UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
COLEGIADO DO CURSO DE ENGENHARIA ELÉTRICA

Desenvolvimento de um Sistema de Planejamento Integrado (SIPLA): *Software* Livre para Aplicação na Área de Sistemas Elétricos de Potência

Sandy Aquino dos Santos

2021

Sandy Aquino dos Santos

Desenvolvimento de um Sistema de Planejamento Integrado (SIPLA): *Software* Livre para Aplicação na Área de Sistemas Elétricos de Potência

Trabalho apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal da Bahia como parte dos requisitos para a obtenção do grau de Engenheiro(a) Eletricista.

Orientador(a): Daniel Barbosa

SALVADOR
2021

AQUINO DOS SANTOS, SANDY.
/ . – , -
139 p. : il. (algumas color.) ; 30 cm.

– , .

1. *Software Livre*. 2. Desenvolvimento de *Software*. 2. Sistemas Elétricos de Potência. I. Daniel Barbosa. II. Universidade Federal da Bahia. III. Escola Politécnica da Universidade Federal da Bahia. IV. Desenvolvimento de um Sistema de Planejamento Integrado (SIPLA): *Software* Livre para Aplicação na Área de Sistemas Elétricos de Potência.

Sandy Aquino dos Santos

Este Trabalho de Graduação foi julgado adequado para a obtenção do grau de Engenheiro(a) Eletricista e aprovado em sua forma final pela Comissão Examinadora e pelo Colegiado do Curso de Graduação em Engenharia Elétrica da Universidade Federal da Bahia.

André Pires Nóbrega Tahim
Coordenador do Colegiado do
Curso de Engenharia Elétrica

Comissão avaliadora

Daniel Barbosa
(Orientador – DEEC/UFBA)

Renato José Pino de Araújo
(Coorientador – DEEC/UFBA)

Kleber Freire da Siva
(Avaliador Interno - DEEC/UFBA)

Fernando Augusto Moreira
(Avaliador Interno - DEEC/UFBA)

Dedico este trabalho a todos aqueles que passaram pela minha vida, seja no mundo físico ou no mundo espiritual, em especial a minha mãe (in memoriam), sozinho não somos nada.

Agradecimentos

Agradeço a Deus primeiramente por ter me permitido chegar até aqui, não há uma folha se quer que caía de uma árvore sem a tua permissão.

A minha mãe, Sandra Maria Aquino dos Santos, (*in memoriam*), por todo amor a mim dedicado enquanto estávamos fisicamente juntos e pela presença constante ainda que em espírito nos longos períodos de dificuldades enfrentados para chegar até aqui, serei eternamente grato.

Ao meu pai, Gilson Fernandes dos Santos, pela companhia durante todo esse período, pelo carinho e atenção para os auxílios necessários durante esse longo percurso acadêmico.

A minha vó Irene e minhas mães do coração, Vilma, Tânia e Ivonia pelo acolhimento em todos os momentos difíceis, pelo cuidado, carinho e incentivo.

A minha companheira, Silvana Oliveira, pelo apoio, paciência e incentivo nos momentos de estresse e de desânimo, pela compressão nos momentos de ausência e pelo cuidado nos momentos de dor. Obrigado pelo amor a mim dedicado.

Aos meus irmãos, Leonardo, Thais, Luiz Carlos, Inaiara, Beatriz, Luan, Camila e Jean pelo incentivo de sempre seguir em frente.

Agradeço à minha família por todo o apoio oferecido, por acreditarem em mim e estarem sempre ao meu lado.

Ao meu Amigo e irmão do coração Leandro Encarnação e sua esposa Helaine Encarnação, pelo amor, amizade e cuidado por todos esses anos. Pelas palavras de conforto, de incentivo, em todos os momentos, vocês foram cruciais para que eu tenha chegado até aqui, serei eternamente grato.

Aos amigos do Centro Espirita Casa de Caridade “Amigos do Bem”pela amizade, carinho e amor fraternal durante esse período, em especial aos amigos espirituais, pelo apoio necessário em todos os momentos durante minha trajetória terrena.

Ao Pré-vestibular Alternativo do Centro de Cultura, Orientação e Estudos Quilombo (Coequilombo), que me acolheu, me formou enquanto homem negro e me permitiu experienciar aulas que foram essências para minha vida e entrada na universidade. A todos os professores que lá estavam serei eternamente grato.

Ao meu orientador Daniel Barbosa, pela oportunidade, por acreditar em mim e no meu trabalho, por me ensinar a não desistir enquanto o problema não for superado. Pelas discussões incansáveis sobre temas diversos que sempre terminavam do jeito que começavam, sem conclusões, mas com lições de convivência e respeito a opiniões divergentes. Por me incentivar, conscientemente ou não a sempre dar o meu melhor, suas contribuições na

minha vida acadêmica e profissional são imensuráveis, serei sempre grato.

Ao meu coorientador, Renato Araújo, pelas oportunidades de aprendizagens, pelas ensinamentos sobre mercado de trabalho, pelas trocas continuas nas conversas informais nas aulas e nos corredores da universidade, com certeza seus conselhos tem feito diferença nas minhas escolhas.

Agradeço a Universidade Federal da Bahia, em especial, a Escola Politécnica e aos meus professores pelas oportunidades de aprendizagens vivenciadas durante todo o curso.

A todos os meus colegas do curso de graduação que compartilharam dos inúmeros desafios que enfrentamos, sempre com o espírito colaborativo. Em especial, Ibsen e Renata pela parceria durante longos e desafiadores semestres, a troca de apoio foi fundamental para que eu continuasse.

Aos meus colegas do Grupo de Pesquisa de Sistemas Elétricos de Potência Integrado, por todas as trocas, pelas resenhas, pelas oportunidades e por sempre me incentivarem a entregar o melhor resultado. O grupo para mim foi e é uma família.

A todas as pessoas que passaram pela minha vida durante esses longos anos, todos deixaram sua marca e contribuíram com a minha caminha, meu muito obrigado.

"A fé na vitória tem que ser inabalável." O Rappa

Tem que acreditar. Desde cedo a mãe da gente fala assim:
"Filho! por você ser preto, você tem que ser duas vezes melhor".
Ai passado alguns anos eu pensei,
"Como fazer duas vezes melhor, se você está pelo menos 100 vezes atrasado?
Atrasado pela escravidão, pela história, pelo preconceito, pelos traumas, pelas psicoses, por tudo que aconteceu, duas vezes melhor como?".
Ou melhora, você é o melhor ou o pior de uma vez, sempre foi assim, se você vai escolher o que estiver mais perto de você, o que tiver dentro da sua realidade, você vai ser duas vezes melhor como? Quem inventou isso dai? Quem foi o pilantra que inventou isso daí?
Acorda pra vida rapaz."
Mano Brown, Edy Rock, Kl Jay, Ice Blue. Racionais MC.

Resumo

AQUINO DOS SANTOS, SANDY. **Desenvolvimento de um Sistema de Planejamento Integrado (SIPLA): *Software* Livre para Aplicação na Área de Sistemas Elétricos de Potência.** 2021. 139p. Trabalho de Conclusão de Curso, Escola Politécnica, Universidade Federal da Bahia, Salvador, 2021.

Um dos meios para compreensão dos fenômenos que envolvem o funcionamento de um Sistema Elétrico de Potência é simular situações em ambientes computacionais que reproduzam os comportamentos reais dos elementos que o compõe. Contudo, maioria dos softwares para estudos no sistema elétrico de potência são produtos comerciais obtidos por meio de licenças com um alto custo de aquisição. Desta forma, os softwares livres surgem como uma alternativa, pois em sua filosofia promovem as liberdades de ser utilizado, estudado, modificado e disponibilizado, tornando-se assim de suma importância na universidade e nas empresas do setor de energia. Este trabalho realizou uma vasta revisão de literatura e fundamentação teórica sobre desenvolvimento de *software* livre, buscando reunir conceitos dessa filosofia aos já tradicionais métodos de desenvolvimento e partir disso, foi desenvolvido um *software* livre voltado para estudos de sistemas elétricos de potência em Python, utilizando o *Framework PyQt5*, usando a metodologia orientada a objetos, fazendo do *software* uma plataforma funcional e promissora para futuros desenvolvimentos.

Palavras-chave: *Software Livre*, Desenvolvimento de *Software*, Sistema Elétrico de Potência.

Abstract

One of the ways to understand the phenomena that involve the functioning of an Electric Power System is to simulate situations in computing environments that reproduce the real behaviors of the elements that compose it. However, most software for studies in the electrical power system are commercial products obtained through licenses with a high acquisition cost. In this way, free software emerges as an alternative, because in its philosophy it promotes the freedom to be used, studied, modified and made available, thus becoming of paramount importance in universities and companies in the energy sector. This work carried out a vast literature review and theoretical foundation on free software development, seeking to bring together concepts of this philosophy to the already traditional development methods and from that, free software was developed aimed at studies of electrical power systems in Python, using the PyQt5 Framework, using object-oriented methodology, making the software a functional and promising platform for future developments.

Keywords:*Free Sofware,Software Development, Power System.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANEEL Agência Nacional de Engenharia Elétrica

BDGD Base de Dados Geográficas das Distribuidoras

CRC Classe Responsabilidade Colaborador

DAS Desenvolvimento Ágil de *Software*

DEEC Departamento de Engenharia Elétrica e Computação

ES Engenharia de *Software*

EPRI *Electric Power Research Institute*

GC Geração Centralizada

GD Geração Distribuída

GPL Lincença Pública Geral

GUI *Graphical User Interface*

IXP Industrial XP

KISS *Keep It Simple Stupid!*

MOO Metodologia Orientada a Objetos

OTAN Organização do Tratado do Atlântico Norte

POO Paradigma Orientado a Objetos

FOSS *Free and Open Source Software*

FSF *Free Software Foundation*

FR Requisitos Funcionais

MIT *Massachusets Institute of Technology*

NFR Requisitos Não Funcionais

OSI *Open Source Initiative*

PCC Ponto Comum de Conexão

PRODIST Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional

SD Sistema de Distribuição

SEP Sistema Elétrico de Potência

SIPLA Sistema de Planejamento Integrado

TFG Trabalho Final de Graduação

UML Linguagem de Modelagem Unificada

XP *Extreme Programming*

Sumário

1 Introdução

Um dos meios para compreensão dos fenômenos que envolvem o funcionamento de um Sistema Elétrico de Potência (SEP) é simular situações em ambientes computacionais que reproduzam os comportamentos reais dos elementos que o compõe. Os autores Oliveira *et al.* (2019) afirmam que o processo de simulação consiste em reproduzir fenômenos do universo, sendo assim, segundo Strieder, Schuch e Frias (2010), a utilização de simuladores em atividades práticas de laboratório é uma tendência cada vez mais requerida para o ensino nas diferentes áreas da engenharia, fato confirmado através dos trabalhos trazidos por Terbuc (2006), Barry (2009) e Lazić, Zorica e Klindžić (2011) onde percebe-se que essa é uma realidade mundial. Portanto, a existência de um sistema computacional no ambiente laboratorial do curso de engenharia elétrica aplicado ao SEP torna-se importante para proporcionar aos estudantes uma vivência de ensino-aprendizagem mais completa, além de contribuir como uma ferramenta de antecipação de cenários que impactem diretamente a realidade operativa de um Sistema de Distribuição (SD), dialogando desta forma com as empresas do setor.

Contudo, maioria dos *softwares* para estudos no SEP são produtos comerciais obtidos por meio de licenças com um alto custo de aquisição (MILANO, 2010). De acordo com Oliveira *et al.* (2019) esses programas proprietários possuem licenças com direitos exclusivos para o produtor, ou seja, seu uso, redistribuição ou modificação são proibidos, e a permissão do desenvolvedor e a aquisição dessas permissões, quando concedida, implica em um custo ainda maior ao usuário.

Desta forma, os *softwares* livres surgem como uma alternativa a esses altos custos de aquisição, pois em sua filosofia promovem as liberdades de ser utilizado, estudado, modificado e disponibilizado, tornando-se assim de suma importância na universidade. Além disso, Gomes *et al.* (2011) afirma que o contato com os aplicativos livres na universidade reflete uma interseção entre a comunidade acadêmica e as empresas de engenharia elétrica, visto que os mesmos são utilizados no desenvolvimento de inúmeros projetos.

1.0.1 Motivação e Justificativa

O setor elétrico, tanto no Brasil quanto em diversos países do mundo, tem evoluído em sua característica tradicional. Historicamente as etapas que configuram o setor da energia elétrica engloba as seguintes atividades hierarquizadas: geração, transmissão, distribuição e comercialização. Sendo a forma tradicional de geração de energia elétrica chamada de Geração Centralizada (GC), que utiliza uma grande fonte geradora para a transformação da energia (BEZERRA, 2019).

Grandes usinas térmicas e hidrelétricas são exemplos de GC, porém estas usinas, em sua grande maioria, estão instaladas em locais distantes dos centros de consumo, sendo necessário a utilização de longas linhas de transmissão e distribuição de energia implicando em perdas técnicas, diminuindo a eficiência de todo o processo. A Geração Distribuída (GD), que está em crescente desenvolvimento no Brasil e no mundo, mostra-se como alternativa para mitigar tais ineficiências, pois são instaladas próxima das cargas minimizando o uso de linhas de transmissão.

Os estímulos à GD, segundo a Agência Nacional de Engenharia Elétrica (ANEEL), se justificam pelos potenciais benefícios que tal modalidade pode proporcionar ao sistema elétrico. Entre eles, estão o adiamento de investimentos em expansão dos sistemas de transmissão e de distribuição, o baixo impacto ambiental, adequação dos níveis de tensão, redução no carregamento das redes, a minimização das perdas elétricas e a diversificação da matriz energética (ANEEL, 2019).

O contexto descrito nos parágrafos precedentes conduz a criação de ambientes cada vez mais competitivos em busca da máxima eficiência dos elementos que compõem o sistema elétrico de potência. Além disso, o avanço tecnológico contínuo impulsiona uma expansão progressiva do SEP e implica na necessidade de planejamento por parte das concessionárias, pois as mesmas devem garantir os níveis adequados de qualidade de serviço e do produto entregue aos consumidores finais (BARBOSA, 2018).

Nesse sentido, o desenvolvimento de ferramentas computacionais que simulem o comportamento do sistema elétrico torna-se imprescindíveis, visto que o avanço em termos de metodologia, equipamentos e modelos devem ser disponibilizados aos profissionais do setor o mais rápido quanto possível (AGOSTINI; DECKER; SILVA, 2002).

Destaca-se também que a maioria dos programas consolidados no setor elétrico como por exemplo o Anarede, Anatem, CYME, ETAP, PowerFactory, PowerWorld Simulator, PSS®E, Power*Tools for Windows e SimPowerSystems são de caráter comercial, os chamados *softwares* proprietários, que só podem ser obtidos por meio de licenças com um alto custo de aquisição o que muitas vezes inviabiliza a sua utilização, sobretudo para usos com fins educacionais (OLIVEIRA *et al.*, 2019).

Como alternativa a esse segmento de softwares com licenças comerciais ou protetoras, surgiu o conceito de programas gratuitos e de código aberto, conhecido como *softwares open-sources Free and Open Source Software* (FOSS). Esse conceito traz como principais características o acesso livre à informação bem como à decisões realizadas em domínio público, promovendo a descentralização, simultaneidade e colaboração, permitindo ao programador a possibilidade de uma completa modificação e adaptação do programa para adequação aos problemas estudados (SARKINEN, 2007).

Um exemplo de *software opensource* é o simulador OpenDSS destinado à representação de sistemas distribuição, mantido e atualizado atualmente por um grupo de

engenheiros funcionários da *Electric Power Research Institute* (EPRI). Por se tratar de um *software* de código aberto existe um grupo de pesquisadores da cultura livre que desenvolveram um pacote em linguagem de programação *Python*, que implementa um conjunto de funções da biblioteca OpenDSS, tornando-o nativo em *Python*, fato que permite que entusiastas da cultura livre criem interfaces específicas para suas aplicações utilizando a linguagem. Chamado de OpenDSSDirect.py o pacote mencionado é multiplataforma e está disponível para *Windows*, *Mac* e *Linux* (PYPI, 2019).

A linguagem *Python* foi escolhida dentre outras linguagens de programação conhecidas na literatura pelo seu potencial crescimento de usabilidade. Junta-se a isso o fato de ser de código aberto, ser de alto nível e possuir uma sintaxe clara e concisa que favorece a legibilidade do código fonte, tornando a linguagem mais produtiva (BORGES, 2014).

A linguagem *Python* em contraponto a metodologia de construção de *softwares* da primeira geração apresenta-se como uma opção promissora para enfrentar os novos desafios da produção de ferramentas computacionais para a indústria da energia elétrica. Trata-se de uma linguagem de programação interpretada, orientada a objetos, utilizando-se da filosofia de Metodologia Orientada a Objetos (MOO) (MANZONI, 2005).

A MOO visa a construção de uma estrutura de dados sólida e consistente, a partir da qual são implementados *softwares* flexíveis, com alto grau de reutilização dos códigos e com facilidades para manutenções e atualizações (MANZONI, 2005).

Nesse sentido, o desenvolvimento de um *software* livre para o setor de sistemas elétricos de potência com aplicação na área educacional, tanto para pesquisa como para fins didáticos, e que também possa ser usado para buscar soluções de mercado é uma solução ideal para o contexto exposto, sobretudo quando é utilizado na sua filosofia de criação o conceito de FOSS e MOO, de modo que o mesmo se mantenha atualizado, evitando sua obsolescência, podendo contribuir para o avanço do setor nessa nova realidade operativa que se apresenta.

1.0.2 Objetivo

O objetivo deste Trabalho Final de Graduação (TFG) é o desenvolvimento de um *software* livre e de código aberto para estudos em SEP, que possua uma arquitetura padrão e modular que permita a fácil modificação e implementação de novas rotinas utilizando o código fonte publicamente disponibilizado. De modo geral, desenvolveu-se uma *Graphical User Interface* (GUI) que permite a conversão automática de dados de um SD fornecidos pela ANEEL, disponível por meio da BDGD, para o formato do *software* livre OpenDSS.

Inicialmente a aplicação tem como foco a análise de fluxo de potência no SD em regime permanente. Além disso, tem-se o objetivo de auxiliar estudantes e professores do Departamento de Engenharia Elétrica e Computação (DEEC) da Universidade Federal da Bahia sendo uma ferramenta de aplicação nas áreas de pesquisa e educação voltada ao

SEP.

Nesse sentido, no longo prazo pretende-se que o *software* funcione como uma plataforma de desenvolvimento colaborativo, onde estudantes de graduação e pós graduação possam desenvolver seus trabalhos e pesquisas somando esforços para o crescimento e aplicabilidade do *software*.

1.0.2.1 Objetivos específicos

Os objetivos específicos deste trabalho final de graduação:

- Realizar um levantamento bibliográfico sobre o tema do TFG, além de uma fundamentação teórica coesa da metodologia e métodos empregados no seu desenvolvimento.
- Desenvolver competência na área de programação especificamente na linguagem de programação *Python*. Pois a GUI citada foi criada utilizando o *framework* Qt, em sua versão *python*, o *PyQT5*, bem como o pacote *OpenDSSDirect.py* que é a versão do OpenDSS nativa.
- Implementar o programa utilizando o paradigma de programação de orientação a objetos.
- Entender e utilizar os dados públicos das concessionárias de distribuição, já que a dinâmica de identificação, acesso e entendimento do dados fornecidos pelo BDGD estão especificados no módulo 10 dos Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional (PRODIST) disponibilizados pela ANEEL de forma transparente no site oficial da agência.
- Criar a base de um *software* que aceitará futuras contribuições de novos TFG, trabalhos de iniciação científica, em diversas áreas tais como: proteção, qualidade de energia, máquinas elétricas e etc.
- Produzir um artigo cientifico com o resultado do presente trabalho, para viabilizar publicação após sua apresentação.

1.0.3 Metodologia

Segundo Köche (2016) o que leva o homem a produzir ciência é a busca por respostas dos problemas. Miguel e Sousa (2012) aponta que existe um caminho básico a ser percorrido durante a pesquisa: problema, método, solução, aplicação e resultado. Zanella (2006) diz que método é a maneira, é a forma que o cientista escolhe para ampliar o

conhecimento sobre determinado objeto, fato ou fenômeno. Mais do que isso, o método científico requer que seus resultados sejam reproduzíveis, de modo que forneça informações suficientes para que outras pessoas possam repetir o estudo (SILVA, 2003).

A pesquisa desenvolvida neste trabalho é de natureza exploratória e aplicada, buscando desse modo, não apenas gerar um novo conhecimento, mas, também aplicá-lo na prática. A técnica inicial foi a pesquisa bibliográfica e o método de pesquisa utilizado é o qualiquantitativo. Além disso, a forma de escrita deste TFG, para além de repetir padrões estabelecidos, tem como objetivo garantir principalmente que ao ler este trabalho, outros autores consigam repetir o método empregado, utilizando dos conceitos e ferramentas aqui compilados, entregando dessa forma uma contribuição significativa a comunidade acadêmica.

Uma das etapas mais importantes de um projeto de pesquisa é a revisão de literatura (SILVA; MENEZES, 2005). Nesse sentido, a primeira etapa do projeto foi a realização de uma revisão bibliográfica do tipo empírica, buscando de maneira robusta encontrar o estado da arte sobre o desenvolvimento de *software* e cultura de *software* livre. O objetivo principal foi entender quais as metodologias que estão sendo utilizadas para desenvolver ferramentas computacionais atualmente, sobretudo para aplicações no setor de energia elétrica.

Convém destacar, que neste trabalho, o conceito de revisão da literatura é diferente de fundamentação teórica. Como explica Azevedo (2017), a revisão de literatura tem como objetivo fornecer uma visão geral das fontes sobre um determinado tópico e tem características de investigação científica, ou seja, ela deve ser sistemática e abrangente. Já a fundamentação teórica representa a base teórica a partir da qual será feita a análise de dados da pesquisa e sua construção evidencia o domínio que o pesquisador tem sobre o tema, cada teoria a ser utilizada ao longo do trabalho será abordado em uma seção ou capítulo específico. Desta forma, foram escritos dois capítulos distintos um para cada propósito.

O referencial teórico escolhido, descrito na fundamentação, baseou-se principalmente em quatro livros principais: Engenharia de *Software*: Uma Abordagem Profissional; Engenharia de *Software*; Fundamentos de Engenharia de *Software*; Engenharia de *software*: fundamentos, métodos e padrões (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2011; CORTéS, 2013; PÁDUA, 2003). Além disso, buscou-se a complementação de conceitos a partir da inserção de contribuições de outros autores conforme pode ser visto no capítulo da fundamentação teórica.

A metodologia de pesquisa bibliográfica utilizada baseou-se na realizada por Álvarez (2019), porém com foco no *software* livre. A pesquisa foi realizada utilizando o portal de periódicos CAPES/MEC, a bases de dados *IEEE Xplore Digital Library* e a *Science Direct*, sendo os resultados sistemáticos apresentados por tabela agregada por termos de

busca, utilizando os critérios usados definidos da seguinte forma:

- Nível superior: revisados por pares;
- Tipo de recurso: conferências, artigos, artigos de jornais e revistas;
- Palavras Chaves: *free software*, *open source software*, *simulation software*, *power distribution*, *power system*, *graphical user interface*, *PyQt5*, *Paradigma Orientado a Objetos (POO)* e *software engineering development*;
- Refinamento por data de publicação: intervalo de busca do ano de 1990 à 2020 agregadas a cada cinco anos. A longevidade das buscas se dão com o objetivo de identificar o desenvolvimento dos conceitos e aplicações ao longo do tempo. Assumiu-se que artigos anteriores a essa data tornam-se obsoletos e uma revisão da literatura mais profunda não se enquadra no escopo deste trabalho;
- Busca específica por documentos de revisão bibliográfica incorporando as palavras de busca *review* e *survey* com o objetivo de ver outros trabalhos relevantes da área.

A Tabela 1 resume a quantidade de publicações encontradas de maneira geral no segmento de *softwares* não comerciais e devido a grande quantidade de material disponível foi necessário uma filtragem manual. Nessa etapa, a metodologia aplicada foi a busca por títulos de trabalho que remetessem ao tema especifico ou similar do presente TFG e posteriormente sendo realizada a leitura do resumo do artigo. A partir da leitura do resumo do artigo identificou-se a pertinência do conteúdo em relação a proposta do TFG e então uma leitura mais profunda do material era realizada incorporando-o ao texto de revisão.

Tabela 1 – Resultados de buscas bibliográficas

Palavra chave	1990 à 1995	1995 à 2000	2000 à 2005	2005 à 2010	2010 à 2015	2015 à 2020
free software	11	27	78	173	242	233
open source software	8	81	844	2747	2279	2584

Ressalta-se que a ferramenta computacional planejada tem a proposta de ser uma interface gráfica amigável. Durante a revisão da literatura foi possível encontrar referências de autores que desenvolveram *softwares* com o mesmo objetivo, e esse diálogo entre autores permitiu a escolha do *framework QT* em sua versão 5 em *Python*, o *PyQT5*. A escolha do *PyQT5*, além do motivo citado acima, foi fundamentalmente pelo fato do mesmo ser compatível com a licença pública geral reduzida (GNU/LGLP), bem documentada e com boa legibilidade , que será mais explicada na fundamentação teórica (SILVA *et al.*, 2016).

Desta forma, a etapa seguinte foi o aprofundamento de conhecimentos específicos sobre a linguagem de programação *Python* com o objetivo de investigar possíveis módulos do *framework PyQT5* que foram exploradas na construção da GUI. Por se tratar de uma linguagem que utiliza a filosofia de modelagem orientada a objetos, durante esta etapa da produção do projeto solidificou-se conceitos sobre MOO e produziu-se as primeiras experimentações de práticas de programação.

Como explicitado nas justificativas do presente projeto, foi utilizado como parte principal do *solver*, ou motor de cálculo, do *software*, o pacote *OpenDSSDirect*. Neste sentido, a terceira etapa da produção buscou entender as funções associadas a essa biblioteca, explorando a forma que deveriam ser usadas na formulação de rotinas específicas da interface.

O banco de dados georreferenciados das distribuidoras é a priori o componente fundamental para a construção dos sistemas elétricos de potência que pretende-se modelar como resultado deste trabalho, pois nele é que se encontra todas as informações referentes às redes elétricas das distribuidoras nacionais. Logo, entender o padrão de organização dos dados alocados neste banco foi imprescindível para o êxito desta tarefa, sendo assim a quarta etapa do projeto debruçou-se no estudo do módulo 10 do PRODIST, documento que explica a metodologia do Sistema de Informação Geográfica Regulatório, disponível no site da ANEEL.

A quinta etapa foi especificamente a elaboração da fundamentação teórica sobre o desenvolvimento de *software*, onde conceitos particulares dessa área da engenharia foram pesquisados e explicitados. A sexta etapa gerou o desenvolvimento do *script* base para leitura e organização da BDGD, criando o escopo da GUI. Nesta etapa já foi possível a construção de redes básicas e realizadas a primeiras simulações parciais.

A finalização do processo metodológico deu-se na etapa sete, onde formulou-se toda a arquitetura do *software* definindo o *layout* final do projeto, trazendo robustez e consolidando simulações contundentes e validadas. Para isso será realizado um estudo de caso mostrando a usabilidade do *software* desenvolvido.

1.0.4 Estrutura da Monografia

Para alcançar os objetivos propostos, além da presente introdução, este TFG se encontra assim estruturado:

- Fundamentação Teórica: O objetivo deste capítulo é apresentar a fundamentação teórica dos assuntos relacionados ao tema e objetivos deste TFG. Será exposto um panorama geral sobre o assunto, buscando elucidar os caminhos teóricos percorridos durante a elaboração da presente monografia, mostrando ao leitor conceitos que

serão utilizados no decorrer do trabalho, ancorados por autoras e autores referências na área de Engenharia e Desenvolvimento de *Software*.

- Revisão Bibliográfica: O objetivo deste capítulo é realizar um levantamento bibliográfico apresentando alguns trabalhos sobre a temática desse TFG, tendo como estratégia a continua revisão e ampliação do texto durante o desenvolvimento das atividades de pesquisa.

- Desenvolvimento: O objetivo desse capítulo é discorrer sobre como os conceitos apreendidos durante a fundamentação teórica e revisão bibliográfica foram implementados na evolução do desenvolvimento do presente trabalho para que alcançasse-se os resultados que serão apresentados.

- Resultados: O objetivo deste capítulo é apresentar uma aplicação do sistema como resultado final da elaboração do *software* SIPLA. Será elaborado um estudo de caso demonstrando sua usabilidade e demonstrando as possibilidades tanto para fins acadêmicos quanto para fins de mercado. O foco não é análise e sim à possibilidade de aplicação, ou seja, não serão dados muitos detalhes sobre o circuito que será mostrado no estudo de caso.

- Conclusão: O objetivo deste capítulo é sintetizar o trabalho realizado até então, destacando os principais pontos elaborados e seus resultados práticos preliminares. São ainda destacados alguns itens que poderão ser contemplados em pesquisas futuras como forma de incentivar outros trabalhos que colaborem no continuo desenvolvimento do *software*.

2 Fundamentação teórica

2.1 O *Software*

Segundo Sommerville (2011), o *software* pode ser definido como um programa de computador com documentação associada. Ou, como um conjunto de instruções que quando executadas, fornece características, funções e desempenhos desejados, segundo Pressman e Maxim (2016).

2.1.1 *Software* Livre

Os direitos autorais de um programa de computador é automaticamente de quem o escreve, permitindo que o autor controle a cópia, o uso e a adaptação do *software*. A disponibilização do sistema para terceiros é subordinada a autorização prévia para determinadas operações, isso é chamado de licença (ENGELFRIET, 2010). De acordo com Freitas (2009) o conceito de licença de *software* determina a forma que o *software* pode ser usado pela comunidade e sabe-se que dependendo dos desejos do autor várias licenças padrões estão disponíveis sendo possível, inclusive, o autor escrever sua própria licença.

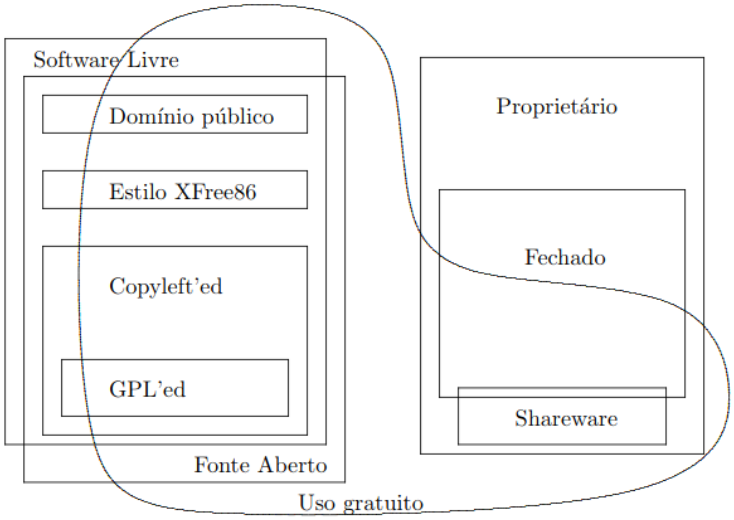
A licença citada está ancorada pelos regimentos das leis de *copyright* explicitada por meio de um documento junto ao *software*. A licença apresenta de forma clara em quais condições o *software* pode ser utilizado. Desta forma, o *software* pode ser caracterizado em função de sua propriedade (FREITAS, 2009; ENGELFRIET, 2010):

- *Software* Proprietário: *software* que proíbe a redistribuição e alteração pelo usuário, apenas o uso através da aquisição do mesmo.
- *Software Freeware*: *software* que permite redistribuição, mas não qualquer modificação nos códigos-fonte e geralmente nem há acesso a essas fontes.
- *Software Shareware*: *software* que permite redistribuição, mas que restringe o uso de acordo com uma condição específica, normalmente associada à um tempo limite de uso.
- *Software* Livre: software que oferece ao usuário o direito de usar, estudar, modificar e redistribuir os códigos-fonte de qualquer maneira e em qualquer tempo.
- *Software Open Source*: software disponível gratuitamente, licenciado para uso e modificação, incluindo atividades comerciais. Qualquer um pode copiar, distribuir ou

incorporá-lo como achar melhor, sem ter que pagar *royalties* ou mesmo negociar um contrato de licença.

A Figura 1 mostra as interseções entre as várias categorias supracitadas.

Figura 1 – Categorias de licenças de software.



Convém destacar que os conceitos de *software livre* e *software opensource* são diferentes embora estejam relacionados. Conforme destaca Silva, Aparicio e Costa (2019), em geral, são referidos como FOSS, sendo que o *software livre* concentra-se no movimento social e na filosofia de uso, enquanto o *software opensource* está focado na metodologia de desenvolvimento e qualidade do *software*.

A definição de *software* livre está em conformidade com a trazida pela *Free Software Foundation* (FSF) definindo-o como aquele que respeita a liberdade e senso de comunidade dos usuários, ou seja, os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o *software*.

Embora seja contra intuitivo, Anand *et al.* (2018) diz que o *software livre* pode ser empacotado e distribuído por uma taxa, sendo o termo livre associado à capacidade de reutilizá-lo, modificado ou não, como parte de outro pacote de *software*. Segundo Beraldo e Fontenelle (2020) *software* livre é uma questão de liberdade, não de preço e conforme destaca Engelfriet (2010) é importante não confundir *software* livre com *software* grátis porque a liberdade associada ao *software* livre de copiar, modificar e redistribuir, independe de gratuidade.

De acordo com Gomes (2013), a filosofia de *software* livre foi criada por Richard Stallman em 1983 por meio de um projeto GNU/GNU's Not *Unix*. Richard Stallman, fundador da FSF, era um pesquisador do Laboratório de Inteligência Artificial do *Massachusets Institute of Technology* (MIT) e desejava desenvolver um sistema operacional que fosse semelhante ao *Unix*, porém que não dependesse de licenças proprietárias

de uso. Desta forma, em contraponto as licenças *copyright*, a FSF criou a Lincença Pública Geral (GPL), popularmente conhecida como *copyleft*, visando evitar o surgimento de proprietários para os *softwares* livres.

Muitas vezes, o termo *software* livre (*free software*) se confunde com *software* aberto (*software open source*), sobretudo após a criação da *Open Source Initiative* (OSI), em 1988. Contudo, ao contrário da FSF, a OSI não tem licenças próprias, porém certifica todas as dezenas licenças atualmente existentes no mercado dentro de seus próprios parâmetros de *software* aberto. A OSI é uma entidade conciliadora, que tenta garantir às empresas que praticam o modelo comercial de desenvolvimento de *software*, uma porta de entrada no movimento de *software* aberto sem necessariamente adotar as licenças da FSF, representando assim uma espécie de alternativa intermediária (CARVALHO, 2011). Para Carvalho (2011) todo *software* livre é, por definição, aberto, porém nem todo *software* aberto é livre.

2.2 Engenharia de *Software*

O termo Engenharia de *Software* (ES) remonta-se à conferência da Organização do Tratado do Atlântico Norte (OTAN) de 1968. Nesse período, a indústria reconheceu que para criar soluções econômicas para problemas práticos, o conhecimento científico teria que ser aplicado aos *softwares* (KAZMAN, 2017). Atualmente, a ES é definida como uma disciplina que utiliza um conjunto de métodos, de técnicas e de ferramentas para analisar, projetar e gerenciar o desenvolvimento e a manutenção do *software*, visando produzir e mantê-lo dentro de prazos, custos e qualidade estimados (CORTéS, 2013).

Sommerville (2011) afirma que a engenharia de *software* é uma tecnologia de importância crítica para o futuro da humanidade. Além disso, o *software* tem um duplo papel, pois é um produto e um veículo para distribuir a informação na contemporaneidade (PRESSMAN; MAXIM, 2016).

Para Sommerville (2011), os serviços e a infraestrutura nacional, nos seguimentos de energia, de comunicação e de transporte dependem de sistemas computacionais cada vez mais complexos. Os sistemas modernos de *software* têm se tornado mais dinâmicos, abertos e hiper conectados, ou seja, o *software* além de seus papéis tradicionais, está desempenhando cada vez mais um papel fundamental na abordagem dos principais desafios da sociedade (KAZMAN; PASQUALE, 2020). Qualquer abordagem da engenharia deve ser fundamentada em um comprometimento organizacional com a qualidade. Neste caso, pode-se destacar a engenharia de *software* como uma tecnologia em camadas, conforme mostrado na Figura 2 (PRESSMAN; MAXIM, 2016).

Quando fala-se em qualidade de *software*, deve-se levar em conta que o mesmo é usado e alterado por pessoas, além de seus desenvolvedores, isto é, o foco não é apenas

Figura 2 – Camadas de Engenharia de *Software*.

Fonte: baseado em Pressman e Maxim (2016).

o que o *software* faz, mas seu comportamento enquanto ele está sendo executado. Devido a essa característica, portanto, deve ser considerado a estrutura, a organização dos programas e a documentação associada (SOMMERVILLE, 2011).

2.3 O Processo de *Software*

O processo de *software* define a metodologia que será estabelecida para a entrega efetiva do sistema, constituindo a base para o controle do gerenciamento de projetos. Pode-se definir processos como um conjunto de atividades de trabalho, de ações e de tarefas realizadas quando algum artefato de *software* necessita ser criado (PRESSMAN; MAXIM, 2016).

De acordo com Dooley (2011), o processo de *software* restringe-se a cinco atividades metodológicas: comunicação, planejamento, modelagem, construção e entrega do *software*. Nesse sentido, Cortés (2013) reforça que a padronização e a sistematização do processo de desenvolvimento de *software* traz algumas vantagens, pois estabelece um passo-a-passo a ser seguindo, de forma que atinja-se o exito ao final da atividade.

Escolher qual tipo de processo, ou Ciclo de Vida de *Software*, implementar é indicado como uma das primeiras decisões a serem tomadas pelo engenheiro de *software*, visto que irá determinar a abordagem sistemática que será seguida para desenvolver o projeto. Entretanto, esta escolha depende dos seguintes fatores: natureza do projeto e da aplicação, métodos e ferramentas a serem usados ou disponíveis, controles requeridos e produtos a serem entregues e prazos e recursos disponíveis (CORTéS, 2013).

O gerenciamento de projetos de desenvolvimento de *software* se divide em dois tipos, modelos orientados a planos tradicionais e os modelos de desenvolvimento ágil (DOOLEY, 2011). Portanto, é importante explicar os modelos de processos existentes na literatura para escolher qual adapta-se ao projeto proposto.

2.3.1 Processos Tradicionais

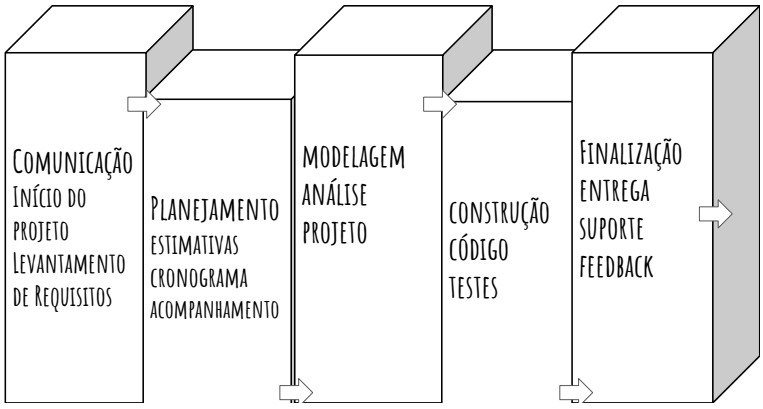
Serão descritos, de forma sistematizada e objetiva, os modelos ditos tradicionais: modelo em cascata, modelos de processo incremental, modelos de processo evolucionários, modelo espiral, modelos concorrentes e, por fim, o modelo conhecido como processo unificado.

2.3.1.1 O Modelo Cascata

De acordo com Dooley (2011), o modelo em cascata foi criado na década de 1970 por Winston Royce, sendo uma das primeiras tentativas de estruturar o processo de desenvolvimento de *software*. Nessa abordagem sequencial um projeto progride através de um conjunto de fases distintas com marcos e artefatos bem definidos (WŁODARSKI; PONISZEWSKA-MARANDA, 2019).

Basicamente, o modelo em cascata pode ser descrito como um modelo linear, pois há um encadeamento entre uma fase e outra do processo (SOMMERVILLE, 2011). Esse modelo também é chamado por alguns autores como ciclo de vida clássico, sugerindo uma abordagem sequencial e sistemática para o desenvolvimento do *software* (PRESSMAN; MAXIM, 2016).

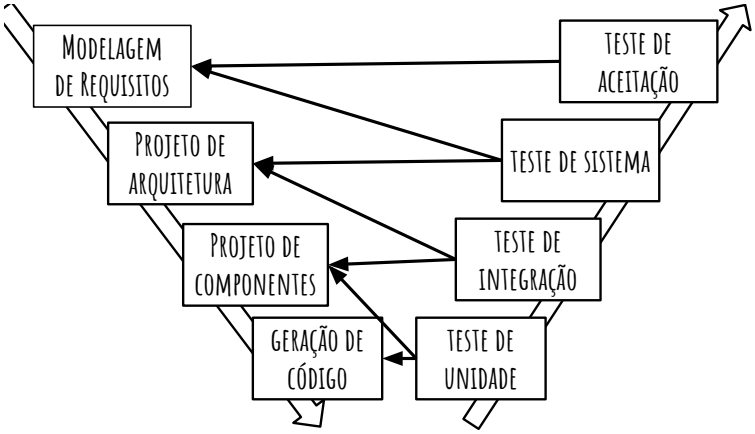
Figura 3 – O modelo cascata.



Fonte: baseado em Pressman e Maxim (2016).

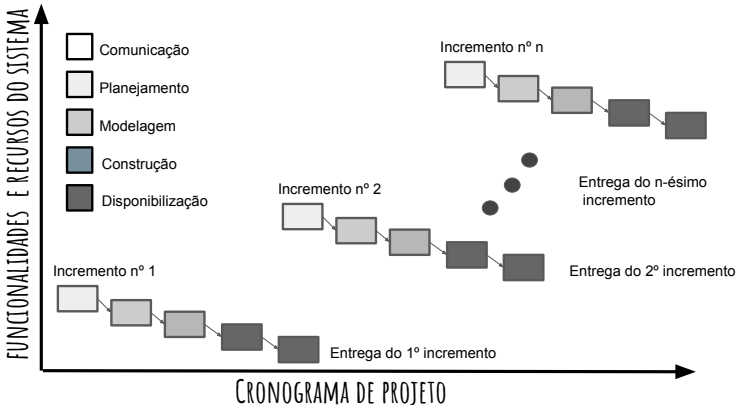
As Figuras 3 e 4 mostram dois esquemáticos de funcionamento do processo no modelo em cascata que demonstram a demarcação conceituada de pontos de controle que facilitam a gestão do projeto tornando-o confiável e utilizável (PÁDUA, 2003). Contudo, segundo Pressman e Maxim (2016) o modelo em cascata é frequentemente considerado inadequado para o trabalho de desenvolvimento de *softwares* atualmente, pois o ritmo de trabalho tem sido acelerado e sujeito a inúmeras mudanças ao decorrer dos projetos, o que torna esse modelo útil apenas para projetos nos quais os requisitos são fixo.

Figura 4 – O modelo cascata formato V.



Fonte: baseado em Pressman e Maxim (2016).

Figura 5 – O modelo incremental.



Fonte: baseado em Pressman e Maxim (2016).

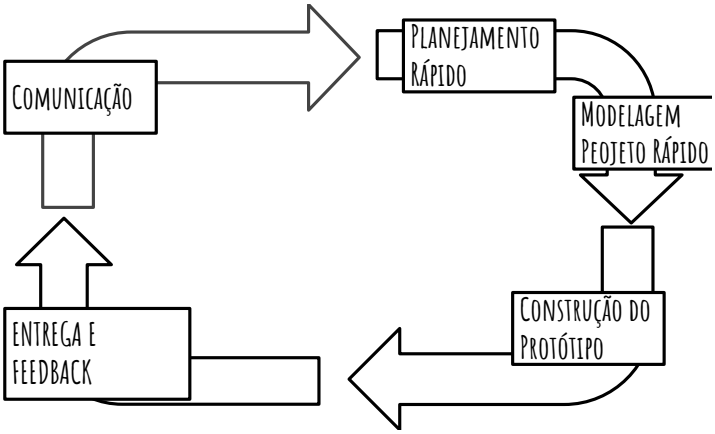
2.3.1.2 O Modelo Incremental

Esse modelo, observado na Figura 5, é aplicável quando o projetista tem uma ideia bem definida dos requisitos iniciais do sistema, porém não consegue implementar sobre uma perspectiva puramente linear (PRESSMAN; MAXIM, 2016).

Cortés (2013) destaca que nesse modelo o *feedback* do usuário é fundamental, pois as etapas do projeto complementam-se na medida em que informações novas são incorporadas, ou seja, ao longo de cada iteração é desenvolvido o *software* e a especificação de forma conjunta.

Este modelo para Sommerville (2011) mostra-se mais fidedigno a forma que pessoas resolvem os problemas. Conforme resume Bezerra (2015), nesse modelo um sistema de *software* é desenvolvido em vários passos similares, o que o torno iterativo, sendo que em cada passo, o sistema é estendido com mais funcionalidades logo, incremental.

Figura 6 – O Paradigma da Prototipação.



Fonte: baseado em Pressman e Maxim (2016).

2.3.1.3 Os Modelos Evolucionários

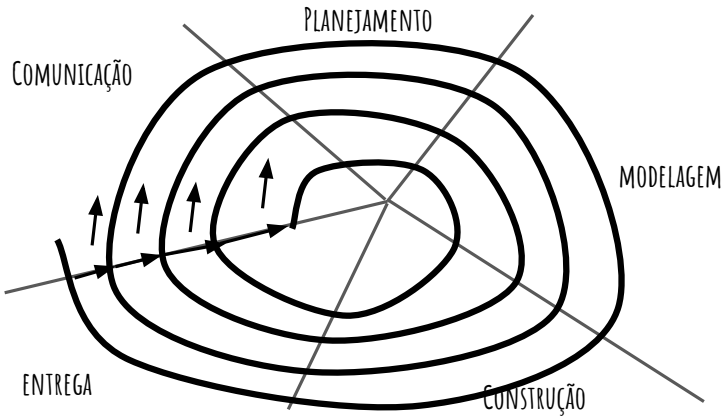
Pressman e Maxim (2016) aponta que a dinâmica atual do desenvolvimento de *software* baseia-se na premissa que ao passo que o projeto avança os requisitos do negócio e do produto mudam com frequência. Além disso, Cortés (2013) destaca que o produto é desenvolvido ao longo de uma sequência de versões cada vez mais detalhada e completa, ou seja, há um refinamento constante fazendo com que o grau de risco de finalização de projeto diminua. Neste contexto, surge dois sub-modelos, enquadrados na categoria evolucionária: a prototipação e o modelo espiral.

Na prototipação, de acordo com Pressman e Maxim (2016), o cliente apresenta uma série de objetivos genéricos ao desenvolvedor sem identificar requisitos específicos e na medida que o projeto avança, o projetista identifica os requisitos e esquematiza de forma a atender as demandas. Sommerville (2011) elucida que um protótipo é uma versão inicial de um sistema de *software*, usado para demonstrar conceitos, experimentar opções de projeto e depurar problemas e buscar suas possíveis soluções. Pode-se observar na Figura 6 o fluxograma denominado Paradigma da Prototipação que mostra todas as fases realizadas neste modelo de processo.

O modelo espiral é a união de dois citados anteriormente: o modelo de prototipação e o modelo rudimentar de cascata, sendo que neste o processo de *software* é representado por uma espiral e não como uma sequência de atividades com alguns retornos de uma para outra, logo cada volta na espiral representa uma fase do processo de *software* (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2011).

No modelo espiral, Figura 7, as atividades de análise de riscos e prototipagem são obrigatórias em todos os estágios do projeto. Os riscos podem ser monitorados de perto e ações preventiva e corretivas podem ser aplicadas minimizando seus impactos (CORTéS, 2013). Contudo, a aplicação desse modelo trás alguns inconvenientes, como por exemplo requerer uma gestão muito sofisticada para ser previsível e confiável (PÁDUA, 2003).

Figura 7 – O modelo espiral adaptado.



Fonte: baseado em Pressman e Maxim (2016).

2.3.1.4 O Modelo Concorrente

O modelo concorrente, por sua vez, permite que o desenvolvedor represente os elementos concorrentes e iterativos de qualquer um dos modelos citados anteriormente. Desta forma, a modelagem concorrente se aplica a todos os tipos de desenvolvimento de *software* e fornece uma imagem precisa do estado atual de um protejo. Resumindo, cada atividade, ação ou tarefa na rede existe simultaneamente com outras atividades, ações ou tarefas (PRESSMAN; MAXIM, 2016).

Os modelos conhecidos como tradicionais não são os únicos modelos de processos explorados pela engenharia de *software*. Exitem diversos outros modelos que podem ser encontrados na literatura dentro da categoria de Modelos de Processo Especializados, como por exemplo: Modelo de Desenvolvimento Baseado em Componentes, Modelo de Métodos Formais e Modelo de Desenvolvimento Orientado a Aspectos (WERNER; BRAGA, 2000; RIBEIRO, 2000; SILVA, 2006). Porém, o detalhamento de cada modelo citado foge o escopo definido para esse trabalho.

Entretanto, um tipo específico de modelo precisa ser descrito, o Processo Unificado, pois os conceitos derivados dele serão bastante utilizados durante a elaboração do projeto proposto.

2.3.2 O Processo Unificado

Criado por Ivar Jacobson, Grady Booch e James Rumbaugh o processo unificado é um processo de desenvolvimento interativo para sistemas complexos, geralmente mais burocráticos com um forte foco nos casos de uso, que por sua vez sugerem requisitos com uma ênfase na escolha da melhor arquitetura (PRESSMAN; MAXIM, 2016; YOUNG, 2013). Como resultado de uma unificação de experiências dos criadores citados, surgiu a UML que contém uma notação robusta para modelagem e desenvolvimento de siste-

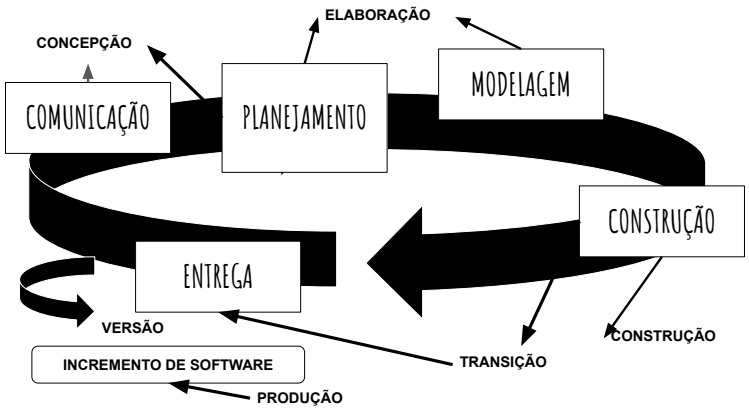
mas orientados a objetos, sendo a mesma explorada com mais detalhes posteriormente. Segundo Manzoni (2001) a programação orientada a objeto está intimamente ligada a UML, contudo, não significa que ambas tecnologias estejam atreladas entre si, visto que a primeira trata-se de técnicas de programação e a segunda trata-se de projeto de *software*.

O processo unificado é normalmente descrito em três perspectivas (SOMMERVILLE, 2011):

- Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo;
- Uma perspectiva estática, que mostra as atividades realizadas no processo;
- Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.

A descrição do processo unificado combina as perspectivas estáticas e dinâmicas em um único diagrama, conforme visto na Figura 8.

Figura 8 – O processo unificado.



Fonte: baseado em Pressman e Maxim (2016).

Pode-se aprofundar os conceitos de cada etapa da seguinte forma (SOMMERVILLE, 2011):

- Concepção: O objetivo da fase de concepção é estabelecer um caso de negócios para o sistema. Deve-se identificar todas as entidades externas (pessoas e sistemas) que vão interagir com o sistema e definir as interações. Então, usa-se as informações para avaliar a contribuição do sistema para o negócio. Se essa contribuição for pequena, então o projeto poderá ser cancelado depois dessa fase.
- Elaboração: As metas definidas para a fase de elaboração são desenvolver uma compreensão do problema dominante, estabelecer um *framework* da arquitetura para o sistema, desenvolver o plano do projeto e identificar os maiores riscos do projeto. ao findar da fase, deve-se obter um modelo de requisitos para o sistema,

que pode ser um conjunto de casos de uso da UML, uma descrição da arquitetura ou um plano de desenvolvimento do *software*.

- Construção: A fase de construção envolve três etapas: projeto, programação e testes do sistema. Durante essa fase, as partes do sistema são desenvolvidas em paralelo e integradas. Ao concluir a fase, deve-se obter um sistema de *software* já funcionando, bem como a documentação associada pronta para ser entregue aos usuários.
- Transição: A fase final do processo unificado implica transferência do sistema da comunidade de desenvolvimento para a comunidade de usuários e em seu funcionamento em um ambiente real. Na conclusão dessa etapa, deve-se obter um sistema de *software* documentado e funcionando corretamente em seu ambiente operacional.

Em suma, os modelos de processos, tradicionais ou específicos, são aplicados na tentativa de organizar e estruturar o desenvolvimento de um *software*. Se o processo for fraco a consequência imediata é um produto final de baixa qualidade, contudo, o conceito de processo deve ser sempre flexível, pois uma rigidez no mesmo também provocará consequências indesejadas, ou seja, encontrar o equilíbrio é o melhor caminho a seguir.

2.4 Desenvolvimento ágil

Observa-se que a aplicação dos modelos tradicionais exige um nível de rigorosidade na sua aplicação, fazendo com que, em algumas situações, engessem o processo de desenvolvimento de *software*, tornando-o pouco flexível. Em 2001, com o objetivo de flexibilizar o processo de desenvolvimento de *software*, surgiu a Aliança dos Ágeis, onde através de um manifesto, Manifesto para o Desenvolvimento Ágil de *Software* (DAS), 17 renomados desenvolvedores, autores e consultores declararam quatro novos princípios para ser valorizado no processo de desenvolvimento de *software* em contraponto aos principias utilizados pelos processos ditos tradicionais (PRESSMAN; MAXIM, 2016).

Por meio do manifesto, estabeleceu-se como princípios do desenvolvimento ágil:

- Indivíduos e interações acima de processos e ferramentas.
- *Software* operacional acima de documentação completa.
- Colaboração dos clientes acima de negociação contratual.
- Respostas a mudanças acima de seguir um plano.

Em resumo, DAS é um conjunto de abordagens de desenvolvimento baseados na incrementalidade. Destacando-se a presença de características como à maleabilidade, planejamento adaptativo, liberação frequente e incremental, desenvolvimento dentro do prazo

e comunicação contínua com o cliente (SINGH; CHAUHAN; POPLI, 2019). Uma evidente característica do DAS é a flexibilidade de se adaptar ao invés de serem preditivas, ou seja, os novos fatores são adaptados durante o desenvolvimento ao invés de predizer tudo o que pode ou não acontecer durante o projeto (SANTOS *et al.*, 2018).

2.4.1 Processo Ágil

A engenharia de *software* ágil defende a satisfação do cliente e a entrega incremental antecipada do projeto, utilizando-se de métodos informais, artefatos de engenharia mínimos, prezando substancialmente por simplicidade do desenvolvimento geral (PRESSMAN; MAXIM, 2016).

Existem alguns métodos da filosofia ágil conhecidos na literatura, como por exemplo: *Scrum*, *Crystal*, desenvolvimento de *software* adaptativo, DSDM e desenvolvimento dirigido a características. Porém, o mais conhecido e amplamente utilizado tem sido o *Extreme Programming* (XP) (SOMMERVILLE, 2011).

2.4.1.1 *Extreme Programming*

A *Extreme Programming* (XP) foi criada por volta de 1995 por Kent Beck e Ward Cunningham, sendo descrita como uma maneira leve, eficiente, de baixo risco, flexível, previsível, científica e divertida de desenvolver *software*, utilizando uma MOO como seu paradigma de desenvolvimento, realizando um conjunto de regras e práticas no contexto de atividades metodológicas de planejamento, projeto, codificação e testes (DOOLEY, 2011; PRESSMAN; MAXIM, 2016).

Na XP os requisitos são expressos como cenários, chamados de estórias de usuários, que são implementados diretamente como uma série de tarefas. Neste processo os clientes estão intimamente envolvidos na especificação e na priorização dos requisitos do sistema, ou seja, o cliente é parte da equipe de desenvolvimento e discute os cenários com outros membros da equipe (SOMMERVILLE, 2011).

As atividades chaves do processo XP serão explicitados a seguir (PRESSMAN; MAXIM, 2016):

- Planejamento: é uma atividade de levantamento de requisitos, a principal característica desenvolvida pela equipe de desenvolvimento é ouvir. A atividade de ouvir conduz a criação de um conjunto de estórias que irão descrever o resultado, as características e as funcionalidades solicitadas para o *software* a ser construído. Não é fixa a quantidade de estórias que podem ser contadas pelo cliente, uma vez que cada estória será responsável por uma versão do produto a ser entregue.
- Projeto: esta atividade está baseada no principio *Keep It Simple Stupid!* (KISS), que significa não complique. O principio estabelece que vale mais um projeto simples a

um complexo. A XP estimula o uso de cartões Classe Responsabilidade Colaborador (CRC) para organizar e identificar as classes orientadas a objeto para o incremento do *software*. Um aspecto central é que a elaboração do projeto ocorre tanto antes quanto depois de ser iniciado a codificação.

- Codificação: após a fase de projeto, a uma etapa conhecida como testes de unidades, onde a implementação de cada história criada é testada gerando versões de incremento de *software*. Na fase de codificação o modelo XP recomenda que duas pessoas trabalhem juntas para criação dos códigos, baseado na ideia de que, geralmente, duas pessoas funcionam melhor do que uma, garantindo uma melhor qualidade no trabalho. À medida que a dupla de programadores conclui o trabalho o código que desenvolveram é incrementado ao trabalho.

- Testes: os testes de unidades criados devem ser implementados usando-se uma metodologia que os capacite a ser automatizadas, com o objetivo de de facilitar a repetição da execução. Nesta metodologia os testes de integração e validação podem ocorrer diariamente permitindo que a equipe tenham uma indicação continua de progresso, ou seja, a correção de possíveis problemas ocorrem de forma otimizada.

O mundo do desenvolvimento de *software* é extremamente evolutivo e adaptativo. Uma evolução do método *Extreme Programming* (XP) é a Industrial XP (IXP), que trás o mesmo espírito: centrado no cliente e orientado a testes. O que as diferenciam, principalmente, é a maior inclusão dos processos de gerenciamento, sendo incorporada seis novas práticas importantíssimas para ajudar na garantia que o projeto XP será finalizado com exito (PRESSMAN; MAXIM, 2016):

- Avaliação Imediata: a equipe IXP verifica se todos os membros do projeto estão a bordo, ou seja, possuem o ambiente correto estabelecido e entendem os níveis de habilidade envolvidos.

- Comunidade de projeto: a equipe IXP determina se as pessoas certas, com habilidades e o treinamento corretos, estão prontas para o projeto.

- Mapeamento do projeto: a equipe IXP avalia o projeto e determina se ele condiz com a estrutura da organização.

- Gerenciamento orientado a testes: a equipe IXP determina metas que avaliem o progresso, realizando seu acompanhamento.

- Retrospectivas: a equipe IXP realiza uma revisão técnica após a entrega de um incremento de *software*. É focado numa revisão que avalie problemas, eventos e lições apreendidas ao longo de todo processo, seja na versão incremental ou na versão final.

- Aprendizagem contínua: a equipe IXP é estimulada a aprender novos métodos e técnicas que possam levar a um produto de qualidade mais alta.

2.4.2 Outros modelos de processos ágeis

Como dito, dentre os processos ágeis existentes o mais utilizado é o XP. Porém, um outro método que tem se destacado é um desenvolvido por Jeff Sutherland e sua equipe nos anos 1990, o *Scrum*. Os princípios do *Scrum* são coerentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades metodológicas: requisitos, análise, projeto evolução e entrega. Em cada atividade metodológica ocorrem tarefas realizadas dentro de um padrão de processo chamado *sprint*.

Cada um desses padrões de processo define um conjunto de atividades de desenvolvimento (PRESSMAN; MAXIM, 2016):

- *Backlog*: uma lista com propriedades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente.
- *Sprints*: consistem em unidades de trabalho solicitadas para atingir um requisito estabelecido no *Backlog* e que precisam ser ajustados dentro de uma meta.
- Reuniões *Scrum*: são reuniões curtas, realizadas diariamente pela equipe *Scrum*. Essas reunião são norteadas por três perguntas chaves: O que você realizou desde a ultima reunião de equipe? Quais obstáculos está encontrando? O que planeja realizar até a próxima reunião da equipe?
- *Demos*: entrega do incremento de *software* ao cliente para que as funcionalidades implementadas possa ser demonstrada e avaliada.

2.5 Modelagem de *Software*

Uma das fases mais importantes de um projeto de desenvolvimento de *software* é sua modelagem. Para tanto, é necessário que se aplique uma série de princípios, conceitos e métodos.

2.5.1 Princípios fundamentais

Segundo Pressman e Maxim (2016), a ES é norteada por um conjunto de princípios fundamentais que colaboram na aplicação de processos de *software* significativo e na execução de métodos de ES eficazes. Os princípios fundamentais que serão descritos não substituem os abordados anteriormente, porém suprem a abstração que eles possuem e

estes devem ser utilizados na metodologia do processo de construção do *software* durante o desenvolvimento deste trabalho.

2.5.1.1 Princípios que orientam processos

Estes princípios conforme lembra Pressman e Maxim (2016) devem ser aplicados na metodologia de todos os processos.

1. Seja ágil: Mantenha a abordagem técnica tão simples quanto possível.

2. Concentre-se na qualidade em todas as etapas: A condição final de toda atividade, ação e tarefa do processo deve se concentrar na qualidade do produto.

3. Esteja prontos para adaptações: No desenvolvimento de processos não há espaços para dogmas, quando necessário adapte a abordagem impostas pelo problema, pelas pessoas ou pelo próprio projeto.

4. Monte uma equipe eficiente: Forme uma equipe que se organize, que preze pela confiança e respeito mútuo.

5. Estabeleça mecanismos para comunicação e coordenação: Evite falhas de projeto devido a omissão de informações, ou falta de esforço por componentes da equipe.

6. Gerencie mudanças: Gerencie mudanças durante o projeto com responsabilidade, estabeleça critérios claros sobre solicitação, avaliação, aprovação e implementação de mudanças.

7. Avalie os riscos: Estabeleça planos de contingência que poderão ser usados como base das tarefas de engenharia de segurança.

2.5.1.2 Princípios que orientam a prática

Estes princípios destinam-se sobretudo a orientação do trabalho técnico e são essenciais para a prática da ES.

1. Divida e conquiste: Um problema será mais fácil de resolver se for subdivididos em conjuntos de interesse.

2. Compreenda o uso da abstração: O termo abstrair está vinculado a simplificação de algum elemento complexo de um sistema. Na prática de ES usa-se diferentes níveis de abstração, cada um incorporando ou implicando um significado que deve ser comunicado.

3. Esforce-se pela consistência: Deve-se buscar a consistência sempre, seja no modelo de analise, desenvolvimento do *software* ou no código fonte. A padronização facilita a compreensão e oferecer uma aplicação ergonomicamente positiva.

4. Construa *softwares* que apresente modularidade efetiva: Qualquer sistema complexo pode ser dividido em módulos, sendo que esses módulos devem se interconectar de uma maneira relativamente simples.

5. Verifique Padrões: Os padrões de projeto geralmente são recomendados para problemas mais amplos, porém sua aplicação na integração de sistemas permite que os componentes evoluam de forma independente.

6. Lembre-se que alguém fará a manutenção do *software*: Defeitos sempre aparecem, nesse sentido a aplicação será corrigida, adaptada ou adicionada novas funcionalidades por partes dos envolvidos. As atividades de manutenção podem ser facilitadas se for aplicada uma pratica de engenharia consistente ao longo do processo.

2.5.2 Princípios das atividades metodológicas

Os princípios descritos a seguir fomentam o sucesso de cada atividade metodológica genérica definida como parte do processo de *software*. Serão abordados apenas princípios que devem ser aproveitadas no escopo desta monografia.

2.5.2.1 Princípios da Comunicação

Essa etapa é fundamental, pois antes mesmo dos requisitos expostos pelo clientes serem modelados, eles devem ser coletados por meio da atividade da comunicação. A comunicação e todos os seus detalhes constituem uma das atividades mais desafiadoras. Serão abordados alguns princípios que se aplicam no escopo deste trabalho.

1. Ouça: Ouvir é mais importante do que dar respostas.

2. Comunicar-se pessoalmente é melhor: No contato presencial com o cliente a clareza das informações tendem a ficar mais precisas, recursos como desenhos ou esboços podem sr usados e servem como foco para discussão.

3. Anote e documente as decisões: As coisas tendem a cair no esquecimento, logo documente tudo quanto possível.

4. Faltando clareza desenhe: A comunicação verbal nem sempre é suficiente, nesse sentido um desenho ou esboço ajudam a dar clareza a informação.

2.5.2.2 Princípios do planejamento

Os princípios da comunicação colaboram com as metas e os propósitos gerais do projeto, porém deve-se definir um plano a ser seguido para alcança-los. Nesse sentido, o ato de planejar consiste em um conjunto de técnicas e práticas gerenciais que estabelecem um roteiro a ser seguido.

Desta forma, o planejamento de um projeto de *software* deve ser construído de forma equilibrada e para tanto, alguns princípios são recomendados (PRESSMAN; MAXIM, 2016):

1. Entenda o Escopo do projeto: saber onde quer chegar é fundamental, o escopo aponta um caminho.
2. *Reconheça que o planejamento é iterativo*: todo planejamento é flexível, pois é previsto que após o inicio de um projeto sempre ocorra modificações.
3. Seja Realista: as pessoas não trabalham 100% de todos os dias. Mudanças ocorrem e omissões e ambiguidades são fatos, considere essa realidade ao estabelecer um plano.
4. Verifique o plano com frequência e faça ajustes necessários: é provável atrasos nos projetos de *softwares*, logo é recomendado verificar diariamente o seu progresso, para identificar de imediato possíveis problemas e buscar soluções.

2.5.2.3 Princípios da modelagem

Quando a aplicação a ser desenvolvida é um *software* o modelo deve ser capaz de representar as informações, a arquitetura e as funções solicitadas pelo usuário. Os modelos devem satisfazer esses objetivos em diferentes níveis de abstrações, inicialmente descreve o *software* do ponto de vista do cliente depois num nível mais técnico.

Na ES exitem duas classes de modelos: modelos de requisitos e de projetos. Os modelos de requisitos se relacionam com as solicitações dos clientes, que descrevem o *software* em três domínios: o domínio de informação, o domínio funcional e o domínio comportamental. Já os modelos de projeto se concentram nas características do *software* que colaboram com os desenvolvedores a construí-lo com eficiência: a arquitetura, a interface do usuário e os detalhes do componente.

1. O objetivo principal da equipe é construir *software*, não criar modelos: a meta deve ser entregar o *software* ao cliente no menor prazo possível, logo deve-se priorizar o princípio da agilidade.
2. Construa modelos que facilitem alterações: deve-se considerar que todo modelo será alterado, mas isso não significa que sua flexibilidade seja relapsa.

3. Crie modelos úteis, mas esqueça a construção de modelos perfeitos: a atividade de desenvolvimento de engenheiros de *software* ao construir modelos de projeto e requisitos por vezes chegam a sua exaustão para obtenção de modelos absolutamente completos e internamente consistentes, porém o esforço não vale os benefícios resultantes. Logo, iterar indefinidamente para tornar um modelo perfeito não supre a necessidade da agilidade.

4. Não seja dogmático quanto a sintaxe do modelo, se ela consegue transmitir o conteúdo, a representação é secundária: é fato que todos os integrantes da equipe devem usar uma notação consistente, porém deve-se considerar que o mais relevante é que a informação seja transmitida para próxima tarefa com sucesso, logos se o modelo viabilizar isso com exito a sintaxe incorreta deve ser perdoada.

2.5.2.3.1 Princípios da modelagem de requisitos

A primeira iniciativa de um trabalho técnico deve ser criar um conjunto de requisitos para todas as tarefas de engenharia, sendo que esta etapa é uma das mais difíceis enfrentados por um engenheiro de *software*. Mais adiante será apresentado com mais profundidade o tema requisitos, mas a seguir apresentaremos alguns princípios para serem aplicados em sua modelagem.

Ressalta-se que cada um dos métodos de análise tem um ponto de vista particular, mas todos estão inter-relacionados por princípios operacionais.

1. O universo de informações de um problema deve ser representado e compreendido: o universo de informações engloba os dados constantes no sistema, os dados que fluem para fora do sistema e a armazenagem de dados que coleta e organiza objetos de dados persistentes.

2. As funções executada pelo software devem ser definidas: as funções do *software* oferecem benefício aos usuários finais do sistema. Elas podem ser descritas em diferentes níveis de abstração, desde uma afirmação geral até uma descrição detalhada dos elementos de processo que devem ser requisitados.

3. O comportamento do *software* deve ser representado: o comportamento de um *software* é comandado por sua interação com o ambiente externo, logo toda informação deve ser explicitada.

4. Os modelos que representam informação, função e comportamento devem ser divididas de modo a revelar detalhes em camadas: a modelagem de requisitos é a primeira etapa da solução de um problema de engenharia de *software*. Um problema grande

e complexo pode ser dividido em subproblemas até que cada um se torne relativamente fácil, esse conceito é uma estratégia chave na engenharia de requisitos e chamasse fracionamento ou separação por interesse.

2.5.2.3.2 Princípios da modelagem de projetos

Para Pressman e Maxim (2016), a modelagem e os projetos assemelha-se a plantas de uma casa feita por um arquiteto, começando pela representação do todo e avançando com o tempo sobre os detalhes, estabelecendo um roteiro para sua construção. Existem algumas formas de se identificar os elementos de que irão compor um projeto, podem ser métodos: voltado a dados, voltado a padrões ou voltado a objetos. Independente do caminho preterido, recomenda-se seguir alguns princípios.

1. O projeto deve ser alinhado para o modelo do requisito: os requisitos informam a área de informação do problema, funções visíveis ao usuário e um conjunto de classes que empacotam os objetos com os métodos utilizados. Neste caso, a função do modelo de projeto será traduzir essas informações em uma arquitetura, um conjunto de subsistemas que implementam as funções e um conjunto de componentes que são a concretização das classes de requisitos elaboradas.

2. Sempre considere a arquitetura do sistema: a arquitetura de *software* é a espinha dorsal do sistema, ela afeta todas as outras camadas. Só após definição da arquitetura deve ser considerado os componentes do sistema.

3. As interfaces devem ser projetadas com cuidado: a forma como os dados fluem entre os componentes do sistema impacta diretamente a eficiência do processamento, uma interface bem elaborada facilita a integração e auxilia os testes.

4. O projeto do usuário deve ser voltado as necessidades do usuário, mas ele deve sempre enfatizar a facilidade de uso: a interface é a manifestação visível do *software*, logo se for falha, não importa a eficiência interna do sistema, ele será considerado um *software* ruim.

5. Representações de projetos devem ser de fácil compreensão: a finalidade dos projetos é transmitir informações a toda a equipe de desenvolvimento, logo se o projeto for de difícil compreensão não servirá como meio de comunicação efetivo.

6. O projeto deve ser desenvolvido iterativamente: as primeiras iterações são realizadas para refinamento e correção de erros, no final deve ser realizada para tornar o projeto tão simples quanto possível.

7. A criação de um modelo de projeto não exclui a abordagem ágil: uma coisa não anula a outra.

Pressman e Maxim (2016) destaca que embora a documentação do código seja útil, é difícil manter o código e suas descrições consistentes. Nesse sentido, o modelo de projeto será vantajoso, pois é criado em um nível de abstração isento de detalhes técnicos desnecessários e é fortemente acoplado aos conceitos e requisitos do sistema em desenvolvimento.

2.6 Engenharia de Requisitos

Considerada uma das tarefas mais difíceis enfrentadas por um engenheiro de *software*, entender os requisitos de um sistema é fundamental, pois para que o projeto de desenvolvimento de *software* seja considerado um sucesso, uma das premissas é que o produto gerado atenda o que o cliente deseja (GONÇALVES; CORTÉS, 2015).

A engenharia de requisitos começa pela concepção do problema, passa pelo levantamento de requisitos e finaliza a primeira etapa na elaboração, onde os requisitos básicos são definidos. Porém, segundo Pressman e Maxim (2016), a engenharia de requisitos constrói uma ponte entre o projeto e a construção abrangendo um conjunto de fases distintas, sendo elas:

1. Concepção: na etapa da concepção do projeto estabelece-se um entendimento básico do problema, as pessoas que querem a solução, a natureza da solução e a eficacia da comunicação preliminar.

2. Levantamento: é categórico que uma etapa importante do levantamento é estabelecer metas e após estabelecimento das metas deve ser estabelecido uma ordem de prioridade.

3. Negociação: usuários podem pedir mais do que o possível, além de poder haver conflitos de especificações entres diferentes clientes do mesmo projeto. Desta forma, é necessário conciliar esses conflitos de modo que cada parte atinja certo nível de satisfação.

4. Especificação: o termo pode assumir diversos significados, desde um documento por escrito, um conjunto de modelos gráficos, um conjunto de cenários de uso ou um protótipo.

5. Validação: os artefatos produzidos pela engenharia de requisitos terão sua qualidade avaliada na etapa de validação.

6. Gestão de requisitos: os requisitos mudam ao longo da vida de um projeto. Nesse sentido, a gestão de requisitos é um conjunto de atividades que ajuda a identificar, controlar e acompanhar as necessidades e suas mudanças a medida que o projeto avança.

Das fases descritas, algumas serão aprofundadas teoricamente, outras apenas serão apenas aplicadas no Capítulo 4.

2.6.1 Levantamento de Requisitos

Para Pressman e Maxim (2016), o levantamento de requisitos combina elementos de solução de problemas, de elaboração, de negociação e de especificação. Existem algumas técnicas que podem ser aplicadas nessa etapa do projeto como: entrevista, observação, demonstração de tarefas, estudo de documentos, substituição de usuários, questionários, *brainstorming*, prototipação e *workshops* ou oficinas de requisitos.

Após aplicação de uma ou mais técnicas e a elicitação dos requisitos ser dada como encerrada, surge uma nova necessidade: escrever aquilo que foi identificado para fique documentado e para que possa ser validado pelo cliente (CORTéS, 2013).

Inicialmente tanto os Requisitos Funcionais (FR) quanto os Requisitos Não Funcionais (NFR) são escritos em um mesmo documento e depois de classificados são divididos. Sendo que os requisitos funcionais evoluem para especificações de caso de uso e os requisitos não funcionais são agrupados em outro documento. Assim sendo, a especificação de requisitos é utilizada com tal finalidade e será aplicada no capitulo de desenvolvimento deste trabalho.

2.6.2 Especificação de Requisitos

Os requisitos de um sistema podem ser classificados em FR e NFR. Os requisitos funcionais descrevem o que o sistema deve fazer e dependem do tipo do *software*, dos usuários e da abordagem geral considerada. Os NFR são os que não estão diretamente relacionados às funções especificas fornecidas pelo sistema, estando relacionados a propriedades emergentes, como confiabilidade por exemplo (GONÇALVES; CORTÉS, 2015).

2.6.2.1 Requisitos Funcionais

Sommerville (2011) destaca que a especificação dos FR de um sistema deve ser completa e consistente, salientando que completude significa que todos os serviços requeridos pelo usuário devem ser definidos e que consistência significa que os requisitos não devem ter definições contraditórias.

Na prática, para sistemas grandes e complexos alcançar completude e consistência dos requisitos é extremamente difícil, quase impossível. Uma razão para isso é que ao elaborar especificações para sistemas complexos é fácil cometer erros e omissões. Outra razão é que em um sistema de grande porte existem muitos *stakeholders* (SOMMERVILLE, 2011).

Um *stakeholder* é uma pessoa ou papel que, de alguma maneira, é afetado pelo sistema e que têm necessidades diferentes e ou inconsistentes. Essas inconsistências podem não ser evidentes em um primeiro momento, quando os requisitos são especificados, e, assim, são incluídas na especificação. Os problemas podem surgir após uma análise mais profunda ou depois de o sistema ter sido entregue ao cliente (SOMMERVILLE, 2011).

2.6.2.2 Requisitos não Funcionais

Características como desempenho, proteção, disponibilidade e compatibilidade são exemplos de requisitos não funcionais e normalmente especificam ou restringem as características do sistema como um todo.

Requisitos não funcionais são frequentemente mais críticos que requisitos funcionais individuais. Os usuários do sistema podem encontrar maneiras de contornar uma função do sistema que realmente não atenda a suas necessidades. No entanto, deixar de atender a um requisito não funcional pode significar a inutilização de todo o sistema (SOMMERVILLE, 2011).

2.7 Modelagem dos Requisitos

Como visto, a ES trás um conjunto de conceitos, de princípios, de métodos e de ferramentas que devem ser consideradas no decorrer de um projeto de *software*. Nesse sentido, a modelagem não só permite que *software* seja projetado com os requisitos necessários para seu perfeito funcionamento, como também que se use uma arquitetura mais adequada para o projeto (LOBO, 2009).

2.7.1 Construção do modelo de análise

Define-se modelo de análise como a descrição das informações, funcionalidades e comportamentos necessários para um sistema de computador. O modelo é dinâmico durante a confecção do projeto e representa basicamente a reprodução dos requisitos em determinado momento do projeto.

Existem algumas formas na ES para selecionar os modelos de análise e diferentes abordagens trazem diferentes pontos de vistas, diferentes elementos genéricos que compõem esses modelos. O elementos genéricos mais comuns usados na literatura são (PRESSMAN; MAXIM, 2016): os elementos baseados em cenários e os elementos baseados em classes.

No primeiro caso, destaca-se que quando a modelagem é feita com aplicação desse método o sistema é descrito sob o ponto de vista do usuário usando uma abordagem baseada em cenários, neste caso os elementos do modelo de requisitos baseado em cenários

são a primeira parte do modelo a ser desenvolvido e servem como direcionamento para criação de outros elementos de modelagem.

No segundo caso, a aplicação desse método baseia-se na ideia que cada cenário de uso implica num conjunto de objetos manipulados à medida que o ator interage com o sistema, sendo que esses objetos são categorizados em classes, entidade que será descrita na seção sobre programação orientada a objetos.

2.7.1.1 Modelagem de requisitos: métodos baseados em cenários

A modelagem baseada em cenários produzirá uma representação textual denominada caso de uso descrevendo uma iteração específica de forma informal, por meio de uma simples narrativa, ou de forma mais estruturada e formal, sendo possível o uso da UML para visualização mais procedural da interação.

Um das primeiras etapas para o desenvolvimento dos casos de uso é enumerar as funções ou atividades realizadas por um ator especifico, ou seja, baseado no levantamento de requisitos (PRESSMAN; MAXIM, 2016).

2.7.1.2 Cenários de Uso

Após a primeira etapa de reunião de requisitos, pode-se ter uma visão geral das funções e características da aplicação, ou seja, inicia-se um processo rudimentar da materialização do *software*. Contudo, afirma Pressman e Maxim (2016), é difícil iniciar as atividades mais técnicas de engenharia de *software* até que se entenda como tais funcionalidades e características de fato serão utilizadas pelo diferentes usuários.

É neste contexto que desenvolvedores e usuários criam um conjunto de cenários que identificam um roteiro de uso para o sistema a ser construído. Esses cenários são chamados na literatura de cenários de uso, ou casos de uso, e segundo Sommerville (2011), a modelagem de caso de uso foi originalmente desenvolvida na década de 1990, e foi incorporada ao primeiro *release* da UML em 1999.

Cortés (2013) elucida que documentos de caso de uso são formas de especificar requisitos de maneira mais completa, detalhada e mais aproximada à implementação do sistema. Sommerville (2011) complementa dizendo que cenários de uso já se tornaram uma característica fundamental da UML.

O primeiro passo ao escrever um caso de uso é definir os “atores” envolvidos na história. Atores são as diferentes pessoas ou dispositivos que usarão o *software* enquanto ele opera. Destaca-se que o levantamento de requisitos é uma atividade evolutiva, logo nem todos os atores necessariamente serão identificado na primeira iteração, normalmente são identificados inicialmente os atores primários. Define-se atores primários como aqueles que interagem para atingir a função necessária e obter o beneficio desejado do sistema, já

os secundários dão suporte ao sistema para que os primários possam realizar seu trabalho (PRESSMAN; MAXIM, 2016).

Após a primeira interação da reunião de requisitos, geralmente, tem-se as informações necessárias para o início da confecção dos casos de uso. Deve-se enumerar as funções ou atividades realizadas, que são obtidas pela lista de requisitos, por um ator específico e preliminarmente escrever de forma narrativa e informal os cenários e depois, caso necessário, reescreve-lo num formato mais estruturado na forma de quadro (PRESSMAN; MAXIM, 2016).

2.7.2 Modelagem de requisitos: métodos baseados em classes

Pressman e Maxim (2016) destaca que o método baseado em classe para modelagem de requisitos utiliza conceitos comuns para produzir uma representação de uma aplicação que possa ser entendida por envolvidos sem conhecimento técnico. Os elementos de um modelo baseado em classes são: classes e objetos, atributos, operações (métodos), modelos CRC, diagrama de colaboração e pacotes.

Uma ideia para começar a identificar as classes é examinar os cenários de uso citados anteriormente. Neste caso, a modelagem vai representar os objetos que o sistema vai manipular, as operações que serão aplicadas aos objeto, as relações entre os objetos e as colaborações que ocorrem entre as classes definidas.

2.7.2.1 Identificação de classes de análise

Como dito, a primeira etapa para identificar classes é examinar os cenários de uso desenvolvidos. As classes são determinadas sublinhando-se cada substantivo ou frase nominal, gerando a partir disso uma tabela simples. É necessária entender como as classes de análise podem ser categorizadas e para isso as opções serão elencadas a seguir (PRESSMAN; MAXIM, 2016):

1. Entidades externas: que produzem ou consomem informações a ser usadas por um sistema baseado em computadores.
2. Coisas: que fazem parte do domínio de infamações para o problema.
3. Ocorrência ou eventos: que ocorrem no contexto de operação do sistema.
4. Papéis: desempenhados pelas pessoas que interagem com o sistema.
5. Unidades Organizacionais: relevantes para uma aplicação.
6. Locais: que estabelecem o contexto do problema e a função global do sistema.
7. Estruturas: que definem uma classe de objetos ou classes de objetos relacionados.

2.7.2.2 Especificação de atributos

Os atributos descrevem uma classe selecionada para inclusão no modelo de analise. Basicamente, definem uma classe e esclarecem o que ela representa. Nesse sentido, é necessário estudar cada caso de uso modelado anteriormente e selecionar as informações que irão compor a classe modelada (PRESSMAN; MAXIM, 2016).

2.7.2.3 Definição das operações

Operações definem o comportamento de um objeto. Geralmente as operações podem ser categorizadas da seguinte forma: operação que manipulam dados, operações que efetuam um cálculo, operação que pesquisam o estado de um objeto e operação que monitora um objeto quanto às ocorrências de um evento de controle (PRESSMAN; MAXIM, 2016).

2.7.2.4 Modelagem *class*-responsabilidade-colaborador

A Modelagem CRC é uma maneira simples de identificar e organizar as classes relevantes para os requisitos do sistema. Os requisitos pode ser esquematizado por um conjunto de fichas padrões que representam as classes. Essas fichas são divididas em três seções: na parte superior escreve-se o nome da classe, no corpo do cartão, no lado esquerdo, escreve-se as responsabilidades e no lado direito os colaboradores. Define-se responsabilidade como atributos e operações relevantes para a classe, ou seja, é qualquer atividade ou ação realizada pela classe. Define-se colaboradores como as classes necessárias para fornecer a uma classe as informações necessárias para completar uma responsabilidade (PRESSMAN; MAXIM, 2016).

De maneira geral, a modelagem baseada em classes usa as informações extraídas do caso de uso, um conjunto de cartões CRC são elaborados e usa-se a a ferramenta UML para diagramar relações existentes das classes modeladas, definindo hierarquias, relações, associações agregações e dependência entre as classes.

2.8 Conclusões Parciais

Desenvolver um *software* vai muito além do processo de codificação, existe uma ciência por trás do processo, a engenharia de *software*. Entender os conceitos envolvidos em todo o processo colabora para a execução da atividade com êxito. A escolha do modelo a ser seguido, a aplicação de metodologias ágeis junto as tradicionais e os sistemas de testes se mostram fundamentais para produção de um *software* funcional.

Sendo assim, a aplicação dos princípios apresentados, desde a comunicação ao planejamento fomentam o caminho a ser seguido para obtenção dos requisitos necessários

para construção de uma ferramenta de alta aplicabilidade. Mais do que isso, pensar a modelagem de um *software* e desenvolvê-lo num ambiente livre e ágil corrobora para que o resultado final do projeto conclua-se conforme planejado.

A elaboração de uma fundamentação teórica robusta corroborou para o desenvolvimento de um *software* sólido, ancorados em conceitos consolidados na área do conhecimento.

3 Revisão bibliográfica

O desenvolvimento de *softwares* não proprietários é um tema de relevância no mundo organizacional e social. Os pesquisadores Silva, Aparicio e Costa (2019) se propuseram realizar um levantamento bibliométrico sobre *software* livre e *software opensource* salientando as diferenças entre os dois conceitos, descrevendo suas principais vantagens e desvantagens e caracterizando suas principais áreas de estudos e aplicações. Os resultados apresentados revelam que o benefício do *software livre* influencia positivamente nas organizações e países com economias em desenvolvimento e afirmam que a filosofia de *software* livre atende as demandas tecnológicas com igual ou superior qualidade em relação aos *softwares* proprietários. Baseado na observação de um crescimento exponencial da literatura durante o processo de revisão bibliométrica o estudo destaca que a disseminação da cultura livre permeia por diversos segmentos.

A diferença entre *software livre* e *software opensource* também é destacada por Bauer e Pizka (2003), que realizaram um estudo com o objetivo de discorrer sobre a contribuição do *software livre* para a evolução do *software* num contexto mais geral. Uma das contribuições do trabalho é desmistificar a ideia que os desenvolvimentos de *softwares* livres são realizados de maneira caótica e sem seguir procedimentos estruturados. Os autores salientam que embora no início de um projeto a organização não seja uma prioridade, a medida que o projeto avança e alcança um nível crítico é necessário uma gerência das complexidades. Os pesquisadores avançam sobre o tema da redução de complexidade por meio da divisão do código e tendência da rescrição do código para sua manutenibilidade.

Em seu trabalho, Fitzgerald (2004) afirma que apesar dos anos de pesquisa e prática no setor de desenvolvimento de *software* proprietários, os principais métodos utilizados estavam longe de ser baseados em processos exatos e previsíveis. Este fato, direta ou indiretamente, tem como consequência projetos que excedem seus orçamentos, descumprem seus cronogramas e não atendem as expectativas dos clientes. Foi nesse contexto que surgiu a conhecida crise de *software* e segundo o autor é para contrapor essa realidade que surgi o FOSS como uma iniciativa de desenvolvimento ágil que aborda sistematicamente trés aspecto: custo, tempo de desenvolvimento e qualidade de entrega. Um dos pontos trazidos pelo autor é que embora seja entediante a documentação de *softwares* para FOSS esse procedimento é vital, sobretudo para sistemas complexos.

Embora discorra elogios sobre essa forma de desenvolvimento, Fitzgerald (2004) se propõe ter um olhar crítico sobre o *opensource* discorrendo também sobre os contratempos envolvendo a comunidade. Ele traz como exemplo, o *SourceXchange* que é um serviço de corretagem para empresas que solicitam desenvolvedores de FOSS e teve suas atividades interrompidas em 2001. Além de afirmar que os estudos de *Freseshmeat.net*

e *SourceForge.net*, dois sites populares de desenvolvimento de FOSS, revelaram que a maioria dos projetos possui apenas um ou dois desenvolvedores.

Contudo, apesar dos apontamentos preocupantes em torno da comunidade, o autor reitera o futuro brilhante do FOSS. Fitzgerald (2004) finaliza seu estudo com uma afirmação curiosa alegando que embora o FOSS pareça evitar princípios básicos da engenharia de *software*, ou seja, nenhuma especificação formal de requisitos ou processo de projeto, nenhuma avaliação de risco ou objetivos mensuráveis, dentre outros, os princípios bem estabelecidos de engenharia de *software* estão no coração do FOSS.

O problema da dificuldade de manutenção em *softwares* ditos tradicionais já havia sido identificado pelos autores Zhu e Lubkeman (1997), que afirmam em seu trabalho que um dos principais obstáculos na atualização e manutenção da maioria dos sistemas existentes decorrem direta ou indiretamente da metodologia de desenvolvimento de *software* orientada a função. Os autores também apresentam em seu trabalho a alternativa de desenvolvimento orientado a objetos, ou seja, aplicação do POO como uma nova maneira de pensar problemas baseados em conceitos do mundo real, afirmando que esta metodologia provou ser uma ferramenta eficaz para atingir esse objetivo.

Fitzgerald (2011) em seu artigo discorre justamente sobre a relação do *software* livre com a engenharia de *software*. O autor destaca que essa relação pode apresentar visões distintas: os adeptos afirmando que o desenvolvimento de um FOSS é de alta qualidade, realizado rapidamente por uma mão de obra muito talentosa e com baixíssimo custo e por outro lado, os criticos dizendo que o *software* possui qualidade incerta, com pouca ou nenhuma documentação e imprevisibilidade jurídica, nesse sentido, o autor com sua pesquisa propõe uma visão mais equilibrada entre os dois extremos. Em relação a ES o pesquisador diz ser evidente que o princípios da ES convencional estão no centro de projetos de desenvolvimento FOSS bem sucedidos, ele afirma que a abordagem modular se aplica à estrutura do projeto e à base de código, ou seja, projetos grandes de FOSS tendem a ser agregações de projetos menores. Fitzgerald (2011) destaca aspectos inquestionáveis que o desenvolvimento FOSS contribui para o tradicional como: inovação aberta, desenvolvimento global de *software*, fonte interna e gerenciamento de liberação baseado em tempo.

A imprevisibilidade jurídica, citada por Fitzgerald (2011), é um tema complexo dentro do contexto geral dos *softwares* não proprietários, sobretudo no que diz respeito a sua licença. Com objetivo que contribuir com essa área os autores Gaff e Ploussios (2012) discutem questões legais básicas relacionadas a pessoas ou empresas que lidam diretamente com o desenvolvimento de *software open sources*.

Os pesquisadores ressaltam as principais vantagens desse tipo de *software*, dentre eles a sua fácil disponibilidade sem a presença de custos vinculados a *royalties* ou taxas de licença. Um dos principais pontos destacados pelos autores é o erro de confundir que um

software open source não possui dono, eles afirmam que os desenvolvedores ou concessionários possuem direitos autorais. O que ocorre segundo os autores é que os proprietários podem disponibilizar seu *software* sob os termos de uma licença específica.

Os autores destacam ainda que para que se copie, modifique, crie-se um produto derivado ou o distribua é necessário possuir os direitos autorais ou ter uma licença do proprietário. No artigo os estudiosos discorrem sobre os tipos de licenças existentes, categorizando-as em três tipos: *restritive licences*, *moderately restrictive licenses* e *permissive licenses* e encerram seu estudo recomendando o uso de *softwares open sources* para empresas com a justificativa que essa adoção permite que elas se concentrem no desenvolvimento interno das funcionalidades, salientando apenas o cuidado na leitura do tipo de licença aplicável.

Um movimento interessante é o direcionamento cada vez maior para a utilização de *softwares* livres para fins educacionais e pesquisas como a realizada por Terbuc (2006) buscam demonstrar a importância da utilização do FOSS nesse contexto. O autor discorre sobre aplicação da filosofia livre e os incentivos governamentais recebidos para aplicação de FOSS em ambientes de ensino e mostram uma parceria realizada entre instituições de ensino na Eslovênia por meio do projeto OKO (Introdução de *software opensource* e o *software* livre nas instituições educacionais). O autor destaca ainda que o FOSS tem o potencial de ajudar a reduzir a barreira de custos, reduzindo o custo de *software*, que é um componente importante das instalações das instituições de ensino. E ainda, afirma que existem inúmeras outras vantagens no uso do *software* livre na educação, incluindo benefícios pedagógicos.

Shatunov *et al.* (2020) começam seu trabalho refletindo a tarefa das universidades modernas em ensinar o aluno a se desenvolver e ser um especialista em seu campo. Para os autores, estudantes na área de engenharia elétrica e de automação precisam realizar simulações para consolidar seus processos de aprendizado, contudo quase todos os *softwares* comerciais são distribuídos com uma licença proprietária que concede direitos limitados ao uso de uma ou mais cópias do programa, o que torna a introdução generalizada da modelagem computacional no processo educacional mais lenta devido ao alto custo. Com o objetivo de suprir essa demanda, os pesquisadores apresentam em seu trabalho uma experiência de utilizar o *Qucs*, um *software* gratuito, na disciplina fundamentos teóricos da engenharia elétrica. Os autores apontam as vantagens e as desvantagens do *Qucs* em relação ao *softwares* proprietários de mesma natureza, salientam que do ponto de vista educacional as desvantagens são insignificantes. O estudo conclui que os desempenhos comparativos entre os *softwares* livres e comerciais são proporcionais, contudo existe um fator que coloca os *softwares* livres na frente, sua flexibilidade e capacidade de acumular a experiência de dezenas ou milhares de entusiastas envolvidos em atividades práticas. Desta forma, segundo os pesquisadores é muito importante chamar a atenção para as

possibilidades de software de código aberto, especialmente no campo da educação em engenharia.

Conforme já destacado por Terbuc (2006), o autor Barry (2009) reafirma que existe um crescimento da adoção de *softwares open sources* por instituições de ensino mundialmente. De acordo com o autor, um dos principais motivos é a tendência de *softwares* de padrões abertos permitirem a interoperabilidade com outros produtos de *software*, além do fato dos *softwares* serem gratuitos, ou com taxas de licenças de custo baixo e fixo, há também o estímulo positivo a educadores e estudantes ao não consumo de cópias ilegais de *softwares* proprietários. O estudo proposto busca relatar a experiência do uso desses *softwares* no Sudão, país em desenvolvimento, de maneira a superar as sanções econômicas impostas. O trabalho elucida as questões educacionais da região, mostrando como a adoção dessas praticas seriam positivas, discutem as vantagens e desvantagens em torno das tecnologias chamando atenção ao fato que *softwares* de código aberto são particularmente adequado para a educação, todavia salientando que *software* livre e o *software* proprietário podem coexistir em instituições que já possuam a cultura de uso em *softwares* proprietários e acham difícil se livrar desse legado.

Nessa mesma linha, Lazić, Zorica e Klindžić (2011) realizaram um estudo sobre *softwares open sources* na educação da Croácia. Os autores afirmam que, no contexto da Croácia, os FOSS devem ser introduzido no currículo dos cursos no inicio do nível universitário. Apontam como consequências positivas a absorção dos preceitos da filosofia do software livre por parte da comunidade acadêmica, além de permitir que os professores tenham consciência das alternativas para *softwares* não comercias para produção de conhecimento.

Nota-se pelo exposto a importância e os desafios em torno do tema, bem como a grande possibilidade e viabilidade da implantação dessa filosofia em ambientes educacionais. Permitindo que estudantes e professores possam além de mais acesso a aplicações que auxiliarão no aprendizado desenvolvam seus próprios softwares livres contribuindo com o setor de maneira geral.

Kober, Manzoni e Lemos (2003) abordam em seu trabalho o fato que a área de sistemas de energia elétrica precisa de um conjunto de softwares para apoiar as atividades de planejamento e operação. Destacam ainda como a POO tem sido utilizada para o desenvolvimento de *softwares* e aplicações em áreas como *design* de GUI, simulação em sistemas de potência, fluxo de potência e desenvolvimento de ferramentas educacionais. Os autores realizam um trabalho de exploração do conceito de POO e aplicam no desenvolvimento de uma interface gráfica para ser útil no setor de distribuição.

Segundo Agostini, Decker e Silva (2002), antigamente os *softwares* tradicionais do setor de energia elétrica de grande porte em sua maioria eram baseados em métodos procedurais. Esses métodos faziam com que fosse mais difícil a integração de novos mó-

dulos desenvolvidos por diferentes equipes, elevando os investimentos em manutenção e atualização dos sistemas. Buscando solucionar esses problemas os autores desenvolveram em seu trabalho uma base computacional para aplicação em sistemas de energia elétrica utilizando-se da técnica de MOO, que segundo os autores é uma técnica de projeto de *software* na qual se está interessado na clareza e organização do projeto, tendo por base uma representação clara e eficiente do mundo real que facilite ao máximo o desenvolvimento e a manutenção do *software*.

A POO é utilizada de maneira sistemática no desenvolvimento de *software* no setor de energia, sobretudo na construção de interfaces gráficas conforme afirmam Foley e Bose (1995) em seu trabalho. Os autores corroboram com o que foi dito anteriormente, ou seja, a principal vantagem do *software* baseado em POO é a facilidade de manutenção e atualização, que historicamente eram um grande problema encontrado pelos programadores, pois consumia tempo de projeto e propiciava erros. Os estudiosos destacam que o uso da POO tornou-se bastante comum, especialmente para GUI e gerenciamento de banco de dados, por isso utilizaram dessas premissas para comparar em seu estudo aplicação de algorítimos de solução de fluxo de potência em topologias de redes elétricas diferentes comparando seus desempenhos.

A empregabilidade da POO e UML é uma realidade no desenvolvimento de *softwares* atualmente. No estudo de caso trazido por Bahri, Pasternak e Bini (2016) é possível observar a aplicação das técnicas na criação de um Sistema de Gerenciamento para Indústria Gráfica, onde os autores utilizam-se da linguagem de programação JAVA para solucionar demandas comerciais reais, resolvendo problemas de gerenciamento duma empresa do setor gráfico, melhorando a rotina de trabalho e otimizando resultados.

Nesta mesma linha, os autores Castro, Cruz e Oddone (2013) descrevem a POO e a UML detalhadamente e utilizam desses conceitos para desenvolverem um sistema para bibliotecas. A partir de um estudo de caso, os pesquisadores constatam que nos conceitos trazidos pela POO e UML possuem influência, ou proximidades lógicas, com as teorias e os conceitos dos sistemas de classificação bibliográficas e concluem o estudo afirmando que para entender o complexo, antes de tudo, é preciso dividi-lo em partes e classificá-lo.

Diante o exposto, nota-se a grande necessidade de desenvolvimento de novos *softwares* que sejam flexíveis e possuam as características de construção em sintonia com o desenvolvimento de *software* contemporâneo, em outros termos, aplicando os conceito trazidos pelo POO. O encapsulamento, polimorfismo, herança e associação de classes são fundamentais para construção de uma GUI que preze pela modularidade e os autores Foley *et al.* (1993) em seu trabalho destacam a utilização desses fundamentos no desenvolvimento de sua pesquisa.

Diversas linguagens de programação tem suporte para POO como por exemplo a *Haskell*, *Java*, *C++*, *Python*, *PHP*, *Ruby* e *Pascal*. Especificamente, a linguagem de

programação *Python* é bastante difundida nas produções acadêmicas, colaborando para analises práticas e consistentes de problemas reais. Os pesquisadores Bezerra, Silva e Lima (2016) utilizam *Python* para implementar um *software* capaz de realizar o cálculo de vigas do tipo *Euller-Bernoulli* com cargas pontuais, destacando as inovações na área da engenharia civil com a aplicação da indústria de softwares, espaço onde computação gráfica, sistemas de informação e métodos matemáticos e físicos aprimoram processos construtivos.

O trabalho realizado por Milano (2013) mostra a importância da linguagem de programação *Python* no desenvolvimento de ferramentas para análises de sistemas elétricos de potência. Os autores desenvolveram um *software* chamado DOME que deve ser usado para ambientes de análise e fins didáticos acadêmicos, eles destacam a aplicação da modularidade e reutilização de código que a linguagem propícia fazendo com que haja potencialidade de crescimento indefinido da ferramenta sem perder qualidade de desempenho. Outro aspecto importante é a possibilidade de utilizar o DOME em diferentes níveis: graduação, mestrado e doutorado o que torna o *software* ainda mais poderoso.

Kulasza e Annakkage (2013) desenvolveram uma ferramenta de *software* para aplicação em sistema de energia, modelagem e simulação. Os autores destacam que um dos principais objetivos do projeto era produzir uma estrutura de *software* que reduzisse a repetição de código, incentivasse o desenvolvimento rápido e garantisse um resultado de código limpo e de fácil leitura. Dessa forma, a linguagem de programação escolhida foi a *Python* devido as grandes possibilidades das bibliotecas e sua natureza modular. Os resultados trazidos são promissores e os trabalhos permanecem avançando no desenvolvimento por meio de um grupo de pós graduação.

Uma proposta que se assemelha à desenvolvida neste trabalho é a feita por Ivanovich e Timofeyevich (2018), onde os autores entusiastas do *software* livre, desenvolvem uma GUI que facilita a interação do usuário com uma ferramenta já existente e aplicada na indústria, a OpenFOAMS. O diferencial é que a ferramenta original se apresenta com difícil uso devido a necessidade do usuário conhecer a fundo sua dinâmica para utiliza-la, visto que funciona por linha de código. Então, os autores projetaram uma interface para ser utilizada como um programa para *desktop*, utilizando como base a linguagem de programação *Python* e o *PyQt5*, disponibilizando seus códigos por meio de repositório repositório *GitHub*.

O trabalho de Kuhmann *et al.* (2018) realizou uma pesquisa internacional à cerca da utilização da abordagem de desenvolvimento híbrido de *software*. Através da aplicação de um questionário constataram que é recorrente a utilização de processos tradicionais de desenvolvimento, complementados por normas, padrões e regras para que se mantenha a qualidade do produto. Porém os autores destacam que essa escolha diminui a flexibilidade a um nível baixo de planeabilidade, previsibilidade e conformidade se comparadas

aos métodos ágeis. Como alternativa, a pesquisa mostrou que as abordagens de desenvolvimento híbrido representam uma solução, independentemente do tipo de empresa e do setor industrial. O Desenvolvimento híbrido permite que as empresas se beneficiem fornecendo aos clientes e à gerência um ambiente seguro e aos desenvolvedores a flexibilidade exigida. Os autores destacam que abordagens de desenvolvimento híbrido surgiram gradualmente: 83,9% dos participantes afirmaram que a abordagem de desenvolvimento emergiu da experiência, embora 52,2% dos participantes tenham declarado que tendencialmente utilizavam uma abordagem padrão. Finalizam o estudo informando que mais de um quarto dos entrevistados afirmaram que as abordagens de desenvolvimento foram selecionadas de forma mais individual e adaptadas durante os projetos em resposta a situações específicas.

Os autores Lanes e Rossoni (2018) desenvolveram uma ferramenta computacional para modelagem de alimentadores reais de distribuição no formato aceito pelo *software OpenDSS*. São utilizados os *softwares Microsoft Excel* e *Matlab* para realização das conversões necessárias para transformação de dados reais das distribuidoras em códigos de sintaxe *OpenDSS*. Os pesquisadores destacam que uma das vantagens de utilizar programas externos para modelagem da rede é a facilidade de encontrar erros e inconsistências nas enormes linhas de código geralmente utilizadas nos estudos recorrentes na área.

Os autores Wang *et al.* (2019) desenvolveram um sistema para simulação de transmissão de imagem sem fio em veículo de inspeção inteligente utilizando como *framework* para aplicação *desktop* o PyQt5. Neste estudo os pesquisadores usaram o arduino como ferramenta principal de gerenciamento e destacam que ligação Qt/Python tem muitas vantagens, como alta eficiência de desenvolvimento, forte estabilidade de programa e suporte entre plataformas.

Os autores Ghosh, Saha e Mou (2010) desenvolveram um estudo comparativo interessante entre dois *softwares* da área de sistemas elétricos de potência: *PSAT*, um *software livre* e *opensource*, e o *PowerWorld* que é um *software* comercial. Os autores fizeram uma descrição de cada uma das ferramentas e concentraram-se na realização de um estudo de caso aplicado a uma rede elétrica e a partir de então avaliaram os resultados e a velocidade de processamento. Segundo os autores os resultados mostraram uma diferença média de 1,7525% nos dados de magnitude da tensão do barramento, que eles consideraram como uma diferença insignificante, além disso verificaram que a velocidade de simulação do *PowerWorld* foi 6,4 vezes mais rápida que a do *PSAT*.

Silva e Pacheco (2018) realizaram em seu trabalho o desenvolvimento de uma GUI *open source* de *design* intuitivo para o usuário com o objetivo de contribuir para aumentar o número de sistemas fotovoltaicos instalados em residências, principalmente nos países em desenvolvimento. A GUI foi desenvolvida por meio do *software MatLab*, numa funcionalidade que o programa comercial permite para criação de interfaces que podem

ser utilizadas e distribuídas gratuitamente e sem a necessidade do *MatLab* instalado nos computadores que que a GUI for utilizada. Os autores salientam que o projeto está em fase inicial, porém julgam que pelo fato de ser desenvolvido como *open source* membros da comunidade de pesquisas e universidades podem colaborar para seu aprimoramento.

Kulasza e Annakkage (2013) relatam em seu artigo a dificuldade dos grupos de pesquisa no processo de construção de algoritmos específicos para o objeto de analise na área de sistemas de energia elétrica. Apontam que além do tempo gasto no desenvolvimento desses *scripts* existem uma perda em relação a integração dos membros e compartilhamento de conhecimentos. Visando suprir essas lacunas os pesquisadores apresentam uma GUI, associada a um projeto chamado PSOA (*Power System Oscillations Analyzer*), com o objetivo de ser uma ferramenta educacional *open source* que foca na implementação de uma estrutura de desenvolvimento de modelos personalizados, que sirva como base apara ensino e pesquisa em modelagem e analise de sistemas de potência.

Kober, Manzoni e Lemos (2003) abordam em seu trabalho o fato que a área de sistemas de energia elétrica precisa de um conjunto de softwares para apoiar as atividades de planejamento e operação. Destacam ainda como a POO tem sido utilizada para o desenvolvimento de *softwares* e aplicações em are as como *design* de GUI, simulação em sistemas de potência, fluxo de potência e desenvolvimento de ferramentas educacionais. Os autores realizam um trabalho de exploração do conceito de POO e aplicam no desenvolvimento de uma interface gráfica para ser útil no setor de distribuição.

Destaca-se de maneira positiva a utilização da linguagem de programação *Python* associada ao conceito de POO para construção de *softwares* promissores. Além disso, como visto na fundamentação teórica a aplicação de técnicas de ES é imprescindível para o exito desse processo de desenvolvimento. Contudo, é interessante notar como a ES pode ser aplicada por profissionais de diferentes áreas e um exemplo prático é o estudo trazido por Silva, Évora e Cintra (2015) onde os pesquisadores desenvolvem um sistema computacional para aplicação em diagnósticos e intervenções de enfermagem. Os autores estruturam sua metodologia de desenvolvimento em três ciclos. A etapa inicial do planejamento é estruturada por meio da comunicação com o cliente, em seguida, é feita a análise do risco, para iniciar a modelagem /construção e finalizam com a verificação.

3.1 Conclusões Parciais

Diante o exposto, é nítida a relevância do desenvolvimento e utilização de *softwares* livres e de código aberto para a construção de um conhecimento sem fronteiras. O *software* livre mostra-se viável pelo seu baixo custo, e pela diversidade de aplicabilidade, sendo promissor tanto no ambiente acadêmico quanto corporativo. A utilização da linguagem de programação *Python*, mostrou-se útil para o desenvolvimento de *software* dessa natureza,

pela comunidade envolvida, pela possibilidade da aplicação da programação orientada a objetos e utilização de *frameworks* consolidados e potentes como é o caso do *PyQt5*. Além disso, nota-se que a união de metodologias tradicionais e ágeis, através de uma metodologia hibrida, apresenta-se como um caminho promissor para o desenvolvimento de *software*.

4 Desenvolvimento

Como visto na fundamentação teórica, o processo de desenvolvimento de um *software* apresenta diversas etapas. O objetivo desse capítulo é discorrer sobre como os conceitos que foram implementados na evolução do desenvolvimento do presente trabalho foram utilizados na obtenção dos resultados.

4.1 Comunicação

As reuniões preliminares funcionaram para absorção dos requisitos fundamentais do sistema, utilizando os princípios da comunicação para solidificar a ideia geral. As reuniões de acompanhamento funcionaram como reuniões de *feedback*, onde a cada evolução de desenvolvimento e entrega de resultados parciais, eram repensados e redefinidos os requisitos do sistema. Nesta fase do desenvolvimento os quatro princípios da comunicação foram essenciais, sobretudo o item 2 precisou ser mais vezes requisitado, visto que o contato pessoal trazia mais clareza as recomendações. Como os requisitos do sistema adaptava-se a cada iteração do processo, anotar sistematicamente as novas solicitações foi imprescindível devido a quantidade de informações trocadas.

Na medida em que solidificavam-se os requisitos do sistema as reuniões presenciais foram diminuindo sua essencialidade. Passou-se então a existir a comunicação por meio de ferramentas de comunicação online, prioritariamente para dúvidas rápidas. Um fator que vale destacar é que no meio do desenvolvimento do projeto houve uma determinação do Ministério da Saúde para que as pessoas fizessem um distanciamento social devido a uma pandemia provocada pelo COVID-19. Neste cenário, a alternativa para as reuniões de acompanhamento, testes e entregas de resultados parciais foi a realização dos encontros por meio da ferramenta online.

No cenário descrito acima a metodologia de comunicação mais utilizada baseou-se nos métodos trazidos pela *scrum*, onde as reuniões onlines eram curtas e quase diárias, este fato fez com que a produtividade do desenvolvimento crescesse bastante.

4.2 Planejamento

Para Pressman e Maxim (2016), o planejamento e o controle são utilizados por uma única e principal razão: administrar a complexidade. Desta forma, essa premissa foi utilizada nesse trabalho para otimizar os recursos de tempo de construção e garantir a modularidade do sistema.

Após a primeira etapa da comunicação, definiu-se o objetivo central do projeto: desenvolver um *software* livre capaz de converter informações retiradas da BDGD em um *script* que descreva um SD e realize um estudo de fluxo de potência. Desta forma, após consolidado o objetivo, um escopo do projeto foi documentado, conforme pode ser visto a seguir.

4.2.1 Escopo do Projeto

Com a definição do objetivo principal o escopo do projeto foi estabelecido. O planejamento de execução foi realizado a partir de um mapeamento do desenvolvimento necessário para entrega do produto final, neste caso o *software* Sistema de Planejamento Integrado (SIPLA). Os princípios do planejamento descritos na fundamentação teórica foram evocados para construção do escopo.

1. **Título do Projeto:** Projeto de desenvolvimento do *software* SIPLA.

2. **Publico Alvo:** Estudantes do curso de engenharia elétrica e possíveis empresas do setor.

3. **Justificativa do Projeto:** O contexto atual do setor elétrico, tanto no Brasil quanto em diversos países do mundo, tem evoluído em sua característica tradicional. Este fato conduz a criação de ambientes cada vez mais competitivos em busca da máxima eficiência dos elementos que compõem o SEP. Além disso, o avanço tecnológico contínuo impulsiona uma expansão progressiva do SEP e implica na necessidade de planejamento por parte das concessionárias, pois as mesmas devem garantir os níveis adequados de qualidade de serviço e do produto entregue aos consumidores finais. Uni-se a isso a existente demanda educacional e acadêmica de *softwares* não proprietários para área de engenharia elétrica, que possam ser usados como ferramentas de estudos para soluções de problemas reais, neste sentido o desenvolvimento do SIPLA tem a possibilidade de abarcar esses objetivos.

4. **Finalidade do projeto:** Desenvolver um *software* livre capaz de converter informações retiradas da BDGD em um *script* que descreva um SD e realize um estudo de fluxo de potência.

5. **Objetivos do projeto:**

    - Desenvolver um *Software Desktop* livre capaz de realizar estudos de fluxo de potência com base em informações contidas na BDGD.
    - Criar uma documentação do *software* desenvolvido.

- Disponibilizar uma plataforma, inicialmente para os alunos do DEEC/UFBA, para futuros desenvolvimentos de TFG, trabalhos de pesquisa, trabalhos de iniciação científica etc.
- Elaboração de um artigo científico para apresentar e dar visibilidade ao *software* para comunidade acadêmica de engenheiros eletricistas.

6. **Descrição do Produto:** O *software* SIPLA será uma *interface* do tipo *desktop*, desenvolvido em linguagem de programação *python* utilizando o *framework PyQt5*. O SIPLA será de caráter modular para facilitar a manutenção e atualização do mesmo. Será capaz de realizar um estudo de fluxo de potência em redes pertencentes a BDGD. Inicialmente pretende-se disponibiliza-lo para os computadores pertencentes aos laboratórios de Engenharia Elétrica da Universidade Federal da Bahia.

7. **Descrição do Produto:**

   - Publico alvo: Estudantes de Engenharia Elétrica.
   - Patrocinador: Grupo de Sistemas Elétricos de Potência Integrado (G-SEPi).
   - Gerente do Projeto: Sandy Aquino dos Santos.

8. **Entregas do Projeto:**

   - *Interface* gráfica do SIPLA.
   - Documentação do *software* SIPLA.
   - Artigo cientifico

9. **Estimativas de tempo e custo:**

   - Custo esperado: R$ 0,00
   - Prazo estimado: 12 meses

10. **Exclusões do Projeto:** Não faz parte do projeto

    - Nenhum especificação além dos definidos nos objetivos do projeto.

11. **Riscos:**

    - Não finalização do *software* no prazo estimado.
    - Não autorização do departamento de engenharia elétrica para implantação do SIPLA nos laboratórios.

Ressalta-se que o reconhecimento do *item 2* dos princípios do planejamento foi importantíssimo durante todo o projeto, visto que o andamento do projeto não engessou-se em seu escopo. O processo de planejamento se mostrou dinâmico e fluido sem prejuízos para o resultado final.

4.2.2 Cronograma do Projeto

O cronograma de um projeto de *software* não difere muito do cronograma de qualquer esforço de engenharia multitarefa, ressaltando sempre sua característica evolutiva a medida que o tempo passa, ou seja, conforme o tempo caminha, cada item é refinado em um cronograma detalhado (PRESSMAN; MAXIM, 2016). Para isso, algumas ferramentas de gerência de projetos foram utilizadas.

4.2.2.1 Gráfico de *Gantt*

O gráfico de *Gantt* é gerado como resultado de informações coletadas do conjunto de tarefas associadas ao projeto. Dados de esforço, duração e data de inicio/fim são definidos para cada tarefa. Basicamente, pode ser entendido como uma matriz que revela graficamente para cada item, em uma escala de tempo, o período que deve ser realizado.

Um gráfico de *Gantt* do projeto é indicado pela Figura 9 registrando as tarefas e prazos macros. Nesta etapa o *item 4* dos princípios do planejamento foi bastante executado, sendo fundamental para o andamento das atividades posteriores.

4.3 Modelagem

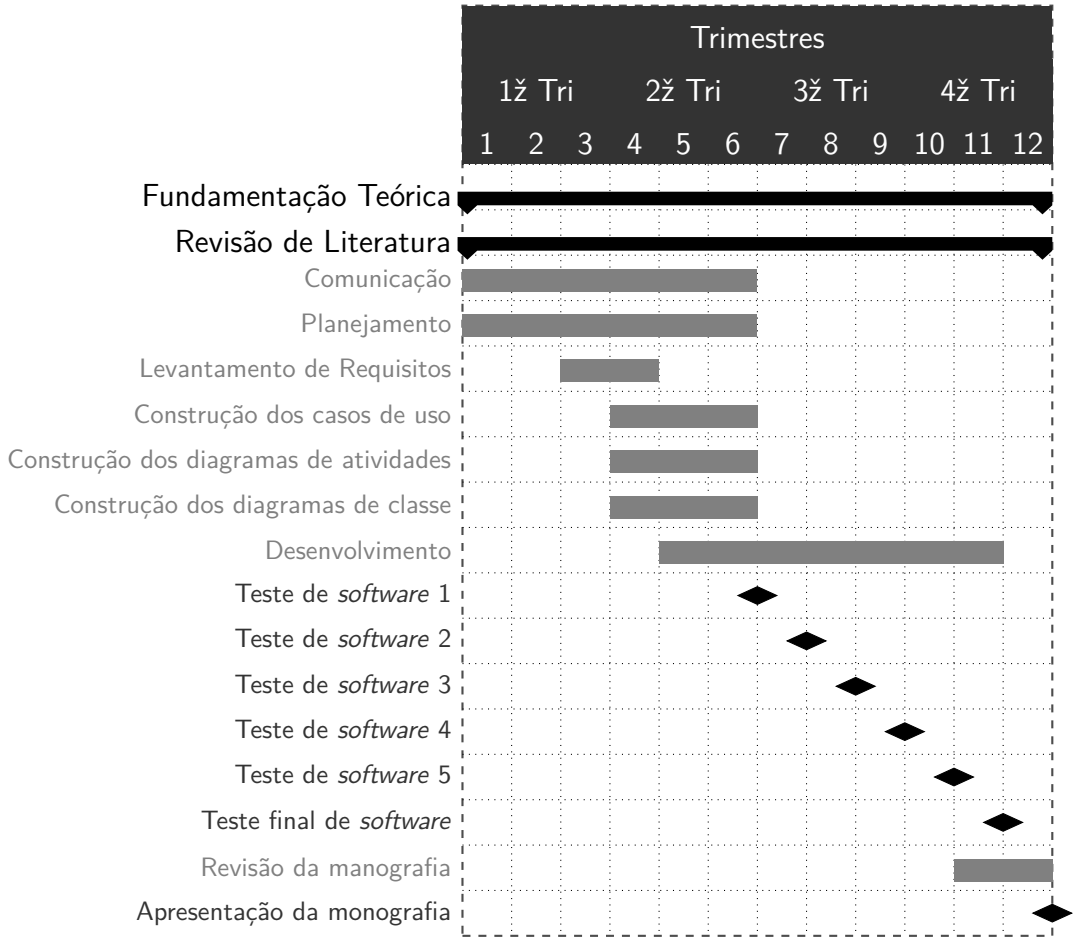
Esta etapa do projeto é dedicada a modelagem do *software* SIPLA concentrando-se especificamente na modelagem de requisitos. A modelagem de requisitos usa a combinação de formas textuais e diagramáticas para descrevê-los e representa-los, garantindo as informações necessárias para o projeto, implementação e realização de testes operacionais.

Inicialmente, na tarefa de elicitação de requisitos foi realizada a coleta de informações preliminares. Posteriormente, foi construído uma lista de requisitos do sistema baseada na elicitação de requisitos coletados no formulário e na comunicação verbal por meio das reuniões de acompanhamento com os clientes do projeto.

4.3.1 Descrição geral do *software*

Como dito, o *software* proposto apresenta-se como uma ferramenta computacional livre, com aplicação na área acadêmica e comercial, que tem como ideia central realizar, a priore, estudos de fluxo de potência em circuitos gerados a partir informações contidas

Figura 9 – Cronograma do Projeto



na BDGD. Dessa forma, serão elencados os requisitos do sistema necessários para sua operacionalização.

4.3.2 Requisitos funcionais (casos de uso)

Nesta seção serão listados os requisitos funcionais do *software* com a finalidade de demonstrar as tarefas a serem desenvolvidas pelo sistema.

1. [**RF001**] Acessar a Base de Dados Geográficos da Distribuidora.

   - **Prioridade:** Essencial

   O *software* deve permitir que o usuário selecione qual BDGD ele quer usar em suas simulações.

2. [**RF002**] Conectar a Base de Dados Geográficos da Distribuidora.

   - **Prioridade:** Essencial

   O *software* deve permitir que o usuário conecte-se a BDGD selecionada.

3. **[RF003]** Selecionar as características da rede que deve ser modelada.

   - **Prioridade:** Essencial

   O *software* deve permitir que o usuário selecione qual rede pertencente ao BDGD selecionado ele deseja modelar. Selecionando subestação de alta tensão, circuito, subestação de média tensão e alimentadores.

4. **[RF004]** Visualizar rede modelada.

   - **Prioridade:** Desejável

   O *software* deve permitir que o usuário visualize as redes que foram configuradas para simulação.

5. **[RF005** Configurar cor de alimentadores selecionados.

   - **Prioridade:** Desejável

   O *software* deve permitir que o usuário selecione as cores dos alimentadores selecionados.

6. **[RF006]** Visualizar equipamentos transformadores da rede selecionada.

   - **Prioridade:** Desejável

   O *software* deve permitir que o usuário selecione se deseja visualizar equipamentos transformadores de distribuição pertencentes a rede selecionada.

7. **[RF007]** Visualizar equipamentos compensadores da rede selecionada.

   - **Prioridade:** Desejável

   O *software* deve permitir que o usuário selecione se deseja visualizar equipamentos compensadores pertencentes a rede selecionada.

8. **[RF008]** Configurar fluxo de potência.

   - **Prioridade:** Essencial

   O *software* deve permitir que o usuário selecione as configurações do modo de simulação do fluxo de potência ele deseja executar: *Snapshot* ou *Daily*.

9. **[RF009]** Configurar curva de carga.

   - **Prioridade:** Importante

   O *software* deve permitir que o usuário selecione as configurações das curvas de cargas a serem utilizadas no modo *Daily*.

10. **[RF010]** Executar fluxo de potência.

    - **Prioridade:** Essencial

    O *software* deve permitir que o usuário execute o fluxo de potência.

11. **[RF011]** Gerar arquivo .DSS.

    - **Prioridade:** Desejável

    O *software* deve permitir que o usuário gere um arquivo .DSS com a descrição dos alimentadores.

12. **[RF012]** Salvar arquivo .DSS.

    - **Prioridade:** Desejável

    O *software* deve permitir que o usuário salve um arquivo .DSS com a descrição dos alimentadores.

13. **[RF013]** Inserção de Medidores de Energia.

    - **Prioridade:** Essencial

    O *software* deve permitir que o usuário insira equipamentos de medição na rede simulada.

14. **[RF014]** Inserção de Monitores.

    - **Prioridade:** Essencial

    O *software* deve permitir que o usuário insira equipamentos de monitoramento na rede simulada.

15. **[RF015]** Visualização dos Resultados Registrados pelos Medidores.

    - **Prioridade:** Essencial

    O *software* deve permitir que o usuário visualize os resultados registrados pelos medidores inseridos na rede simulada.

16. **[RF016]** Plotagem Gráfica.

    - **Prioridade:** Essencial

    O *software* deve permitir que o usuário visualize de forma gráfica as grandezas elétricas monitoradas pelos equipamentos monitores.

17. **[RF017]** Visualização de grandezas elétricas simuladas pós fluxo de potência.

- **Prioridade:** Essencial

O *software* deve permitir que o usuário visualize em forma de tabelas as grandezas elétricas resultantes do fluxo de potência.

4.3.3 Requisitos não funcionais

Nesta seção serão listados os requisitos não funcionais do *software*.

1. **[NF001]** Compatibilidade com sistemas operacionais *Linux e Windows*.

   - **Prioridade:** Essencial
   - **Categoria:** Compatibilidade

   O *software* deve funcionar em sistemas operacionais *linux* e *windows*.

4.4 Modelagem de caso de uso (especificação de requisitos do sistema)

A metodologia utilizada para construção dos casos de uso será baseada em três aspectos: inicialmente descreve-se de forma literal o que pretende-se de cada caso de uso, posteriormente descreve-os de forma estruturada e por último os reproduzem diagramaticamente utilizando duas ferramentas UML: diagrama de caso de uso e diagrama de atividades.

A modelagem dos casos de uso podem ser vista nas Tabelas 3, 4, 5, 6, 7, 8, 9, 10 e também nas Tabelas 11, 12, 13 14, 15, 16, 17, 18, 19, tendo sua compilação agregada na Tabela 2. A união de todos os diagramas de caso de uso culminam no diagrama geral de caso de uso representado na Figura 10. Nesse caso, finaliza-se o processo de modelagem de caso de uso do sistema e pode-se começar o método de modelagem de classes.

4.5 Modelagem das classes do sistema

A metodologia aplicada nessa etapa do projeto é a construção dos cartões CRC. O objetivo identificar as classes do sistema e posteriormente utilizar a estrutura diagramática de classe disponível pela UML para condensar as informações e construir o diagrama de classe do *software*.

A primeira classe a ser modelada é a classe *cmainwindow* responsável pelo *loop* infinito que dá origem a interface gráfica principal mostrada na Figura 11. A Tabela 37 trás as características que compõe a classe e a Figura 66 a sua modelagem em UML,

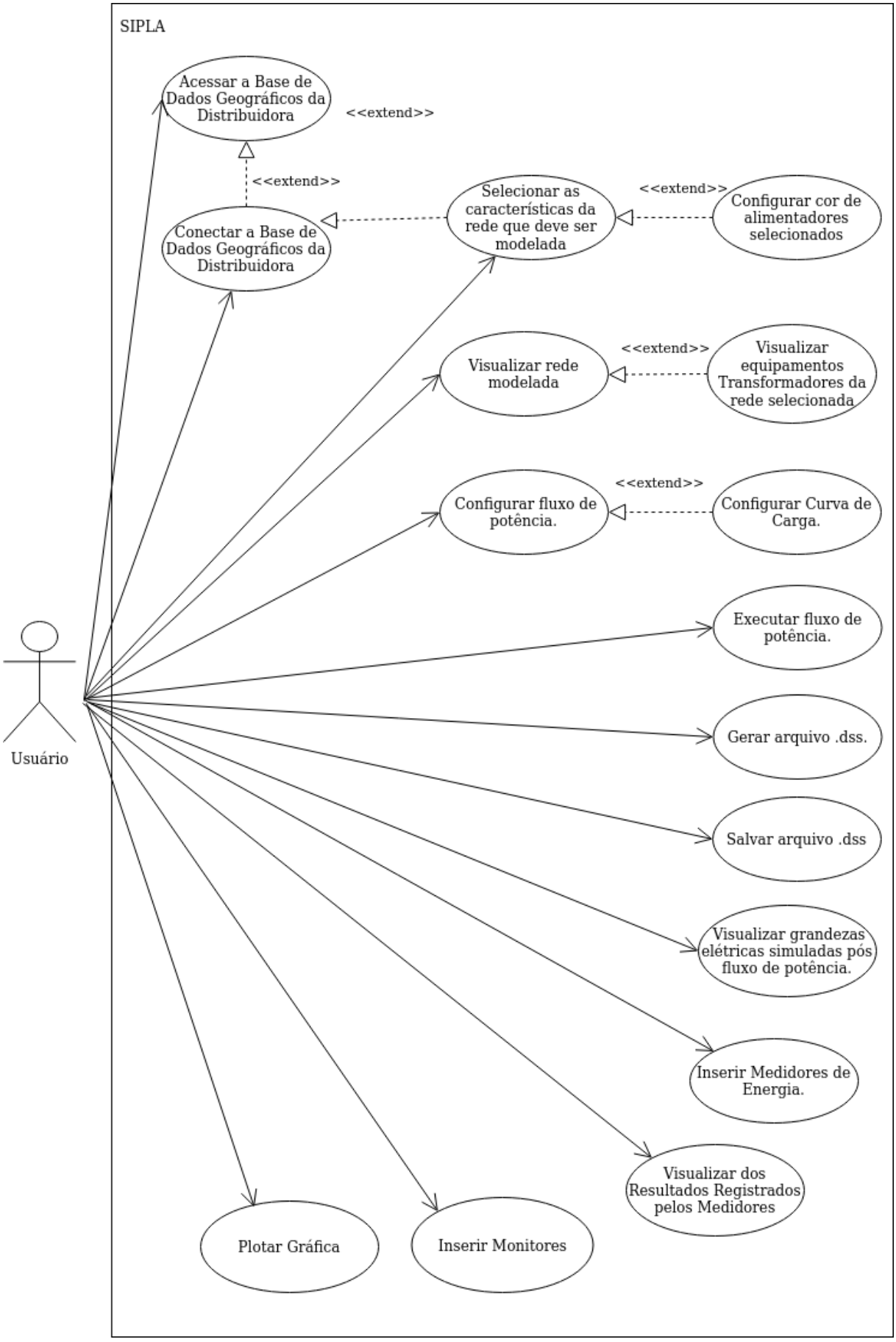


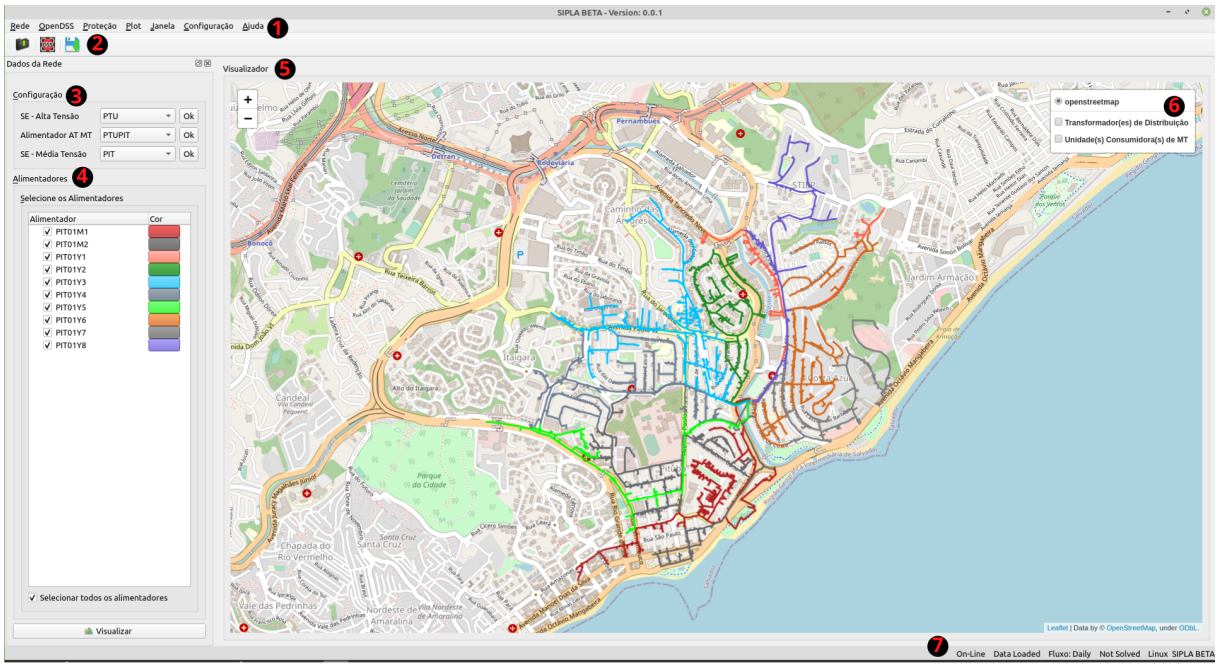
Figura 10 – Diagrama de caso de uso geral - UML

Tabela 2 – Modelagem dos caso de uso geral.

Nome do caso de uso	Tabela
Acessar a Base de Dados Geográficos da Distribuidora	**Tabela 3**
Conectar a Base de Dados Geográficos da Distribuidora	**Tabela 4**
Selecionar as características da rede que deve ser modelada	**Tabela 5**
Visualizar rede simulada	**Tabela 6**
Configurar cor de alimentadores selecionados	**Tabela 7**
Visualizar equipamentos Transformadores da rede selecionada.	**Tabela 8**
Visualizar equipamentos Compensadores da rede selecionada.	**Tabela 9**
Configurar fluxo de potência	**Tabela 10**
Configurar Curva de Carga	**Tabela 11**
Executar fluxo de potência	**Tabela 12**
Gerar arquivo .dss	**Tabela 13**
Salvar arquivo .dss	**Tabela 14**
Inserir Medidores de Energia	**Tabela 15**
Inserir Monitores de Energia	**Tabela 16**
Visualizar dos Resultados Registrados pelos Medidores	**Tabela 17**
Plotagem gráfica	**Tabela 18**
Visualizar grandezas elétricas simuladas pós fluxo de potência	**Tabela 19**

ressalta-se que o método *initui* possui em seu escopo todos os acionamentos de gerência de interface que pode ser mais explorado na analise da codificação.

Figura 11 – Tela principal do *software SIPLA*.



Outra classe importante é *cmenutoolbar* responsável por montar a estrutura dos *menus* e da barra de ferramenta, indicados pelos itens 1 e 2 da Figura 11, tornando-se fundamental para interação usuário/ação do ponto de vista ação/comando. A Tabela 38 descreve sua estrutura, apontando suas relações com outra classe do sistema chamada *cmainactions* que será descrita posteriormente. A Figura 67 mostra a modelagem em UML com os seus objetos e métodos, sendo os detalhes expostos na codificação.

O principal *layout* da GUI é modelado pela classe *cmainpanel*, tendo seu CRC exposto na Tabela 39 que mostra a presença do *centralwidget*, item 5 da Figura 11, responsável por abrigar o visualizador da rede. Outra classe que compõe o *layout* principal é a *cnetpanel* exposta na Tabela 40, que irá abrigar a estrutura que fará o gerenciamento das informações que constitui a modelagem da rede de distribuição a ser estudada. As Figuras 69 e 70 representam a diagramação das classes em UML.

Geralmente, em aplicações *desktops*, o *layout* da janela principal possui uma barra de *status* e no *software* SIPLA isso não é diferente, como visto no item 7 da Figura 11. Sua modelagem é realizada pela classe *statusbar*, seu cartão CRC pode ser visto por meio da Tabela 41 e sua estrutura em UML na Figura 72.

A classe *mainactions* é responsável pela gerência das ações do sistema, ou seja, é nela que estão os métodos que acionam os principais requisitos do *software*, as janelas do tipo *dialogs* ou módulos específicos, como por exemplo a atualização do *status*. Na Tabela 42 é mostrada as principais responsabilidades da classe *mainactions* e a Figura 81 sua diagramação em UML.

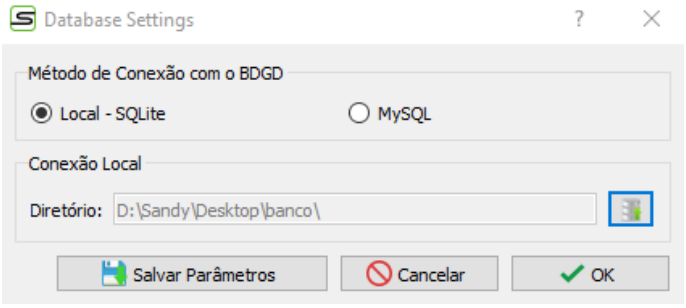
A medida que o usuário interage com o *software* é possível que ocorram erros de execução ou procedimentos, nesse momento é necessário que o sistema mostre ao usuário qual tipo de erro impede o seguimento da tarefa executada. A Tabela 43 irá descrever as principais responsabilidades para o funcionamento desses *feedbacks* de exeção. A classe que gerencia esse processo é a classe pai *cerror* que vai interagir com as classes filhas *conndatabaseerror*, *execdatabaseerror*, *filedatabaseerror*, *execopendss*, *execselectionfields*, *execconfigopendss* e *execenergymeter* que herdam seus atributos e métodos executando desta forma as mensagens necessárias. A diagramação em UML das classes podem ser vista na Figura 85.

Nota-se a grande quantidade de classes e modularidade do *software* SIPLA e com o objetivo de organizar o módulos desenvolvidos foi realizado o empacotamento de classes com funções relacionadas nos denominados pacotes de análise.

Existe um conjunto de classes que se relacionam especificamente com o gerenciamento da base de dados que será usada no sistema. A primeira a ser apresentada é a classe *cconfigdialog* responsável por estabelecer a conexão do sistema com a base de dados, permitindo que o usuário escolha se fará a conexão via banco de dados local ou via servidor, disponibilizando desta forma o acesso aos dados para o sistema. A Tabela 44

mostra de forma básica sua responsabilidade e indica sua diagramatização por meio da Figura 88, o resultado na interface gráfica pode ser visto na Figura 12.

Figura 12 – Conexão com BDGD.



Contudo, a conexão interna as informações pertencentes a BDGD é de responsabilidade da classe *cdbaseconn* tendo suas atribuições expostas de maneira objetiva na Tabela 45 e seus principais métodos sinalizados na Figura 87.

Algumas informações são fundamentais para modelagem da rede que pretende-se estudar, como por exemplo: a lista de subestação de alta tensão, a lista de circuitos de alta para média tensão e a lista de subestação de média tensão pertencentes a BDGD selecionada e essas informações são gerenciadas pela classe *cdbase* tendo suas responsabilidades expostas na Tabela 46 com seus principais métodos expostos na Figura 86. O *groupbox* configuração, mostrado no item 3 da Figura 11 é o resultado da execução da classe aplicado na interface gráfica.

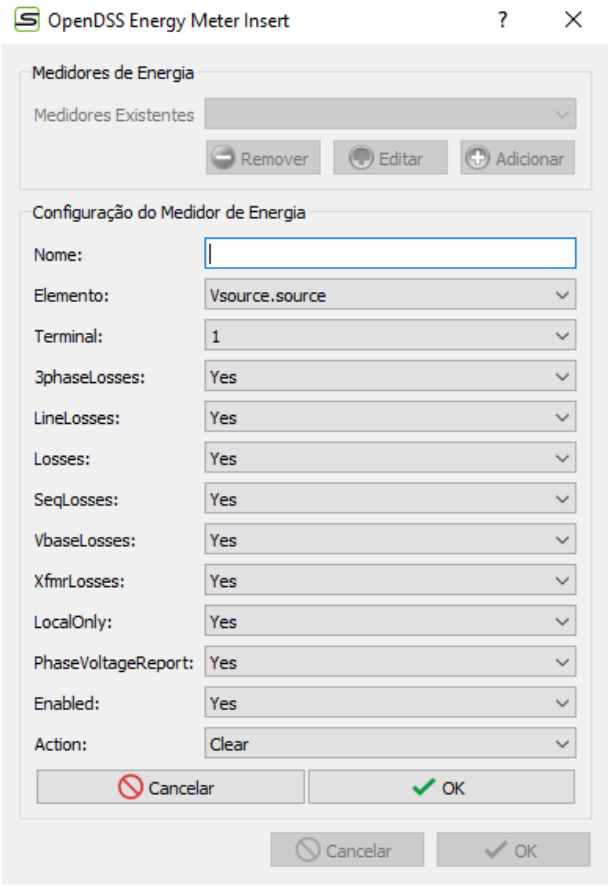
Como as informações dos alimentadores e equipamentos pertencentes as redes de distribuição são retiradas de uma banco de dados georreferenciados é possível usá-las para o desenho dos diagramas. As informações geográficas precisam ser gerenciadas para serem utilizadas no momento propício e quem faz esse gerenciamento é a classe *cdbasecoord* através de métodos que armazenam e utilizam as informações relevantes a cada necessidade no fluxo da utilização do *software*. A Tabela 47 mostra o cartão CRC e a Figura 84 seu diagrama em UML. O resultado dessas interações pode ser visto no traçado dos alimentadores selecionados e mostrados no visualizador de rede na Figura 11.

Outro pacote de analise desenvolvido reúne todas as classes relacionadas ao motor de cálculo *OpenDSS*. A Tabela 50 irá descrever as principais responsabilidades de uma classe extremamente importante para o funcionamento do sistema, a classe *copendss*. Convém destacar que o cartão CRC descrito não expressa todos os detalhes da classe citada, visto que esses detalhes só podem ser vistos na codificação, contudo, é possível vislumbrar suas principais ações ao analisar a sua descrição na Figura 80. Pode-se afirmar que a classe *copendss* é o coração do SIPLA, pois faz todo o gerenciamento das informações que serão necessárias para a preparação da solução do fluxo de potência, colhendo, armazenando e organizando os dados pertinentes.

A classe que realiza a interação com as informações da BDGD selecionada pelo usuário é a *cdata*. Os métodos realizam a captação dos dados e armazenam em objetos que serão utilizados no momento oportuno pelo sistema. A Tabela 61 apresenta as responsabilidades dessa classe extremamente necessária para construção da sintaxe dos alimentadores e seu diagrama UML indicado pela Figura 71.

É possível inserir alguns equipamentos na rede elétrica com o objetivo de registrar ou monitorar as grandezas elétricas. As classes que são responsáveis por realizar essas ações são: *cinsertenergymeterdialog* e *cinsertmonitordialog* tendo seus cartões CRC apresentados pelas Tabelas 55 e 54 respectivamente, bem como seus diagramas UML nas Figuras 75 e 77.

Figura 13 – *SubMenu* OpenDSS - *insert energymeter* extendido.



Note nas Figuras 13 e 14 as interfaces que executam a inserção do medidor de energia e monitor no circuito, respectivamente. O usuário pode nomear o equipamento, escolher o elemento do circuito que será conectado e setar os seus parâmetros e usar as informações coletadas após execução do fluxo de potência.

Para que o usuário realize o fluxo de potência é necessário que seja executada algumas configurações. As configurações são gerenciadas pela classe *cconfigdialog*, onde o usuário do sistema interage com o *software*, indicando algumas informações essenciais para

Figura 14 – *SubMenu* OpenDSS - Monitor extendido.

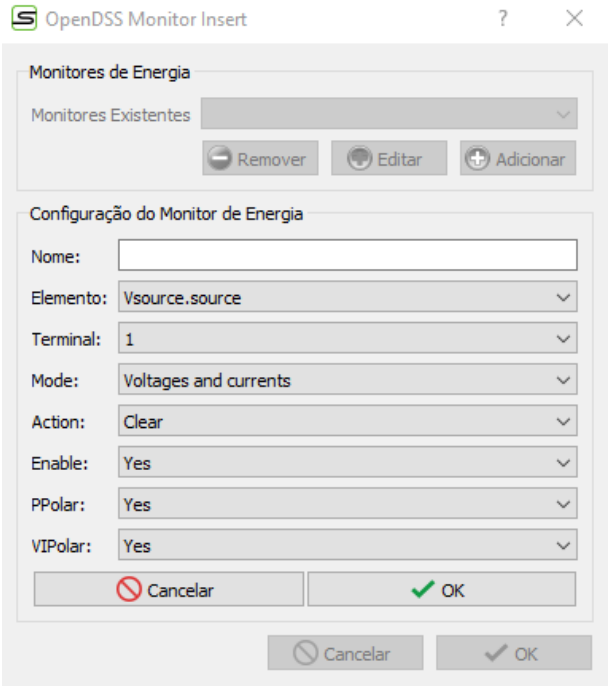
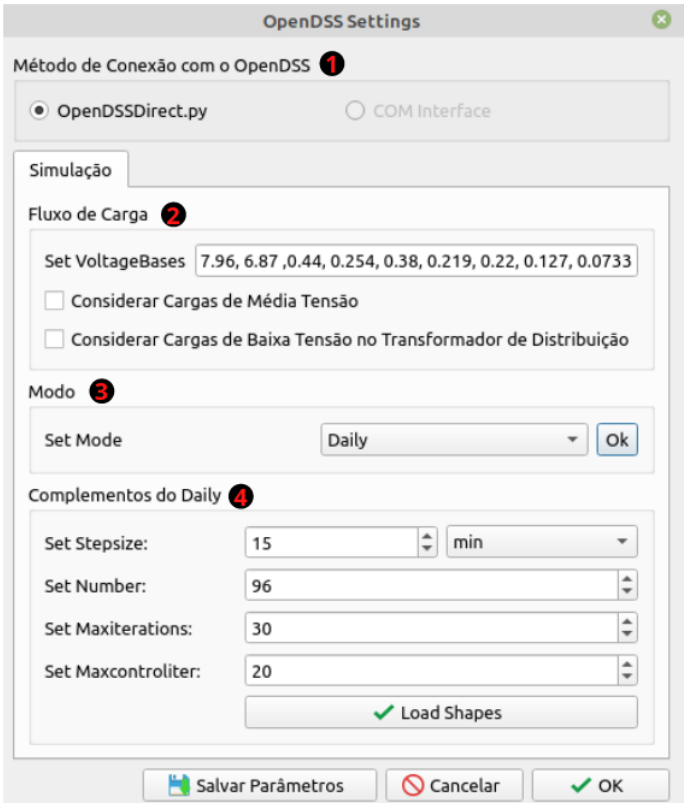


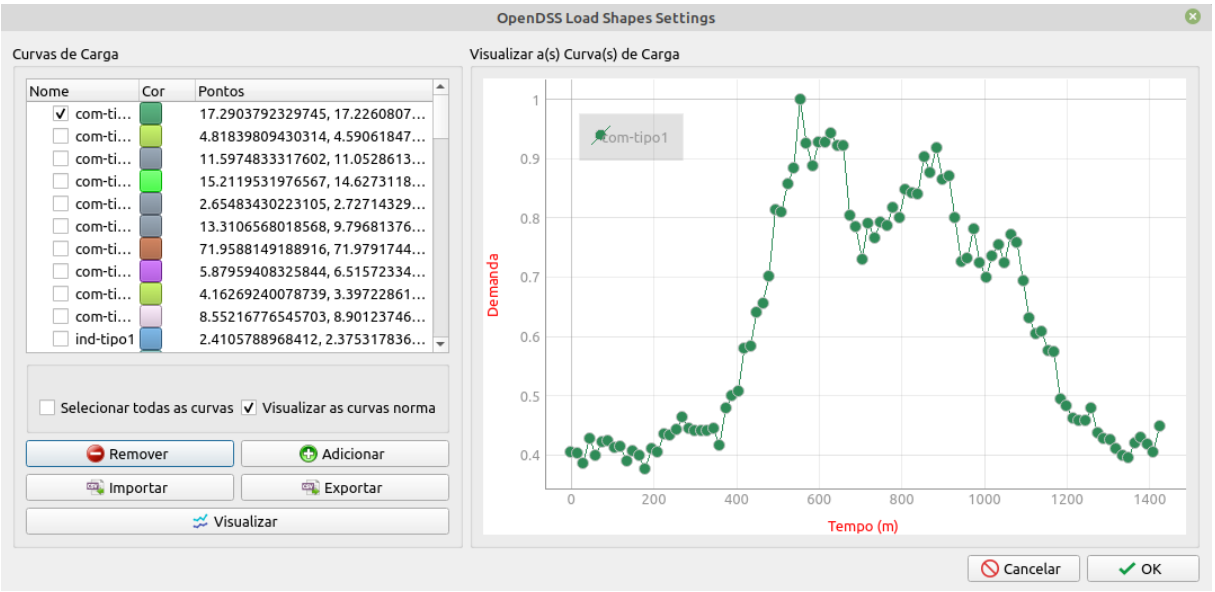
Figura 15 – Configuração do Sistema.



realização do fluxo, como por exemplo, indicando qual o motor de cálculo será utilizado, quais as tensões bases que devem ser consideradas, qual o modo de solução do fluxo de

potência, quais cargas devem ser utilizadas nas construção do alimentador e quais curvas de cargas devem ser consideradas, como pode ser visto nos itens de 1, 2, 3 e 4 da Figura15. A Tabela 51 compila as responsabilidades dessa classe e seu diagrama UML indicado pela Figura 79.

Figura 16 – Dialog loadshape.



Quando o usuário seleciona o modo de solução de fluxo de potência *daily* é necessário também que ele selecione qual o perfil das curvas de carga que compõe cada unidade consumidora. A classe *cconfigloadshapedialog* é responsável por gerir esse processo, seu cartão CRC pode ser visto na Tabela 51, sua modelagem em UML na Figura 78 e execução de sua interface por meio da Figura 16 onde uma curva aleatória foi plotada como exemplo.

É possível inserir alguns equipamentos na rede elétrica com o objetivo de registrar ou monitorar as grandezas elétricas. As classes que são responsáveis por realizar essas ações são: *cinsertenergymeterdialog* e *cinsertmonitordialog* tendo seus cartões CRC apresentados pelas Tabelas 55 e 54 respectivamente, bem como seus diagramas UML nas Figuras 75 e 77. Após inserir um monitor de energia num ponto da rede e realizar o fluxo de potência é possível fazer o acompanhamento da grandeza monitorada por meio de gráficos. A construção da janela responsável por intermediar essa ação do usuário é feita pela classe *cconfigplotdialog*, o cartão CRC associado encontra-se descrito na Tabela 53 e sua diagramação em na Figura 76.

Os resultados são gerados pelo *solver* do SIPLA, o *software* OpenDSS, e como dito nos capítulos anteriores, existem duas formas da interface se comunicar com seu motor de cálculo: via OpenDSS nativo ou via comunicação DLL. A classe que importa os

métodos necessários para as ações especificas é a classe pai *CConn* com suas classes filhas *copenDSSDirectConn* e *copenDSScoconn* que possuem seus cartões CRC representados pelas Tabelas 58, 59 e 60 respectivamente tendo diagramação compilada UML verificada pela Figura 68.

Os documentos da extensão *.py* que compõe o *software* SIPLA não se esgotam nos apresentados anteriormente. Existem outros arquivos que auxiliam na construção do sistema e são responsáveis por ações específicas, como por exemplo, armazenar dados de tabelas convertidas da BDGD ou aglutinar imagens que serão usadas na construção da interface, porém não serão descritos no detalhe e podem ser vistos na codificação.

4.6 Conclusões Parciais

O desenvolvimento de um *software* é complexo. A comunicação assertiva com o cliente é fundamental para o êxito do processo, visto que o entendimento dos requisitos do sistema é etapa *sine qua non* para que resultado final atenda as especificações.

Construir um planejamento fluido corrobora para que no processo de desenvolvimento, a não rigidez do projeto permita que adaptações sejam incorporadas e contribuam com a entrega.

Aplicar conceitos tradicionais de modelagem de sistemas ajuda na coerência entre requisitos e aplicabilidade, porém a dinamicidade trazida pela aplicação de metodologias ágeis, permite que a interação com o cliente final seja mais rotineira, gerando *feedbacks* mais rápidos e auxiliando na assertividade das aplicações.

Modelar um sistema modular acrescenta desafios surpreendentes, todavia permite a possibilidade de integração mais rápida de módulos diversos que aumentam o *range* de aplicabilidade do *software projetado*. Utilizar ferramentas UML facilita a construção de uma identidade visual do projeto, auxiliando numa visão sistêmica durante o processo de modelagem. Desta forma, entender conceitos tanto tradicionais quanto ágeis de desenvolvimento de *softwares* apresenta-se como melhor estratégia para uma entrega de qualidade.

5 Resultados

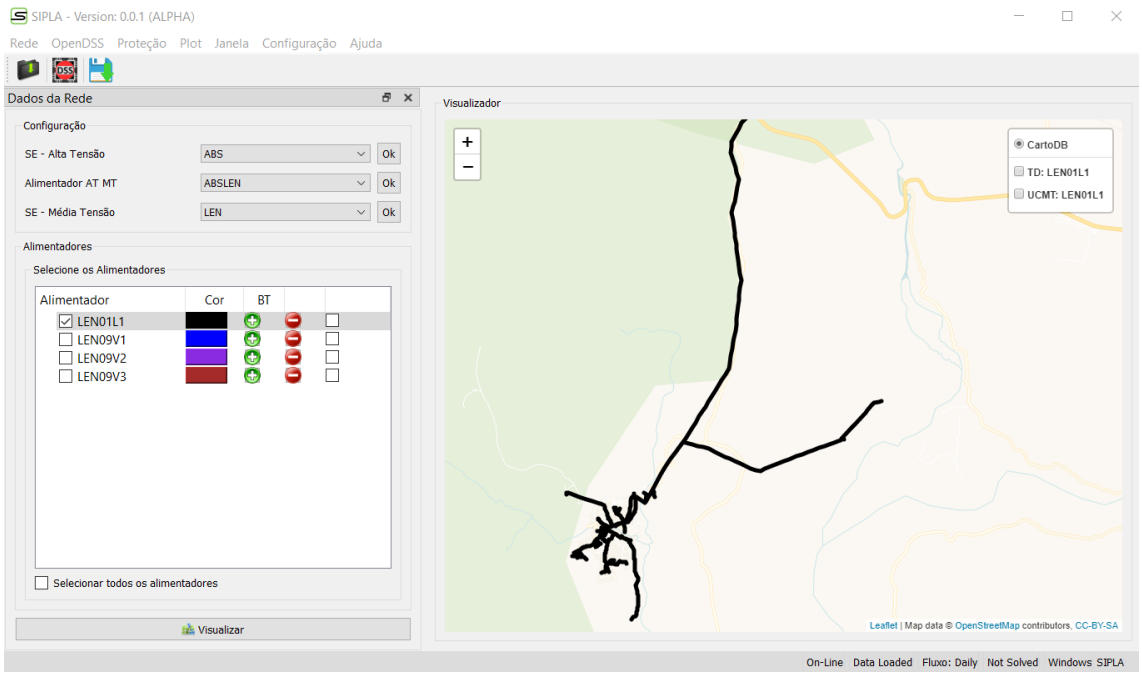
O objetivo deste capítulo é apresentar uma aplicação prática do resultado final da elaboração do *software* Sistema de Planejamento Integrado (SIPLA). Desta forma, será elaborado um estudo de caso, simples, demonstrando sua usabilidade para observações de alguns cenários operativos de um SD. Vale ressaltar, que o propósito do estudo é mostrar, sobretudo, a viabilidade da utilização do *software* para fins educacionais, trazendo introduções de conceitos e observação de fenômenos.

5.1 Estudo de Caso

Para o estudo de caso, foi escolhido um alimentador específico de um Sistema de Distribuição (SD) gerado a partir dos dados extraídos da Base de Dados Geográficas das Distribuidoras (BDGD) de uma concessionária de energia elétrica localizada no Nordeste Brasileiro. O alimentador apresenta característica radial e pertence a uma região rural do interior da Bahia, conforme pode ser visto na Figura 17.

Os sistemas solares fotovoltaicos possuem diversas aplicações, dentre elas: sistemas fotovoltaicos isolados, sistemas fotovoltaicos conectados à rede, sistemas de bombeamento de água, sistemas de telecomunicação e monitoramento remoto e dessalinização da água conforme cita Campos *et al.* (2019) e neste estudo, considerou-se o sistema fotovoltaico conectado a rede elétrica.

Figura 17 – Alimentador Radial.

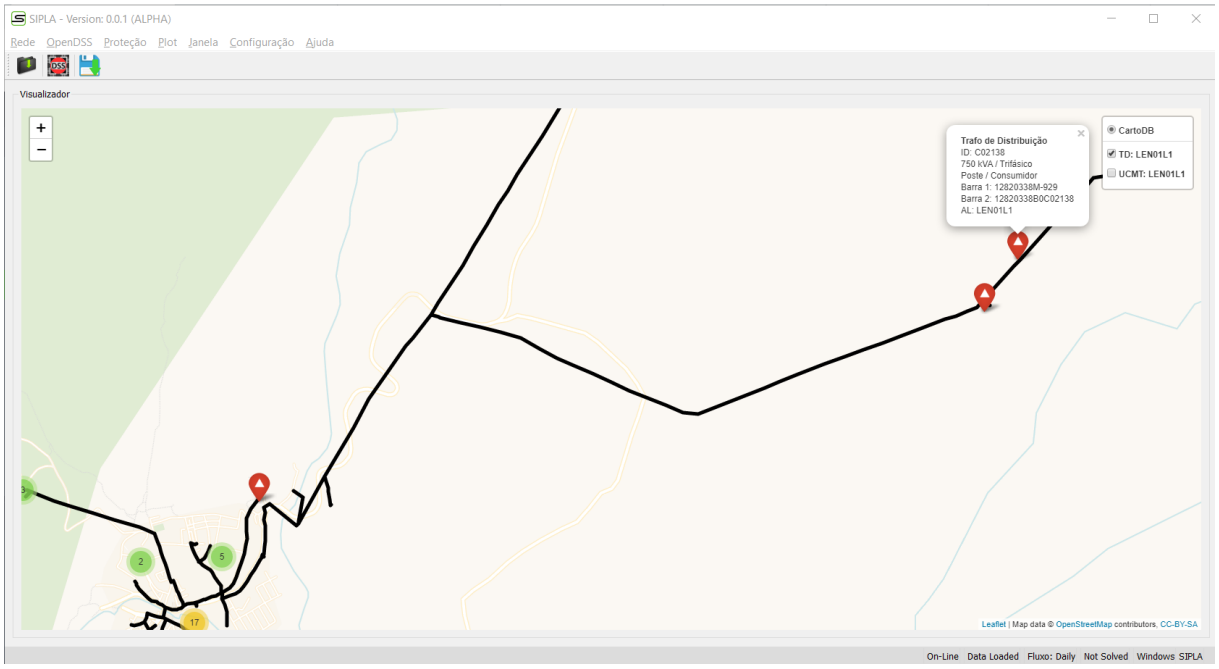


O cenário operativo escolhido baseia-se na observação do comportamento da tensão, potência e perda elétrica em determinado ponto do alimentador. Deve-se avaliar o comportamento das grandezas citadas, com ou sem a presença de um sistema solar fotovoltaico acoplados no ponto. Utilizou-se o conceito de PCC, visto que os sistemas fotovoltaicos foram considerados como cargas, tomando como referência o estudo trazido por Salvatierra *et al.* (2017).

Após avaliação do alimentador de distribuição escolhido, foi selecionado o transformador de código de identificação *C02138* para ser o ponto de análise, ou seja o PCC. O transformador em questão possui $750kVA$ de potência nominal, trifásico e possui como terminal primário a barra *12820338M-929* e secundário a barra *12820338B0C02138*. As informações que modelam o transformador são extraídas diretamente da BDGD, e como dito essa função é realizada pela classe *cdata* que acessa essas informações, trata e descreve o equipamento no formato que o *solver* do sistema reconheça no circuito.

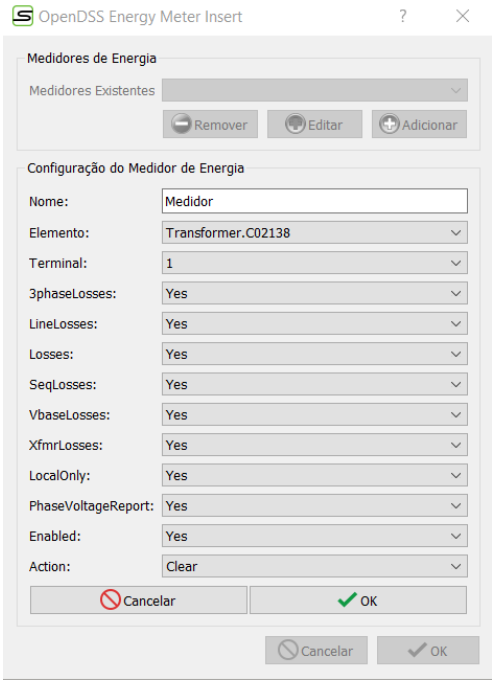
A Figura 18 mostra a localização do transformador no alimentador selecionado.

Figura 18 – Transformador *C02138*



Para que as grandezas elétricas sejam registradas após a execução do fluxo de potência é necessário a inserção de um medidor de energia. Desta forma, foi inserido um medidor no PCC, conforme mostrado na Figura 19.

Figura 19 – Medidor de Energia.



Além dos registros das grandezas elétricas foi necessário o seu monitoramento. Foram instalados monitores no PCC para visualizar a variação da grandeza elétrica ao longo do dia. Instalou-se dois monitores: um para análise da potência, Figura 20, e outro para análise de tensão, conforme visto na Figura 21.

Figura 20 – Monitor de potência.

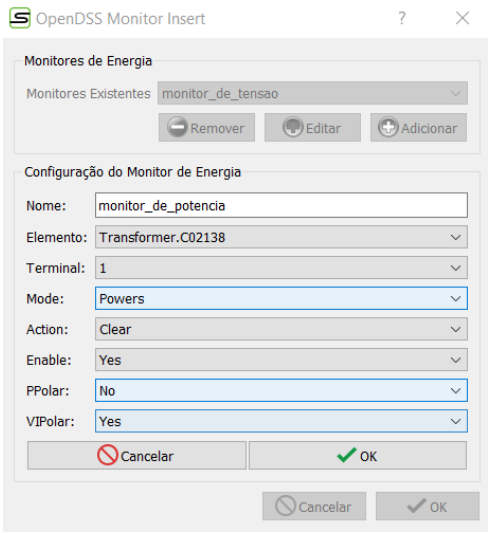
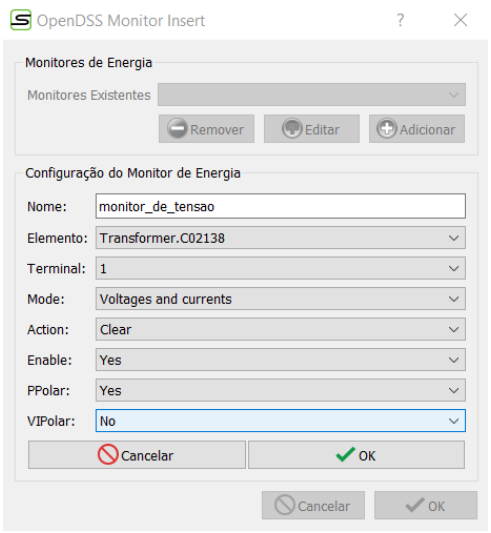


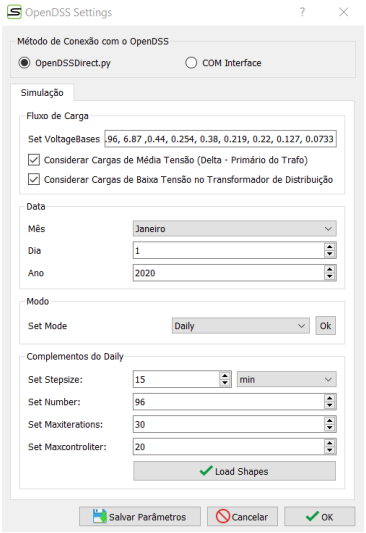
Figura 21 – Monitor de tensão.



Para execução do fluxo de potência, configura-se alguns parâmetros iniciais como: o modo de cálculo de fluxo de carga que será aplicado será o diário, ou *daily*, cujo fluxo de potência é executado para cada hora do dia. É possível verificar na Figura 22 que alguns parâmetros já possuem escolhas pré definidas, que é considerado o modo *default*

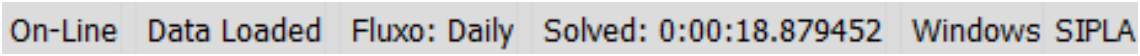
de configuração de fluxo de potência. As cargas consideradas no estudo são de média e de baixa tensão pertencentes ao alimentador estudado. Em relação as cargas de baixa tensão, contudo, ressalta-se que foi utilizado o artificio de programação de realizar um somatório de todas as curvas de carga individuais das unidades consumidoras e conectando-as ao secundário do transformador, ou seja, a carga foi concentrada.

Figura 22 – Configuração do Fluxo de Potência.



Após a configuração de todos os parâmetros de pré analise, foi realizada a execução do *solver*. Note na Figura 23, que a barra de *status* oferece informações sobre o fluxo, e que o tempo de processamento foi de 18 segundos, tempo razoável, sobretudo para fins didáticos, já que não existe necessidade de muita velocidade para apresentar os resultados.

Figura 23 – *Status* após fluxo de Potência.



Observa-se na Figura 24 o perfil de tensão no PCC antes da inserção do *PVSystem*. Ao avaliar os valores monitorados verifica-se que no período entre 00h:00min e aproximadamente 07h:30min as tensões registradas são em torno de 0,49pu e encontram-se críticas, de acordo aos critérios de nível de tensão em regime permanente estabelecidos no Módulo 8 do Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional (PRODIST). E no período entre 07h:30min e 23h:58min os níveis de tensão são considerados precários, visto que medem em torno de 0,91pu.

Na Figura 25 é mostrado o perfil de potência monitorada no transformador de distribuição. Ao analisar o perfil da curva de carregamento do transformador, sugere-se que o carregamento aproxima-se de curvas típicas comerciais, conforme pode ser visto em Tabares e Hernández (2008).

Figura 24 – Variação de tensão no PCC antes da inserção do *PVSystem*

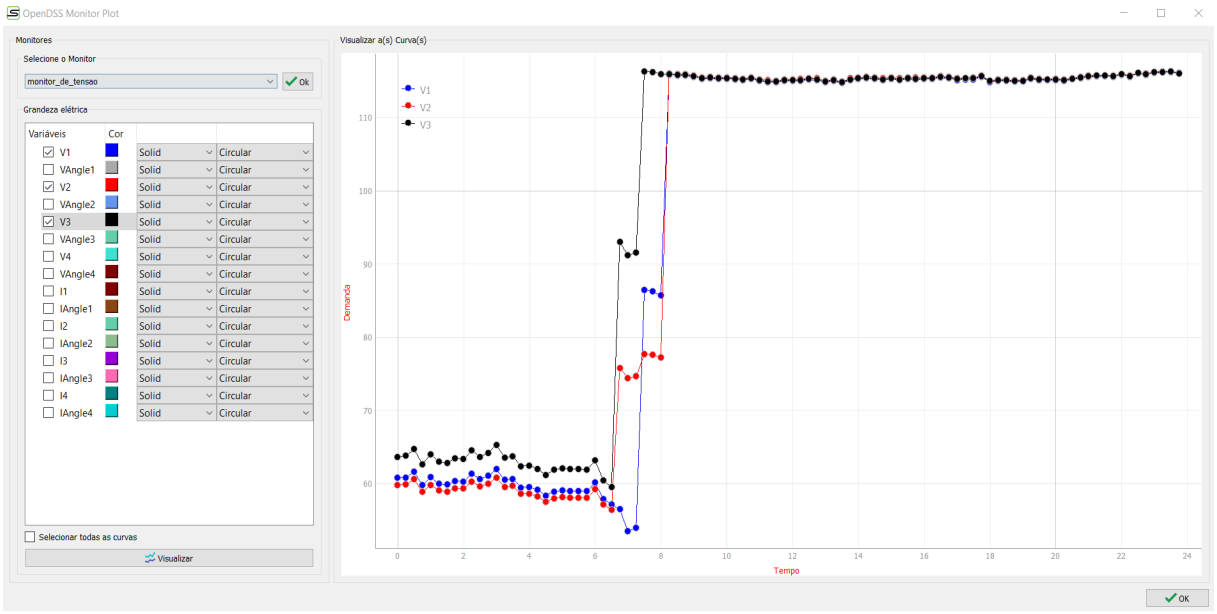
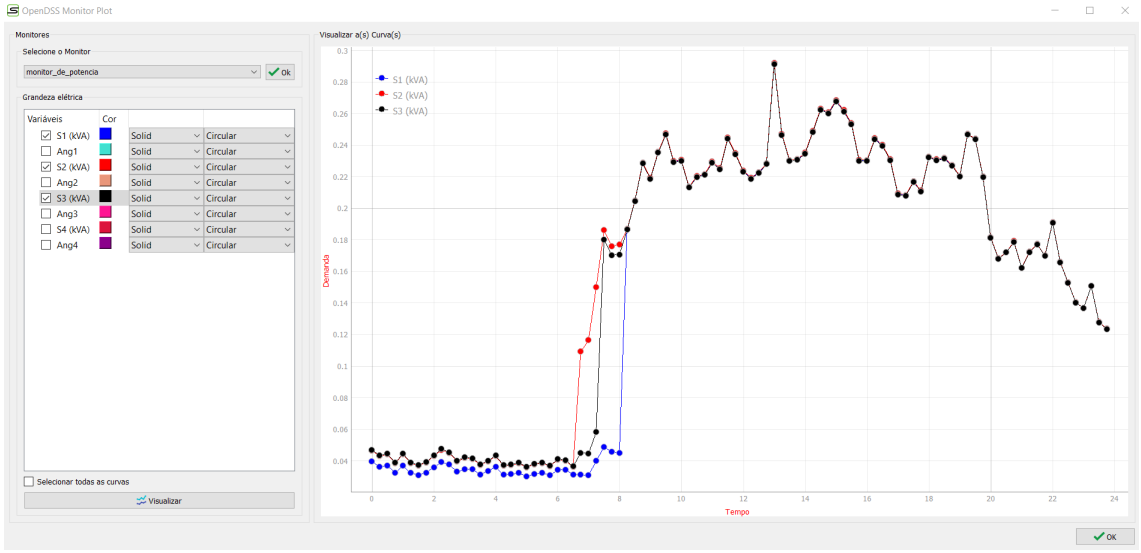


Figura 25 – Variação de potência no PCC antes da inserção do *PVSystem*.

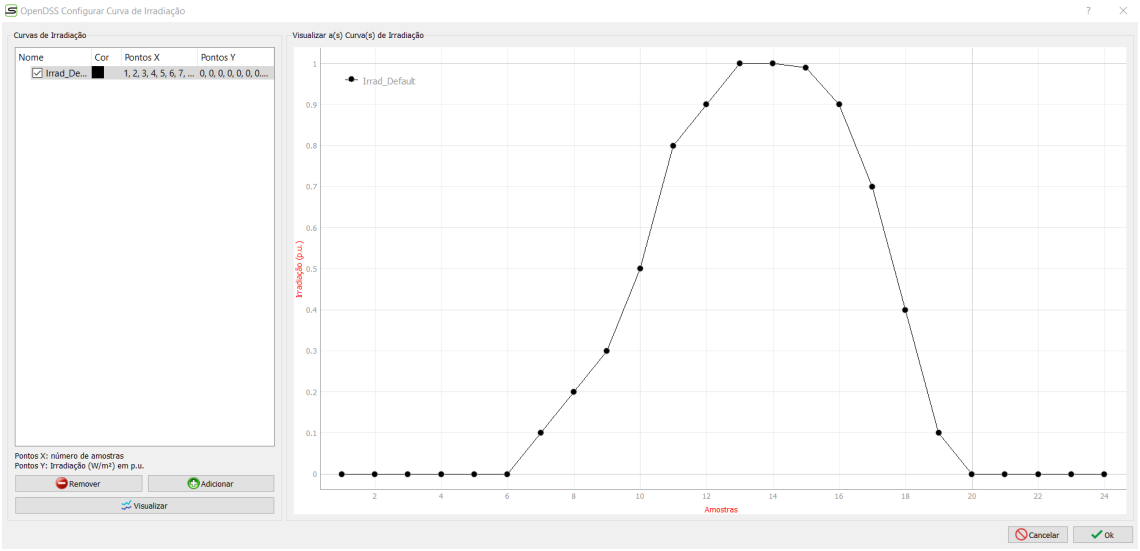


Segundo Shayani (2010) não existe um consenso na definição de índice de penetração, sendo que esse valor pode ser em função do percentual de algumas grandezas que envolvem o SEP analisado, como por exemplo em função da demanda total do alimentador. Neste trabalho, será usada a definição de índice de penetração como um percentual da demanda instantânea máxima do transformador de distribuição que servirá como PCC, neste caso, 290 kVA.

Definido o conceito de índice de penetração, a próxima etapa foi a inserção de um sistema solar fotovoltaico no PCC. Inicialmente, foi executada uma simulação com 50%

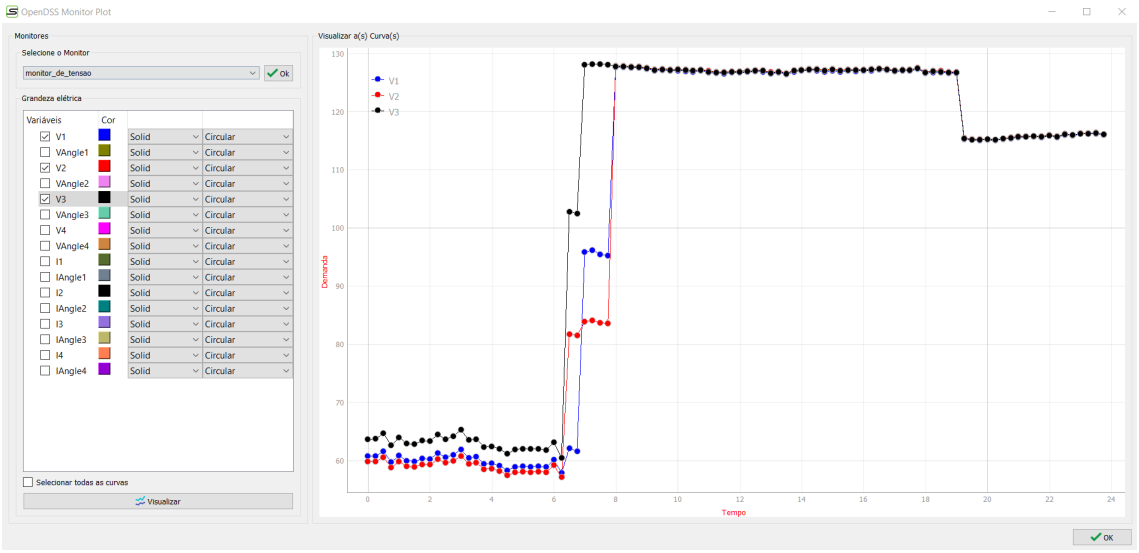
de índice de penetração. A curva de irradiância utilizada foi típica de um dia ensolarado como pode ser visto na Figura 26.

Figura 26 – Perfil da curva de irradiância do *PVSystem*.



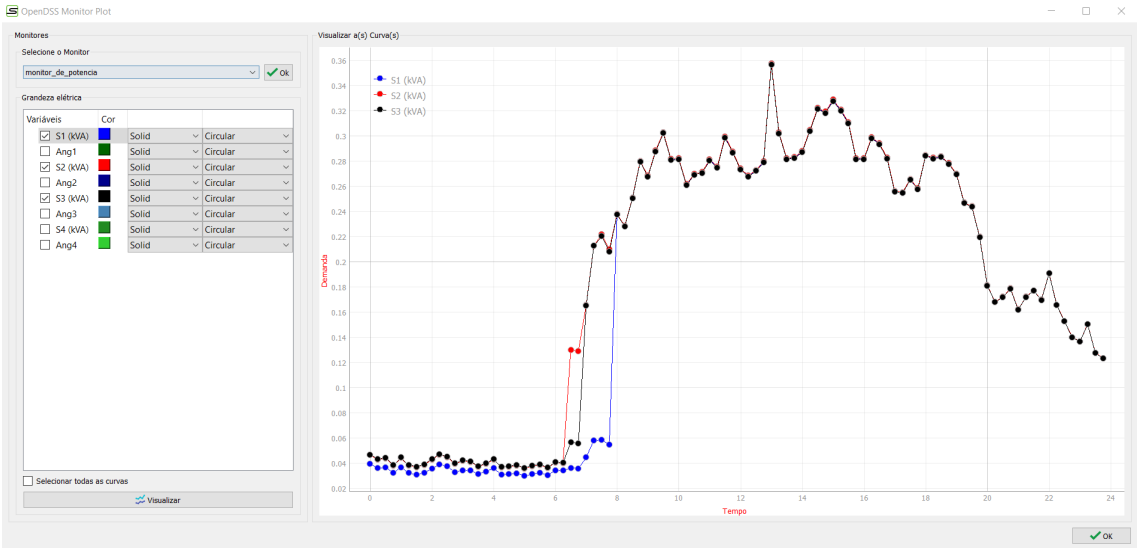
Ao avaliar o perfil da variação de tensão, mostrado na Figura 27, nota-se que no período entre 00h:00min e aproximadamente 07h:30min, as tensões permanecem inalteradas em torno de 0,49pu, visto que nesse período a inserção de potência injetada pelo *PVSystem* é muito baixa, conforme observada na Figura 26. Porém, ao analisar o período entre 07h:30min e 18h:30min verifica-se que as tensões medidas são de 1pu, e que como esperado há um aumento nos valores de tensão no PCC, melhorando os níveis de tensão de regime permanente, o que está de acordo com os resultados trazidos por Paixao, Abaide e Filho (2018), elevando seu *status* para adequado segundo o PRODIST. Após as 18h:30min os valores de tensão retornam para os valores iguais ao cenário sem *PVSystem*, como esperado, visto que nesse momento já não existem injeção de potência por parte do *PVSystem*.

Figura 27 – Variação de tensão no PCC depois da inserção do *PVSystem* - 50% de indíce de penetração.



A variação da potência após a inserção do *PVSystem* pode ser observada na Figura 28 e nota-se que como esperado existe uma elevação da potência no período coincidente com a curva de irradiância do *PVSystem*, visto que nesse período é o momento que o sistema solar fotovoltaico injeta potência na rede e conforme afirma Liu *et al.* (2008) a potência produzida pelo *PVSystem* é proporcional a irradiância e independente do nível de tensão ao qual está conectado.

Figura 28 – Variação de potência no PCC depois da inserção do *PVSystem*.

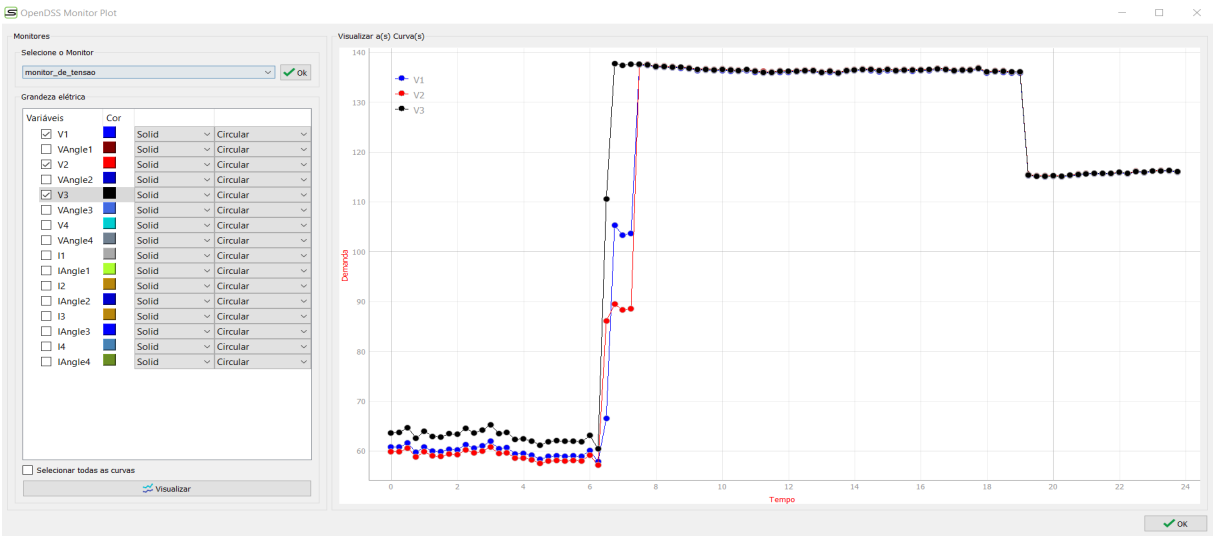


A próxima etapa de simulação é aumentar o índice de penetração para avaliar a resposta do sistema. Como o objetivo principal deste TFG não é analise profunda do

comportamento do sistema, a estratégia utilizada foi de apenas avaliar o sistema estressado com um cenário de 90% de penetração.

O resultado obtido em termos de níveis de tensão de regime permanente pode ser observado na Figura 29, note que no momento de maior injeção de potência pelo *PVSystem* a tensão monitorada é de 1,07pu e segundo o PRODIST também ultrapassa o limite superior estabelecido, voltando para o *status* de tensão crítica.

Figura 29 – Variação de tensão no PCC depois da inserção do *PVSystem* - 90% indice de penetração.



Outro parâmetro que pode ser observado ao avaliar cenários do comportamento de um SEP, com ou sem a presença de *PVSystem*, é a perda técnica. Por meio das Figura 30 e 31, observar-se que as perdas técnicas medidas no PCC aumentaram cerca 33% com a elevação do índice de penetração, e conforme aponta Pesaran (2017), de fato, uma geração distribuída com capacidade e localização inadequadas podem ocasionar efeitos indesejados, como por exemplo o aumento das perdas técnicas.

5.2 Conclusões Parciais

Pelo exposto, o *software* SIPLA se mostrou eficiente para o objetivo proposto: ser um *software* com aplicação em sistemas elétricos de potência e que sirva como uma plataforma para que outros desenvolvedores possam contribuir com novos módulos. Através de um estudo de caso simples como a ferramenta dialoga com possibilidades de aplicações para o empresas do setor elétrico, mostrando uma aplicação prática de estudos de cenários operativos atuais de um Sistema de Distribuição (SD) com a presença de sistemas solares fotovoltaicos.

Figura 30 – Variação das perdas ativas no PCC antes da inserção do *PVSystem* - 90% indice de penetração.

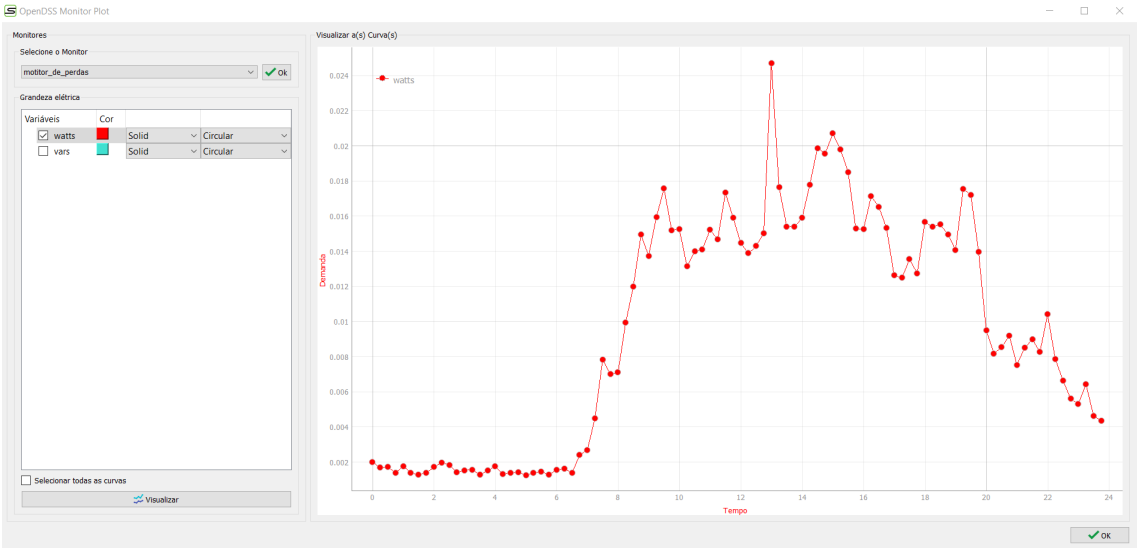
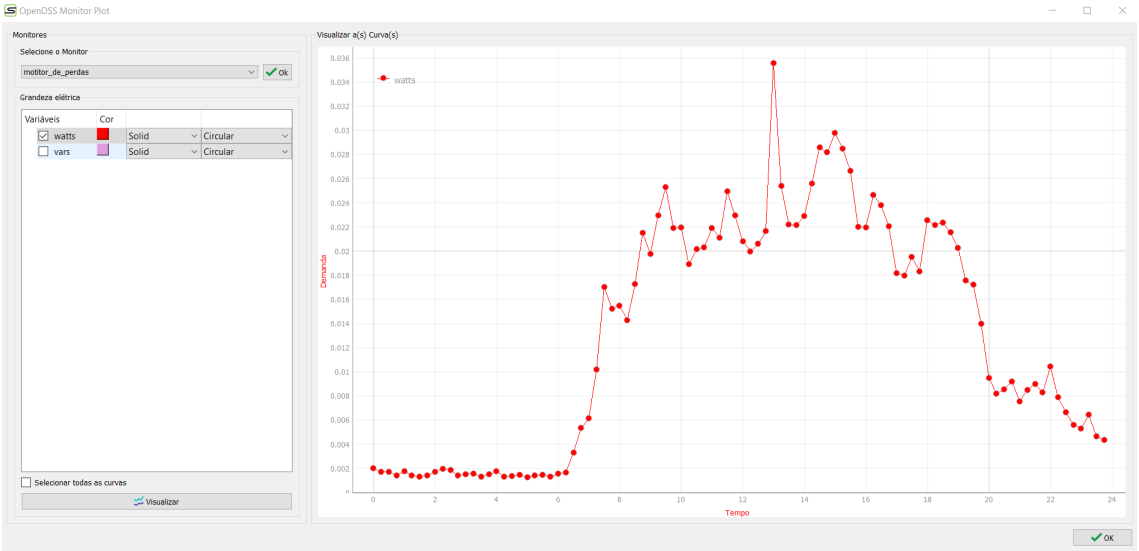


Figura 31 – Variação das perdas ativas no PCC depois da inserção do *PVSystem* - 90% indice de penetração.



6 Conclusões e Sugestões

6.1 Conclusões

Este capítulo tem como objetivo apresentar as conclusões do presente trabalho final de graduação e sumarizar as perspectivas futuras. Desta forma, será realizada a consolidação das análises parciais expostas em cada capítulo, dando assim, uma visão geral dos estudos realizados no trabalho, bem como as suas contribuições.

Inicialmente apresentou-se um capitulo introdutório com o objetivo de contextualizar e justificar o desenvolvimento do trabalho, descrevendo as principais motivações e relevância do tema abordado. Utilizou-se de autoras e autores referências para preambular a necessidade e utilidade do projeto proposto e introduzir o tema central da manografia, além disso, destacou-se os objetivos principais estabelecidos, bem como a metodologia a ser empregada para alcança-los.

Para iniciar o processo de execução foi dedicado um capítulo robusto para fundamentação teórica a fim de entender os principais conceitos a cerca do tema e projetar os melhores caminhos a serem percorridos durante a execução das metas estabelecidas pelo projeto. Entender a definição de *software*, *software livre* e processos para o seu desenvolvimento foi fundamental para a qualidade e agilidade durante o processo da construção do sistema. Adicionalmente, aprofundou-se o conteúdo sobre modelagem de *softwares*, explorando a modelagem de requisitos, tanto usando métodos baseados em classe, quanto baseado em cenários, sendo fundamentais para o exito de todo processo e entrega dos resultados esperados.

Concomitantemente a construção da fundamentação teórica, foi elaborada uma revisão bibliográfica destacando-se os principais trabalhos sobre a temática da monografia. Evidenciando-se as diferenças entre os conceitos de *software livre* e *software opensource*, discorrendo sobre a diversidade dos tipos de licenças possíveis e a sua importância para a área de desenvolvimento de *softwares*. Foi mostrada a diferença dos paradigmas de programação, apresentando o conceito de FOSS e POO que foram importantes durante a fase de desenvolvimento.

Ainda no capítulo de revisão bibliográfica foi destacada a importância da utilização de *softwares livres* em ambientes educacionais, mostrando inclusive exemplos nacionais e internacionais do exito de sua aplicação em universidades. Foi trazido trabalhos que utilizaram a técnica de unir POO e UML para o desenvolvimento de interfaces gráficas utilizando diversas linguagem de programação, inclusive a usada nesse projeto, a linguagem de programação *python*.

No capitulo de desenvolvimento foi mostrado o caminho seguido para execução do que foi planejado. Foi explicitado desde a fase da comunicação à fase da elaboração do projeto, detalhando todos os níveis necessários para implementação, criando escopo do projeto, cronograma e modelagem do sistema. Todas as classes desenvolvidas foram descritas e construídas conforme conceitos apreendidos nos capítulos anteriores.

Após executar todas as etapas citadas foi descrito o resultado final do projeto, mostrando e descrevendo todos os artefatos de *softwares* criados. Todos os itens que compunham o sistema foram detalhados a fim de explicitar ao leitor todas as funções disponíveis. Ademais, foi demonstrado um estudo de caso para solidificar e apresentar a usabilidade e parte da aplicação do SIPLA.

Ressalta-se o principal objetivo e contribuição deste trabalho final de graduação: desenvolvimento de um software livre de código aberto, gráfico e multiplataforma para estudos em sistemas elétricos de potência, possuindo uma arquitetura de software padrão e modular. Além disso, propõe-se que o *software* sirva como uma plataforma para que outros pesquisadores possam melhorar e incrementar sua usabilidade desenvolvendo outros módulos pertinentes a sua aplicação. O *software* desenvolvido, em sua versão atual, é estável e pode auxiliar como ferramenta as áreas de pesquisa e educação, além de aplicações práticas em SEP.

Embora desenvolvido durante a elaboração desta monografia, a versão atual do SIPLA já foi testada para auxilio na aplicação de atividades na disciplina de Proteção de Sistemas Elétricos na Universidade Federal da Bahia mostrando seu potencial de uso.

6.2 Sugestões para trabalhos futuros

Alguns trabalhos já tem sido desenvolvidos após a finalização da construção do SIPLA, foram criados módulos específicos para incrementar as funções e aplicações do mesmo, conforme mostrado abaixo:

- Módulo de Curto Circuito: elaborado para execução de estudos de curto circuito.
- Módulo de PVSystem: elaborado para execução de estudos de sistemas fotovoltaicos.
- Módulo de Storage: elaborado para execução de estudos de sistemas armazenadores de energia.
- Módulo Controle VoltVAr: elaborado para execução de estudos de técnicas controle voltvar.

Porém, outros módulos ainda podem ser desenvolvidos, pois o SIPLA é um programa livre e de código aberto, possui arquitetura de *software* padrão e modular e tais

perspectivas de implementações em pesquisas futuras são aqui sucintamente sumarizadas, como por exemplo:

- Módulo de Harmônicos: o *software* tem suporte para desenvolvimento de um módulo de execução de estudos em harmônicos.
- Módulo de Máquinas: o *software* tem suporte para desenvolvimento de um módulo de execução de estudos em máquinas.
- Módulo de Wind: elaborado para execução de estudos de sistemas de geração eólica.

Referências

AGOSTINI, M. N.; DECKER, I. C.; SILVA, A. S. Desenvolvimento e implementação de uma base computacional orientada a objetos para aplicações em sistemas de energia elétrica. *Sba: Controle & Automação Sociedade Brasileira de Automatica*, SciELO Brasil, v. 13, n. 2, p. 181–189, 2002.

ANAND, A. *et al.* Comparative analysis between proprietary software vs. open-source software vs. free software. In: *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. [S.l.: s.n.], 2018. p. 144–147.

ANEEL. *Geração Distríbuida*. 2019. <https://www.aneel.gov.br/geracao-distribuida>.

AZEVEDO, D. *Revisão de Literatura, Referencial Teórico, Fundamentação Teórica e Framework Conceitual em Pesquisa–diferenças e propósitos*. [S.l.], 2017.

BAHRI, L.; PASTERNAK, W. D.; BINI, E. M. Sistema de gerenciamento para indústria gráfica. 2016.

BARBOSA, M. F. L. Avaliação probabilística do impacto da microgeração fotovoltaica distribuída em redes de distribuição de energia elétrica. Universidade Federal do Rio de Janeiro, 2018.

BARRY, B. I. A. Using open source software in education in developing countries: The sudan as an example. In: *2009 International Conference on Computational Intelligence and Software Engineering*. [S.l.: s.n.], 2009. p. 1–4.

BAUER, A.; PIZKA, M. The contribution of free software to software evolution. In: *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 170–179.

BERALDO, R.; FONTENELLE, R. *O que é Software Livre?* 2020. <http://www.gnu.org/philosophy/free-sw.html>.

BEZERRA, A. A. B.; SILVA, L. M. S. d.; LIMA, A. W. d. Desenvolvimento de um programa computacional para análise de vigas euler-bernoulli utilizando a linguagem python. 2016.

BEZERRA, D. F. Panorama da infraestrutura no nordeste do basill: Energia elétrica. *Caderno Setorial ETENE*, v. 4, n. 65, 2019.

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. 3. ed. [S.l.]: CAMPUS, 2015.

BORGES, L. E. *Python para desenvolvedores: aborda Python 3.3*. [S.l.]: Novatec Editora, 2014.

CAMPOS, C. A. A. *et al.* Análise da capacidade de hospedagem de geração distribuída em uma rede radial de distribuição. Insitituto Federal de Educação, Ciência e Tecnologia de Goiás, 2019.

CARVALHO, M. *Uma história do software livre*. 2011.

CASTRO, F. R.; CRUZ, F. M. D.; ODDONE, N. E. O paradigma da orientação a objetos, a linguagem unificada de modelagem (uml) e a organização e representação do conhecimento: um estudo de caso de um sistema para bibliotecas. *Informação & Informação*, v. 18, n. 1, p. 82–105, 2013.

CORTéS, M. I. *Fundamentos de Engenharia de Software*. [S.l.: s.n.], 2013.

DOOLEY, J. *Software development and professional practice*. [S.l.]: Apress, 2011.

ENGELFRIET, A. Choosing an open source license. *IEEE Software*, v. 27, n. 1, p. 48–49, 2010.

FITZGERALD, B. A critical look at open source. *Computer*, IEEE, v. 37, n. 7, p. 92–94, 2004.

FITZGERALD, B. Open source software: Lessons from and for software engineering. *Computer*, v. 44, n. 10, p. 25–30, 2011.

FOLEY, M.; BOSE, A. Object-oriented on-line network analysis. *IEEE Transactions on Power Systems*, v. 10, n. 1, p. 125–132, 1995.

FOLEY, M. *et al.* An object based graphical user interface for power systems. *IEEE Transactions on Power Systems*, v. 8, n. 1, p. 97–104, 1993.

FREITAS, D. N. d. Metodologia de desenvolvimento de software livre com arquiteturas orientadas a serviços. 2009.

GAFF, B. M.; PLOUSSIOS, G. J. Open source software. *Computer*, v. 45, n. 6, p. 9–11, 2012.

GHOSH, S.; SAHA, A.; MOU, N. I. Load flow simulation of western grid of bangladesh power system using psat and performance-based comparison with powerworld. In: *2010 International Conference on Intelligent Systems, Modelling and Simulation*. [S.l.: s.n.], 2010. p. 292–295.

GOMES, D. A. T. Software livre na educação. Universidade Presbiteriana Mackenzie, 2013.

GOMES, R. V. d. C. C. *et al.* Software livre e engenharia elétrica. In: *Anais do Congresso Nacional Universidade, EAD e Software Livre*. [S.l.: s.n.], 2011. v. 1, n. 2.

GONÇALVES, E. J. T.; CORTÉS, M. I. Análise e projeto de sistemas. *História*, v. 9, p. 3, 2015.

IVANOVICH, C. D.; TIMOFEYEVICH, K. S. Development of user interface for openfoam software environment used in design and technological subdivisions of machine-building enterprises. In: IEEE. *2018 Global Smart Industry Conference (GloSIC)*. [S.l.], 2018. p. 1–7.

KAZMAN, R. *COMPUTING: THE NEXT 50 YEARS Software Engineering*. [S.l.]: IEEE COMPUTER SOC 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA ..., 2017.

KAZMAN, R.; PASQUALE, L. Software engineering in society. *IEEE Software*, v. 37, n. 1, p. 7–9, 2020.

Kober, F.; Manzoni, A.; Lemos, F. A. B. An objected-oriented approach to development and integration of graphical user interface and power system framework. In: *2003 IEEE Bologna Power Tech Conference Proceedings,*. [S.l.: s.n.], 2003. v. 3, p. 7 pp. Vol.3–.

KÖCHE, J. C. *Fundamentos de metodologia científica*. [S.l.]: Editora Vozes, 2016.

KUHMANN, M. *et al.* Hybrid software development approaches in practice: a european perspective. *IEEE Software*, IEEE, v. 36, n. 4, p. 20–31, 2018.

KULASZA, M.; ANNAKKAGE, U. Design of a software framework for research in power system modeling and simulation. In: IEEE. *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. [S.l.], 2013. p. 1–5.

LANES, A. L. F.; ROSSONI, A. Ferramenta computacional para modelagem de alimentadores reais de distribuição no opendss. 2018.

LAZIć, N.; ZORICA, M. B.; KLINDžIć, J. Open source software in education. In: *2011 Proceedings of the 34th International Convention MIPRO*. [S.l.: s.n.], 2011. p. 1267–1270.

LIU, Y. *et al.* Distribution system voltage performance analysis for high-penetration pv. In: IEEE. *2008 IEEE Energy 2030 Conference*. [S.l.], 2008. p. 1–8.

LOBO, E. J. *Guia prático de engenharia de software*. [S.l.]: Universo dos Livros Editora, 2009.

MANZONI, A. Desenvolvimento de um sistema computacional orientado a objetos para sistemas elétricos de potência: aplicaçao a simulaçao rápida e análise da estabilidade de tensao. *Rio de Janeiro*, 2005.

MANZONI, L. V. Uso de sistema de gerência de workflow para apoiar o desenvolvimento de software baseado no processo unificado da rational estendido para alcançar níveis 2 e 3 do modelo de maturidade. 2001.

MIGUEL, P. A. C.; SOUSA, R. O método do estudo de caso na engenharia de produção. 2012.

MILANO, F. *Power system modelling and scripting*. [S.l.]: Springer Science & Business Media, 2010.

MILANO, F. A python-based software tool for power system analysis. In: IEEE. *2013 IEEE Power & Energy Society General Meeting*. [S.l.], 2013. p. 1–5.

OLIVEIRA, T. L. *et al.* Desenvolvimento de um programa computacional livre, gráfico, e multiplataforma para analisar sistemas elétricos de potência em regime permanente e dinâmico. Universidade Federal de Uberlândia, 2019.

PÁDUA, W. d. *Engenharia de software: fundamentos, métodos e padrões*. [S.l.: s.n.], 2003.

PAIXAO, J. L.; ABAIDE, A. R.; FILHO, G. P. Impact evaluation of the photovoltaic generation input on a concessionaire's network. In: IEEE. *2018 Simposio Brasileiro de Sistemas Eletricos (SBSE)*. [S.l.], 2018. p. 1–6.

PESARAN, H. M., pd huy, and vk ramachandaramurthy. *A review of the optimal allocation of distributed generation: Objectives, constraints, methods, and algorithms*, p. 293–312, 2017.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software: Uma Abordagem Profissional*. [S.l.]: McGraw Hill Brasil, 2016.

RIBEIRO, L. Métodos formais de especificaçao: gramáticas de grafos. *VIII Escola de Informática da SBC-Sul*, p. 1–33, 2000.

SALVATIERRA, B. G. *et al.* Power quality analysis of a low-voltage grid with a solar photovoltaic system. In: *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. [S.l.: s.n.], 2017. p. 1–6.

SANTOS, W. B. *et al.* Simplicidade no desenvolvimento agil de software: Um mapeamento sistemático da literatura. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, v. 1, n. 8, 2018.

SARKINEN, J. An open source (d) controller. In: IEEE. *INTELEC 07-29th International Telecommunications Energy Conference*. [S.l.], 2007. p. 761–768.

SHATUNOV, A. N. *et al.* Electrical circuits simulation in free gpl software. In: *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. [S.l.: s.n.], 2020. p. 173–175.

SHAYANI, R. A. Método para determinação do limite de penetração da geração distribuída fotovoltaica em redes radiais de distribuição. 2010.

SILVA, A. C. R. Metodologia da pesquisa aplicada. *São Paulo: Atlas*, 2003.

SILVA, A. L. *et al.* Interface gráfica para redução de espectros ópticos. Universidade Estadual de Feira de Santana, 2016.

SILVA, D.; APARICIO, M.; COSTA, C. Estudo bibliométrico de software livre e open source. In: . [S.l.: s.n.], 2019.

SILVA, E. L. D.; MENEZES, E. M. Metodologia da pesquisa e elaboração de dissertação. *UFSC, Florianópolis, 4a. edição*, v. 123, 2005.

SILVA, J. R. C. da; PACHECO, G. M. Open source software development in matlab for sizing photovoltaic systems. In: *2018 13th IEEE International Conference on Industry Applications (INDUSCON)*. [S.l.: s.n.], 2018. p. 256–262.

SILVA, K. d. L.; ÉVORA, Y. D. M.; CINTRA, C. S. J. Desenvolvimento de software para apoiar a tomada de decisão na seleção de diagnósticos e intervenções de enfermagem para crianças e adolescentes. *Revista Latino-Americana de Enfermagem*, SciELO Brasil, v. 23, n. 5, p. 927–935, 2015.

SILVA, L. F. da. *Uma Estratégia Orientada a Aspectos para a Modelagem de Requisitos*. Tese (Doutorado) — Tese de Doutorado, Computer Science Department, PUC-Rio, 2006.

SINGH, M.; CHAUHAN, N.; POPLI, R. A framework for transitioning of traditional software development method to distributed agile software development. In: *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. [S.l.: s.n.], 2019. v. 1, p. 1–4.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: McGraw Hill Brasil, 2011.

STRIEDER, A.; SCHUCH, C. M.; FRIAS, A. R. Utilização de simuladores de processos como ferramenta para o ensino de engenharia. *Revista Ciência e Tecnologia*, v. 9, n. 14, 2010.

TABARES, H.; HERNÁNDEZ, J. Mapeo curvas típicas demanda de energía eléctrica del sector residencial, comercial e industrial de la ciudad de medellín, usando redes neuronales artificiales y algoritmos de interpolación. *Revista Facultad de Ingeniería Universidad de Antioquia*, Universidad de Antioquia, n. 46, p. 110–118, 2008.

TERBUC, M. Use of free/open source software in e-education. In: *2006 12th International Power Electronics and Motion Control Conference*. [S.l.: s.n.], 2006. p. 1737–1742.

WANG, S. *et al.* Simulation of wireless image transmission system on multi-functional intelligent inspection vehicle. In: *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*. [S.l.: s.n.], 2019. p. 543–546.

WERNER, C. M. L.; BRAGA, R. M. M. Desenvolvimento baseado em componentes. *XIV Simpósio Brasileiro de Engenharia de Software SBES2000 Minicursos e Tutoriais–pg*, p. 297–329, 2000.

WŁODARSKI, R.; PONISZEWSKA-MARANDA, A. Applying a traditional software development process to drive projects in higher education. In: *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2019. p. 309–316.

YOUNG, D. Software development methodologies. *White paper*, 08 2013.

ZANELLA, L. C. H. *Metodologia da pesquisa*. [S.l.]: SEAD/UFSC, 2006.

ZHU, J.; LUBKEMAN, D. L. Object-oriented development of software systems for power system simulations. *IEEE Transactions on Power Systems*, v. 12, n. 2, p. 1002–1007, 1997.

ÁLVAREZ, G. A. R. Diagramabarra: Uma interface para ensino e estudo de sistemas elétricos. 2019.

Anexos

ANEXO A – Modelagem de Casos de Uso.

Tabela 3 – Modelagem do caso de uso 01.

Nome	Acessar a Base de Dados Geográficos da Distribuidora
Descrição literal	Esse caso de uso refere-se a possibilidade do usuário selecionar o local, nativo ou na nuvem, que contém os arquivos que compõe a base de dados utilizada para modelagem das redes de distribuição. O *script* do sistema acessará essas informações e fará a conversão para uma sintaxe executável pelo motor de cálculo do *software*.
Etapas	**Local no documento**
Descrição formal	Tabela 20
Diagrama de caso de uso UML	Figura 32
Diagrama de atividades UML	Figura 49

Tabela 4 – Modelagem do caso de uso 02.

Nome	Conectar a Base de Dados Geográficos da Distribuidora
Descrição literal	Após o usuário selecionar o diretório de trabalho da BDGD que deseja trabalhar, ele deve se conectar a esses dados de modo que o sistema consiga acessá-los para oferecer as opções de modelagem de redes convenientes. Ou seja, essa conexão permite que o sistema identifique, por exemplo, as subestações de alta tensão existentes na BDGD selecionada e a partir daí valide outros requisitos do sistema serem acionados.
Etapas	**Local no documento**
Descrição formal	Tabela 21
Diagrama de caso de uso UML	Figura 33
Diagrama de atividades UML	Figura 50

Tabela 5 – Modelagem do caso de uso 03.

Nome	Selecionar as características da rede que deve ser modelada
Descrição literal	Após o usuário já está conectado a BDGD escolhida deve ser permitido a ele escolher as características da rede que será modelada. Logo, deve ficar disponível para o usuário as subestações de alta tensão, os circuitos, as subestações de média tensão e os alimentadores que farão parte da rede modelada para o estudo. A medida que o usuário faz uma escolha, uma escolha subsequente é destravada até que todas as seleções de configuração de rede é realizada. Desse modo, o sistema colhe todas as informações necessária para construção do *script* de modelagem da rede.
Etapas	**Local no documento**
Descrição formal	Tabela 22
Diagrama de caso de uso UML	Figura 34
Diagrama de atividades UML	Figura 51

Tabela 6 – Modelagem do caso de uso 04.

Nome	Visualizar rede simulada
Descrição literal	A rede de distibuição é modelada a partir das informações colhidas na BDGD que são georreferênciadas. Desta forma, o *script* que realiza a modelagem deve utilizar recursos de programação para permitir a visualização da rede de distribuição modelada via o recurso *Open Street Maps*.
Etapas	**Local no documento**
Descrição formal	Tabela 23
Diagrama de caso de uso UML	Figura 35
Diagrama de atividades UML	Figura 52

Tabela 7 – Modelagem do caso de uso 05.

Nome	Configurar cor de alimentadores selecionados.
Descrição literal	Como a rede será visualizada utilizando a ferramenta do *Open Street Maps* é interessante que o usuário possa selecionar a cor dos alimentadores que serão estudados, isso ajudará na experiência do usuário com a utilização do *software*. Sendo assim, existe duas possibilidades: ou as cores são geradas automáticamente, ou o usuário ao selecionar o alimentador indica a cor associada, de modo que o *script* do sistema colha essa informação e use-a objetivamente.
Etapas	**Local no documento**
Descrição formal	Tabela 24
Diagrama de caso de uso UML	Figura 36
Diagrama de atividades UML	Figura 53

Tabela 8 – Modelagem do caso de uso 06.

Nome	Visualizar equipamentos Transformadores da rede selecionada.
Descrição literal	Esse caso de uso refere-se a possibilidade do usuário visualizar os transformadores de distribuição presentes na rede modelada, visto que as informações referente aos transformadores contidas na BDGD são georreferênciadas. Desse modo, o usuário poderá ter mais condições de análise do sistema em seus estudos, pois a visualização dos transformadores devem indicar sua identificação e algumas de suas características técnicas.
Etapas	**Local no documento**
Descrição formal	Tabela 25
Diagrama de caso de uso UML	Figura 37
Diagrama de atividades UML	Figura 54

Tabela 9 – Modelagem do caso de uso 07.

Nome	Visualizar equipamentos Compensadores da rede selecionada.
Descrição literal	Esse caso de uso refere-se a possibilidade do usuário visualizar os compensadores de distribuição presentes na rede modelada, visto que as informações referente aos transformadores contidas na BDGD são referenciadas.
Etapas	**Local no documento**
Descrição formal	Tabela 26
Diagrama de caso de uso UML	Figura 38
Diagrama de atividades UML	Figura 55

Tabela 10 – Modelagem do caso de uso 08.

Nome	Configurar fluxo de potência.
Descrição literal	Para estudar fluxo de potência o usuário deve configurar inicialmente algumas opções, como por exemplo o modo de operação que pode ser *daily*, *snapshot* ou *direct*. Caso selecione *daily* o usuário deve selecionar curvas de cargas para serem associadas a cada consumidor, bem como determinar os intervalos correspondentes. Além disso, o usuário deve indicar quais as tensões de base que devem ser consideradas para à apresentação de resultados em valores por unidade.
Etapas	**Local no documento**
Descrição formal	Tabela 27
Diagrama de caso de uso UML	Figura 39
Diagrama de atividades UML	Figura 56

Tabela 11 – Modelagem do caso de uso 09.

Nome	Configurar Curva de Carga.
Descrição literal	Como dito, o modo *daily* permite que o usuário realize o estudo do fluxo de potência de forma diária. Para esse modo é necessário que o usuário selecione as curvas de carga associadas a cada consumidor pertencente a rede escolhida, essas curvas podem ser *default* ou inseridas pelo usuário.
Etapas	**Local no documento**
Descrição formal	Tabela 28
Diagrama de caso de uso UML	Figura 40
Diagrama de atividades UML	Figura 57

Tabela 12 – Modelagem do caso de uso 10.

Nome	Executar fluxo de potência.
Descrição literal	O modo *daily* permite que o usuário realize estudo do fluxo de potência de forma diária. Para esse modo é necessário que o usuário selecione as curvas de carga associadas a cada consumidor pertencente a rede escolhida, essas curvas podem ser *default* ou inseridas pelo usuário. Basicamente, o sistema deve permitir que o usuário visualize os perfis de curva e os habilite a ser usado pelo *script* que modela a rede de distribuição que será estudada.
Etapas	**Local no documento**
Descrição formal	Tabela 29
Diagrama de caso de uso UML	Figura 41
Diagrama de atividades UML	Figura 58

Tabela 13 – Modelagem do caso de uso 11.

Nome	Gerar arquivo .dss
Descrição literal	A rede modelada pelo SIPLA pode ser utilizada pelo *software* OpenDSS. Desse modo, é conveniente permitir que o usuário tenha acesso ao script interno de conversão de informações BDGD para o formato openDSS e isso é realizado por meio desse requisito.
Etapas	**Local no documento**
Descrição formal	Tabela 30
Diagrama de caso de uso UML	Figura 42
Diagrama de atividades UML	Figura 59

Tabela 14 – Modelagem do caso de uso 12.

Nome	Salvar arquivo .dss
Descrição literal	Esse caso de uso tem relação com o caso de uso exposto anteriormente e realiza o salvamento do *script* gerado.
Etapas	**Local no documento**
Descrição formal	Tabela 31
Diagrama de caso de uso UML	Figura 43
Diagrama de atividades UML	Figura 60

Tabela 15 – Modelagem do caso de uso 13.

Nome	Inserir Medidores de Energia
Descrição literal	Medidores de energia armazenam informações das grandezas elétricas associadas ao ponto de conexão ou as zonas de interesse, devem ser inseridos em elementos pertencentes as redes modeladas. As informações que compõe sua especificidade técnica são dadas pelo usuário e colhidas pelo sistema para construção do *script* responsável por sua inserção.
Etapas	**Local no documento**
Descrição formal	Tabela 32
Diagrama de caso de uso UML	Figura 44
Diagrama de atividades UML	Figura 61

Tabela 16 – Modelagem do caso de uso 14.

Nome	Inserir Monitores
Descrição literal	Os monitores de energia são equipamentos que monitoram as grandezas elétricas ou comportamentos específicos de determinados equipamentos presentes na rede modelada, eles podem ser instalados em elementos do sistema analisado.
Etapas	**Local no documento**
Descrição formal	Tabela 33
Diagrama de caso de uso UML	Figura 45
Diagrama de atividades UML	Figura 62

Tabela 17 – Modelagem do caso de uso 15.

Nome	Visualizar dos Resultados Registrados pelos Medidores
Descrição literal	Os medidores de energia registram valores, porém é necessário visualiza-los de alguma forma e isso é feito pela *interface* visualizar resultados registrados.
Etapas	**Local no documento**
Descrição formal	Tabela 34
Diagrama de caso de uso UML	Figura 46
Diagrama de atividades UML	Figura 63

Tabela 18 – Modelagem do caso de uso 16.

Nome	Plotar gráfica.
Descrição literal	É possível visualizar as grandezas elétricas de forma gráfica. Esse fato é importantíssimo para as análises gerais dos estudos realizados através do SIPLA.
Etapas	**Local no documento**
Descrição formal	Tabela 35
Diagrama de caso de uso UML	Figura 47
Diagrama de atividades UML	Figura 64

Tabela 19 – Modelagem do caso de uso 17.

Nome	Visualizar de grandezas elétricas simuladas pós fluxo de potência
Descrição literal	É possível visualizar as grandezas elétricas no modo *daily* de forma gráfica no decorrer do tempo e isso depende de onde o usuário instalou equipamentos monitores.
Etapas	**Local no documento**
Descrição formal	Tabela 36
Diagrama de caso de uso UML	Figura 48
Diagrama de atividades UML	Figura 65

ANEXO B – Casos de Uso Estruturados.

Tabela 20 – Caso de uso 01: Acessar a Base de Dados Geográficos da Distribuidora.

CU001	**Acessar a Base de Dados Geográficos da Distribuidora**
Referência	Acessar a Base de Dados Geográficos da Distribuidora.RF001
Sumário	O caso de uso é responsável por acessar os dados pertencentes ao BDGD
Pré-condições	A pasta BDGD deve esta armazenada no diretório de trabalho com os arquivos das bases de dados pertinentes devidamente baixados.
Atores	Usuário do *software*.
Descrição	1 - O *software* se inicia abrindo sua interface principal. 2 - O usuário seleciona no menu o item configuração. 3 - O usuário seleciona no menu o subitem configurar o BDGD. 4 - Abre-se uma janela do tipo *dialog* para configuração e escolha da base de dados. 5 - O usuário seleciona a base de dados. 6 - O usuário confirma seleção clicando em um botão ok.
Alternativas	1 - Caso o usuário queira salvar o diretório basta clicar num botão salvar configuração.
Exceção	Se o usuário selecionar o diretório que não possui os arquivos da BDGD deve-se mostrar uma mensagem de exceção

Tabela 21 – Caso de uso 02: Conectar a Base de Dados Geográficos da Distribuidora

CU002	**Conectar a Base de Dados Geográficos da Distribuidora**
Referência	Conectar a Base de Dados Geográficos da Distribuidora.RF002
Sumário	O caso de uso é responsável por conectar o sistema a BDGD selecionada.
Pré-condições	A seleção da BDGD já deve sido executada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item configuração. 2 - O usuário seleciona no menu o subitem conectar ao BDGD. 3 - O *software* exibe as subestações de alta tensão disponíveis no BDGD selecionado no local indicado.
Alternativas	
Exceção	Se o usuário não tiver selecionado a BDGD deve-se mostrar uma mensagem de exceção.

Tabela 22 – Caso de uso 03: Selecionar as características da rede que deve ser modelada

CU003	**Selecionar as características da rede que deve ser modelada**
Referência	Selecionar as características da rede que deve ser modelada.RF002
Sumário	O caso de uso é responsável por permitir que o usuário configure a rede que deseja simular.
Pré-condições	A BDGD já deve ter sido selecionada e conectada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona a subestação de alta tensão desejada. 2 - O usuário confirma seleção clicando em um botão ok. 3 - O *software* mostra todas os circuitos disponíveis associados a subestação de alta tensão selecionada anteriormente. 4 - O usuário seleciona o circuito e confirma seleção clicando em um botão ok. 5 - O *software* mostra a subestação de média tensão associada ao circuito selecionado. 6 - O usuário seleciona subestação de média e confirma seleção clicando em um botão ok. 7 - O *software* mostra todos os alimentadores associados a subestação de média selecionada. 8 - Fica disponível os alimentadores para seleção do usuário.
Alternativas	1 - Caso o usuário queira modificar qualquer etapa de seleção as opções subsequentes devem ser atualizadas.
Exceção	Caso o usuário não esteja conectado a BDGD os botões de seleções devem permanecer desativados.

Tabela 23 – Caso de uso 04: Visualizar rede modelada.

CU004	**Visualizar rede simulada.**
Referência	Visualizar rede simulada.RF004
Sumário	O caso de uso é responsável por permitir que o usuário visualize a rede que foi modelada.
Pré-condições	A BDGD já deve ter sido selecionada e conectada.O usuário já deve ter selecionado a subestação de alta tensão, o circuito de alta para média e a subestação de média tensão.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona os alimentadores que deseja visualizar. 2 - O usuário confirma a visualização clicando num botão visualizar rede.
Alternativas	1 - Opção para o usuário selecionar todos os alimentadores disponíveis.
Exceção	Essa função não deve está disponível caso as pré condições não sejam atendidas, neste caso o botão deve está desativado.

Tabela 24 – Caso de uso 05: Configurar cor de alimentadores selecionados.

CU005	**Configurar cor de alimentadores selecionados.**
Referência	Configurar cor de alimentadores selecionados.RF005
Sumário	O caso de uso é responsável por permitir que o usuário selecione a cor de cada alimentador selecionado para visualização.
Pré-condições	A BDGD já deve ter sido selecionada e conectada. O usuário já deve ter selecionado a subestação de alta tensão, o circuito de alta para média e a subestação de média tensão.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona os alimentadores que deseja visualizar. 2 - O usuário seleciona a cor do alimentador clicando num botão.
Alternativas	1 - Caso o usuário não selecione uma cor especifica o *software* deve selecionar uma cor aleatória.
Exceção	Não há.

Tabela 25 – Caso de uso 06: Visualizar equipamentos Transformadores da rede selecionada.

CU006	**Visualizar equipamentos Transformadores da rede selecionada.**
Referência	Visualizar equipamentos Transformadores da rede selecionada.RF006
Sumário	O caso de uso é responsável por permitir que o usuário visualize os transformadores de distribuição associados aos alimentadores selecionados.
Pré-condições	A BDGD já deve ter sido selecionada e conectada. O usuário já deve ter selecionado a subestação de alta tensão, o circuito de alta para média, a subestação de média tensão e os alimentadores desejados. A ação visualizar rede já deve ter sido executada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona se deseja visualizar os transformadores.
Alternativas	1 - O *software* deve mostrar a identificação de cada transformador.
Exceção	Não há.

Tabela 26 – Caso de uso 07: Visualizar equipamentos Compensadores da rede selecionada.

CU007	**Visualizar equipamentos Compensadores da rede selecionada.**
Referência	Visualizar equipamentos Compensadores da rede selecionada.RF007
Sumário	O caso de uso é responsável por permitir que o usuário visualize os compensadores associados aos alimentadores selecionados.
Pré-condições	A BDGD já deve ter sido selecionada e conectada. O usuário já deve ter selecionado a subestação de alta tensão, o circuito de alta para média, a subestação de média tensão e os alimentadores desejados. A ação visualizar rede já deve ter sido executada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona se deseja visualizar os compensadores.
Alternativas	1 - O *software* deve mostrar a identificação de cada compensador.
Exceção	Não há.

Tabela 27 – Caso de uso 08: Configurar fluxo de potência.

CU008	**Configurar fluxo de potência.**
Referência	Configurar fluxo de potência.RF008
Sumário	O caso de uso é responsável por permitir que o usuário configure o modo de simulação que ele deseja realizar o fluxo de potência.
Pré-condições	Não há.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário seleciona no menu OpenDSS o subitem Configurar. 3 - O usuário seleciona o modo de simulação. 4 - O usuário configura os parâmetros associado ao modo escolhido. 5 - O usuário confirma a configuração clicando num botão.
Alternativas	1 - O usuário deve selecionar qual motor de calculo o *software* deve operar.
Exceção	Não há.

Tabela 28 – Caso de uso 09: Configurar Curva de Carga.

CU009	**Configurar Curva de Carga.**
Referência	Configurar Curva de Carga.RF009
Sumário	O caso de uso é responsável por permitir que o usuário selecione qual curva de carga será utilizada no modo *Daily*.
Pré-condições	O modo selecionado deve ser o modo solução *Daily*.
Atores	Usuário do *software*.
Descrição	1 - O usuário clica num botão de acesso a uma *dialog loadshap*. 2 - Uma nova janela é aberta. 3 - O usuário pode adicionar, importar, remover ou exportar curvas de carga. 4 - O usuário seleciona as curvas para serem inseridas. 5 - O usuário confirma seleção.
Alternativas	1 - O usuário pode selecionar todas as curvas importadas. 2 - O usuário pode visualizar o perfil das curvas selecionada.
Exceção	Não há.

Tabela 29 – Caso de uso 10: Executar fluxo de potência.

CU010	**Executar fluxo de potência.**
Referência	Executar fluxo de potência.RF010
Sumário	O caso de uso é responsável por permitir que o usuário rode o fluxo de potência configurado.
Pré-condições	O modo selecionado deve ser o modo solução *Daily*.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu Execute.
Alternativas	1 - O usuário pode selecionar o Execute pelo atalho F3.
Exceção	A rede deve ter sido modelada e configurada.

Tabela 30 – Caso de uso 11: Gerar arquivo .dss.

CU011	**Gerar arquivo .dss.**
Referência	Gerar arquivo .dss.RF011
Sumário	O caso de uso é responsável por gerar os arquivos no formato .dss da rede modelada.
Pré-condições	Ter configurado a rede e ter rodado o fluxo de potência.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu Sub-process. 3 - O usuário aciona a opção gerar .dss.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada e o fluxo de potência rodado.

Tabela 31 – Caso de uso 12: Salvar arquivo .dss.

CU012	**Salvar arquivo .dss.**
Referência	Salvar arquivo .dss.RF012
Sumário	O caso de uso é responsável por salvar os arquivos no formato .dss da rede modelada.
Pré-condições	Ter configurado a rede, ter rodado o fluxo de potência e ter gerado o arquivo .dss.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu Sub-process. 3 - O usuário aciona a opção gerar .dss. 4 - O usuário aciona a opção salvar .dss.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada e o fluxo de potência rodado.

Tabela 32 – Caso de uso 13: Inserir Medidores de Energia.

CU013	**Inserir Medidores de Energia.**
Referência	Inserção de Medidores de Energia.RF013
Sumário	O caso de uso é responsável por inserir medidores de energia na rede modelada.
Pré-condições	Ter configurado a rede, ter rodado o fluxo de potência ou ter visualizado a rede modelada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu inserir. 3 - O usuário aciona a opção inserir medidores.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada, o fluxo de potência rodado ou rede visualizada, caso contrário o item deve estar desabilitado .

Tabela 33 – Caso de uso 14: Inserir Monitores.

CU014	**Inserir Monitores.**
Referência	Inserção de Monitores.RF014
Sumário	O caso de uso é responsável por inserir monitores na rede modelada.
Pré-condições	Ter configurado a rede, ter rodado o fluxo de potência ou ter visualizado a rede modelada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu inserir. 3 - O usuário aciona a opção inserir monitores.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada, o fluxo de potência rodado ou rede visualizada, caso contrário o item deve estar desabilitado .

Tabela 34 – Caso de uso 15: Visualizar Resultados Registrados pelos Medidores.

CU015	**Visualizar dos Resultados Registrados pelos Medidores.**
Referência	Visualização dos Resultados Registrados pelos Medidores.RF015
Sumário	O caso de uso é responsável por visualizar os resultados registrados pelos medidores.
Pré-condições	Ter configurado a rede, ter rodado o fluxo de potência ou ter visualizado a rede modelada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item OpenDSS. 2 - O usuário aciona o submenu inserir. 3 - O usuário aciona a opção inserir monitores.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada, o fluxo de potência rodado ou rede visualizada, caso contrário o item deve estar desabilitado.

Tabela 35 – Caso de uso 16: Plotar Gráfico.

CU016	**Plotar Gráfica.**
Referência	Plotagem Gráfica.RF016
Sumário	O caso de uso é responsável por permitir que o usuário visualize as grandezas monitoradas de forma gráfica.
Pré-condições	Ter configurado a rede, ter rodado o fluxo de potência ou ter visualizado a rede modelada.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item Plot. 2 - O usuário aciona o submenu gráfico dos monitores.
Alternativas	Não há.
Exceção	A rede deve ter sido modelada, configurada, o fluxo de potência rodado ou rede visualizada, caso contrário o item deve estar desabilitado .

Tabela 36 – Caso de uso 17: Visualizar grandezas elétricas simuladas pós fluxo de potência.

CU017	**Visualizar de grandezas elétricas simuladas pós fluxo de potência.**
Referência	Visualização de grandezas elétricas simuladas pós fluxo de potência.RF017
Sumário	O caso de uso é responsável por permitir que o usuário visualize as grandezas pós fluxo de forma tabelada.
Pré-condições	Ter configurado a rede e ter rodado o fluxo de potência.
Atores	Usuário do *software*.
Descrição	1 - O usuário seleciona no menu o item Janela. 2 - O usuário aciona o submenu Visualizar resultados em tabela.
Alternativas	Não há.
Exceção	Não há.

ANEXO C – Diagramas de Caso de Uso

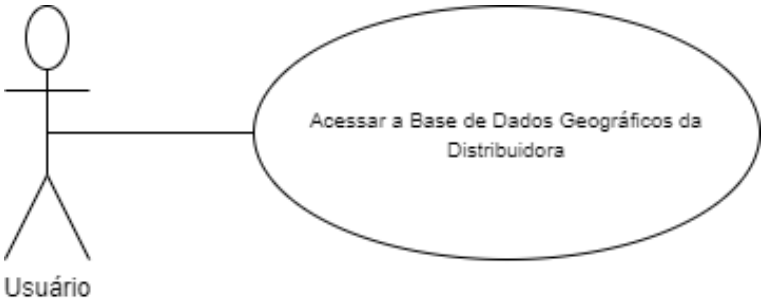


Figura 32 – Diagrama de Caso de uso 01 - UML

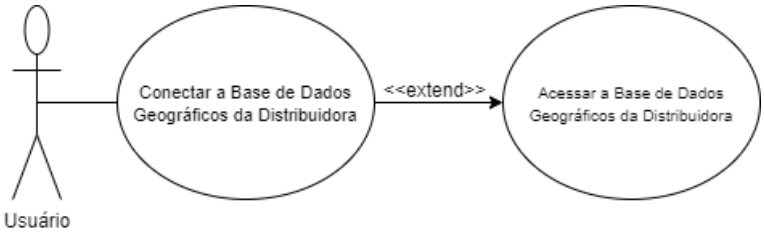


Figura 33 – Diagrama de Caso de uso 02 - UML

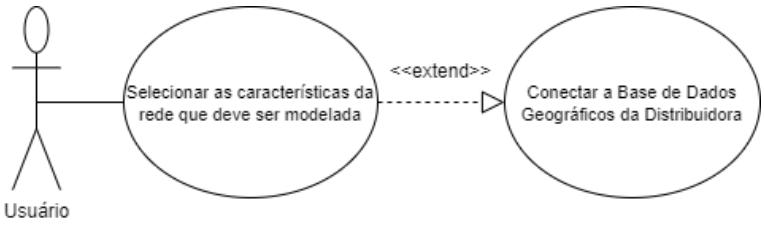


Figura 34 – Diagrama de Caso de uso 03 - UML

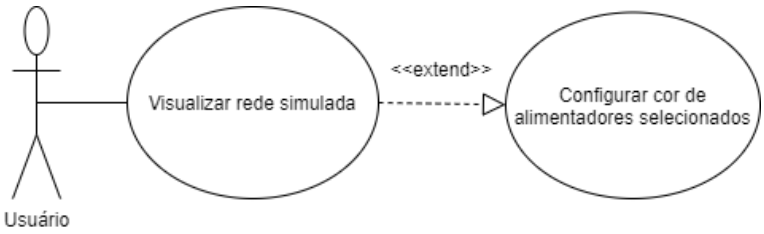


Figura 35 – Diagrama de Caso de uso 04 - UML

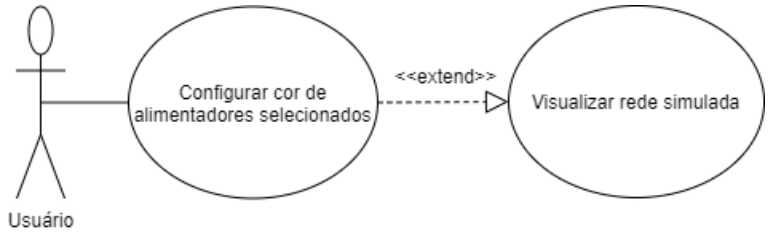


Figura 36 – Diagrama de Caso de uso 05 - UML

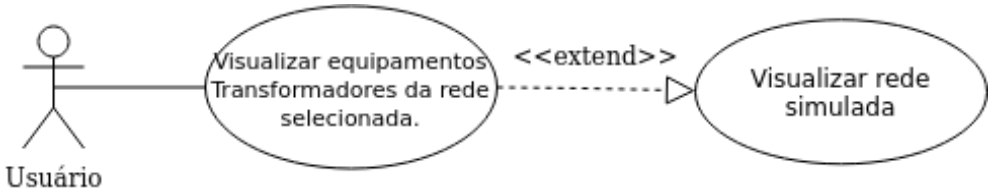


Figura 37 – Diagrama de Caso de uso 06 - UML

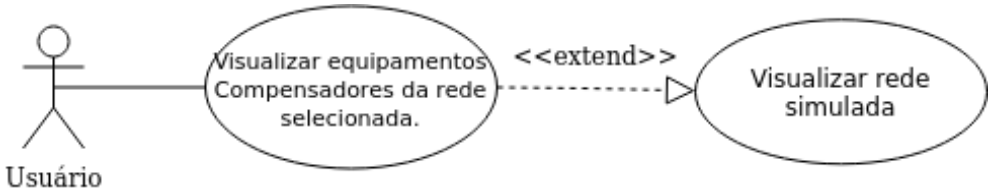


Figura 38 – Diagrama de Caso de uso 07 - UML

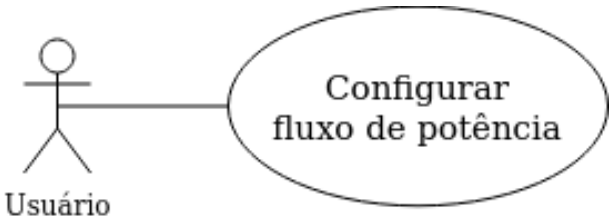


Figura 39 – Diagrama de Caso de uso 08 - UML

Figura 40 – Diagrama de Caso de uso 09 - UML

Figura 41 – Diagrama de Caso de uso 10 - UML

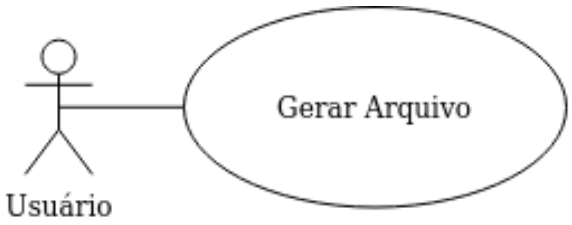


Figura 42 – Diagrama de Caso de uso 11 - UML

Figura 43 – Diagrama de Caso de uso 12 - UML

Figura 44 – Diagrama de Caso de uso 13 - UML

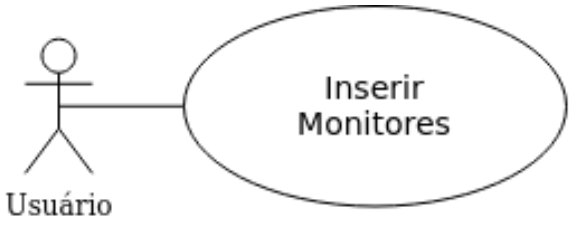


Figura 45 – Diagrama de Caso de uso 14 - UML

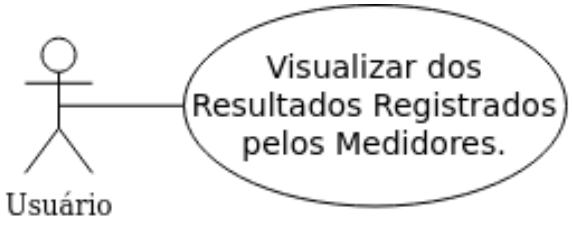


Figura 46 – Diagrama de Caso de uso 15 - UML

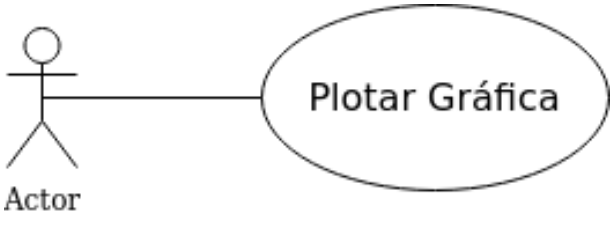


Figura 47 – Diagrama de Caso de uso 16 - UML

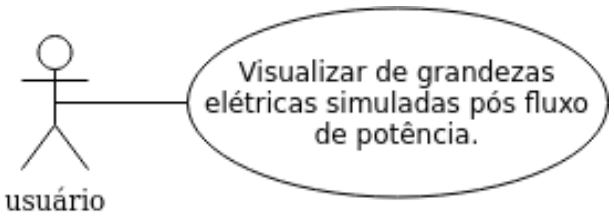


Figura 48 – Diagrama de Caso de uso 17 - UML

ANEXO D – Diagramas de Atividades

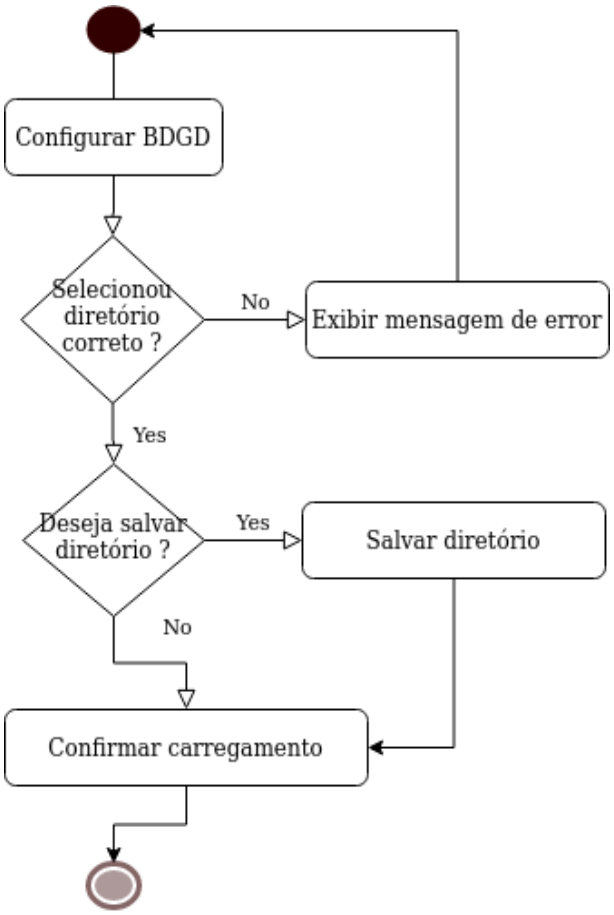


Figura 49 – Diagrama de Atividade 01 - UML

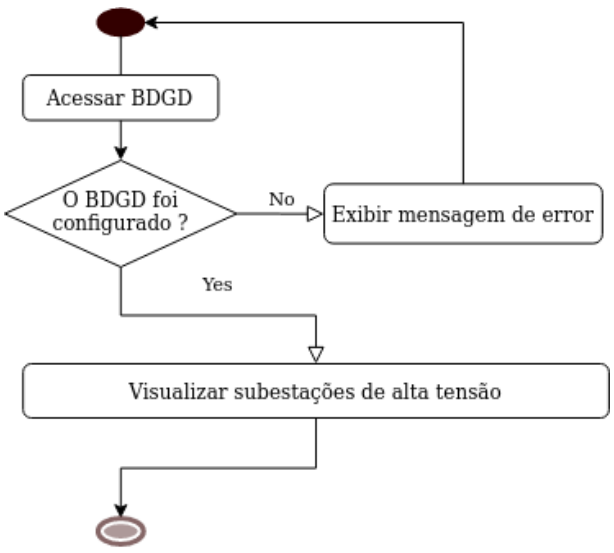


Figura 50 – Diagrama de atividade 02 - UML

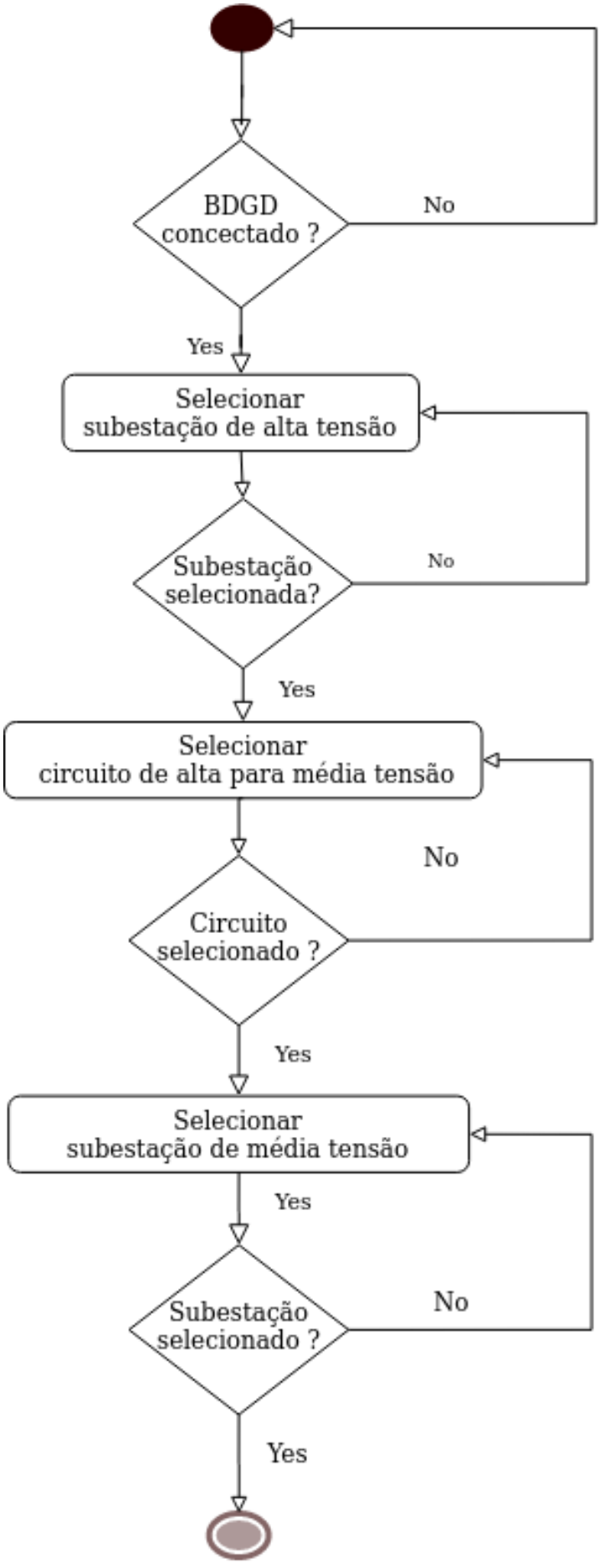


Figura 51 – Diagrama de atividade 03 - UML

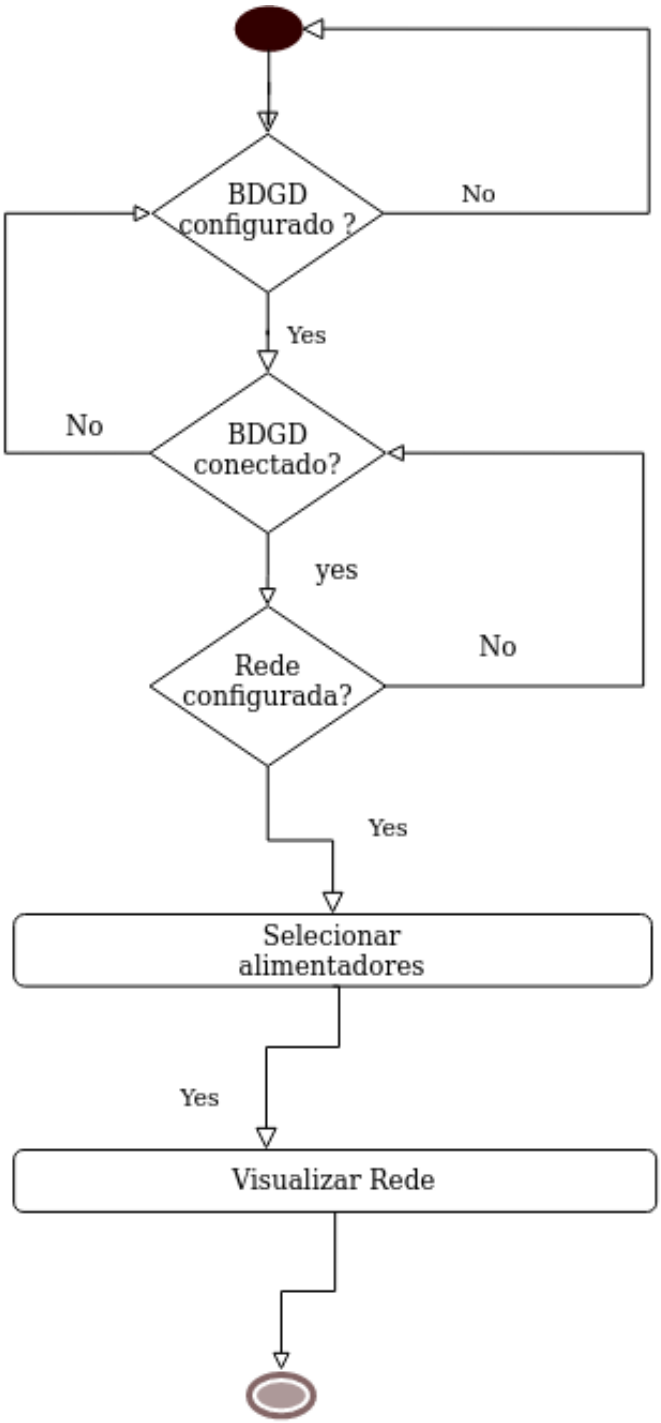


Figura 52 – Diagrama de atividade 04 - UML

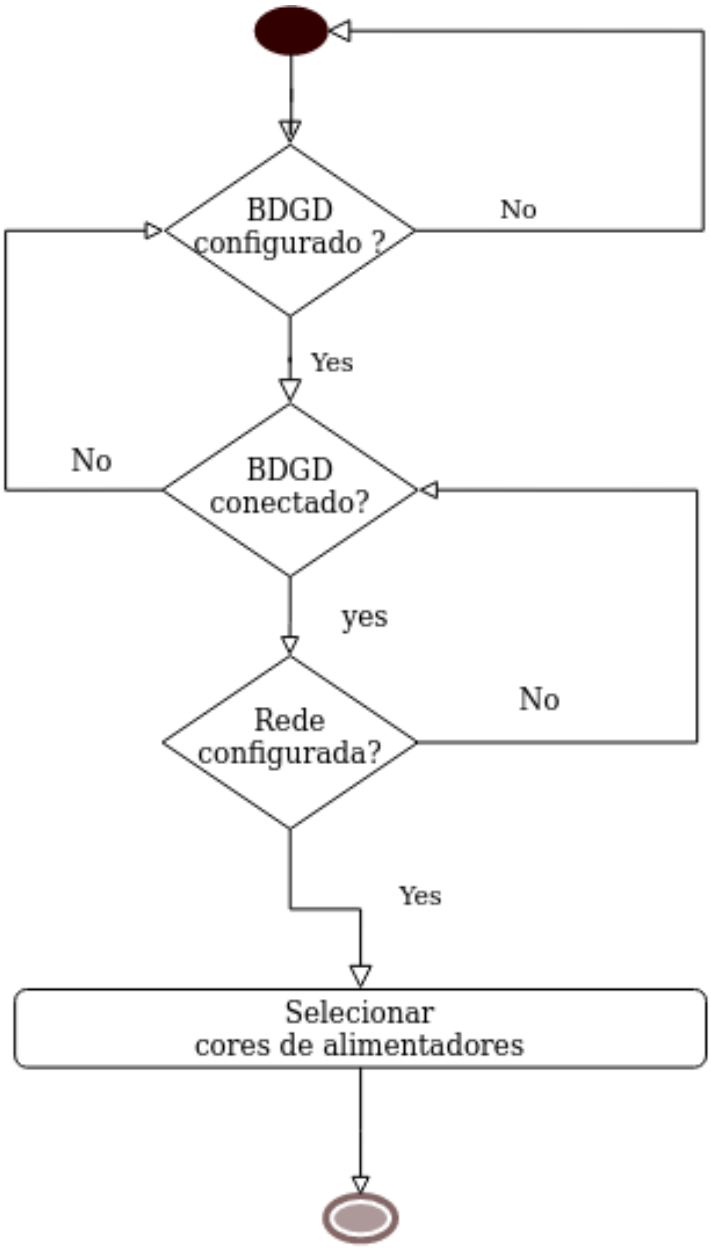


Figura 53 – Diagrama de atividade 05 - UML

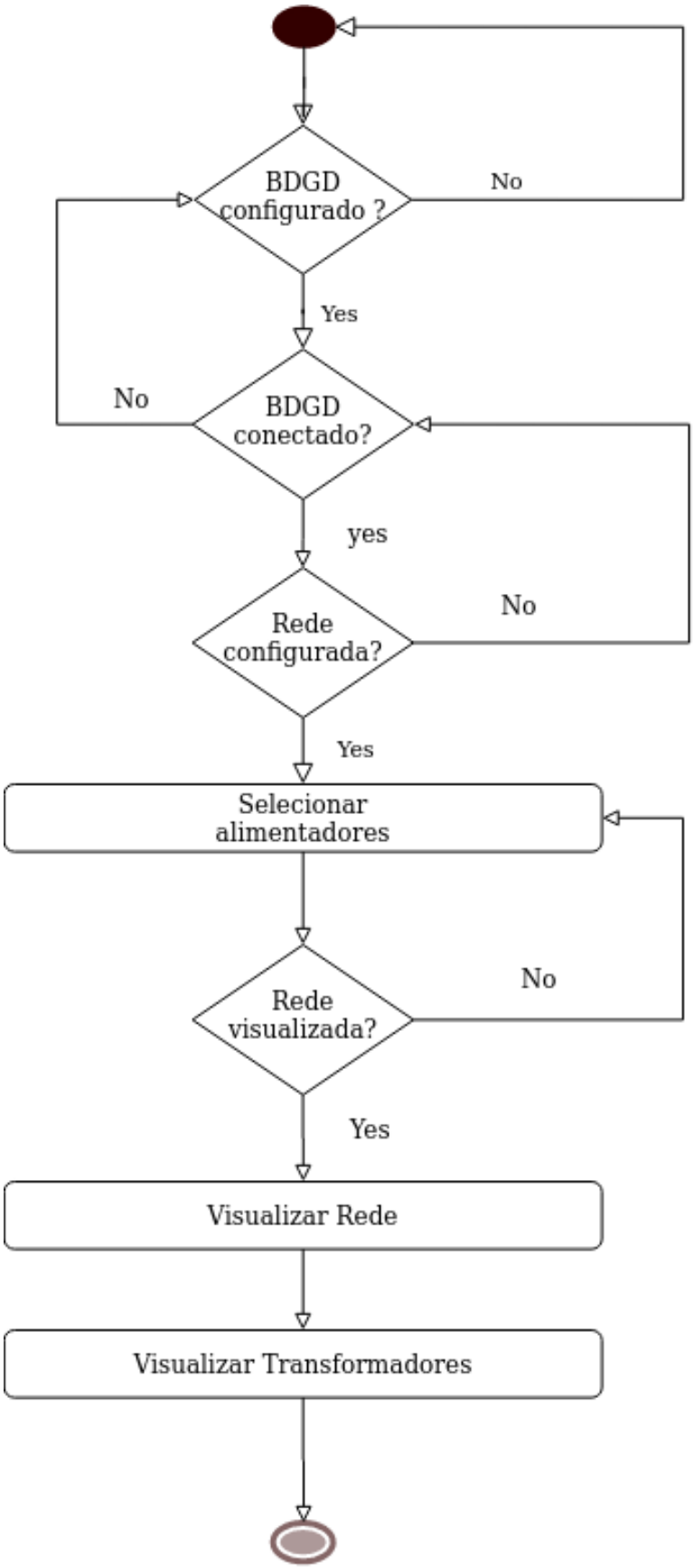


Figura 54 – Diagrama de atividade 06 - UML

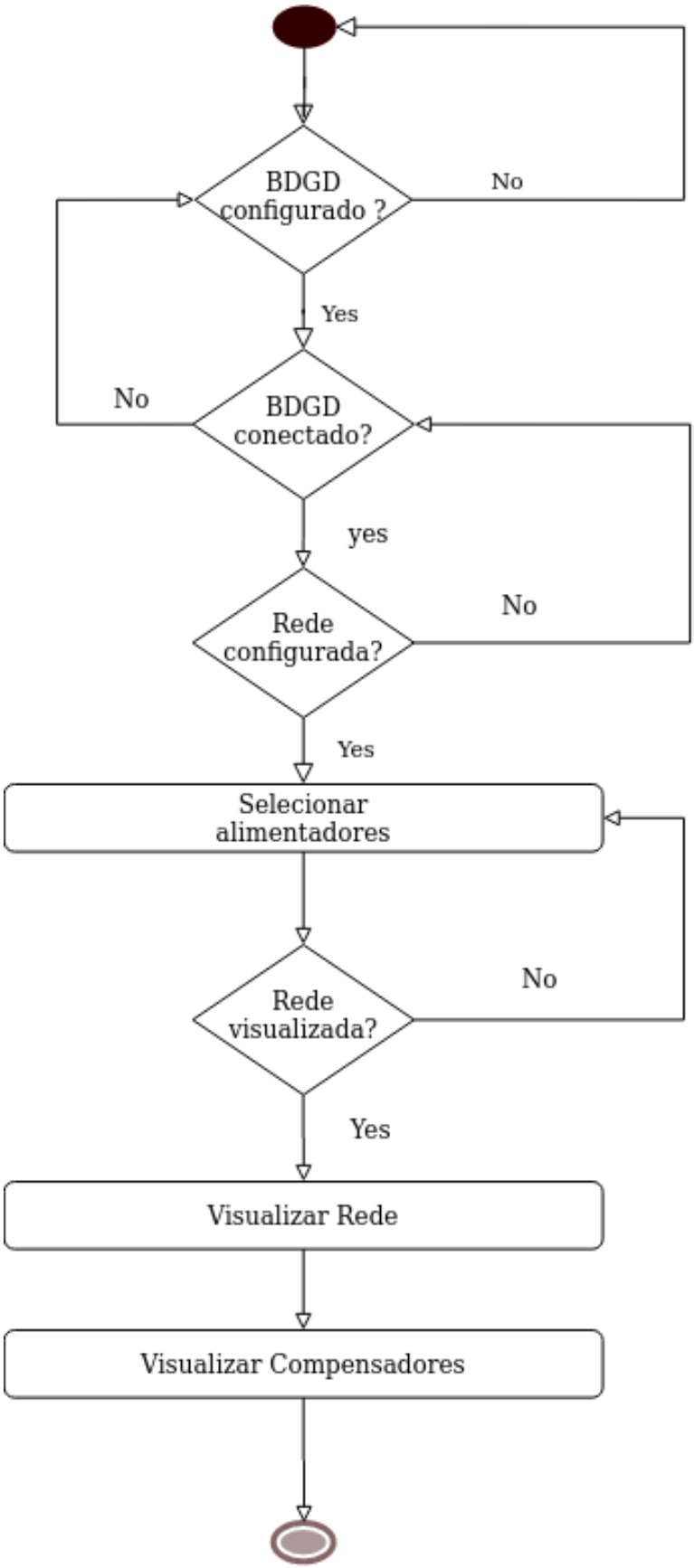


Figura 55 – Diagrama de atividade 07 - UML

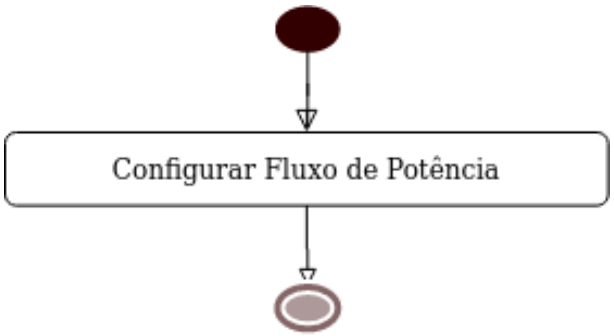


Figura 56 – Diagrama de atividade 08 - UML

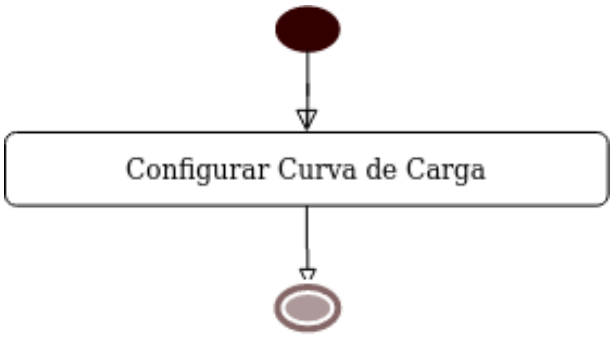


Figura 57 – Diagrama de atividade 09 - UML

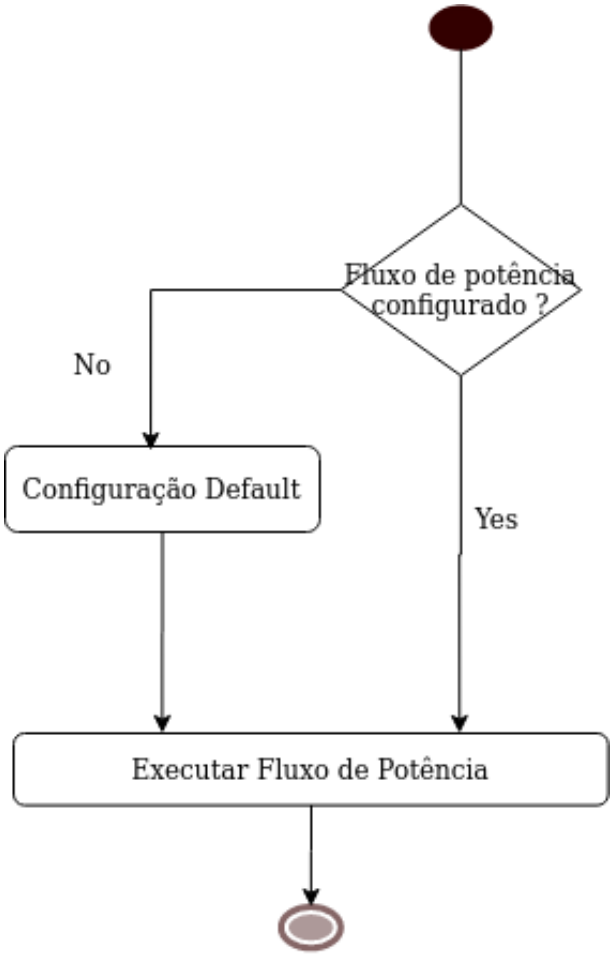


Figura 58 – Diagrama de atividade 10 - UML

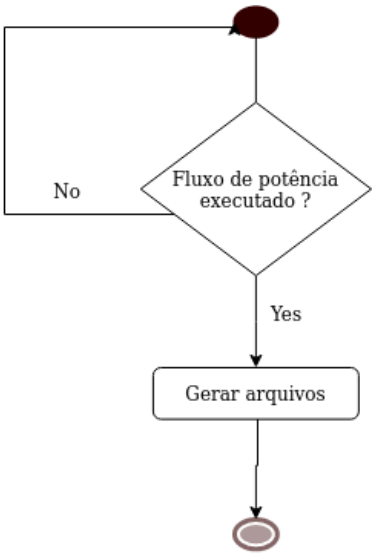


Figura 59 – Diagrama de atividade 11 - UML

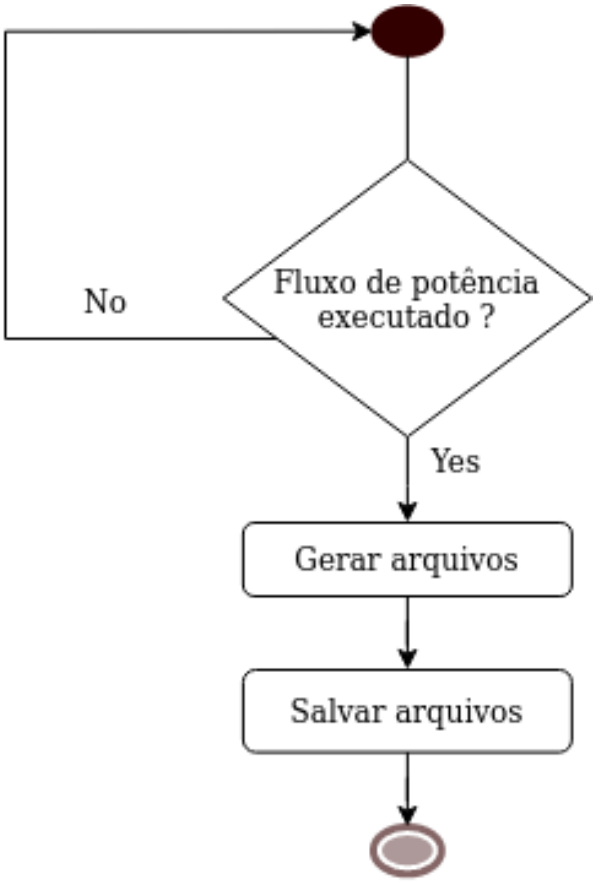


Figura 60 – Diagrama de atividade 12 - UML

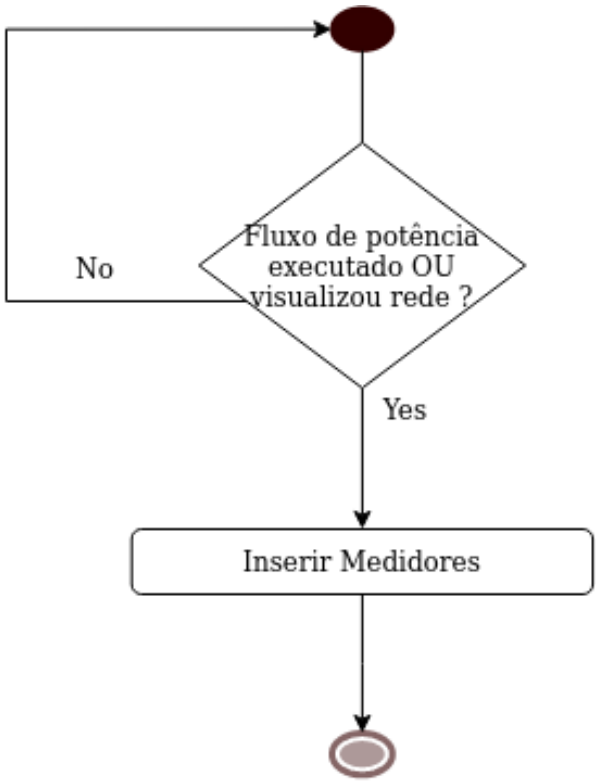


Figura 61 – Diagrama de atividade 13 - UML

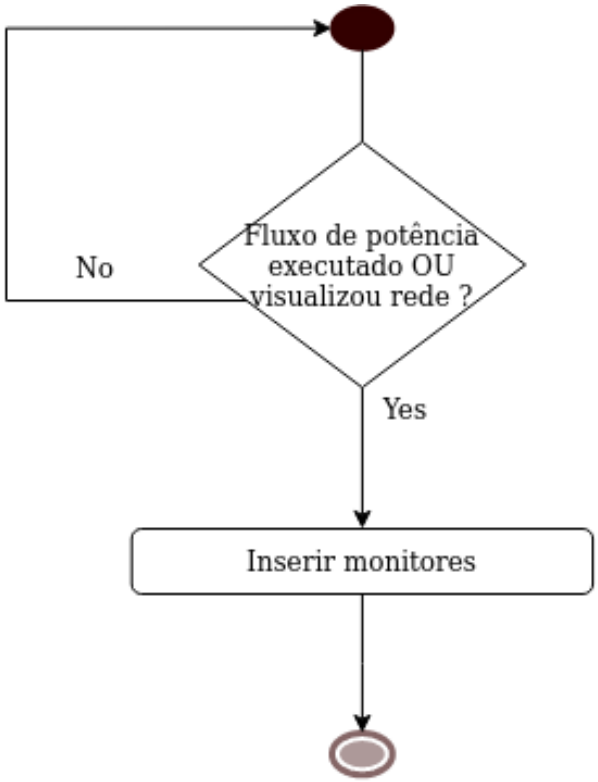


Figura 62 – Diagrama de atividade 14 - UML

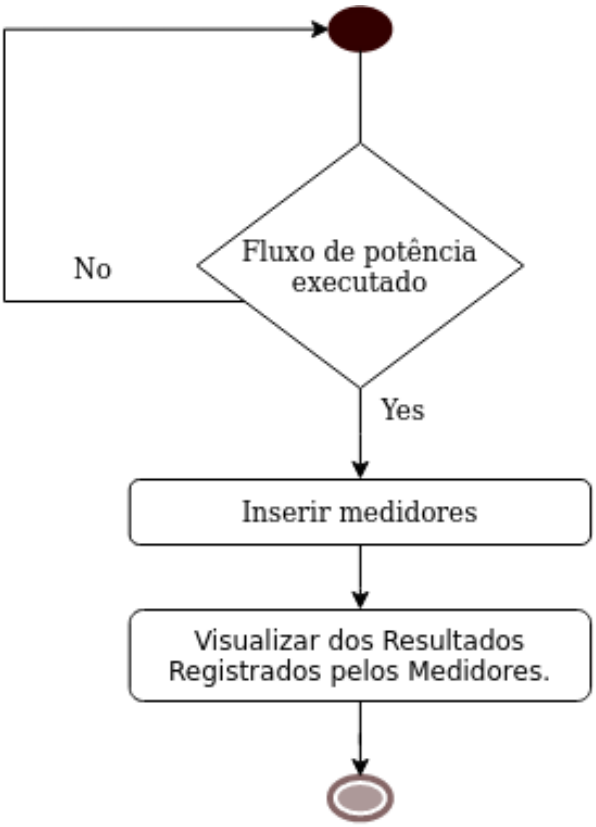


Figura 63 – Diagrama de atividade 15 - UML

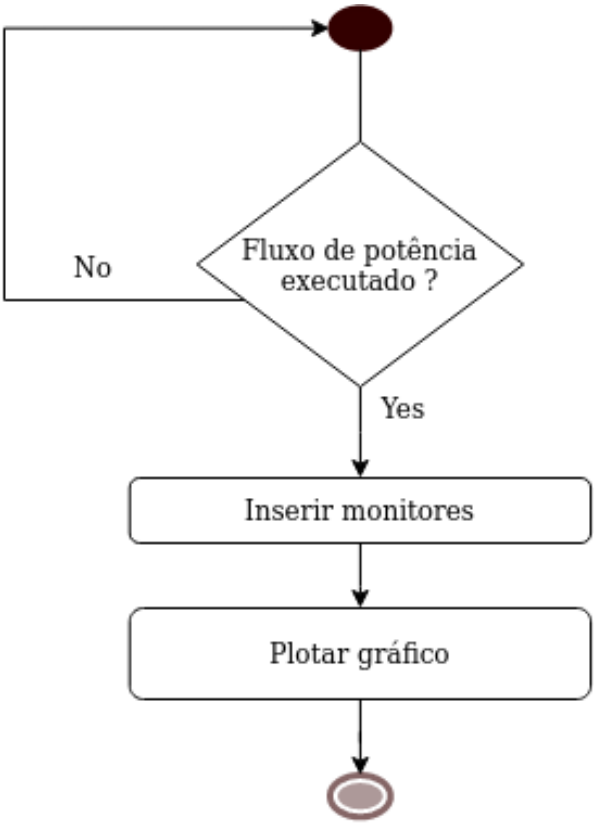


Figura 64 – Diagrama de atividade 16 - UML

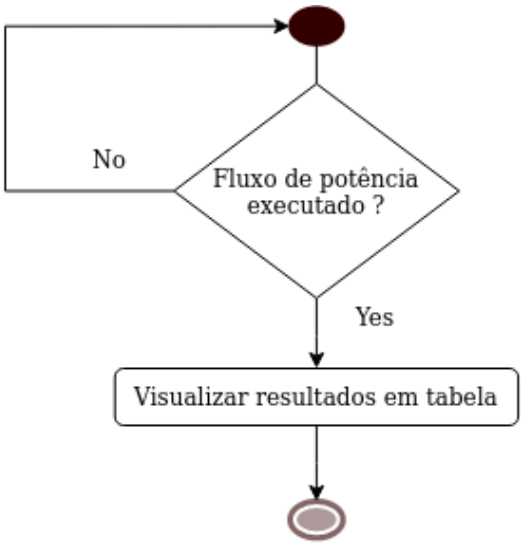


Figura 65 – Diagrama de atividade 17 - UML

ANEXO E – Cartões Classe-Responsabilidade-Colaborador

Tabela 37 – CRC: classe CMainWindow.

Classe	**CMainWindow.**
Descrição	Classe responsável por estruturar a interface principal.
Responsabilidade	**Colaborador**
Definir o nome do *software*	
Definir o ícone do *software*	
Definir o tamanho da janela	
Definir o *style* da janela	
Incorporar o *menu* do *software*	MenuToolBar
Incorporar a *toolbar* do *software*	MenuToolBar
Incorporar a barra de *status* do *software*	MenuToolBar
Incorporar o painel central do *software*	MainPanel
Incorporar o DockResults do *software*	DockResults
Mostrar o Mapa do *streetmaps*	Viewer

Tabela 38 – CRC: classe CMenuToolBar.

Classe	CMenuToolBar
Descrição	Classe responsável por estruturar a barra de ferramentas do *software*.
Responsabilidade	**Colaborador**
Carrega a Base de Dados	CMainActions
Conecta a Base de Dados	CMainActions
Carrega rede a ser simulada	CMainActions
Configura simulação	CMainActions
Visualiza rede modelada	CMainActions
Executa fluxo de potência	CMainActions
Gera arquivo com rede modelada	CMainActions
Salva arquivo gerado com rede modelada	CMainActions
Inserir medidores de energia	CMainActions
Inserir monitores	CMainActions
Visualiza Resultados gerados	CMainActions

Tabela 39 – CRC: classe CMainPanel.

Classe	CMainPanel
Descrição	Classe responsável pelo *layout* principal da GUI.
Responsabilidade	**Colaborador**
Criar a caixa do *centralwidget* do visualizador da rede	

Tabela 40 – CRC: classe CNetPanel.

Classe	CNetPanel
Descrição	Classe responsável pelo complemento/suporte do layout principal da GUI.
Responsabilidade	**Colaborador**
Criar o *layout* que abriga a seleção das informações nescessárias para construção da rede a ser estudada	
Criar o *layout* que abriga os resultados em forma de tabela	

Tabela 41 – CRC: classe CStatusBar.

Classe	*CStatusBar*
Descrição	Classe responsável pela configuração da barra de *status do software*.
Responsabilidade	**Colaborador**
Indicar o *status* da conexão com a Base de Dados - *online ou offline*	
Indicar o *status* do carregamento da Base de Dados	
Indicar o *status* do tipo do fluxo de potência	
Indicar o *status* da solução do fluxo de potência	
Indicar o *status* do sistema operacional	

Tabela 42 – CRC: classe *MainActions*.

Classe	*CMainActions*
Descrição	Classe responsável pela gerência das ações do *software*.
Responsabilidade	**Colaborador**
Atualiza o *Status* por meio do *updateStatusBar*	*CDBaseConn*
Atualiza o *Menu* por meio *updateToobarMenu*	*MainWindowToolBar*
Conectar com o banco de dados	*CDBaseConn*
Configurar inicialmente o sistema	*CConfigDialog*, *CNetPanel*
Acionar/mostrar a subestação de alta tensão	*CNetPanel*
Acionar/mostrar alimentadores de distribuição	*MainActions*
Mostrar informações sobre o *software*	*CAboutDialog*
Acionar visualização do Mapa	*CViewer*
Gerar arquivo .dss	*COpenDSS*
Aciona a janela de inserir medidores	CInsertEnergyMeterDialog
Aciona a janela de inserir monitores	*CInsertMonitorDialog*
Aciona a janela de plotar monitores	*CInsertMonitorDialog*
Acionar o carregamento das informações que compõe .dss *COpenDSS*	

Tabela 43 – CRC: classe *CError*.

Classe	*CError*
Descrição	Classe responsável pela gerência das menssagens de exeção do sistema.
Responsabilidade	**Colaborador**
Exibir menssagem de error caso o banco de dados não esteja carregado	ConnDataBaseError
Exibir menssagem de error caso o banco de dados não esteja conectado	ExecDataBaseError
Exibir menssagem de error caso os procediemntos de execução do opendss sejam violados	ExecOpenDSS
Exibir menssagem de error caso os alimentadores não sejam selecionados	ExecSelectionFields
Exibir menssagem de error de inserção de equipamentos	ExecEnergyMeter

Tabela 44 – CRC: classe *CConfigDialog*.

Classe	*CConfigDialog*
Descrição	Classe responsável pela gerência da configuração de conexão com a base de dados do sistema
Responsabilidade	**Colaborador**
Selecionar o diretório que possui a base de dados que será utilizado pelo sistema	

Tabela 45 – CRC: classe*CDBaseConn*.

Classe	*CDBaseConn*
Descrição	Classe responsável pela gerência da configuração de conexão com a base de dados do sistema.
Responsabilidade	**Colaborador**
Conectar ao BDGD selecionado para interação do sistema com as informações necessárias para seu funcionamento pleno	
Conectar com a BDGD apenas no formato leitura	

Tabela 46 – CRC: Classe *CDBase*.

Classe	*CDBase*
Descrição	Classe responsável pela gerência da configuração de conexão com a base de dados do sistema.
Responsabilidade	**Colaborador**
Obter lista de subestação de alta tensão presentes na BDGD selecionada	
Obter lista de circuitos de alta tensão para média tensão presentes na BDGD selecionada	
Obter lista de subestações de média tensão presentes na BDGD selecionada	

Tabela 47 – CRC: Classe *CDBaseCoord*.

Classe	*CDBaseCoord*
Descrição	Classe de fronteira responsável pela gerência da configuração de conexão com a base de dados do sistema.
Responsabilidade	**Colaborador**
Coletar os códigos dos alimentadores de uma subestação de média tensão	CDBaseConn
Coletar as coordenadas geográficas de um alimentador de uma subestação de média tensão	CDBaseConn
Coletar o nome dos equipamentos transformadores de média tensão	CDBaseConn

Tabela 48 – CRC: Classe *CDBaseData*.

Classe	*CDBaseData*
Descrição	Classe de fronteira responsável pela gerência da configuração de conexão com a base de dados do sistema.
Responsabilidade	**Colaborador**
Coletar informações sobre o equivalente de Thevenin	CDBaseConn
Coletar informações sobre transformadores de alta para média tensão	CDBaseConn
Coletar informações sobre condutores	CDBaseConn
Coletar informações sobre seccionadoras de alta tensão	CDBaseConn
Coletar informações sobre seccionadoras de média tensão	CDBaseConn
Coletar informações sobre reguladores tensão	CDBaseConn
Coletar informações sobre seccionadoras de alta tensão	CDBaseConn
Coletar informações sobre unidade consumidora de média tensão	CDBaseConn
Coletar informações sobre transformadores de distribuição	CDBaseConn
Coletar informações sobre condutores de baixa tensão	CDBaseConn
Coletar informações sobre unidade compensadora de média tensão	CDBaseConn

Tabela 49 – CRC: Classe *CViewer*.

Classe	*CViewer*
Descrição	Classe de fronteira responsável pela gerência da configuração de conexão com a base de dados do sistema.
Responsabilidade	**Colaborador**
Criar o mapa de visualização da rede	CDBaseConn, CDataBaseConn
Mostrar o mapa de visualização da rede	
Configurar informações da rede para serem exibidas	

Tabela 50 – CRC: classe *COpenDSS*.

Classe	*COpenDSS*
Descrição	Classe de fronteira responsável pela gerência da construção da modelagem e execução da rede no formato opendss.
Responsabilidade	**Colaborador**
Criar o carregamento do *script* que modela a rede	
Criar o carregamento dos complementos do carregamento da rede	
Gerenciar a inserção de equipamentos medidores e monitores na rede modelada	
Executa comandos internos de ação do sistema	
Adquirir informações importantes sobre a rede modelada	

Tabela 51 – CRC: classe *CConfigDialog*.

Classe	*CConfigDialog*
Descrição	Classe de fronteira responsável pela gerência da configuração de simulação
Responsabilidade	**Colaborador**
Selecionar o método de conexão via COM ou direct	
Selecionar as tensões de base para se usada na simulação	
Selecionar o modo de simulação	
Configurar o modo de simulação	
Configurar tipo de consumidores e curvas de carga	

Tabela 52 – CRC: classe *CConfigLoadShapeDialog*.

Classe	*CConfigLoadShapeDialogg*
Descrição	Classe de fronteira responsável pela gerência da configuração de Curvas de carga
Responsabilidade	**Colaborador**
Selecionar curva de carga para ser utilizada na simulação	
Adicionar curva de carga	
Importar curva de carga	
Exportar curva de carga	
Visualizar curva de carga	

Tabela 53 – CRC: classe *CConfigPlotDialog*.

Classe	*CConfigPlotDialog*
Descrição	Classe de fronteira responsável por mostrar os resultados gráficos dos equipamentos monitores
Responsabilidade	**Colaborador**
Selecionar monitores na rede analisada para serem visualizados graficamente	
Mostrar resultados gráficos dos monitores inseridos na rede analisada	COpenDSS

Tabela 54 – CRC: classe *CInsertMonitorDialog*.

Classe	*CInsertMonitorDialog*
Descrição	Classe de fronteira responsável pela gerência dos equipamentos monitores
Responsabilidade	**Colaborador**
Inserir monitores na rede analisada	
Editar monitores inseridos na rede analisada	COpenDSS
Configurar monitores inseridos na rede analisada	COpenDSS

Tabela 55 – CRC: classe *CInsertEnergyMeterDialog*.

Classe	*CInsertEnergyMeterDialog*
Descrição	Classe de fronteira responsável pela gerência dos equipamentos medidores de energia
Responsabilidade	**Colaborador**
Inserir medidores de energia na rede analisada	
Editar medidores de energia inseridos na rede analisada	COpenDSS
Configurar medidores inseridos na rede analisada	COpenDSS

Tabela 56 – CRC: classe *CLoadDataProcess*.

Classe	*CLoadDataProcess*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**

Tabela 57 – CRC: classe *CLoadDataThread*.

Classe	*CLoadDataThread*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**

Tabela 58 – CRC: classe *CConn*.

Classe	*CConn*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**

Tabela 59 – CRC: classe *COpenDSSDirectConn*.

Classe	*COpenDSSDirectConn*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**

Tabela 60 – CRC: classe *COpenDSSCOMConn*.

Classe	*COpenDSSCOMConn*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**

Tabela 61 – CRC: classe *CData*.

Classe	*CData*
Descrição	Classe de fronteira
Responsabilidade	**Colaborador**
Obtenção do equivalente de thevenin	CDBaseData
Obtenção dos transformadores AT/MT	CDBaseData
Obtenção dos tipos de condutores	
Obtenção do identificador do alimentador	
Obtenção das chaves seccionadoras de alta tensão	CDBaseData
Obtenção dos controles das chaves seccionadoras de alta tensão	CDBaseData
Obtenção das chaves seccionadoras de média tensão tipo óleo	CDBaseData
Obtenção do controle das chaves seccionadoras de média tensão tipo óleo	CDBaseData
Obtenção das chaves seccionadoras de média tensão tipo faca	CDBaseData
Obtenção do controle das chaves seccionadoras de média tensão tipo faca	CDBaseData
Obtenção das chaves seccionadoras de média tensão tipo faca tripolar	CDBaseData
Obtenção do controle das chaves seccionadoras de média tensão tipo faca tripolar	CDBaseData
Obtenção das chaves seccionadoras de média tensão tipo fusível	CDBaseData
Obtenção do controle das chaves seccionadoras de média tensão tipo fusível	CDBaseData
Obtenção do identificador do alimentador	

ANEXO F – Diagramas de Classes

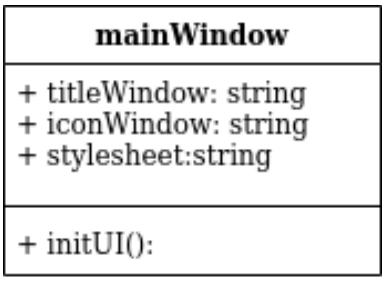


Figura 66 – Classe MainWindow

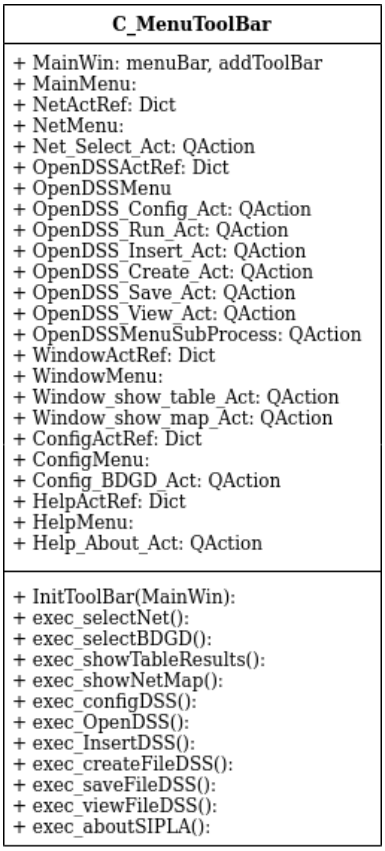


Figura 67 – Classe CMenuToolBar

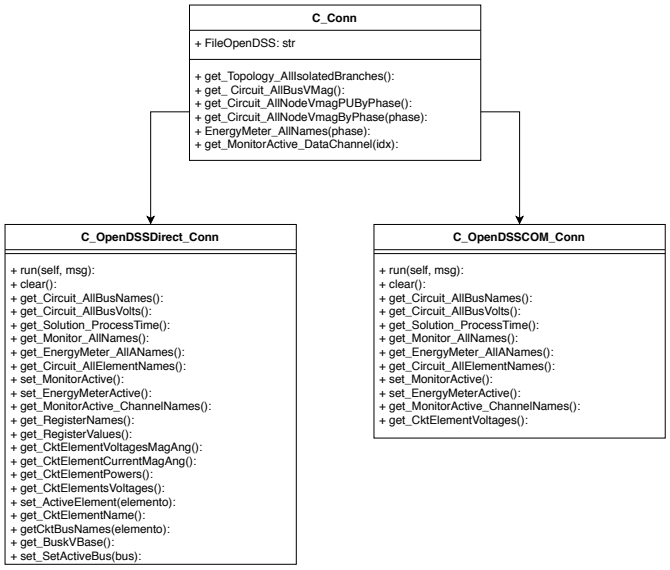


Figura 68 – Classe *CConn*, *COpenDSSDirectConn*, *COpenDSSCOMConn*

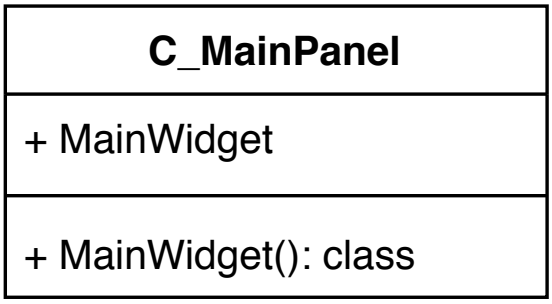


Figura 69 – Classe CMainPanel

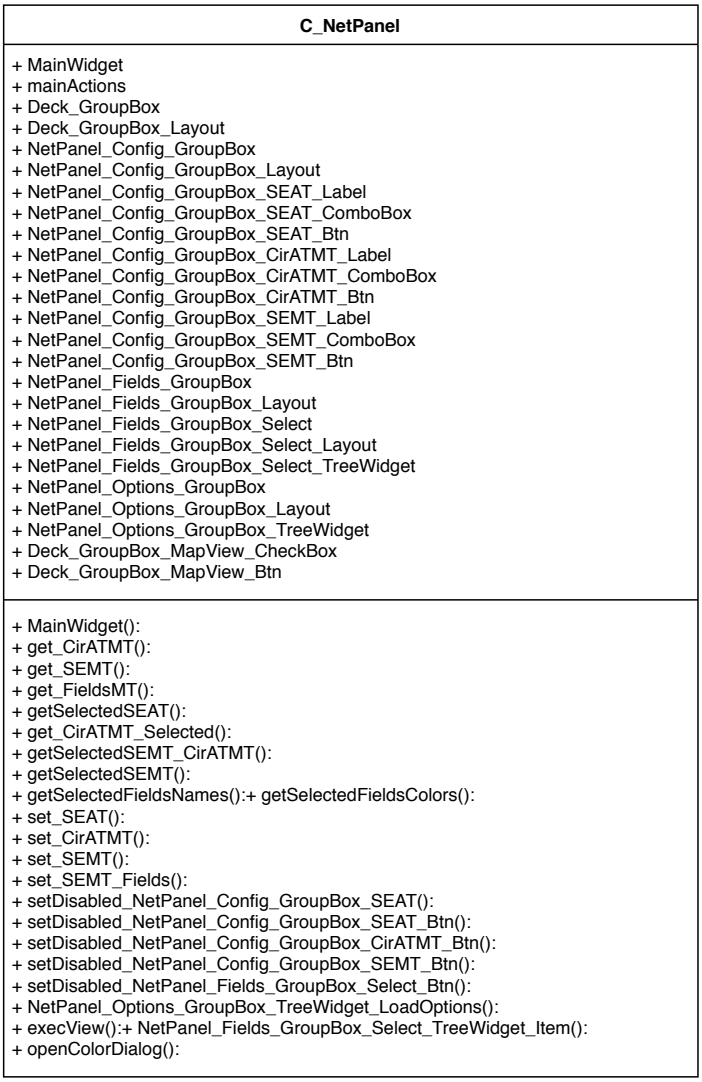


Figura 70 – Classe CNetPanel

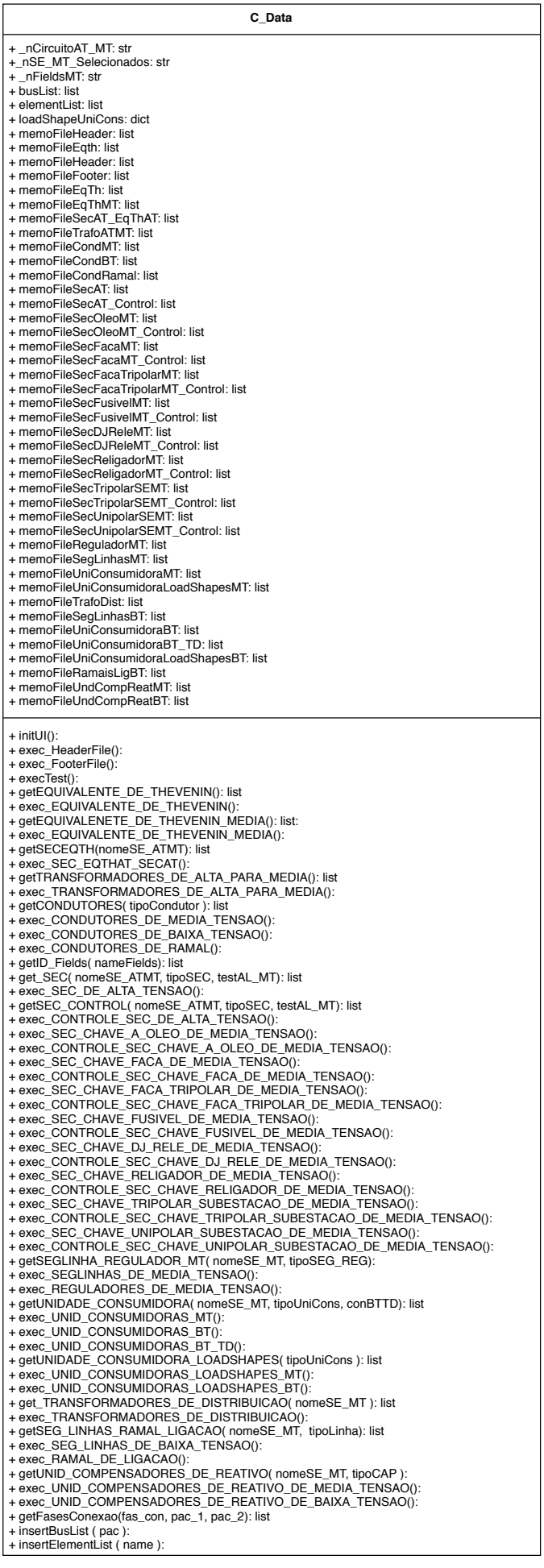


Figura 71 – Classe *CData*

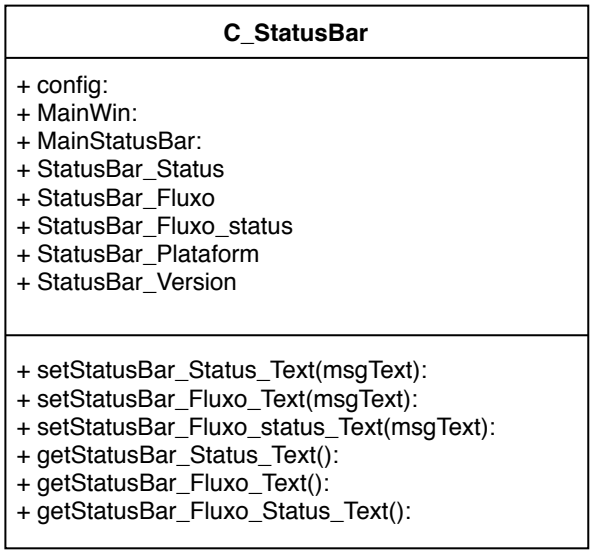


Figura 72 – Classe *CStatusBar*

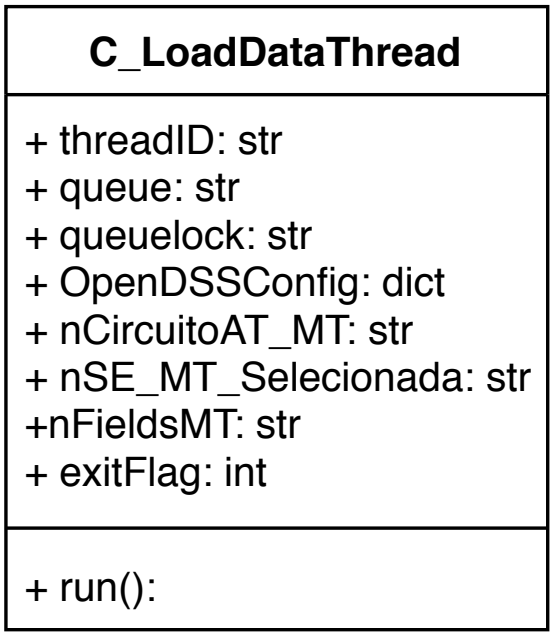


Figura 73 – Classe *CLoadDataThread*

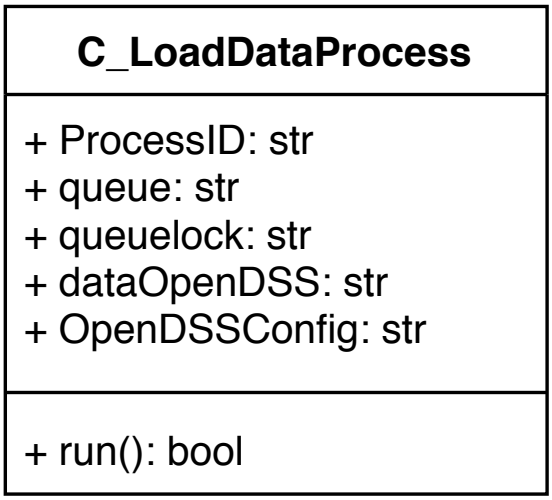


Figura 74 – Classe *CLoadDataProcess*

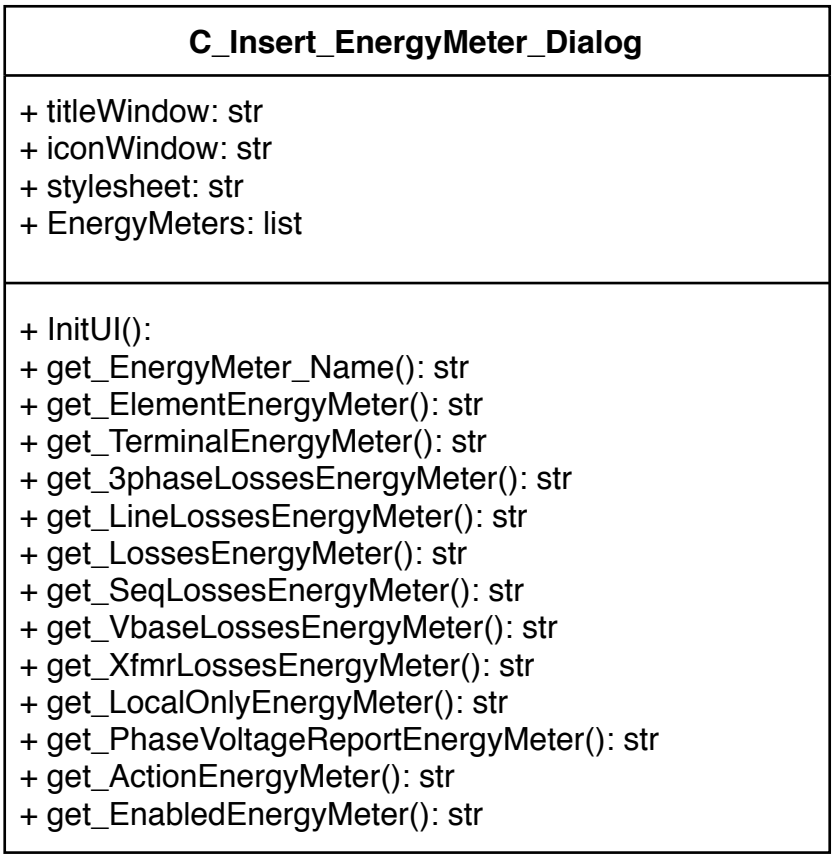


Figura 75 – Classe *CInsertEnergyMeterDialog*

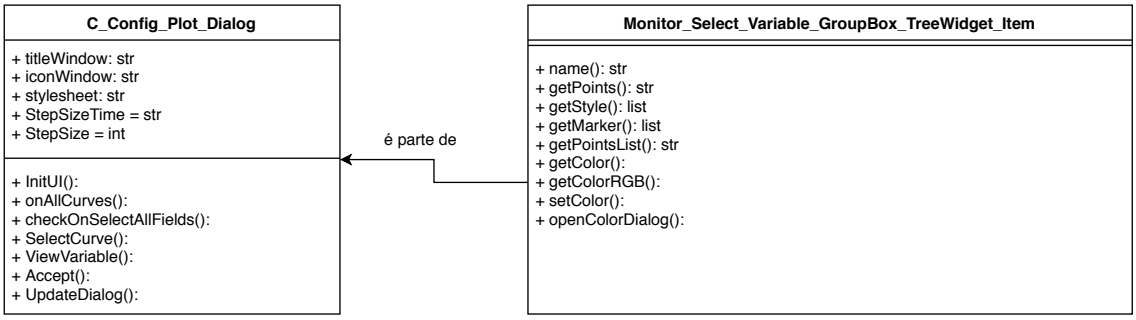


Figura 76 – Classe *CConfigPlotDialog*

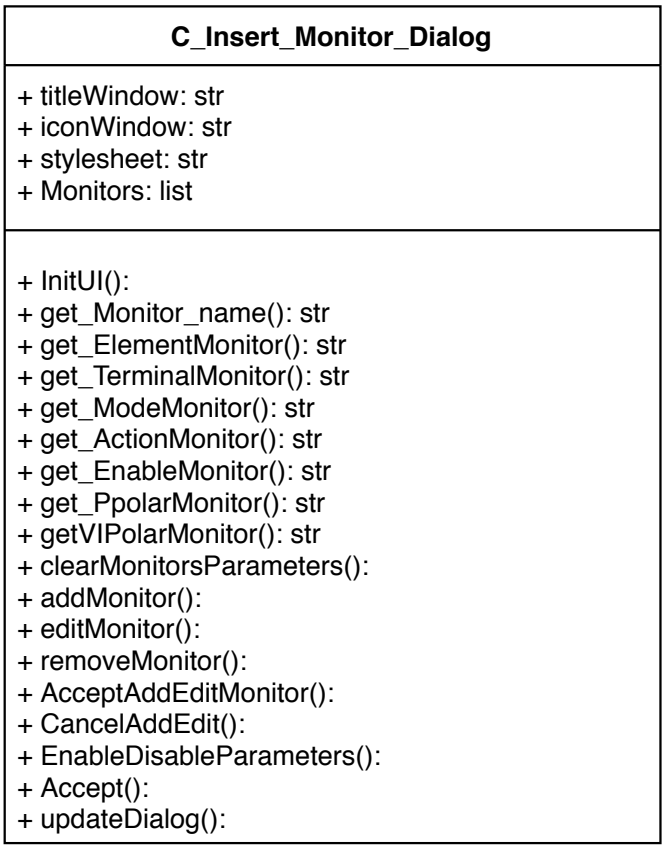


Figura 77 – Classe *CInsertMonitorDialog*

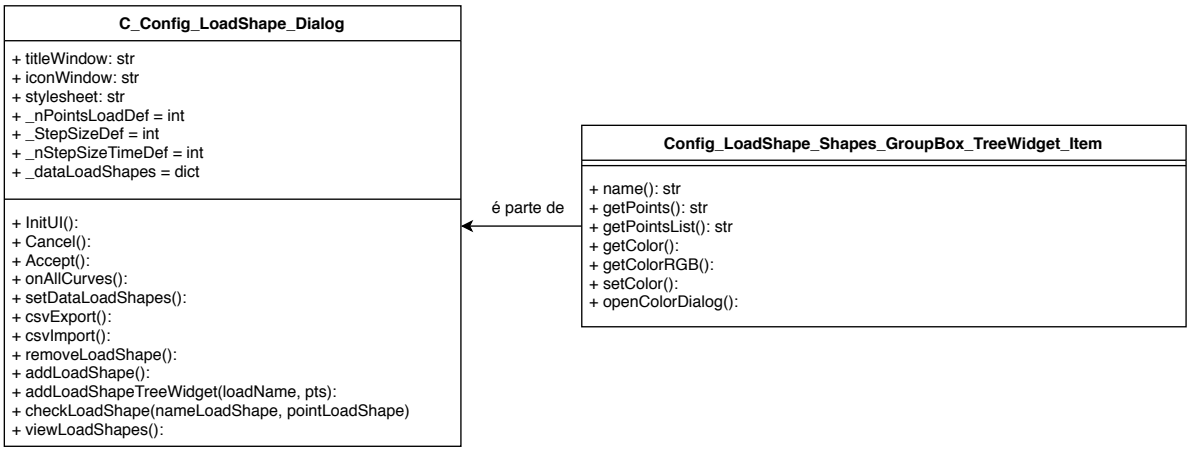


Figura 78 – Classe *CConfigLoadShapeDialog*

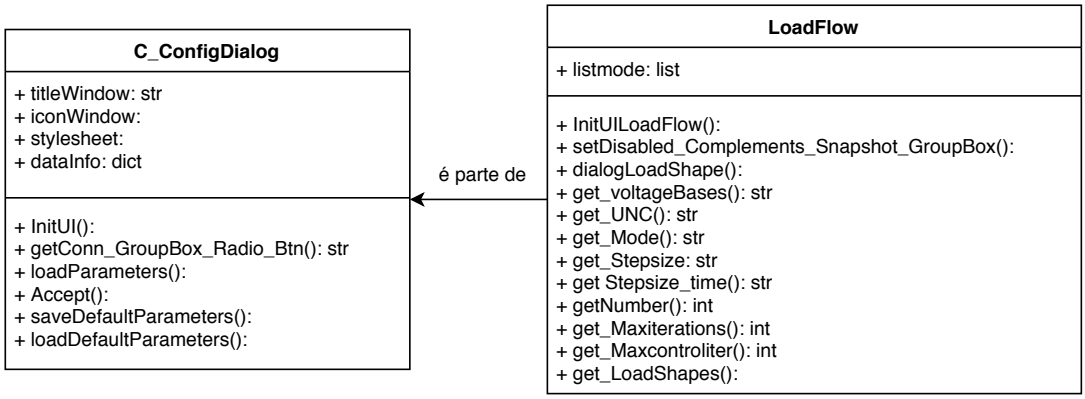


Figura 79 – Classe *CConfigDialog*

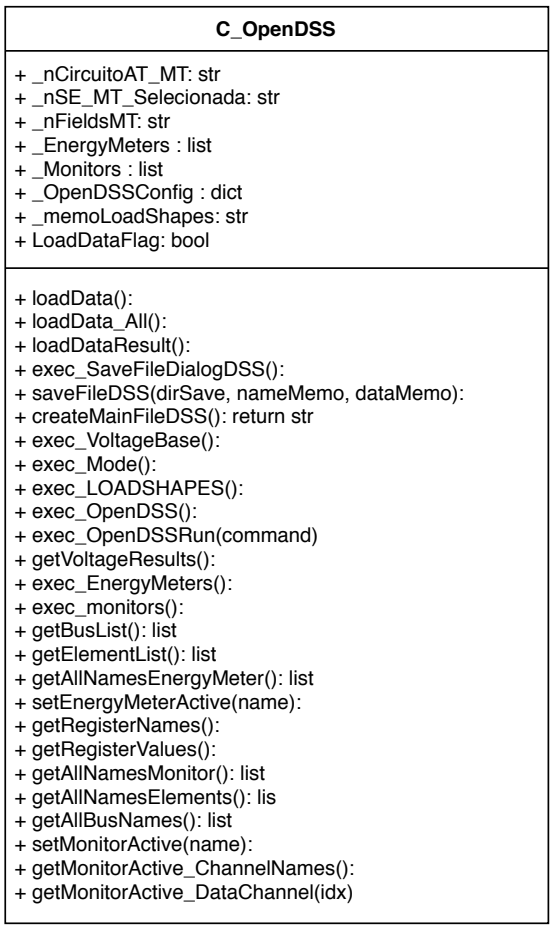


Figura 80 – Classe *COpenDSS*

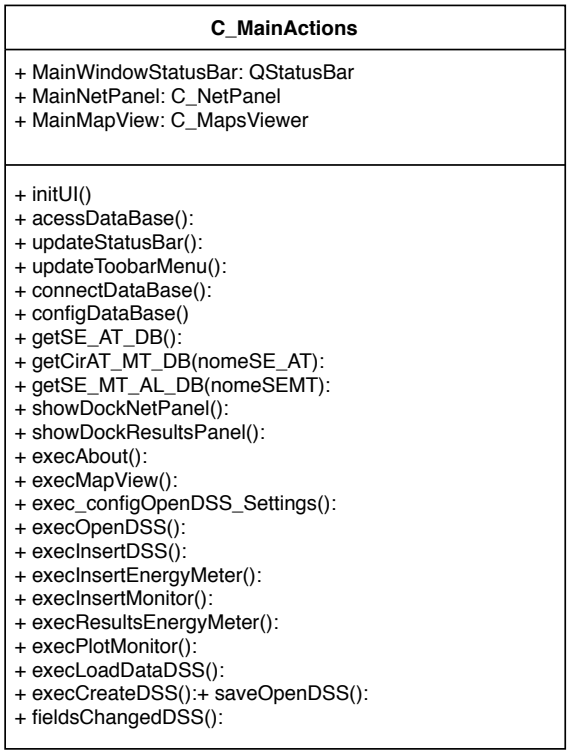


Figura 81 – Classe *MainActions*

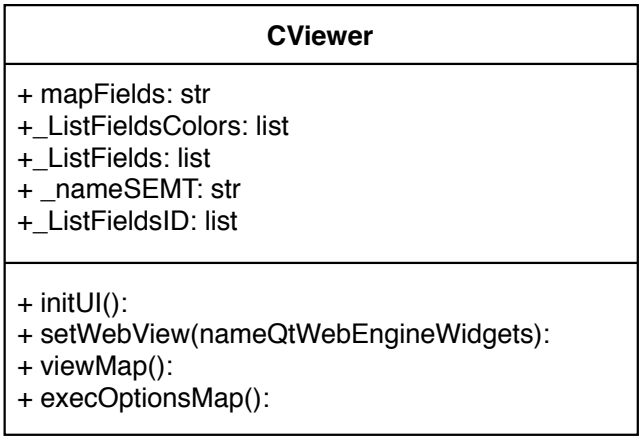


Figura 82 – Classe *CViewer*

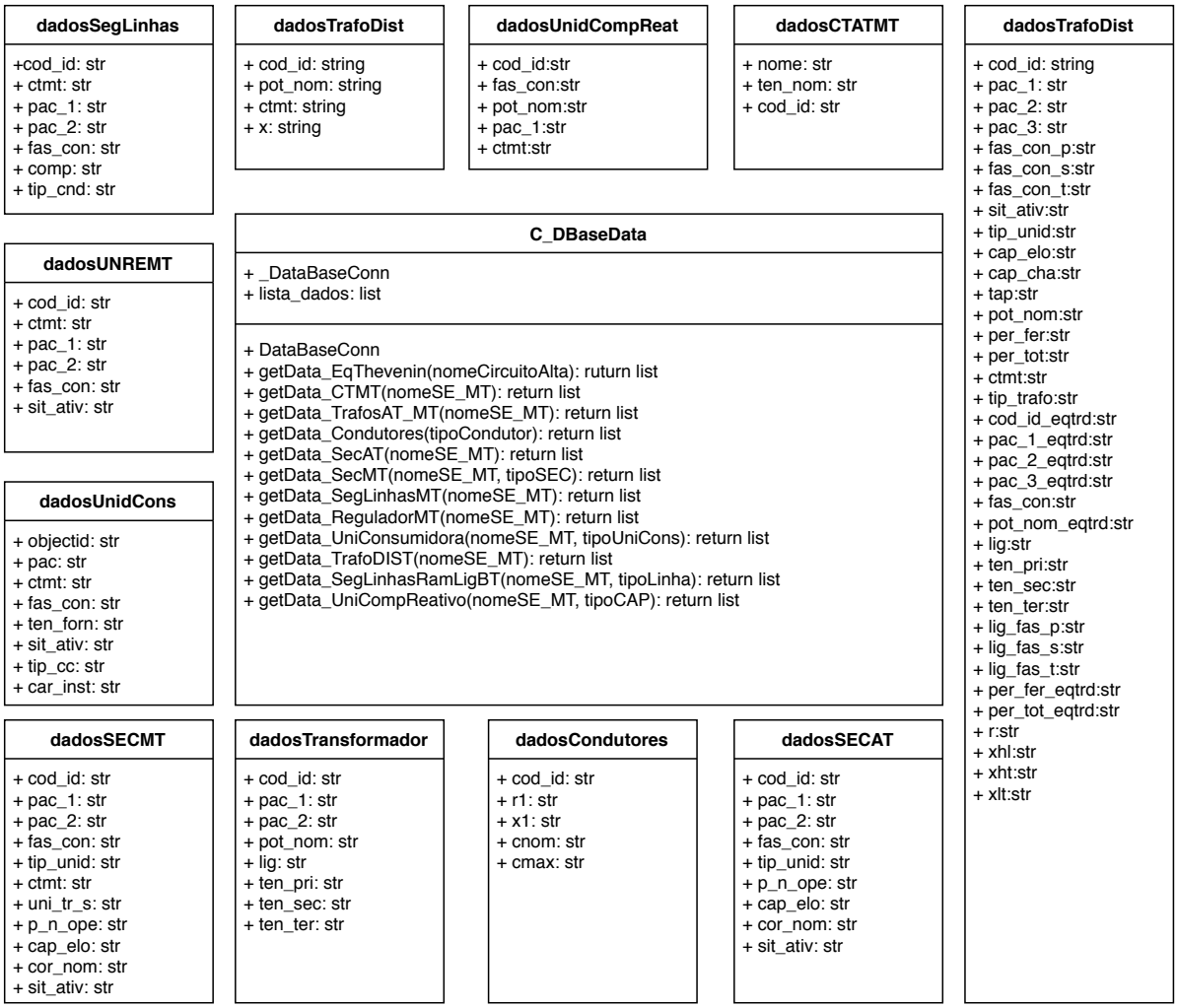


Figura 83 – Classe *CDBaseData*

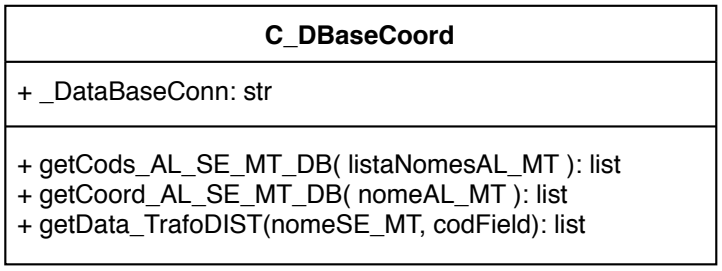


Figura 84 – Classe *CDBaseCoord*

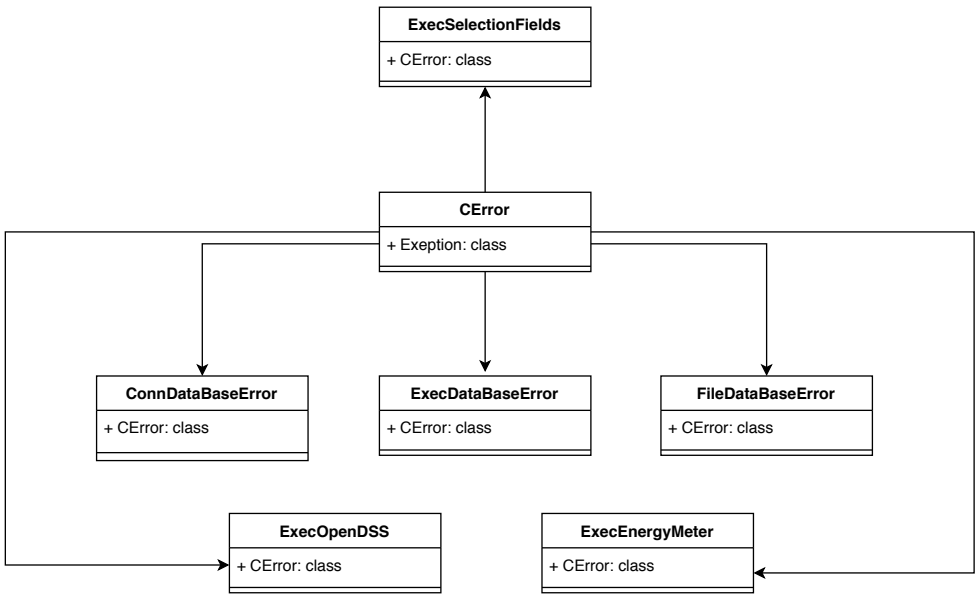


Figura 85 – Classes de gerenciadoras de mensagens de *Exceção*

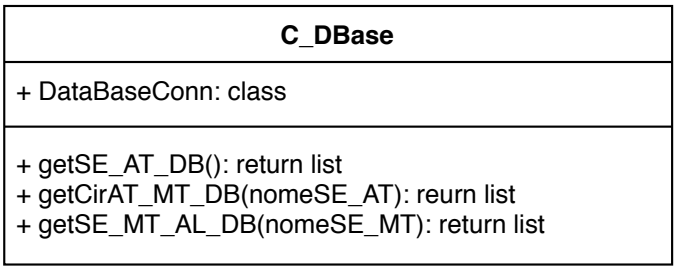


Figura 86 – Classe *CDBase*

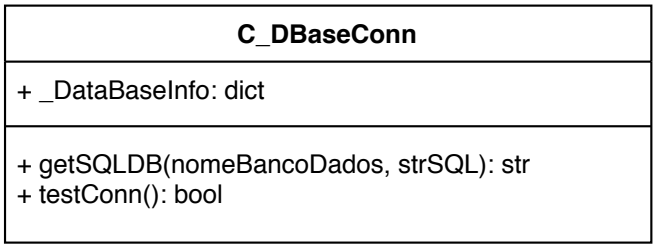


Figura 87 – Classe CDBaseConn

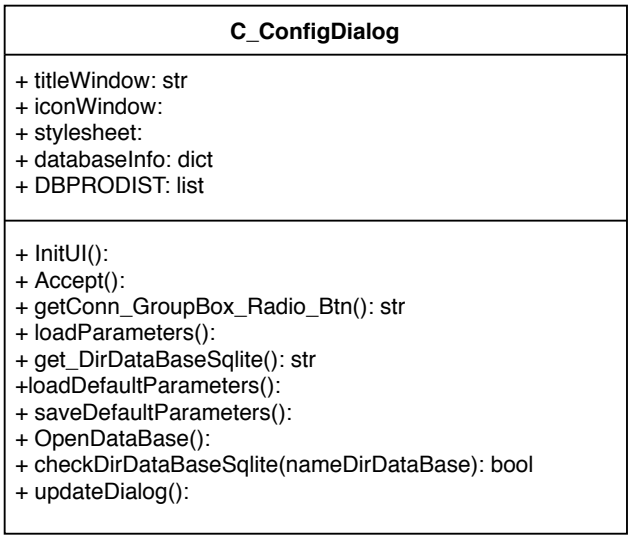


Figura 88 – Classe ConfigDialog