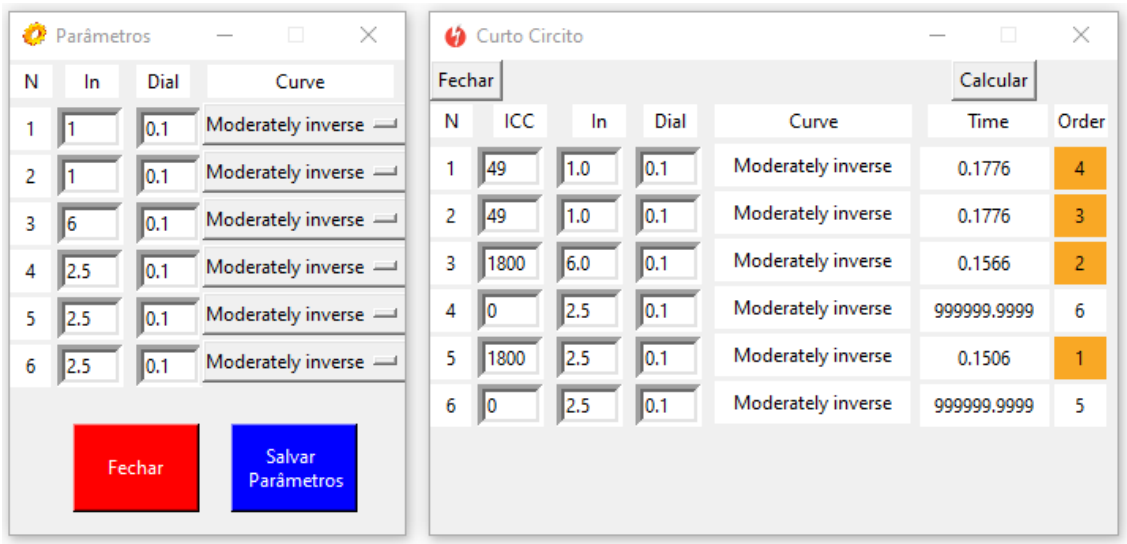
Figura 33 – Interface com botões destacados.



Fonte: O Autor (2020).

satisfatórios para os relés, pode ser utilizado o fluxo de cálculo indicado na seção 2.5.2.2. A partir desse cálculo, podemos encontrar um valor de aproximadamente 0,8, 1,1 e 1,6, respectivamente, para o dial de tempo dos relés 3, 2 e 1. O resultado da segunda simulação pode ser observado na Figura 35. Este é satisfatório, pois a diferença no tempo de atuação de cada dispositivo e seguinte é aproximadamente o intervalo de coordenação definido. Na mesma Figura, pode-se

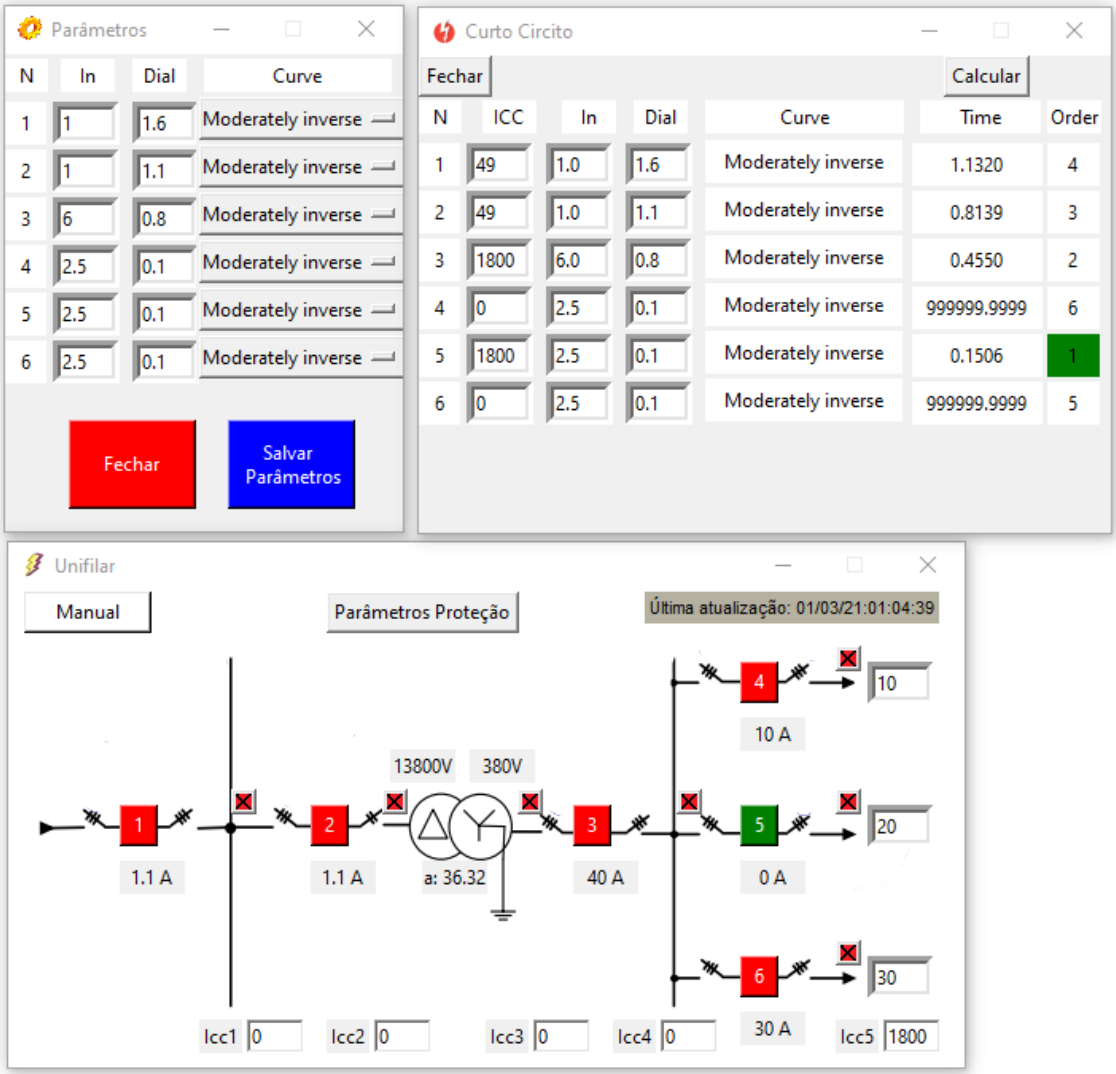
Figura 34 – Resultado do simulação de falta com valores iniciais.



Fonte: O Autor (2020).

observar a mudança do estado do relé 5, o que também é percebido pelo sistema SCADA.

Figura 35 – Resultado do simulação de falta com valores ajustados.



Fonte: O Autor (2020).

5 CONCLUSÃO E TRABALHOS FUTUROS

5.1 Conclusão

Neste trabalho, foi possível desenvolver satisfatoriamente um sitema simulador integrado a um de Supervisão Controle e Aquisição de Dados, utilizando-se apenas tecnologias livres em todas as suas etapas, desde a programação da interface gráfica, passando pelo protocolo de comunicação até o *software* SCADA. Isso foi realizado em um contexto que também permitiu que fossem explorados conceitos relacionados à Proteção de Sistemas Elétricos de Potência. Também foi possível aprofundar em conceitos relacionados à programação que podem ser relevantes para a formação de qualquer engenheiro.

Grande parte da relevância desse documento está na proposição de integração do ScadaBR a partir da linguagem Python. Isso é verdade pois alia a capacidade de um sistema SCADA de se comunicar com diversos dispositivos por meio de múltiplos protocolos de comunicação com a versatilidade de uma das linguagens de programação mais populares no mundo. Essa integração gera diversas possibilidades para trabalhos futuros sem a necessidade da utilização de softwares proprietários, cujas desvantagens foram discutidas no primeiro capítulo.

5.2 Trabalhos Futuros

Entre os possíveis trabalhos futuros, destaco:

- Aprimoramento do software para estudo completos de proteção e Curto Circuito;
- Integração dos dados SCADA a bancos de dados e serviços on-line a partir de bibliotecas de código livre;
- Monitoramento e Controle de dispositivos conectados por meio de diferentes protocolos a partir interface desenvolvida em Python;
- Integração de múltiplos dispositivos de monitoramento e atuação a algorítimos de previsão e visualização.
- Integração do monitoramento de múltiplos dispositivos a tecnologias padrões de BI do mercado como *Power BI Web Service*.

REFERÊNCIAS

CARBONNELLE, P. Pypl popularity of programming language. **URL: http://pypl. github. io/PYPL. html**, 2018.

COLLINS, G. **Pymodbus Documentation**. [S.l.]: June, 2013.

FOVINO, I. N.; CARCANO, A.; MASERA, M.; TROMBETTA, A. Design and implementation of a secure modbus protocol. In: PALMER, C.; SHENOI, S. (Ed.). **Critical Infrastructure Protection III**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 83–96. ISBN 978-3-642-04798-5.

GERS, J. M.; HOLMES, E. J. **Protection of electricity distribution networks**. [S.l.]: IET, 2004. v. 47.

GONZALEZ, I.; CALDERON, A. J. Integration of open source hardware arduino platform in automation systems applied to smart grids/micro-grids. **Sustainable Energy Technologies and Assessments**, v. 36, p. 100557, 2019. ISSN 2213-1388. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2213138819304230>.

GRAYSON, J. E. **Python and Tkinter programming**. [S.l.]: Manning Publications Co. Greenwich, 2000.

Guarese, G. B. M.; Sieben, F. G.; Webber, T.; Dillenburg, M. R.; Marcon, C. Exploiting modbus protocol in wired and wireless multilevel communication architecture. In: **2012 Brazilian Symposium on Computing System Engineering**. [S.l.: s.n.], 2012. p. 13–18.

KAZALA, R.; STRACZYNSKI, P. The most important open technologies for design of cost efficient automation systems. **IFAC-PapersOnLine**, v. 52, n. 25, p. 391–396, 2019.

KAZALA, R.; STRACZYNSKI, P. The most important open technologies for design of cost efficient automation systems. **IFAC-PapersOnLine**, v. 52, n. 25, p. 391–396, 2019.

KINDERMANN, G. Proteção de sistemas elétricos de potência. **Editora da UFSC**, v. 1, 1999.

LOHIA, K.; JAIN, Y.; PATEL, C.; DOSHI, N. Open communication protocols for building automation systems. **Procedia Computer Science**, v. 160, p. 723 – 727, 2019. ISSN 1877-0509. The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops. Disponível em: <http://www.sciencedirect.com/science/article/pii/S187705091931720X>.

MCKINNEY, W. *et al.* pandas: a foundational python library for data analysis and statistics. **Python for High Performance and Scientific Computing**, Seattle, v. 14, n. 9, 2011.

MODBUS-IDA. Modbus application protocol specification v1.1b. 12 2006.

PALACH, J. **Parallel programming with Python**. [S.l.]: Packt Publishing Ltd, 2014.

PETERS, T. The zen of python. In: **Pro Python**. [S.l.]: Springer, 2010. p. 301–302.

PINHEIRO, J. M. S. Introdução às redes de supervisão e controle. **Projeto de Redes**, 4 2006. <http://engineering.purdue.edu/~mark/puthesis> Acesso em: 12. Nov. 2020.

PYTHON. **multiprocessing — Process-based parallelism**. [S.l.]: The Python Software Foundation, 2020.

ROSSUM, G. V. *et al.* Python programming language. In: **USENIX annual technical conference**. [S.l.: s.n.], 2007. v. 41, p. 36.

SENSORWEB. Scada br cases. 2017. <http://http://www.scadabr.com.br/index.php/cases-2/> Acesso em: 13. Nov. 2020.

SENSORWEB. Scada br website homepage. 2017. <http://www.scadabr.com.br/> Acesso em: 13. Nov. 2020.

SILVA, M. R.; OLIVEIRA Ângelo Rocha de; CARMO, M. J. do; JUNIOR, L. O. de A. ImportÂncia da ferramenta scadabr para o ensino em engenharia. In: COBENGE. Centro Federal de Educação Tecnológica de Minas Gerais, Campus Leopoldina Rua José Peres, 558 – Centro 36700-000 – Leopoldina – MG, 2013.

APÊNDICE A – CÓDIGO-FONTE INTERFACE E SIMULAÇÃO

Código-fonte 1 – Interface de simulação

```
#Impotacao das bibliotecas
from tkinter import Tk,Label,Entry,Button,PhotoImage,
   Toplevel,StringVar,OptionMenu,PhotoImage
from PIL import Image
import pandas as pd
import time as tm

#Importar imagem para plano de fundo
im = Image.open("Unifilar3.png")
im.close()

#Criacao das funcoes

def place_all(): #Posiciona todos os Witgets (Atualiza seus
    visuais)

    #"Reles"
    b[0].place(x=70, y = 135)
    b[1].place(x=190 ,y =135)
    b[2].place(x=355, y =135)
    b[3].place(x=460, y =45)

    b[4].place(x=460, y =135)
    b[5].place(x=460, y =230)

    #Botoes do curto
    bcix = [140,235,320,420,520,520,520]
    bciy = [125,125,125,125,35,125,220]

```

```
28
29      for bci in range(7):
30          bc[bci].place(x=bcix[bci], y = bciy[bci])
31
32      #Rel gio
33      clock.place(x=400, y=1)
34
35      #Botao Teste Curto Circuito
36      CCB.place(x=200, y=1)
37
38      #Selecao Autom tico/Manual
39      auto_manu.place(x=10, y=1)
40
41      #Posicionar valor label "relacao de tranformacao"
42
43      l_v_prim.place(x = 240,y = 100)
44      l_v_sec.place (x = 290,y = 100)
45      l_transf.place(x = 260,y = 170)
46
47      #Labels indicadores de corrente
48      i[0].place(x=70 , y=170)
49      i[1].place(x=190, y=170)
50      i[2].place(x=355, y=170)
51      i[3].place(x=460, y= 80)
52      i[4].place(x=460, y=170)
53      i[5].place(x=460, y=265)
54
55      #Caixas de entrada para as cargas
56      e[0].place(x=540, y=45)
57      e[1].place(x=540, y=135)
58      e[2].place(x=540, y=230)
59
```

```
60      #Posisionar 'caixas' das impedancias
61      zx = [120,200,300,380,520]
62      zy = [270,270,270,270,270]
63
64      for zi in range(5):
65          z_label[zi].place(x=zx[zi], y=zy[zi])
66          z_entry[zi].place(x=zx[zi]+30, y=zy[zi])
67
68  def mudar_fechar(j): # Altera estado do rele
69      state[j] = 1
70      b[j] = Button(root,text = str(j+1), height=1, width=2,
           command =lambda: mudar_abrir(j),bg = 'red',fg = '
           white')
71      place_all()
72
73  def mudar_abrir(k): # Altera estado do rele
74      state[k] = 0
75      b[k] = Button(root,text = str(k+1), height=1, width=2,
           command =lambda: mudar_fechar(k),bg = 'green',fg = '
           white')
76      place_all()
77
78  def find_index(address,offset): #Auxilia encontrar o
       endereco de cada dado no .csv
79      return (int(address[-1]) - 1) * 10 + offset
80
81  def Atualizar(): #Atualiza vari veis , l  /escreve dados
       externos
82
83      global leitura
84
85      try:
```

```
86          leitura = pd.read_csv("inputs.csv") #Comandos do
               Scada
87
88          #Se comando for autom tico, realiazr alteracoes de
                acordo com comandos Scada
89          if comand != 'Manual':
90              ste_input = []
91              for j in range(6): #Para cada rele, identificar
                    o 'Value' correnpondente e 'abrir' ou '
                   fechar de acordo'
92                  ste_input.append(leitura.loc[find_index('
                       127.2.2.'+str(j+1),0),'Value'])
93                  if ste_input[j] == 1:
94                      mudar_abrir(j)
95                  if ste_input[j] == 0:
96                      mudar_fechar(j)
97
98          #Atualizaar Hora e 'agendar' proxima atualizacao
99          hora =tm.strftime(' ltima  atualizacao: %D:%H:%M:%S
               ')
100         clock['text'] = hora
101         root.after(1000,Atualizar)
102
103         #Iniciar valores de corrente como 0
104         values  = [0,0,0,0,0,0]
105
106         #3 Ultimos reles tem a corrente demandada pela
               carga se os    jusante estiverem '1'
107         for j in range(3):
108             values[j] = int(e[j].get()) * state[0] * state
                   [1]* state[2] * state[j+3]
109             i[j+3].configure(text = str(values[j]) + " A")
```

```
110
111         #calculo das correntes mais proximas da fonte
112         i[0].configure(text =  str(round(sum(values) * (1/
                rt),2))+ " A")
113         i[1].configure(text =  i[0]['text'] )
114         i[2].configure(text =  str(sum(values) * 1) + " A")
115
116         #Escrever dados que serao direcionados para o scada
117         #Valores de corrente
118         teste.loc[find_index('127.2.2.1',0),'Value'] = int(
                sum(values) * 1000 * (1/rt))
119         teste.loc[find_index('127.2.2.2',0),'Value'] = int(
                sum(values) * 1000*(1/rt))
120         teste.loc[find_index('127.2.2.3',0),'Value'] = int(
                sum(values)) *1000
121         teste.loc[find_index('127.2.2.4',0),'Value'] = int(
                values[0]) *1000
122         teste.loc[find_index('127.2.2.5',0),'Value'] = int(
                values[1])*1000
123         teste.loc[find_index('127.2.2.6',0),'Value'] = int(
                values[2])*1000
124         #Estado dos reles
125         teste.loc[find_index('127.2.2.1',1),'Value'] = int(
                not state[0])
126         teste.loc[find_index('127.2.2.2',1),'Value'] = int(
                not state[1])
127         teste.loc[find_index('127.2.2.3',1),'Value'] = int(
                not state[2])
128         teste.loc[find_index('127.2.2.4',1),'Value'] = int(
                not state[3])
129         teste.loc[find_index('127.2.2.5',1),'Value'] = int(
                not state[4])
```

```
130         teste.loc[find_index('127.2.2.6',1),'Value'] = int(
                not state[5])
131
132         #Exportacao dos dados
133         teste.to_csv("Regs.csv",index=False)
134
135         #Atualizacao dos widgets
136         place_all()
137
138     except: #Caso erro na atualizacao
139         print('erro')
140         Atualizar()
141
142 def tipo_comando():
143     global comand
144     global t_com
145     #Mudar tipo de comando
146     if t_com == 1:
147         t_com = 0
148         comand = 'Manual'
149     elif t_com == 0:
150         t_com = 1
151         comand = 'Automatico'
152
153     #Muda texto do Botao
154     auto_manu.configure(text = comand)
155     print("t_com " + str(t_com))
156
157     #Atualizar widgets
158     place_all()
159
160 def t_trip (curva, I_pu, I_fault, dial):
```

```
161     if I_pu * I_fault * dial > 0:
162         curvas = ["Moderately inverse","Very inverse","
              Extremely inverse"]
163         alpha = [0.02,2,2]
164         beta  = [0.0515,19.61,28.2]
165         L     = [0.114,0.491,0.1217]
166         const = pd.DataFrame(
167             {'Desc': curvas,
168             'Alpha': alpha,
169             'Beta': beta,
170             'L'   : L
171             })
172         c = const[const.Desc == curva]
173         a = c.loc[:,"Alpha"]
174         b = c.loc[:,"Beta"]
175         L = c.loc[:,"L"]
176         t_act = (dial*b)/(((I_fault)/(I_pu))**a-1) + L
177     else:
178         t_act = 999999.9999
179 
180     return t_act
181 
182 def Win_CC_Param():
183     print(z_value)
184     WCC = Toplevel(root)
185     WCC.iconbitmap("conf.ico")
186     WCC.title("Par metros")
187     WCC.geometry("250x300")
188     WCC.resizable(False,False)
189     close_cc = Button(WCC,text = "Fechar",command= lambda:
          WCC.destroy(),width = 10,height = 3,borderwidth = 2,
          bg = 'Red',fg = 'white')
```

```
190     close_cc.place(x=40,y=230)
191     save_cc = Button(WCC,text = "Salvar \n Par metros",
           command= lambda: att_params(),width = 10,height = 3,
            borderwidth = 2,bg = 'blue',fg = 'white')
192     save_cc.place(x=140,y=230)
193
194     curvas = ["Moderately inverse","Very inverse","
           Extremely inverse"]
195     headers = ["N","In","Dial","Curve"]
196     axysp = [0,35,80,125,180,315]
197     tam =   [3,4 ,4 ,16  ,17 ,10,20]
198     head = []
199
200     selected = []
201     drops = []
202     Entb  = []
203     Entc  = []
204     equip = []
205
206     #Alocar widgets
207     #Headers
208     for i in range(4):
209         print(str(i) +': ' +str(tam[i]))
210         head.append(Label(WCC,text = headers[i],bg = 'white
               ',width = tam[i]))
211         head[i].place( x = axysp[i],y = 3)
212     #Inputs
213     for i in range(6):
214
215         equip.append(Label(WCC,text = str(i+1),bg = 'white'
               ,width = 2,borderwidth = 5))
216         Entb.append(Entry(WCC,width = 5,borderwidth = 5))
```

```
217         Entc.append(Entry(WCC,width = 5,borderwidth = 5))
218
219         selected.append(StringVar())
220         selected[i].set(curvas[0])
221         drops.append(OptionMenu(WCC,selected[i],*curvas))
222
223         equip[i].place(x = 0   ,y =(i+1)*30 )
224         Entb[i].place( x = 30, y = (i+1)*30 )
225         Entc[i].place( x = 80, y = (i+1)*30 )
226         drops[i].place(x= 120,y=(i+1)*30 - 5)
227         drops[i].config(width=15)
228
229         Entb[i].insert(0,'1')
230         Entc[i].insert(0,'1')
231
232         global in_param
233         global dial_param
234         global curvas_param
235
236         def att_params():
237             for i in range(6):
238                 in_param[i] = float(Entb[i].get())
239                 dial_param[i] = float(Entc[i].get())
240                 curvas_param[i] = str(selected[i].get())
241             print(str(in_param) + '\n' + str(dial_param)+ '
                \n' + str(curvas_param))
242
243 def Win_CC(iccs = [0,0,0,0,0,0]):
244
245     def att_parametros():
246         tempos = []
247         for i in range(6):
```

```
248             global in_param
249             global dial_param
250             Entb[i].delete(0,'end')
251             Entc[i].delete(0,'end')
252             Entb[i].insert(0,in_param[i])
253             Entc[i].insert(0,dial_param[i])
254             # selected[i].set(curvas_param[i])
255             drops[i].config(text = curvas_param[i])
256             icc_ =  float(Enta[i].get())
257             in_ =   float(Entb[i].get())
258             dial_ = float(Entc[i].get())
259             curva_ = curvas_param[i]
260
261             try:
262                 t_act = float(t_trip (curva_, in_, icc_,
                        dial_))
263             except:
264                 t_act = 0
265
266             resul[i].config(text = "{:.4f}".format(t_act))
267             tempos.append(t_act-0.00001*i)
268
269         t_ordenado = sorted(tempos)
270         ordem = []
271         t_gap = 0.3
272
273         cores = ['white','white','white','white','white','
                white']
274
275         if t_ordenado[0] < t_ordenado[1] - t_gap:
276             cores[0] = 'Green'
277             global comand
```

```
278             comand = 'Manual'
279             tipo_comando()
280             mudar_abrir(tempos.index(t_ordenado[0]))
281         else:
282             for i in range(6):
283                 if t_ordenado[i] < t_ordenado[0] + t_gap:
284                     cores[i] = '#f9a825'
285 
286         for i in range(len(tempos)):
287             ordem.append(t_ordenado.index(tempos[i])+1)
288             t_act_label[i].configure(text = int(t_ordenado.
                index(tempos[i])+1))
289             t_act_label[i].configure(bg = cores[t_ordenado.
                index(tempos[i])])
290         print(ordem)
291 
292     print(z_value)
293     WCC = Toplevel(root)
294     WCC.iconbitmap("Curto.ico")
295     WCC.title("Curto Circito")
296     WCC.geometry("435x300")
297     WCC.resizable(False,False)
298     close_cc = Button(WCC,text = "Fechar",command= lambda:
            WCC.destroy())
299     close_cc.place(x=0,y=0)
300 
301     curvas = ["Moderately inverse","Very inverse","
            Extremely inverse"]
302     headers = ["N","ICC","In","Dial","Curve","Time","Order"
            ]
303     axysp = [0,40,90,135,180,315,395]
304     tam = [3,4,4,4,17,10,4]
```

```
305     head = []
306 
307     selected = []
308     drops = []
309     Enta  = []
310     Entb  = []
311     Entc  = []
312     equip = []
313     resul = []
314     t_act_label = []
315     #Alocar widgets
316 
317     for i in range(7):
318         head.append(Label(WCC,text = headers[i],bg = 'white
                ',width = tam[i]))
319         head[i].place( x = axysp[i],y = 28)
320 
321     for i in range(6):
322 
323         equip.append(Label(WCC,text = str(i+1),bg = 'white'
                ,width = 2,borderwidth = 5))
324         Enta.append(Entry(WCC,width = 5,borderwidth = 5))
325         Entb.append(Entry(WCC,width = 5,borderwidth = 5))
326         Entc.append(Entry(WCC,width = 5,borderwidth = 5))
327 
328         selected.append(StringVar())
329         selected[i].set(curvas[0])
330         # drops.append(OptionMenu(WCC,selected[i],*curvas))
331         drops.append(Label(WCC,text = curvas[0],bg = 'white
                ',width = 2,borderwidth = 5))
332         resul.append(Label(WCC,text = "",bg = 'white',width
                 = 10,borderwidth = 5))
```

```
333         t_act_label.append(Label(WCC,text = "",bg = 'white'
                ,width = 3,borderwidth = 5))
334
335         equip[i].place(x = 0   ,y =(i+1)*30   + 25)
336         Enta[i].place( x = 30, y = (i+1)*30 + 25)
337         Entb[i].place( x = 80, y = (i+1)*30 + 25)
338         Entc[i].place( x = 130,y =(i+1)*30   + 25)
339         drops[i].place(x=180,y=(i+1)*30+23)
340         drops[i].config(width=16)
341         resul[i].place( x = 310,y =(i+1)*30   + 25)
342         t_act_label[i].place( x = 395,y =(i+1)*30   + 25)
343
344         Enta[i].insert(0,int(iccs[i]))
345         Entb[i].insert(0,'0')
346         Entc[i].insert(0,'0')
347
348             # print(a + '|'+ b +'|' + c + '|' + d)
349     #Calcular tempos e atualizar valores
350     att_cc = Button(WCC,text = "Calcular",command=
            att_parametros)
351     att_cc.place(x=330,y=0)
352
353     att_parametros()
354
355 def calc_z(a):
356
357     imp = 0
358
359     #Reles que compartilham a mesma Icc
360     if a > 4:
361         b = 4
362     else:
```

```
363             b = a
364
365         try:
366             imp = round(float(z_entry[b].get()),5)
367             print(imp)
368         except:
369             pass
370
371         print(atravessar(a))
372         icc_calc = imp # 1000/imp
373         icc_calc_vec = [h * icc_calc for h in atravessar(a)]
374         print(icc_calc_vec)
375         Win_CC(icc_calc_vec)
376
377 def atravessar(n):
378         if n > 2:
379             fp = 1/rt
380             fs = 1
381         else:
382             fp = 1
383             fs = 0
384         atv = []
385         atv.append([fp,0,0,0,0,0])
386         atv.append([fp,fp,0,0,0,0])
387         atv.append([fp,fp,0,0,0,0])
388         atv.append([fp,fp,fs,0,0,0])
389         atv.append([fp,fp,fs,fs,0,0])
390         atv.append([fp,fp,fs,0,fs,0])
391         atv.append([fp,fp,fs,0,0,fs])
392         return(atv[n])
393
394 #Inicializar objeto principal
```

```
395 root = Tk()
396 root.resizable(False,False)
397 root.title("Unifilar")
398
399 root.geometry(str(im.size[0]) + "x" + str(round(im.size
      [0]/2)))
400 root.iconbitmap("Raio.ico")
401
402 #Definie background
403 filename = PhotoImage(file = "Unifilar3.png")
404 background_label = Label(root, image=filename)
405 background_label.place(x=0, y=0, relwidth=1, relheight=1)
406 #Label de data/hora
407 clock = Label(root,font='ariel 8',bg='#b5b1a1',fg='black')
408
409 #Dataframe com os valores iniciais
410 try:
411     teste = pd.read_csv('Regs.csv')
412 except:
413     pass
414
415 #Lista dos estados dos reles
416 global state
417 state = [0,0,0,0,0,0]
418
419 #Vari vel Relacao de transformacao, com entrada e label
420 global rt
421
422 v_prim = 13800
423 v_sec  = 380
424
425 V_barras = [v_prim,v_prim,v_sec,v_sec,v_sec]
```

```
426 rt = v_prim/v_sec
427
428 l_v_prim = Label(root,text = str(round(v_prim,2)) + "V",
      width= 5 ,height = 1)
429 l_v_sec =  Label(root,text = str(round(v_sec,2))  + "V",
      width= 5 ,height = 1)
430 l_transf = Label(root,text = "a: "+str(round(rt,2)),width=
      5 ,height = 1)
431
432 global comand   # String tipo de comando
433 global t_com    # Tipo de comando (1 ou 0)
434 #Comando inicialmente autom tico, definicao do botao de
      selecao
435 t_com = 1
436 comand = 'Automatico'
437 auto_manu = Button(root,text = comand, height=1, width=10,
      command = tipo_comando,bg = 'white')
438
439 global leitura  # Comandos do scada
440 leitura = pd.read_csv("inputs.csv")#Introduzir valores
      comandos iniciais vindos do scada.
441
442 #6 Labels para indicacao das correntes
443 i = []
444 for j in range(6):
445     i.append(Label(root,text = "0 A",width= 5 ,height = 1))
446
447 #3 caixas de para imput das cargas
448 e = []
449 for t in range(3):
450     e.append(Entry(root ,width = 5,borderwidth = 5))
451     e[t].insert(0,(t+1)*10)
```

```
452
453 # Criacao dos botoes, representando os reles.
454 b = []
455
456 for ib in range(6):
457     b.append(Button(root,text = str(ib+1), height=1, width
        =1, command =lambda: mudar_fechar(ib),bg = 'green',
        fg = 'white'))
458
459 # Criacao das entradas das impedancias
460 z_entry = []
461 z_label = []
462 z_value = [0,0,0,0,0]
463
464 for zi in range(5):
465     z_entry.append(Entry(root ,width = 5,borderwidth = 2))
466     z_entry[zi].insert(0,z_value[zi])
467     z_label.append(Label(root,text = 'Icc' + str(zi+1),
        width= 3 ,height = 1))
468
469 bc = []
470 #Botao para selecionar o curto
471 photo_x = PhotoImage(file = r"C:\Users\guilh\Desktop\TCC\
    Programas\X.png")
472
473 #Vetor valor das correntes tem cada dispositivo
474
475 bc.append(Button(root, command =lambda: calc_z(0),image =
    photo_x))
476 bc.append(Button(root, command =lambda: calc_z(1),image =
    photo_x))
```

```
477 bc.append(Button(root, command =lambda: calc_z(2),image =
        photo_x))
478 bc.append(Button(root, command =lambda: calc_z(3),image =
        photo_x))
479 bc.append(Button(root, command =lambda: calc_z(4),image =
        photo_x))
480 bc.append(Button(root, command =lambda: calc_z(5),image =
        photo_x))
481 bc.append(Button(root, command =lambda: calc_z(6),image =
        photo_x))
482
483 CCB = Button(root,text = "Par metros Protecao",command =
        Win_CC_Param,fg = 'Black')
484 global in_param
485 global dial_param
486 global curvas_param
487
488 in_param = [0,0,0,0,0,0]
489 dial_param = [0,0,0,0,0,0]
490 curvas_param = []
491 for k in range(6):
492     curvas_param.append("Moderately inverse")
493
494 place_all() #Posicionar widgets na tela
495 Atualizar() #Atuzlizar valores
496 root.mainloop() #Loop principal da interface
```

APÊNDICE B – CÓDIGOS-FONTE PARA COMUNICAÇÃO COM SCADA

Código-fonte 2 – Comunicação paralela com protocolo MODBUS

```
1
2  # Importacao das bibliotecas
3  from multiprocessing import Pool,Process
4  import time as t
5  import pandas as pd
6  from pyModbusTCP.server import ModbusServer, DataBank
7  import time
8  import pandas as pd
9  from os import listdir,getcwd
10
11 # Funcao de leitura e escrita com o protocolo MODBUS a ser
       replicada no processamento paralelo.
12 def write(name):
13     #Execu a  coninua
14     while True:
15         #Parametros necess rios e inicilizacao do servidor
               .
16         address = name
17         server = ModbusServer(address,12345,no_block=True)
18         server.start()
19
20         #Importacao dos dados para escrita e criacao da
               lista para os dados de leitura e escrita
21         Data = pd.read_csv("Regs.csv")
22         Saida = pd.DataFrame()
23         S_data = Data[Data.Address == address]
24         offset = S_data.Offset.tolist()
25         Value = S_data.Value.tolist()
26
```

```
27          #Escrita de dados
28          for i in range(len(S_data)):
29              time.sleep(.1)
30              print(int(Value[i]))
31              DataBank.set_words(offset[i], [int(Value[i])])
32
33          #Leitura de dados
34          try:
35              for i in range(len(S_data)):
36                  time.sleep(.1)
37                  valor = DataBank.get_words(i+10)
38                  df2 = pd.DataFrame({"Address":[name],
39                                      "Base":['127.2.2.'],
40                                      "N":[1],
41                                      "Port":[502],
42                                      "Offset":[i+10],
43                                      "Value":valor})
44                  Saida =  Saida.append(df2)
45
46              #Exportcao das leituras em arquivo .csv
47              Saida.to_csv(address+".csv")
48              print(t.time()-t1)
49          except:
50              pass
51
52      return name
53
54
55  if __name__ == '__main__':
56
57          #Leitura dos dados da simula  o.
58         csv_data = pd.read_csv("Regs.csv")
```

```
59          #Listagem dos endere os a serem utilizados.
60          ends = csv_data.Address.unique().tolist()
61          # Escrever os enderecos que serao argumentnos para
                cada funcao paralela
62          ends = ends[0:6]
63          print(ends)
64          #Cria   o do obejeto "Pool"
65          p = Pool(6)
66          #Inicializacao do paralelismo.
67          p.map(write, ends)
```

Código-fonte 3 – Unificação dos dados exportados pelos programas concorrentes

```
1
2   #Importacao das bibliotecas
3   import pandas as pd
4   from os import listdir,getcwd
5   from time import sleep,time
6
7   # Listar arquivos .csv no diretorio
8   def find_csv_filenames( path_to_dir, suffix=".csv" ):
9       filenames = listdir(path_to_dir)
10      return [ filename for filename in filenames if filename
            .endswith( suffix )]
11
12  #Funcao Principal
13  if __name__ == '__main__':
14
15      #Contador para identificar o tamanho do arquivo de
            saida
16      t = 0
17
```

```
18      #Executar continuamente
19      D = True
20      while D:
21          #Buscar e concatenar arquivos desejados.
22          merged = pd.DataFrame()
23          filenames = find_csv_filenames(getcwd())
24
25          for name in filenames:
26              if name[0:4] == "127.":
27                  try:
28                      df = pd.read_csv(name)
29                      merged =  merged.append(df)
30                  except:
31                      pass
32
33          #Exportar caso os dados estejam completos, apenas
34          if t <= merged.count()[1]:
35              t = merged.count()[1]
36              try:
37                  merged.to_csv('inputs.csv')
38              except:
39                  pass
40              sleep(0.1)
```