UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CAMPUS PAU DOS FERROS
CURSO DE ENGENHARIA DE SOFTWARE

RARYSSON GUILHERME DA COSTA

PROJETO DE UM MODELADOR E SIMULADOR DE REDE DE PETRI

PAU DOS FERROS – RN
2020

RARYSSON GUILHERME DA COSTA

PROJETO DE UM MODELADOR E SIMULADOR DE REDE DE PETRI

Monografia apresentada a Universidade Federal Rural do Semi-Árido, campus Pau dos Ferros como requisito para obtenção do título de Bacharel em Egenharia de Software.

Orientador: Prof. Dr. Reudismam Rolim de Sousa

PAU DOS FERROS – RN
2020

C837p Costa, Rarysson Guilherme da.
PROJETO DE UM MODELADOR E SIMULADOR DE REDE DE PETRI / Rarysson Guilherme da Costa. - 2020.
51 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia de Software, 2020.

1. Rede de Petri. 2. PIPE. 3. Desenvolvimento. 4. Programação Concorrente e Distribuída. 5. Interface. I. Sousa, Reudismam Rolim de, orient. II. Título.

RARYSSON GUILHERME DA COSTA

PROJETO DE UM MODELADOR E SIMULADOR DE REDE DE PETRI

Monografia apresentada a Universidade Federal Rural do Semi-Árido, campus Pau dos Ferros como requisito para obtenção do título de Bacharel em Engenharia de Software.

Aprovado em: 11 / 12 / 2020.

BANCA EXAMINADORA

_____

Prof. Dr. Reudismam Rolim de Sousa – UFERSA

Presidente

_____

Prof. Dr. Vinícius Samuel Valério de Souza – UFERSA

Membro da banca

_____

Profa. Dra. Samara Martins Nascimento – UFERSA

Membro da banca

PAU DOS FERROS – RN

2020

AGRADECIMENTOS

Para ter chegado a situação de eu concluir o curso de Engenharia de Software e ter que escrever essa sessão foram necessários muitos eventos ocorrerem, e nesses eventos aleatórios do universo (ou não tão aleatórios, dependendo da sua crença) encontrei diversas pessoas que talvez eu não tenha mais contato, mas que de alguma forma causaram algum impacto em mim, e para de certa forma agradecer vou dedicar esse espaço a elas.

Apesar de eu não ser uma pessoa religiosa, gostaria de agradecer a Deus por ter me guiado durante esse tempo, mesmo que a direção indicada não parecia ter feito muito sentido, mas quem sou eu para duvidar.

Família sempre é complicado, mas de certa forma todos estarão perto de você para ajudar nos momentos difíceis. Por isso gostaria de agradecer a minha mãe Zefinha por ter me educado da melhor forma possível. Ao meu irmão Robson por sempre tentar me ajudar nos últimos tempos quando preciso de ajuda. Ao meu irmão Reirysson que sempre foi meu amigo. Também gostaria de agradecer ao meu irmão Halyson, apesar de a gente não ter muito contanto, ainda creio que de alguma forma você me influenciou. Já os outros parentes não tenho tanto contato, mas gostaria de agradecer a minha prima Valzinha por nesses últimos tempos da faculdade ter se tornado minha amiga.

Já na UFERSA me encontrei com as pessoas mais diversas possíveis, e isso me foi bastante útil para expandir a minha visão e entender melhor o mundo. Primeiramente, gostaria de agradecer aos professores da banca, Samara e Vinícius, por terem me ensinado e por tomarem uma parte do seu tempo para corrigir este trabalho, principalmente, por Reudismam, pela sua orientação e seu jeito de ser e ensinar. Também não poderia me esquecer de: Marco Diego, Thiago Rique, Claudio Callejas, Fernando Henrique (sem você eu ainda estaria pagando Cálculo), Helder e Jarbele.

Durante esse período da faculdade vi diversas pessoas aparecerem na minha vida, apesar de nem sempre ficarem, elas foram bastantes úteis para melhorar a minha perspectiva de vida. Pode ser que eu esqueça de alguém, mas gostaria de agradecer: Dyego Magno, Lucas Santana, Edglê, Lucas Lima (Lusca), Josaias, Tássio, Assunaueny, Adby, Emerson, Belarmino, Igor Ramon, Jadson, Laís, Rafael Luan, Wallace, Remyson, Silvio, Allan Matheus, Liliane, Luiz Paulo, Raimundo Jácome (Raí), Dangels, Rafaela Duarte, Dmytres, Tenório, Camila Perin, Leogilton, Samuel Dias.

RESUMO

A modelagem de sistemas por meio de redes de Petri é importante para várias atividades do ensino e do mercado. Nesse sentido, alguns *softwares* foram desenvolvidos para auxiliar os projetistas a modelarem e simularem essas redes, tais como HpetriSim e IOPT Tools. Porém, um dos *softwares* mais empregados é o PIPE. No entanto, esse *software* apresenta alguns problemas no tocante a modelagem da rede e a dificuldade de evoluir o *software* para atender demandas específicas. Nesse sentido, a proposta do trabalho possui como propósito o desenvolvimento de uma ferramenta capaz de modelar e simular Redes de Petri como uma alternativa às funções de modelar e simular do *software* PIPE. Foram analisados um conjunto de softwares que são utilizados para a modelagem de redes de Petri. As principais características desses sistemas foram consideradas para o desenvolvimento da ferramenta proposta, por exemplo, a disponibilização delas *online*, buscando também reduzir os aspectos negativos apresentados por elas. Para tornar possível o desenvolvimento da ferramenta, foram seguidos processos definidos para o desenvolvimento de *software*, gerando artefatos para auxiliar na idealização da interface. O resultado obtido foi satisfatório para o cenário que se propôs. A ferramenta proposta também pode ser evoluída em trabalhos futuros para atuar de forma a atender outras necessidades que se demonstrarem necessárias para a modelagem geral de uma Rede de Petri.

Palavras-chave: Rede de Petri. PIPE. Desenvolvimento. Programação Concorrente e Distribuída. Interface.

ABSTRACT

The modeling of systems using Petri nets is important for various teaching and market activities. In this sense, some software was developed to help designers to model and simulate these networks, such as HpetriSim and IOPT Tools. However, one of the most used software is PIPE. However, this software has some problems with regard to network modeling and the difficulty of evolving the software to meet specific demands. In this sense, the purpose of the work is to develop a tool capable of modeling and simulating Petri Nets as an alternative to the modeling and simulating functions of the PIPE software. A set of software that are used for modeling Petri nets was analyzed. The main characteristics of these systems were considered for the development of the proposed tool, for example, making them available online, also seeking to reduce the negative aspects presented by them. To make the development of the tool possible, defined processes for software development were followed, generating artifacts to assist in the idealization of the interface. The result obtained was satisfactory for the proposed scenario. The proposed tool can also be developed in future works to act in order to meet other needs that prove necessary for the general modeling of a Petri Net.

Keywords: Petri net. PIPE. Development. Concurrent and Distributed Programming. Interface.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CPU – *Central Processing Unit*

PRISMA – *Petri networks Simulation and Modeling*

PIPE – *Platform Independent Petri-net Editor*

SUMÁRIO

1 INTRODUÇÃO

A área de computação evoluiu, significativamente, com a demanda crescente por tecnologia nas últimas décadas, passando do uso de válvulas eletrônicas a transistores e de computadores que ocupavam um galpão para aqueles que cabem na palma da mão (TANENBAUM, 2013). Da mesma forma, o poder computacional evoluiu em ritmo similar. Anteriormente, o objetivo era aumentar o *clock* de um núcleo da CPU (do inglês *Central Processing Unit* (CPU) conhecido como Unidade Central de Processamento, agora o objetivo passou a ser aumentar o número de núcleos de CPU (STALLINGS, 2010). O aumento no número de núcleos de CPU fez surgir novos paradigmas de programação para aproveitar esse novo poder computacional, dentre elas a programação concorrente e distribuída. Paradigma que foi teorizado para aproveitar ao máximo o poder computacional extra de múltiplos núcleos ou, até mesmo, múltiplos computadores conectados em conjunto (TANENBAUM; STEEN, 2008).

A disciplina de Programação Concorrente e Distribuída, na Universidade Federal Rural do Semi-Árido, Centro Multidisciplinar de Pau dos Ferros, Rio Grande do Norte, do curso de Bacharelado em Tecnologia da Informação (UFERSA, 2014) foi criada com o objetivo de abordar esse novo paradigma de programação. Essa disciplina aborda a parte prática dos conhecimentos teóricos adquiridos na disciplina Sistemas Distribuídos do referido curso, sendo a disciplina Sistemas Distribuídos dividida em duas etapas fundamentais. A primeira delas, o ensino da modelagem em Rede de Petri para problemas da programação, como a realização de um programa entre múltiplos processos e a segunda etapa, a aplicação prática do aprendizado de programação concorrente e paralela, em que é aplicado o conceito de processos e *threads* para analisar como se dá a execução desses tipos de programas.

A etapa de modelagem em Rede de Petri da disciplina Programação Concorrente e Distribuída também é dividida em duas etapas. A primeira etapa é a modelagem de sistemas simples, para, dessa forma, permitir entender os aspectos básicos de modelagem de Rede de Petri. Neste primeiro momento, a modelagem é realizada no papel, sem o auxílio de um sistema computacional, uma vez que os aspectos envolvidos são mais simples de abordar. De outra forma, a segunda etapa aborda a modelagem de sistemas mais complexos, que devido à complexidade, necessitam de um sistema de *software* para serem modelados. Durante a disciplina, para realizar a modelagem de Rede de Petri no computador, geralmente, são

utilizados sistemas como o *software* PIPE (do inglês *Platform Independent Petri-net Editor*) (BONETE et al., 2007). Esse *software* permite organizar a rede e simulá-la e também possui recursos mais robustos, como a verificação de um *deadlock* na rede modelada, recursos extras esses que geralmente não são utilizados durante a disciplina.

Apesar do *software* PIPE ser robusto, foi possível perceber alguns problemas com o mesmo: algumas vezes, o *software* não permitia salvar o arquivo da rede modelada, o que causava, em alguns casos, a perda do progresso feito na Rede de Petri. Além disso, o *software* também não era consistente na hora da modelagem, uma vez que, em alguns casos, a edição da rede era prejudicada devido a algum erro na movimentação dos objetos da Rede de Petri. Adicionalmente, o PIPE é um *software* que possui características gerais, evoluído para atender as diferentes aplicações do software, o que leva a demora e a dificuldade de resolver problemas como os citados anteriormente.

Visando corrigir esses problemas e fornecer um *software* que seja capaz de atender as demandas de disciplinas voltadas para a modelagem e simulação de redes de Petri, neste trabalho é proposto o desenvolvimento de um *software* para modelagem e simulação de Rede de Petri. O *software* é denominado de PRISMA (do inglês *Petri networks Simulation and Modeling*), uma plataforma Web para simulação e modelagem de redes de Petri. Por ser desenvolvido tomando por base as características específicas da disciplina Programação Concorrente e Distribuída, a ferramenta pode contribuir para a realização de atividades práticas na disciplina e também pode ser evoluída a partir do uso da plataforma pelos discentes do campus.

1.1 Problematização

Ferramentas de *software* capazes de modelar e simular uma Rede de Petri são importantes em diversas áreas, com um caso específico na aplicabilidade das disciplinas da UFERSA, tais como Programação Concorrente e Distribuída e Métodos Formais de Engenharia de *Software*, uma vez que a construção de uma Rede de Petri com papel e caneta pode se tornar uma tarefa não trivial para redes complexas. Em disciplinas que não utilizam ferramentas, por exemplo a componente Métodos Formais de Engenharia de *Software*, no formato atual, essa necessidade foi notada. Em contrapartida, a disciplina Programação Concorrente e Distribuída, utiliza, dentro dos diversos *softwares* para modelar uma Rede de

Petri (HAMBURG, 2020), o *software* PIPE. No decorrer dos semestres letivos, os alunos notaram algumas falhas do *software*, sendo uma das falhas fundamentais a dificuldade com respeito à usabilidade. Esse problema é decorrente tanto da falta de confiabilidade no estado da Rede, quanto na própria modelagem da Rede. A falta de confiabilidade ocorre devido a inconsistências no armazenamento do modelo da Rede, em que eventualmente ao fechar o *software* e abrir novamente, os lugares, transições e conexões dos lugares a transições não estarem da forma que o usuário modelou. Ocorre também problemas na modelagem da Rede, em que ao posicionar lugares ou ao tentar realizar alguma conexão, o *software* não efetiva a ação, podendo colocar o lugar em outro ponto da rede ou realizar a conexão do lugar com um evento incorreto, ou vice-versa. Além disso, por ser um *software* de terceiros, a customização da ferramenta para atender demandas específicas se torna dificultosa.

1.2 Objetivos

Considerando o que foi apresentado na problematização, o trabalho possui como objeto geral o de desenvolver um *software* capaz de modelar e simular uma Rede de Petri que minimize os problemas de usabilidade apresentados pelo *software* PIPE. Para tornar a realização do objetivo geral, foram definidos os seguintes objetivos específicos:

- Revisão bibliográfica sobre o funcionamento de uma Rede de Petri.
- Análise do *software* PIPE e seus concorrentes.
- Análise da eficácia e consistência da solução proposta.
- Analisar possíveis extensões de funcionalidades do *software* desenvolvido.
- Escrita de um Trabalho de Conclusão de Curso.

1.3 Justificativa

O trabalho possui como proposta a de construir um *software* capaz de modelar e simular uma Rede de Petri. Dentre as várias funcionalidades do *software* PIPE, o trabalho foi focado na modelagem e simulação, foco do uso do *software* na componente Programação Concorrente e Distribuída. Espera-se desenvolver um *software* capaz de modelar e simular uma Rede de Petri sem as inconsistências do PIPE.

Apesar da quantidade de *softwares* disponíveis para modelar uma Rede de Petri demonstrada em Hamburg (2020), nem todos atendem às necessidades da disciplina, pois uma pequena parte é voltada para Rede de Petri Colorida, uma versão melhorada da Rede de Petri. Alguns dos sites dos projetos estão indisponíveis, impossibilitando o acesso ao *software*. Outros estão desatualizados, tendo como requisito um sistema operacional antigo. Dessa maneira, a criação de outro *software* para modelar e simular uma Rede de Petri se tornaria um projeto capaz de aumentar a gama desses tipos de *software*, assim, oferecendo mais escolhas ao usuário.

Além disso, o *software* poderia ser customizado ao longo do tempo para se adequar cada vez mais às demandas dos discentes, tarefa que seria inviável com *softwares* de terceiros. Nesse sentido, o projeto fomenta a aprendizagem na graduação de Tecnologia da Informação, Engenharia de Computação e Engenharia de *Software* e também possibilita o desenvolvimento de outros projetos que busquem aperfeiçoar este trabalho, podendo acarretar, após evoluções, num projeto com funcionalidades semelhantes ao PIPE, mas adaptado às necessidades dos discentes dos cursos citados acima.

2 FUNDAMENTAÇÃO TEÓRICA

Para a melhor compreensão deste trabalho, é necessário que alguns fundamentos teóricos relacionados a Sistemas Distribuídos, Programação Concorrente e Distribuída, Rede de Petri e interface sejam conhecidos. Visando apresentar esses fundamentos, as próximas seções fundamentarão os tópicos mencionados.

2.1 Sistemas Distribuídos

A definição de um sistema distribuído pode variar de autor para autor. Para Tanenbaum e Steen (2008), um sistema distribuído é um conjunto de computadores conectados em que o usuário interpreta como um único computador. O uso de sistemas distribuídos vem crescendo com a necessidade de obter mais poder computacional para realizar tarefas mais robustas. De forma, ao se apresentar como um único computador ao usuário, uma etapa fundamental é a criação de um *middleware* (COLOURIS et al., 2013), um *software* que faz a comunicação do usuário com os diversos computadores conectados e pode apresentar diferentes qualidades, como: (i) acesso aos recursos, (ii) transparência da distribuição, (iii) abertura e (iv) escalabilidade (TANENBAUM; STEEN, 2008). O acesso aos recursos representa a facilidade do usuário em acessar os recursos desejados e a segurança por trás deles; a transparência da distribuição representa como o acesso à informação se apresenta ao usuário, por exemplo, informa se o usuário precisa saber de onde o dado ou a notificação de falhas do recurso vem; a qualidade de abertura diz respeito a como o *middleware* deve apresentar suas interfaces de comunicação, pois o *middleware* apenas faz a comunicação da requisição do acesso ao recurso com o usuário, podendo o usuário utilizar diferentes interfaces gráficas para um mesmo *middleware*; a qualidade de escalabilidade estabelece como o sistema deve ser escalável, seja em quantidade de usuários ou em relação a localização geográfica (o usuário pode acessar o sistema em diferentes locais sem graves problemas). Devido ao escopo deste trabalho, os sistemas distribuídos não serão discutidos em detalhes. Para mais informações, consulte Tanenbaum e Steen (2008).

O desenvolvedor deve analisar a necessidade de utilizar um sistema distribuído em cada caso, uma vez que adicionar as capacidades de um sistema distribuído pode tornar o desenvolvimento de um sistema complexo. No entanto, caso seja necessário que diversos

computadores possam atuar de forma a melhorar a performance do sistema, o uso dos sistemas distribuídos é encorajado. Um exemplo de um sistema distribuído bastante utilizado é o Dropbox (DROPBOX, 2020), que mantém diversos servidores que guardam os arquivos pessoais do usuário e ele tem a impressão que está acessando um único computador.

2.2 Programação Concorrente e Distribuída

Apesar dos computadores serem peças fundamentais para o mundo moderno de hoje, a sua presença é recente. Os computadores começaram a ser utilizados na década de 50 em situações bem específicas, como para fins militares (RAINER; CEGIELSKY, 2012). No entanto, com a evolução dos processadores o seu uso começou a se difundir para mais áreas, a exemplo de universidades e grandes empresas. Porém, foi somente com a invenção dos transistores que o poder computacional dos computadores e a sua disseminação se desenvolveu em um ritmo acelerado, em que os computadores passaram de ocupar um galpão para um ocupar um espaço que coubesse numa mesa de um cidadão comum. A evolução constante dos processadores abriu precedentes para Gordon E. Moore formular o que se tornaria a Lei de Moore, que estabelece que a cada 18 meses o processador dobraria de poder computacional e continuou verdadeira desde a sua proposição, mas atualmente algumas restrições físicas estão dificultando a validade desta lei: os processadores não são mais capazes de colocar mais transistores num menor espaço (FREITAS et al., 2009). Neste sentido, outras abordagens começaram a ser utilizadas na produção de processadores, em vez de aumentar o poder computacional de um núcleo, passou-se a colocar diversos núcleos num mesmo processador, conseguindo paralelizar o poder computacional, e assim permitir executar diversas tarefas ao mesmo tempo (DEITEL et al., 2005).

Com o advento de processadores com múltiplos núcleos e a possibilidade de utilizar o poder computacional de múltiplos computadores, foi se tornando necessário desenvolver um novo paradigma de programação que conseguisse utilizar o novo poder computacional. Dessa maneira, surgiu a Programação Concorrente e Distribuída, que tem foco em permitir a utilização do poder computacional de múltiplos núcleos e computadores (TANENBAUM, 2010). A principal característica desse paradigma de programação é a utilização de *threads* e processos para permitir a paralelização da execução do programa. Ressalta-se que o uso de *threads* e processos é mais conectado a parte concorrente, que consiste em desenvolver um

software que execute em múltiplos núcleos de um mesmo computador, mas não exclui a utilização em conjunto com a programação distribuída, que dada a consolidação dos sistemas distribuídos, facilita essa integração.

Com o aumento de núcleos e sistemas mais complexos, os sistemas distribuídos começaram a ser desenvolvidos para aproveitar o poder de múltiplos computadores e se tornou uma tendência conseguir programar para usufruir o máximo da paralelização computacional, em que algumas linguagens de programação foram desenvolvidas com esse intuito, como a linguagem de programação brasileira Elixir (ELIXIR, 2020).

2.3 Rede de Petri

A Rede de Petri surgiu da necessidade de conseguir modelar sistemas discretos de maneira mais eficiente em que um sistema discreto é aquele que altera seu comportamento em intervalos precisos de tempo (ROSEN, 2009). Dessa maneira, o advento dos computadores tornou as Redes de Petri bastante úteis para modelar o comportamento desses novos sistemas, uma vez que um computador pode ser visto como um sistema discreto. E com a sua utilidade demonstrada, outros autores evoluíram com outras características, por exemplo, Rede de Petri Colorida, em que permite codificação na modelagem. Dentre esses diversos tipos de Rede de Petri, o trabalho focou-se nas Redes de Petri Ordinárias.

Uma Rede de Petri é dividida em dois tipos de modelagens: gráfica e matemática (CARDOSO; VALLETE, 1997). A modelagem gráfica é realizada pela definição de objetos que representam algo do mundo real, enquanto que, a modelagem matemática é realizada por definições matemáticas que representam algo do mundo real. O escopo deste trabalho está limitado a alguns aspectos da modelagem gráfica da Rede de Petri. Dessa maneira, quando a modelagem em Rede de Petri, for citada, está será a modelagem gráfica.

De acordo com Cardoso e Vallete (1997), a modelagem gráfica da Rede de Petri é feita através de três objetos fundamentais, que podem ser vistos na Figura 1: (i) lugar (representado na opção a), (ii) transição (representado na opção b), (iii) ficha (representado na opção c) e (iv) arco (representado na opção d). Um lugar é representado graficamente por um círculo e representa um estado ou um recurso de algo do mundo real. Uma transição é representada graficamente por um retângulo (podendo ser preenchido ou não) e representa o acionamento de um evento do objeto no mundo real. Uma ficha é representada graficamente

por um ponto dentro de um lugar e representa a veracidade do estado ou presença de recurso. Já um arco representa a conexão de um lugar com uma transição ou vice-versa.

Figura 1 – Objetos da Rede de Petri

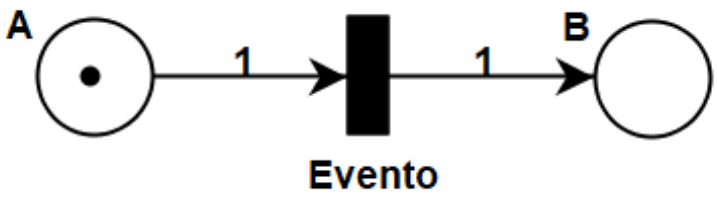
Fonte: Autor (2020)

A modelagem gráfica de uma Rede de Petri pode se tornar complexa, principalmente, se esta envolver conceitos como paralelização e recursos compartilhados, devido às inúmeras possibilidades de modelar uma Rede de Petri. Neste trabalho serão apresentados dois exemplos bases que ajudarão na compreensão de outras modelagens de Rede de Petri. Vale ressaltar que a teoria envolvida nas Redes de Petri é mais complexa do que a apresentada aqui, mas devido ao fato deste trabalho não apresentar mais exemplos de Rede de Petri, além desta seção, não serão apresentados estes aspectos. Mas, uma teoria fundamental que deve ser conhecida é as características envolvidas para o disparo de uma transição. Para o disparo de uma transição ocorrer é necessário que o lugar de origem conectado àquela transição tenha o número de fichas mínimas, em que esse número mínimo é indicado pelo número presente no arco conectado àquele lugar, e esse número presente no arco é nomeado de peso. Por exemplo, se o arco possui o peso 2, o lugar conectado àquele arco deve possuir no mínimo duas fichas para disparar a transição. Após os requisitos terem sidos obedecidos e o disparo realizado, é removido a quantidade de fichas do lugar de origem de acordo com o peso do arco, e de acordo com o peso do arco para o lugar de destino, é depositada essa quantidade de fichas no lugar de destino. Por exemplo, se o arco de destino possui peso 1, o lugar de destino receberá 1 ficha.

O primeiro exemplo consiste na compreensão básica de uma leitura de uma modelagem de Rede de Petri. A Figura 2 representa o estado inicial e a Figura 3 denota o estado final. Em ambos os estados estão presentes dois lugares, uma transição e uma ficha, em que no estado inicial, a ficha está presente no lugar A e no estado final ela está presente no lugar B. Esta troca de lugares da ficha ocorre quando o evento é acionado, e para esse evento

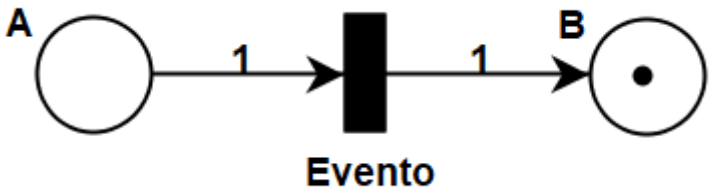
ser acionado, o lugar conectado anteriormente a ele deve possuir um número de fichas necessárias para conseguir disparar o evento. A indicação do número de fichas necessárias para disparar o evento é descrita com um número próximo da seta conectando o lugar com a transição, essa indicação é opcional se o número de fichas for 1.

Figura 2 – Estado inicial de uma Rede de Petri



Fonte: Autor (2020)

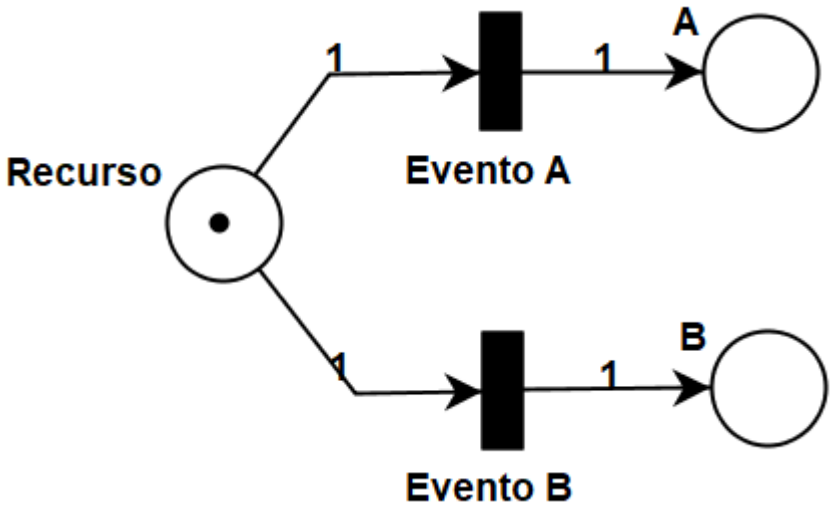
Figura 3 – Estado final de uma Rede de Petri



Fonte: Autor (2020)

O segundo exemplo consiste na compreensão da execução de uma modelagem com um recurso compartilhado entre dois objetos, demonstrado na Figura 4. Nesse exemplo é possível observar um lugar chamado “Recurso”, em que esse lugar é requisitado para a ativação de duas transições distintas. Quando esse cenário ocorre, há uma seleção aleatória pela Rede para determinar qual dos dois lugares receberá o recurso. Dessa maneira, tanto o lugar A pode receber o recurso ou o lugar B. Destaca-se que ao determinar qual transição ocorrerá, a outra não poderá ser executada, visto que a existência de fichas necessárias não será cumprida.

Figura 4 – Exemplo de recurso compartilhado entre lugares



Fonte: Autor (2020)

Dada as características de alguns sistemas em se tornarem confiáveis, técnicas de desenvolvimento de *software* foram desenvolvidas com esse propósito. Uma dessas técnicas é a aplicação das Redes de Petri. A modelagem gráfica com base na matemática, tornou possível prever o comportamento de sistemas, aumentando a qualidade e confiabilidade do *software* (SPIVEY, 2008).

2.4 Interfaces

Uma interface denota a ocorrência de contato físico ou conceitual de um usuário com um sistema ou com partes dele (BARBOSA e SILVA, 2010). O contato físico ocorre pela interação de um usuário com dispositivos de *hardware* e *softwares* ligados a ele, tais como teclado, *mouse e microfone.* Por outro lado, o contato conceitual está associado à percepção do usuário das ações realizadas ao longo do contato físico, de forma a permitir a tomada de decisões e os passos futuros a serem realizados a partir das respostas do sistema (BARBOSA e SILVA, 2010).

As características de uma boa interface estão associadas ao conceito de *affordance*, que denota o quão simples para o usuário é entender como interagir com um sistema/objeto mesmo na ausência de um contato prévio. Neste caso, as características físicas dos artefatos

evidenciam as formas de utilizá-los (BARBOSA e SILVA, 2010). Um exemplo de um objeto com uma boa *affordance* é uma maçaneta de uma porta, em que o usuário é capaz de entender como os processos de abrir e fechar sem prévia explicação.

Algumas características evidenciam a qualidade da interação do usuário com as interfaces (BARBOSA e SILVA, 2010). Uma delas é a usabilidade, que está associada tanto ao aprendizado, como ao uso da interface e a satisfação do usuário em relação ao uso (NIELSEN, 1993). A usabilidade se refere a fatores como a cognição do usuário e também as suas capacidades de compreender as respostas promovidas pelo sistema. Fatores esses que denotam aspectos emocionais e sentimentais do usuário e são denominados coletivamente de experiência do usuário (BARBOSA e SILVA, 2010).

3 METODOLOGIA

Para ter tornado o trabalho possível, foi feito inicialmente uma revisão bibliográfica mais detalhada sobre como a Rede de Petri funciona e suas características, para, dessa forma, ter permitido uma compreensão melhor acerca do tema para possibilitar o desenvolvimento da ferramenta de modo a modelar e simular uma Rede de Petri.

Com a análise sobre Rede de Petri feita, foi prosseguido para uma análise do *software* PIPE e uma análise de possíveis *softwares* concorrentes ao PIPE, de forma a alcançar uma visão ampla sobre esses ambientes e incorporar o melhor de cada possibilidade. Da análise dos *softwares* concorrentes foram obtidos dois *softwares* com propostas relacionadas: HPetriSim (HPETRISIM, 2020) e OPT Tools (TOOLS, 2020).

Com as análises realizadas, o projeto passou para a fase de desenvolvimento, que foi dividida em outras fases. A primeira delas foi a análise das tecnologias a serem utilizadas, sendo escolhidas as seguintes: Vue (VUE.JS, 2020) e JointJS (JOINTJS, 2020). O Vue foi utilizado para permitir a criação do *software* na Web de maneira mais flexível, visto que permite separar partes das funcionalidades em componentes e reutilizar esses componentes pelo *software*. O JointJS foi utilizado como biblioteca gráfica, onde nele foi possível acelerar o desenvolvimento, já que nele vêm embutidas várias funcionalidades importantes, como a criação, renderização e o gerenciamento de objetos gráficos, além do mais, seu diferencial é que ele possui funcionalidades exclusivas para quem deseja criar Redes de Petri, facilitando ainda mais o desenvolvimento.

Após ter elencado as tecnologias, foi desenvolvido um protótipo de média fidelidade para servir como um guia para o desenvolvimento do *software*. E, com o protótipo realizado, passou-se para o desenvolvimento de um *software* capaz de modelar e simular uma Rede de Petri, resultados esses que serão demonstrados na seção que aborda o desenvolvimento da ferramenta proposta. Para gerenciar esse desenvolvimento foi utilizado o método Kanban (TRELLO, 2020) com ajustes específicos para este trabalho com o auxílio do *Card Sorting* (USABILITY.GOV, 2020) para organizar os requisitos e o uso do Trello (TRELLO, 2020) para manter toda essa estrutura *online*. O método criado consiste em 3 principais fases: criação, desenvolvimento e término. Vale ressaltar, que este método foi escolhido com base nas opiniões experiências prévias do autor.

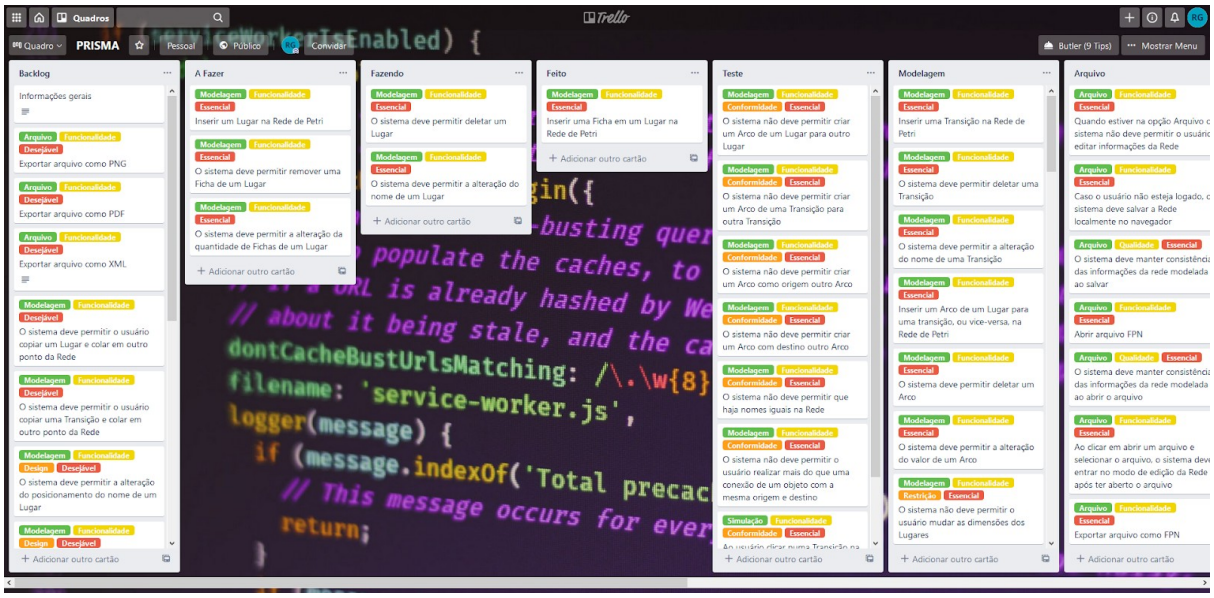
A fase criação consiste em descrever a funcionalidade em um cartão, colocar esse cartão na trilha de *Backlog*, e colocar as devidas *tags* no cartão, para, dessa maneira, permitir a organização dos cartões de acordo com a sua área. As *Tags* foram divididas em cores que representam categorias distintas, que são: verde, amarelo, laranja e vermelho. Verde representa uma funcionalidade mais abstrata do software. Essa cor possui as seguintes categorias: arquivo, autenticação, desenvolvimento, modelagem e simulação. Amarelo representa o que aquele cartão acrescenta ao sistema e possui as seguintes categorias: *bug*, funcionalidade e qualidade. Laranja representa uma subcategorização da *tag* amarela, para dessa maneira tornar mais claro o que está associado aquela funcionalidade, tendo as seguintes categorias: conformidade, design e restrição. Vermelho representa a prioridade do cartão e possui as seguintes categorias: urgente, essencial, importante e desejável.

A fase de desenvolvimento consiste em pegar um cartão do *Backlog* e colocar na trilha de “A Fazer”. Quando o cartão estivesse em desenvolvimento movia o cartão para a trilha “Fazendo”. Quando terminasse o desenvolvimento do conteúdo do cartão, esse era movido para a trilha “Feito”. Nessa trilha discutia-se com o orientador se aquela funcionalidade necessitava de realizar testes unitários, caso positivo movia o cartão para a trilha “Teste”.

A fase de término é mais simples, pois só é o processo de mover o cartão da trilha de Feito/Teste para a trilha com o nome da *tag* verde que aquele cartão possui. E quando esse processo é feito, representa que aquela funcionalidade foi completamente feita, e qualquer alteração que seja necessário deverá ser feita através da criação de outro cartão.

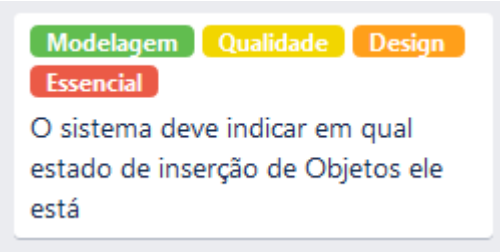
Todo esse processo no Trello pode ser visualizado na Figura 5, e pode ser visualizado com mais detalhes no seguinte link: https://trello.com/b/zzL01Eto/prisma. E, um exemplo de como um cartão de requisito é composto pode ser observado na Figura 6. Vale destacar que não será detalhado cada requisito pelo fato da grande quantidade de requisitos gerados. Dessa maneira, o foco voltou-se para o desenvolvimento da ferramenta.

Figura 5 – Representação do método de desenvolvimento



Fonte: Autor (2020)

Figura 6 – Composição de um cartão de requisito



Fonte: Autor (2020)

Por fim, com toda a base realizada e o *software* desenvolvido, foi redigida a monografia deste Trabalho de Conclusão de Curso, descrevendo os processos considerados para o desenvolvimento do projeto e os seus resultados, assim como os trabalhos futuros que podem ser desenvolvidos a partir deste.

4 TRABALHOS RELACIONADOS

Como citado anteriormente na introdução, as opções de ferramentas para modelar e simular Redes de Petri não são satisfatórias. Ao realizar pesquisas superficiais não foi possível achar tantas opções, e ao descobrir uma fonte (HAMBURG, 2020) que tenta manter uma lista das possíveis ferramentas de maneira centralizada, foi possível notar a falta de novas opções. Pois, ao realizar a checagem de todas as ferramentas da lista, foi possível observar ferramentas que só funcionam em computadores antigos, ferramentas que não são mais possíveis de obter, pois o *site* não está mais disponível, outras que só possuem o aspecto de modelar e outras para Rede de Petri Colorida, que é uma versão atualizada da Rede de Petri.

Depois de ter realizado os testes necessários, foi escolhido apenas três ferramentas. As ferramentas foram: PIPE, HPetriSim e OPT Tools. PIPE foi escolhida devido a ser a ferramenta base para o melhoramento, e ser uma das mais completas no quesito de modelar e simular Redes de Petri. HPetriSim foi escolhida por possuir uma melhor usabilidade, no sentido de apresentar um efeito mais intuitivo do que está acontecendo ao modelar. E, a OPT Tools foi escolhida por ser a única ferramenta que oferece o recurso de permitir modelar e simular Redes de Petri de maneira *online*.

Outras ferramentas foram desconsideradas devido a não acrescentar novos recursos em comparação ao PIPE, ou por serem para outros tipos de redes ou a versão disponível não é compatível com os computadores de hoje. Algumas dessas ferramentas descartas são: AlPiNA, Artifex e CPN Tools. Nesse sentido, nos próximos subtópicos, apresentam-se em mais detalhes as ferramentas selecionadas, abordando de maneira mais abrangente os concorrentes do PIPE e mais detalhes sobre a ferramenta PIPE.

4.1 HPetriSim

Nesta seção apresenta-se mais detalhes sobre a ferramenta HPetriSim e problemas notados durante o uso do *software*.

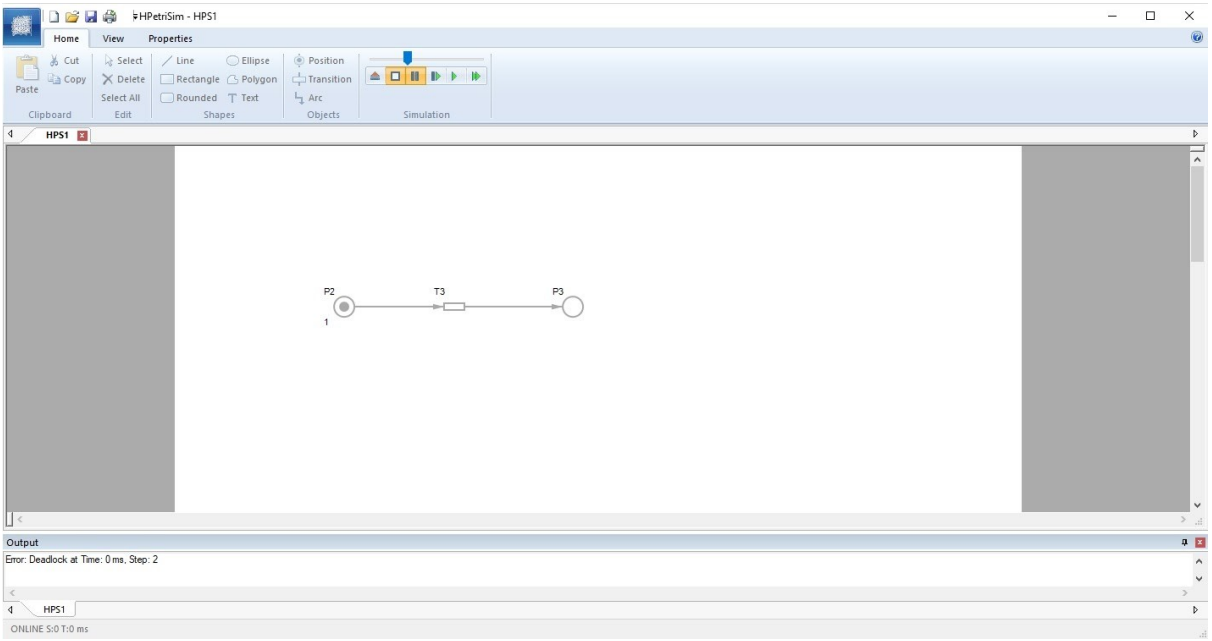
A funcionalidade que se demonstrou mais marcante em comparação com a ferramenta PIPE foi o efeito de animação ao ocorrer o disparo de um evento. No PIPE, o comportamento padrão é o de remover uma ficha do lugar de origem e adicionar essa ficha no lugar de destino. Enquanto que no HPetriSim, o comportamento é de remover uma ficha do lugar de

origem, realizar uma animação que faz percorrer essa ficha pelos arcos da origem ao destino, e adicionar a ficha no lugar de destino. Esse efeito torna mais simples para o usuário determinar a mudança de estado da rede.

No entanto, essa ferramenta apresenta vários outros problemas. A usabilidade e a interface não são agradáveis. Funcionalidades adicionais são acrescentadas à ferramenta, sendo que não ajudam a modelar a rede, e só acrescentam complexidade para a utilização da ferramenta. Como por exemplo, no menu de inserção de objetos, existe a opção de adicionar formas geométricas na rede, sendo que essas formas geométricas não interagem com a rede de nenhuma maneira.

Apesar de possuir um bom *feedback* visual na simulação, o ato de simular a rede não é trivial, e ela também não permite ao usuário decidir qual transição disparar durante a simulação, tudo é feito de maneira automática. Assim como a modelagem e edição da rede se demonstraram tarefas não triviais. Devido à impossibilidade de demonstração do efeito ao simular, só será demonstrado uma visão inicial da ferramenta, que pode ser observada na Figura 7.

Figura 7 – Interface da ferramenta HPetriSim



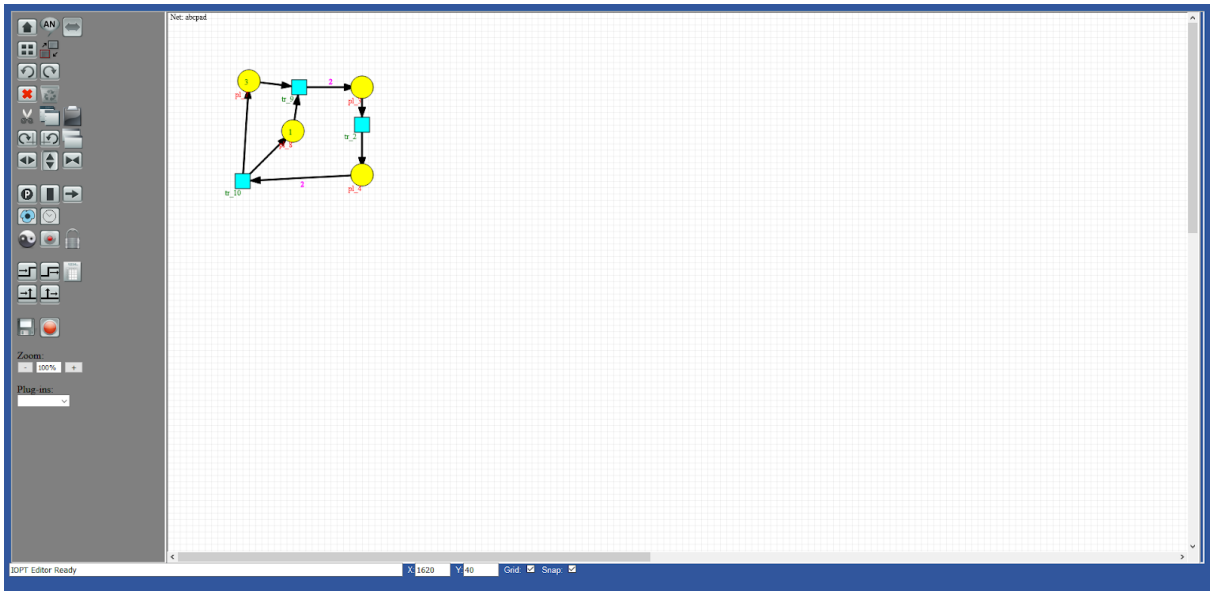
Fonte: Autor (2020)

4.2 OPT Tools

Nesta seção apresenta-se mais detalhes sobre a ferramenta OPT Tools e problemas notados durante o uso do *software*.

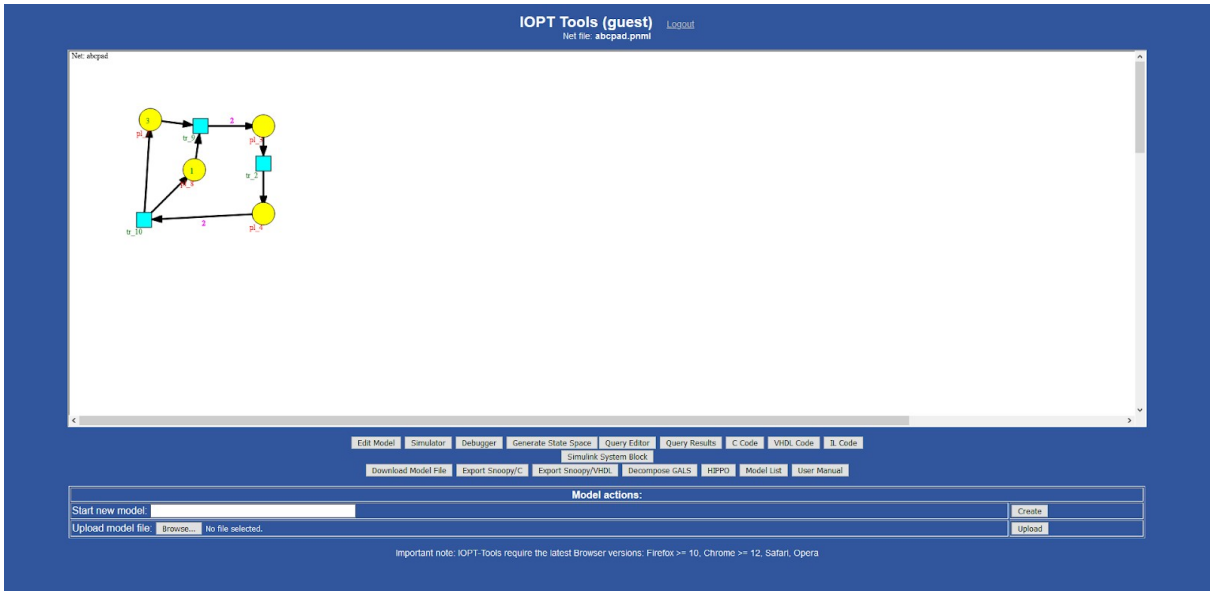
O maior fator de destaque dessa ferramenta foi o fato dela ter sido desenvolvida para sistemas Web, dessa maneira, o usuário não necessita instalar a ferramenta no seu computador. Em contrapartida, a interface do sistema não se demonstrou tão agradável, pois possui uma complexidade visual das funcionalidades e faltam indicações do que cada botão faz, já que a indicação do significado daquele botão se dá através de um ícone. Essa complexidade visual pode ser visualizada na Figura 8, em que pode ser vista a interface do sistema na modelagem da rede. O *design* do sistema também se demonstrou muito simplório, fato que pode ser tanto negativo quanto positivo. O lado positivo é que o tempo dedicado no desenvolvimento é mais focado nas funcionalidades do que no *design*. O lado negativo é que essa abordagem resulta no desinteresse por parte do usuário, já que um *design* mais atraente instiga mais a atenção do usuário. Esse *design* mais simples é possível observar na Figura 9, em que pode ser vista a interface inicial ao logar no sistema.

Figura 8 – Interface de modelagem do OPT Tools



Fonte: Autor (2020)

Figura 9 – Interface inicial do OPT Tools

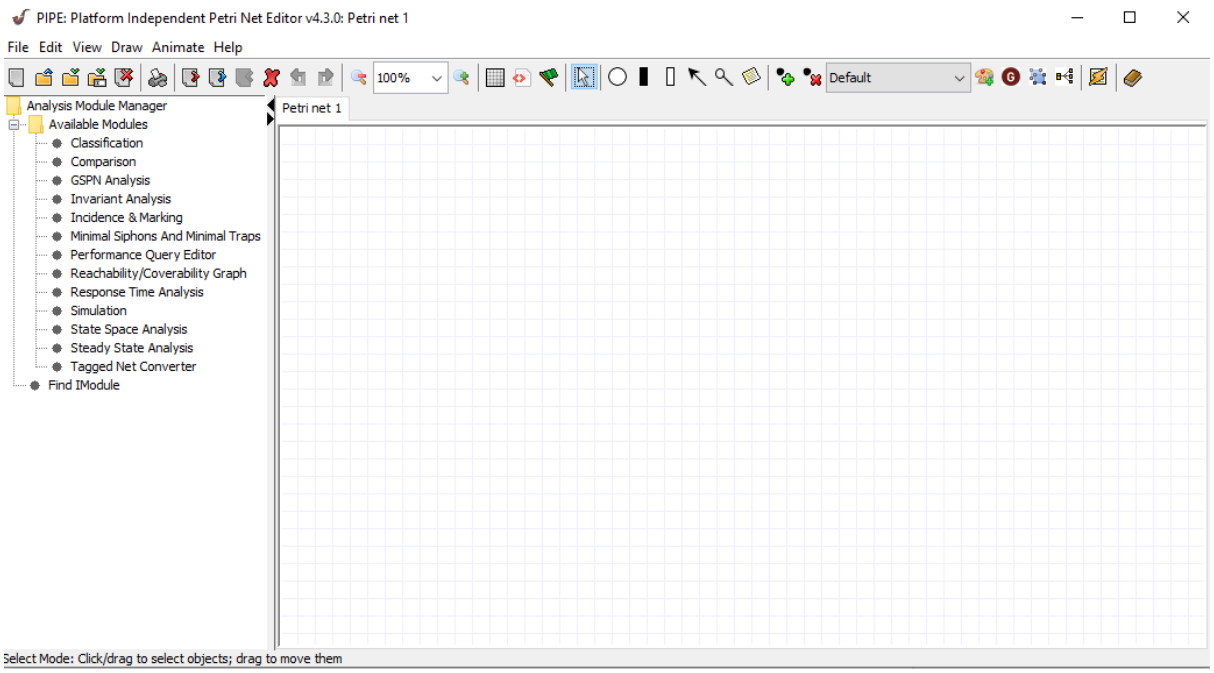


Fonte: Autor (2020)

4.3 PIPE

O PIPE foi criado em 2002, por estudantes de mestrado do I*mperial College of Science, Technology and Medicine* em Londres. Onde ele foi o resultado do trabalho de diversos estudantes durante os anos, que acarretou em diversas evoluções do *software*, desta maneira, o gabaritando na área de modelagem de Rede de Petri. O *software* permite modelar uma Rede de Petri, e a partir desta, conseguir simular a ocorrência dos eventos, permitindo a ativação das transições e o consumo das fichas, resultando em novos estados. O *software* também possui módulos mais complexos, como o cálculo de profundidade e complexidade da rede, determinação de *deadlock* da rede, entre outras funcionalidades. Uma breve visualização da interface do *software* pode ser vista na Figura 10.

Figura 10 – Interface do PIPE



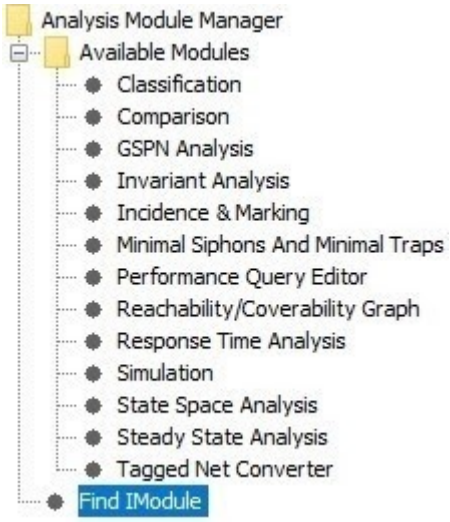
Fonte: Autor (2020)

O PIPE com o decorrer do tempo conseguiu agregar diversas funcionalidades fundamentais e importantes para a modelagem e simulação de Rede de Petri. No entanto, essa característica de pertencer a uma instituição também possui seus lados negativos, pois como é um projeto da instituição e não de uma pessoa, o desenvolvimento da ferramenta depende da vontade dos novos alunos em continuar o projeto. E, esse lado negativo é possível notar no repositório (TATTERSALL, 2020) do GitHub (GITHUB, 2020) do PIPE, em que já se passaram quatro anos até o momento desta escrita desde a última modificação do código fonte. O *software* atualmente está num estado com usabilidade deficitária. A versão 5 do projeto foi lançada em 2016 sem todas as funcionalidades e com falhas não corrigidas da versão 4 do projeto. Mas, mesmo com essas falhas, o PIPE ainda é bastante utilizado, devido ao reduzido número de seus concorrentes.

As funcionalidades do PIPE são diversas, mas o seu núcleo é ser um *software* capaz de modelar e simular Redes de Petri. Para permitir a modelagem da rede, são fornecidos meios para criar e remover Lugares, Transições, Fichas e Arcos. Também são fornecidas funcionalidades extras para facilitar a modelagem, como a possibilidade de adicionar comentários na rede, mover a rede, aumentar ou reduzir o tamanho da rede (*zoom*), diferenciar fichas por cores, entre outras funcionalidades. Já na parte de modelagem, é possível simular a

rede, desfazer o disparo de uma transição, refazer o disparo de uma transição, e avançar a rede para um estado aleatório. Já as outras funcionalidades são de análise da rede, em que as possibilidades podem ser observadas na Figura 11. O foco da demonstração das funcionalidades será o tratado na disciplina Programação Concorrente e Distribuída, às demais funcionalidades não são atacadas, uma vez que não são o foco do trabalho.

Figura 11 – Funcionalidades de análise do PIPE



Fonte: Autor (2020)

Apesar de o PIPE ser um *software* que se destaca na sua área, ele apresenta diversos problemas em relação ao seu uso. Os primeiros problemas percebidos é em relação a interface logo ao abrir o *software* (Figura 10). Nota-se que o foco de visão é dividido em duas partes, a área para a criação da rede e as funcionalidades de análise (Figura 11), isso pode se configurar um erro devido ao fato das funcionalidades de análise serem funcionalidades avançadas, e que nem sempre serão utilizadas por usuários iniciantes. Dessa maneira, ao decidir dar destaque às funcionalidades de análise, corre-se o risco do usuário iniciante tentar usá-las sem que tenha conhecimento prévio para realizar esse uso. Ainda sobre a interface, um problema notado foi a escolha de como demonstrar algumas funcionalidades por meio de ícones, pois com exceção dos ícones de inserção de objetos da modelagem, fica difícil uma primeira interpretação do que aquele ícone representa, como pode ser observado na Figura 12. Vale ressaltar que se o usuário estiver com o *mouse* sobre o ícone, após um tempo aparecerá uma mensagem informando brevemente o que é aquela funcionalidade. Existem também um menu de texto

logo acima do menu de ícones (Figura 10), mas essas características exigem que o usuário realize mais do que uma ação para identificar o que aquele ícone significa. A falta de significado para os ícones foi notada com maior ênfase no ícone de iniciar a simulação da rede, que é uma bandeira verde, o que pode fazer com o usuário tenha dificuldade de realizar a simulação da rede.

Figura 12 – Ícones de funcionalidades do PIPE

Fonte: Autor(2020)

Os outros problemas foram sendo notados durante o uso do PIPE, tais como de usabilidade e confiabilidade. Os principais problemas de usabilidade foram o fato de não ser fluido o processo de aumentar o tamanho da área da rede. Também nem sempre a criação dos arcos é estável. A fluidez no aumento do tamanho da rede não ocorre porque é necessário o usuário arrastar algum objeto da rede para além dos limites, já que o sistema não permite que o usuário ajuste para o tamanho desejado sem realizar os passos descritos. Já na criação dos arcos, ocorrem casos em que a rede contém vários objetos, e devido a algum fator, ao definir um objeto como destino para o arco, é escolhido outro objeto como destino, ou acontece das vértices do arco estarem desalinhadas. O principal problema de confiabilidade foi o fato de que nem sempre ao salvar a rede, o *software* realiza essa operação de maneira consistente, pois, em alguns casos, ao realizar as operações de salvar, fechar o *software* e abrir novamente, alguns arcos somem, os nomes modificados não são preservados e o caminho definido do arco não é preservado. Dessa maneira, resultando num usuário inseguro de utilizar o *software*.

5 PRISMA

Neste capítulo serão discutidos os resultados obtidos durante o desenvolvimento da ferramenta. Na Subseção 5.1 são discutidos os resultados da fase de prototipagem e na Subseção 5.2 são discutidos os resultados da fase de desenvolvimento do *software*.

5.1 Prototipagem

Para tornar o processo do desenvolvimento do *software* possível, foi realizado um protótipo de média fidelidade (BARBOSA e SILVA, 2010) para servir como base para o desenvolvimento das interfaces. A escolha pela prototipagem em média fidelidade em vez de alta foi tomada para direcionar o foco no desenvolvimento dos elementos principais da modelagem da ferramenta, em vez de detalhes visuais, tais como cores, ícones e outras características visuais do projeto. Pois, com um protótipo de média fidelidade, essas escolhas são implícitas, já que o foco é em como a interface do *software* deverá ser organizada e não como deve ser estilizada. Para fixar a diferença entre prototipagem de média e alta fidelidade, é que a prototipagem de média fidelidade é realizada com foco no posicionamento dos elementos da interface sem um detalhamento maior, como cores e ícones, podendo ou não permitir um meio para que o usuário interaja com o protótipo, já a prototipagem de alta fidelidade possui seu foco em demonstrar ao usuário como será o resultado final da interface, em que para isso são definidos todos os aspectos da interface, como cores, ícones e efeitos, assim como, são fornecidos meios para que o usuário possa interagir com o protótipo (BARBOSA e SILVA, 2010).

Na prototipagem, focou-se em organizar a estrutura visual do *software*, contendo três principais funcionalidades: arquivo, modelagem e simulação. Arquivo está relacionado a tudo que envolve manipular a informação da Rede gerada, como por exemplo: salvar, importar e exportar. A modelagem está relacionada à definição dos elementos da Rede. E, a simulação está relacionada com a execução da Rede.

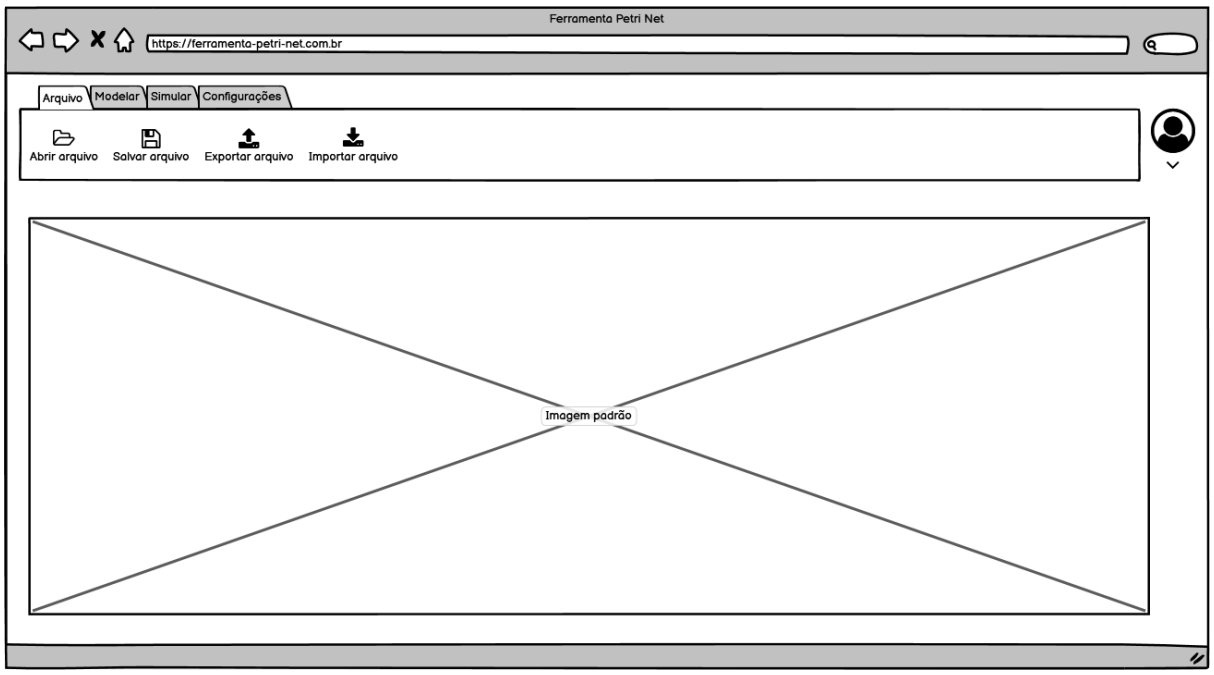
Para a prototipagem da ferramenta, foi utilizado o *software* Balsamiq (BALSAMIQ, 2020), que permite a prototipação rápida de *softwares*, contendo diversas funcionalidades que facilitam o processo de criação de protótipos de média fidelidade, podendo até permitir a interação do usuário com os protótipos. Trata-se de uma ferramenta paga, mas que permite o

uso gratuito durante 30 dias. Após esse período, a prototipagem pode ser consultada, mas não pode ser editada.

Tomando os pontos positivos e negativos analisados dos outros possíveis *softwares* para modelar e simular uma Rede de Petri, discutidos na Seção 4, os protótipos apresentados nas figuras abaixo, correspondem à idealização da ferramenta proposta.

Na Figura 13 pode ser vista a tela de entrada, na parte central é mostrado uma imagem padrão que é a logo da ferramenta. Na parte superior é apresentado um menu tabular dividido pelas funcionalidades do sistema, em que o ícone no lado direito representa a opção de logar no sistema.

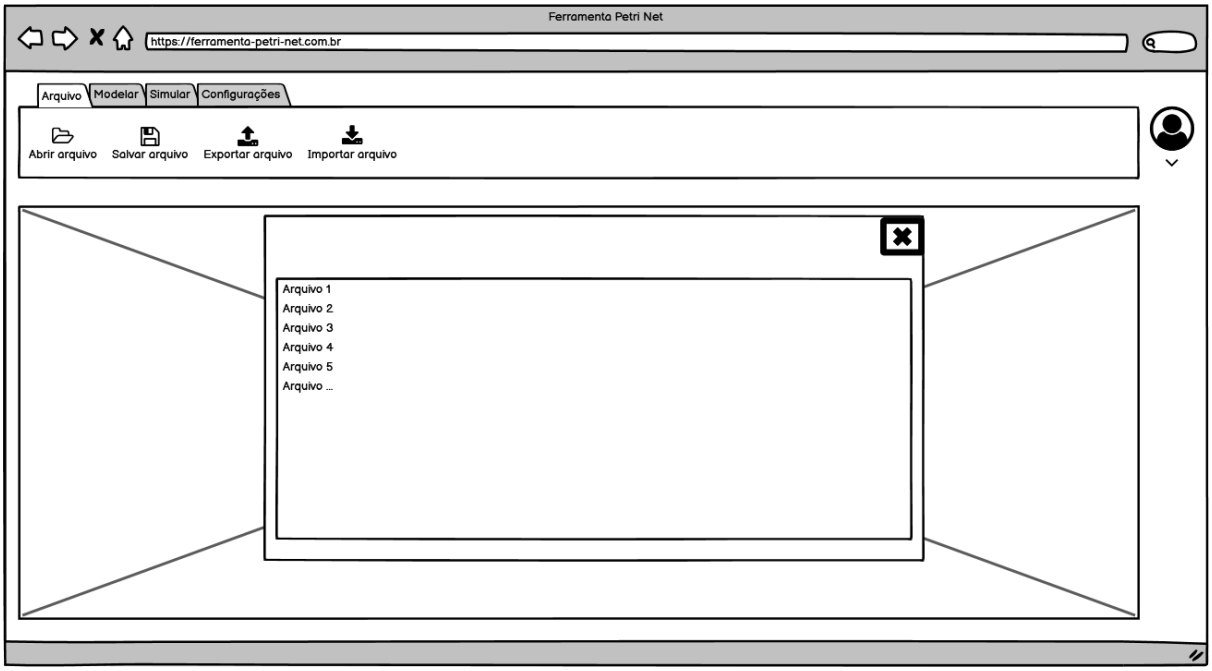
Figura 13 – Tela protótipo de entrada



Fonte: Autor (2020)

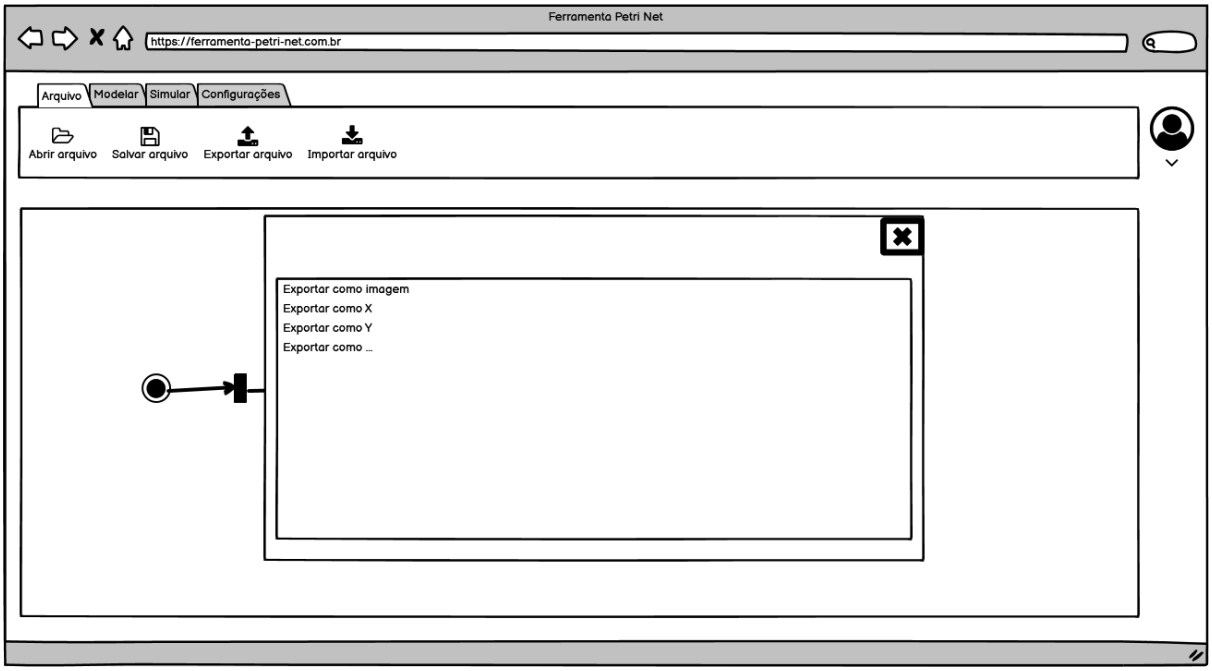
Na Figura 14 é demonstrada a *modal* que é aberta ao clicar em abrir arquivo, em que nesta *modal* são listados os arquivos salvos pelo usuário. Por sua vez, na Figura 15 é demonstrada a *modal* que é aberta ao clicar em exportar arquivo, em que nesta *modal* são listadas as opções de exportação da rede. Na Figura 16 é mostrada a *modal* que é aberta ao clicar em importar arquivo, em que nesta *modal* é aberta uma janela para selecionar um arquivo no computador do usuário.

Figura 14 – Tela protótipo de abrir arquivo



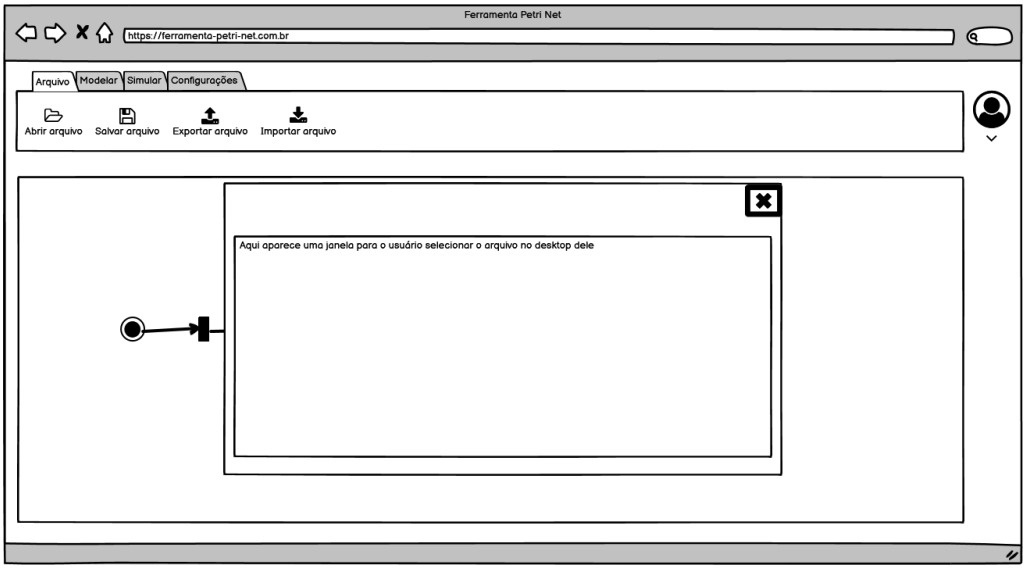
Fonte: Autor (2020)

Figura 15 – Tela protótipo de exportar arquivo



Fonte: Autor (2020)

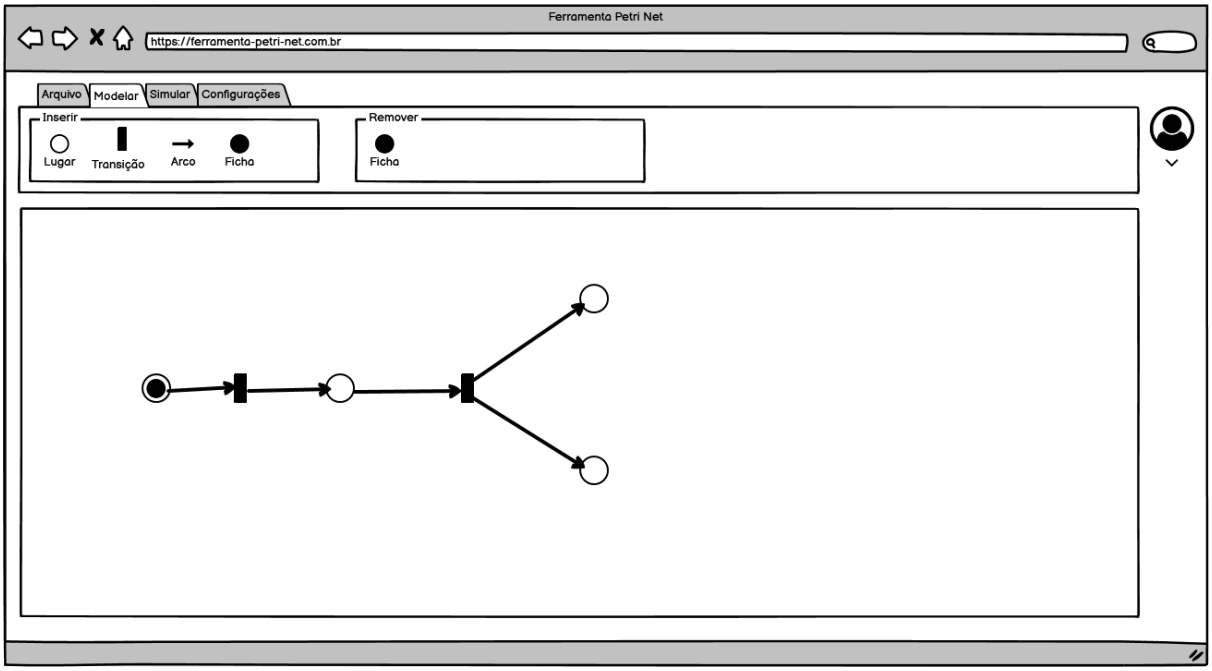
Figura 16 – Tela protótipo de importar arquivo



Fonte: Autor (2020)

Na Figura 17 é representada a tela de modelagem de uma rede, em que na parte superior são posicionadas as opções para inserir e remover objetos na rede.

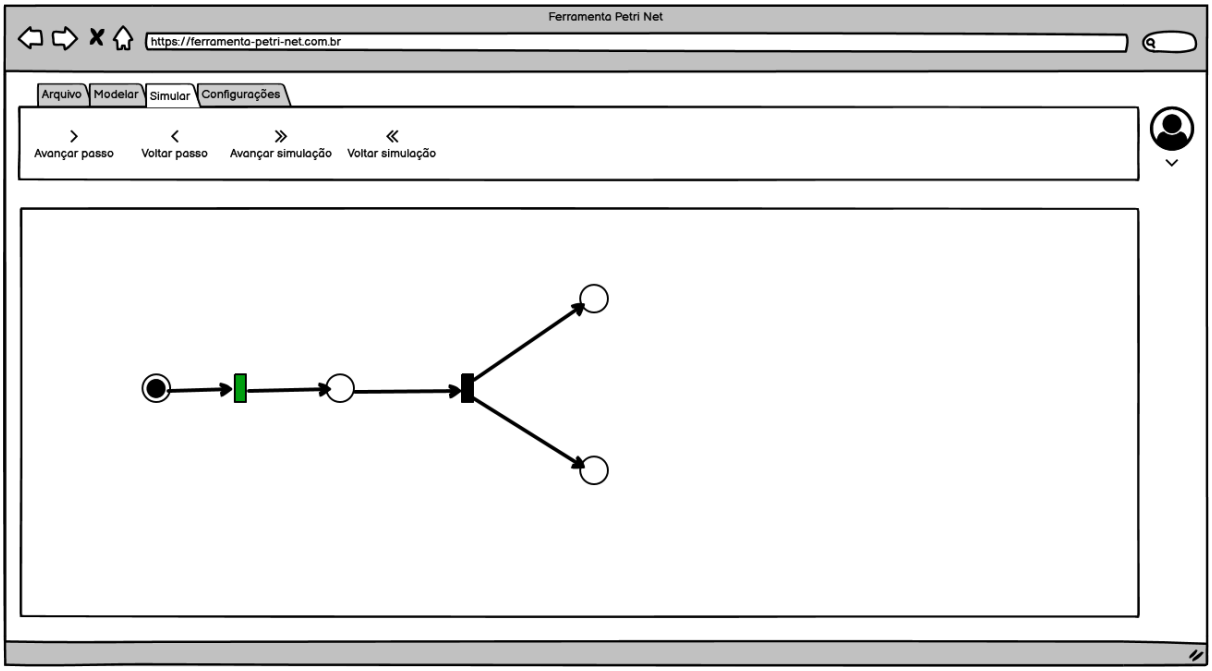
Figura 17 – Tela protótipo de modelar



Fonte: Autor (2020)

Na Figura 18 é representada a tela de simulação de uma rede, em que na parte superior são posicionadas as opções de simulação. Com a característica de indicar a possibilidade de disparo de transição com a mudança da cor da transição de preto para verde.

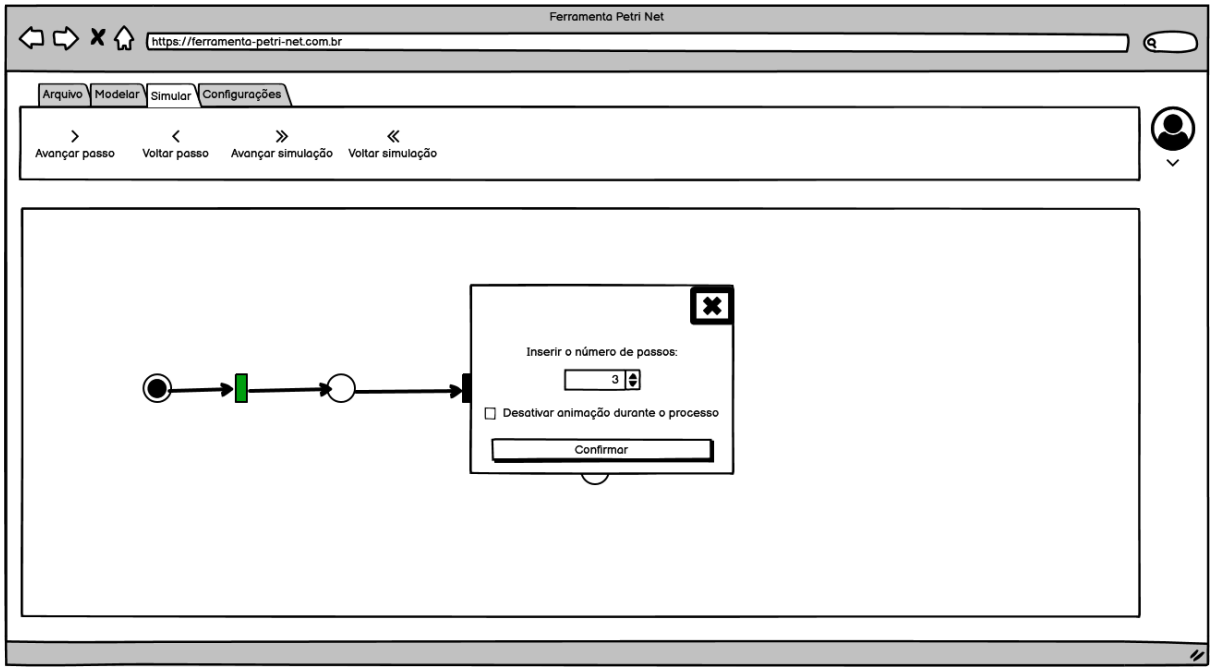
Figura 18 – Tela protótipo de simular



Fonte: Autor (2020)

Por sua vez, na Figura 19 é apresentada a *modal* que é aberta ao clicar em avançar simulação ou voltar simulação, onde nesta *modal* o usuário pode indicar a quantidade de passos a serem disparados, e possui a opção de desativar a animação durante os disparos. Vale ressaltar que ao avançar a simulação, os disparos são feitos de forma aleatória.

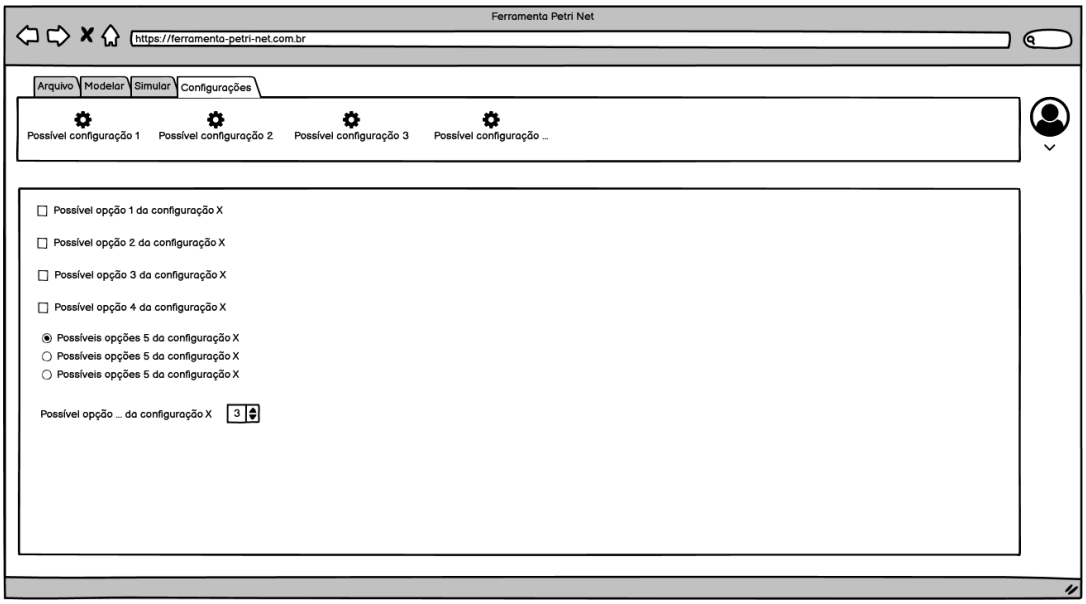
Figura 19 – Tela protótipo de avançar/voltar simulação



Fonte: Autor (2020)

Na Figura 20 são exibidas as possíveis configurações da ferramenta, em que na parte superior é apresentada cada possível opção de configuração da rede, e ao clicar nessa opção são demonstradas na parte central as configurações daquela opção.

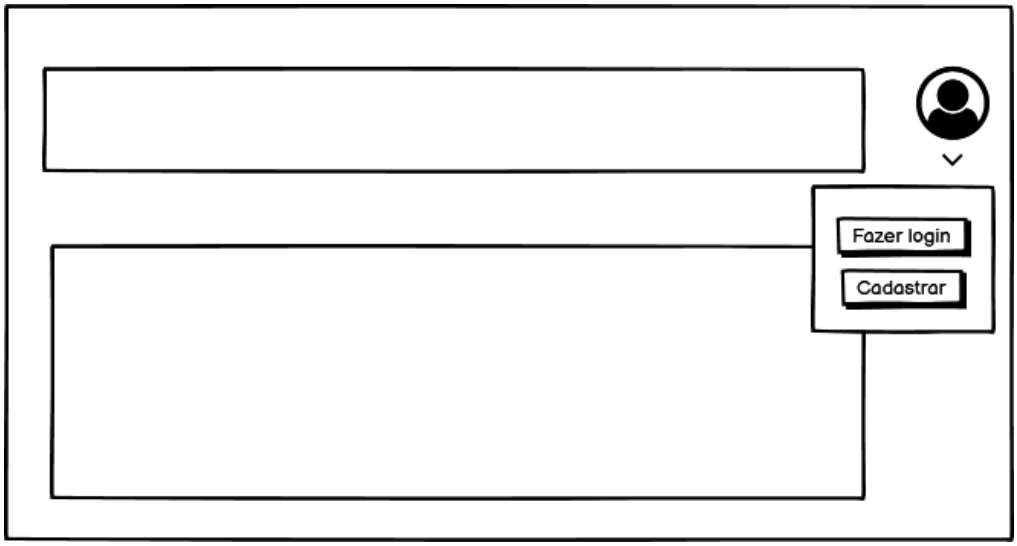
Figura 20 – Tela protótipo de configurações



Fonte: Autor (2020)

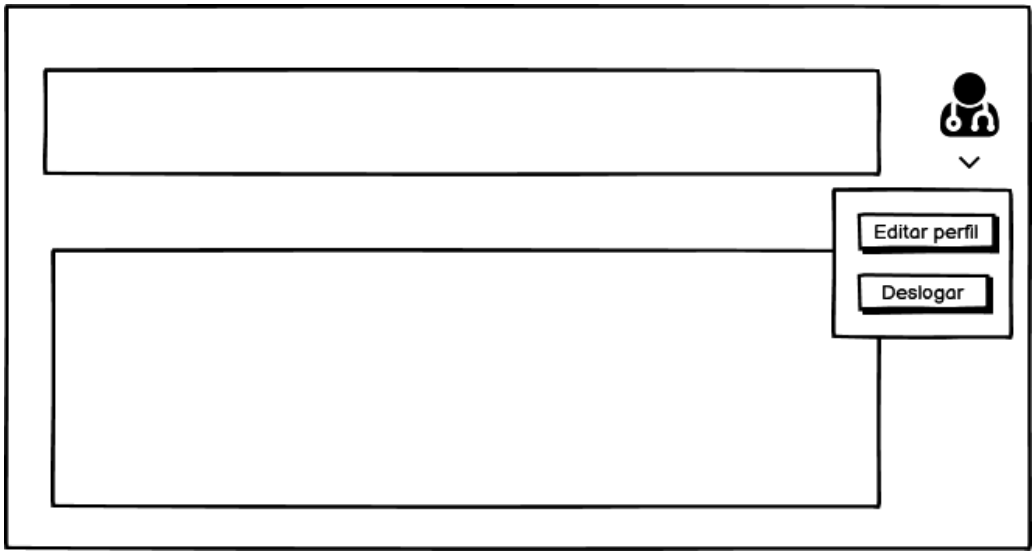
Na Figura 21 é representada a visualização do menu de logar ao usuário passar o mouse em cima. Em que são demonstradas as opções caso o usuário não esteja logado, e na Figura 22 é representada esta mesma ação caso o usuário esteja logado.

Figura 21 – Tela protótipo de usuário não logado *hover*



Fonte: Autor (2020)

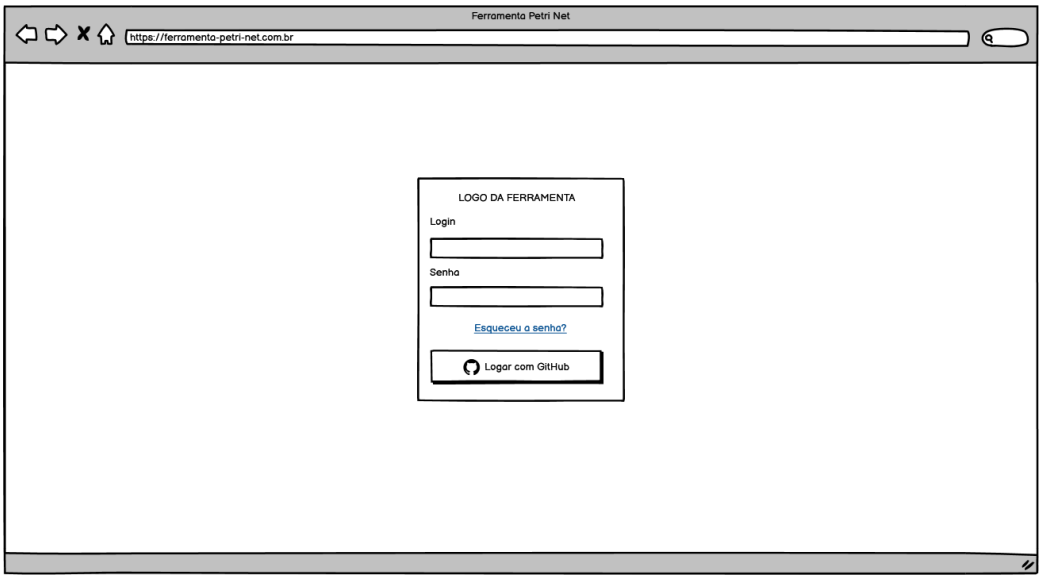
Figura 22 – Tela protótipo de usuário logado *hover*



Fonte: Autor (2020)

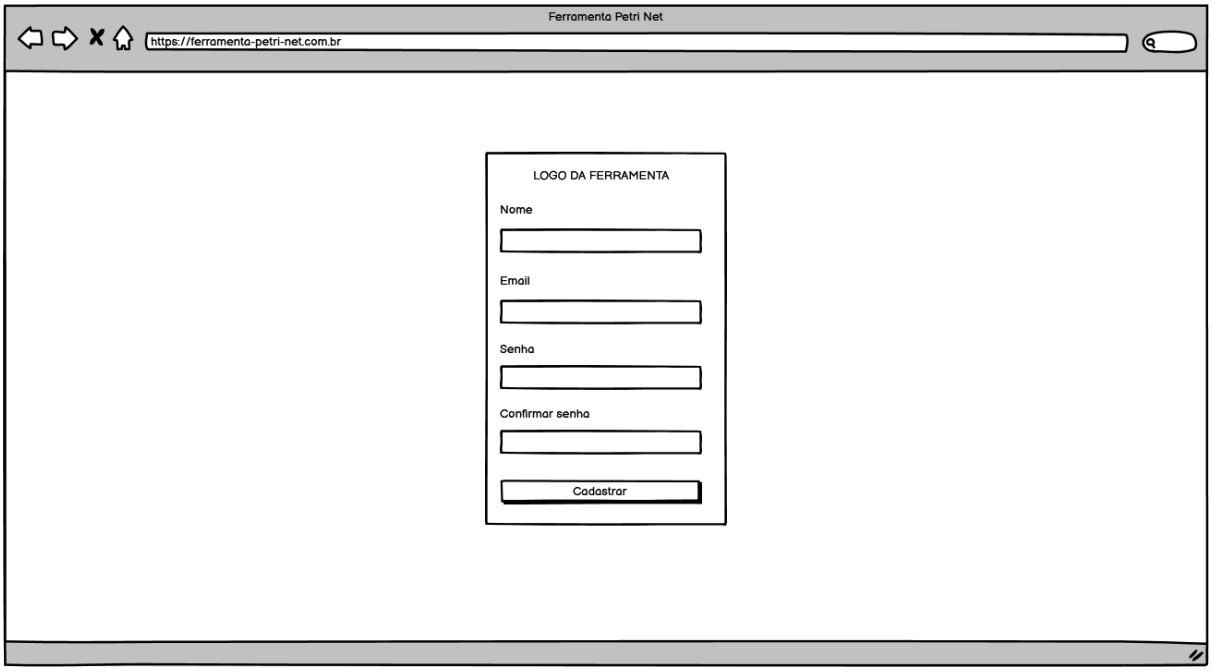
Na Figura 23 é representada a tela de *login*, em que nela o usuário pode logar com seu GitHub. A Figura 24 representa a tela de cadastrar o usuário e na Figura 25 é representada a tela de atualizar as informações do usuário.

Figura 23 – Tela protótipo logar usuário



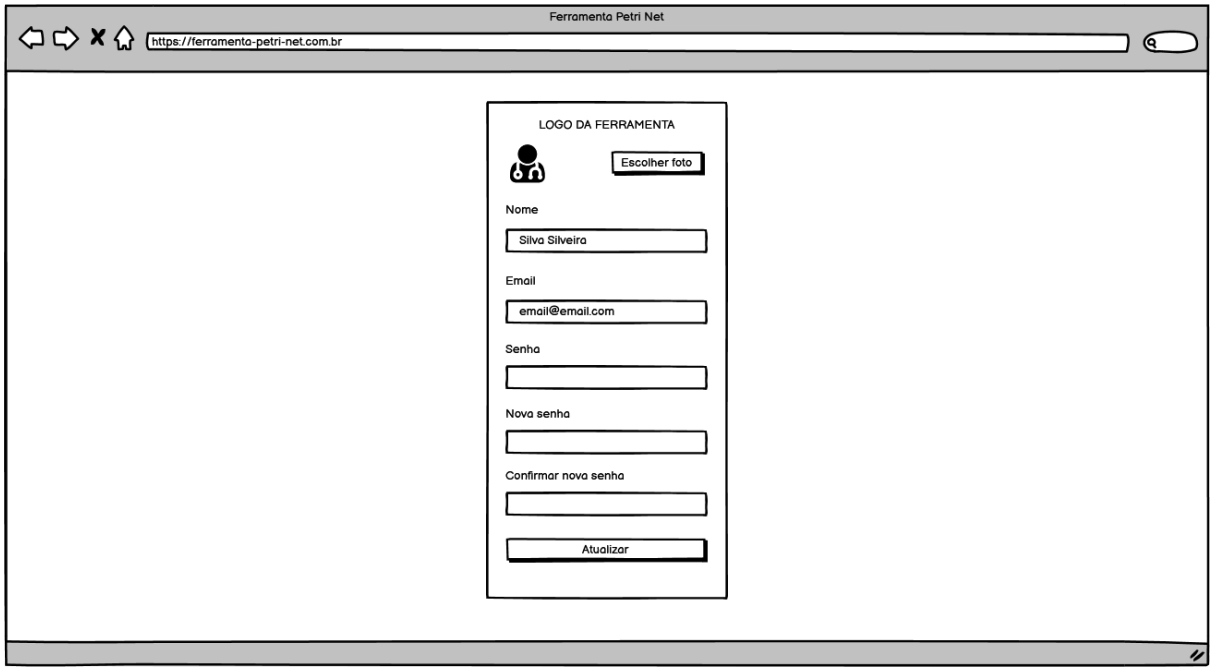
Fonte: Autor (2020)

Figura 24 – Tela protótipo de cadastrar usuário



Fonte: Autor (2020)

Figura 25 – Tela protótipo de atualizar perfil de usuário



Fonte: Autor (2020)

5.2 Desenvolvimento da ferramenta

Após o término da fase de prototipagem das telas da ferramenta, deu-se início a fase de desenvolvimento do *software*. O código fonte gerado, acompanhado das instruções para usar e instalar estão no repositório do GitHub: https://github.com/rarysson/PRISMA. Vale ressaltar que diferente do PIPE, o PRISMA é de propriedade do autor deste trabalho e não da instituição, dessa forma, futuras atualizações e correções poderão ser feitas sem a necessidade da alocação de novos alunos. Outros desenvolvedores/pesquisadores, além do autor deste trabalho, podem contribuir com o projeto, relatando problemas e fornecendo sugestões no repositório. O projeto também está sob uma licença bastante permissiva, permitindo que outras pessoas criem versões diferentes do *software* a partir do código base sem a necessidade de informar ao autor.

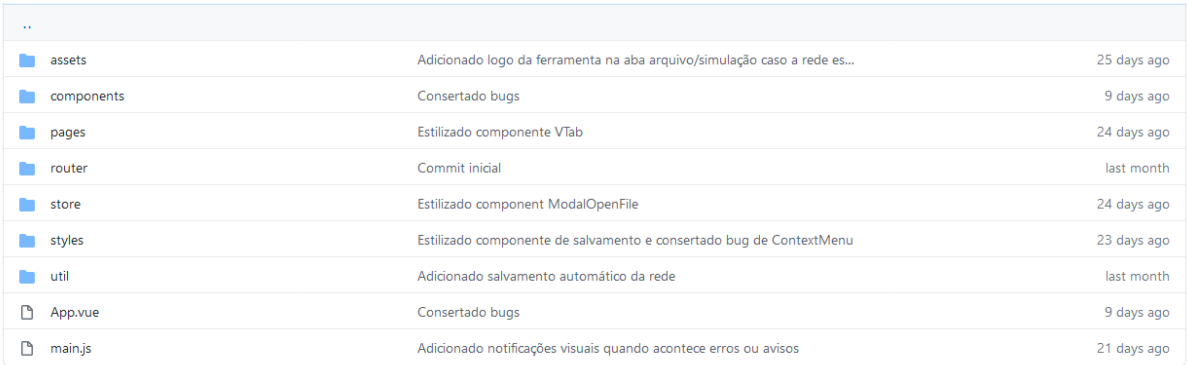
Visando a possibilidade de outras pessoas estenderem o projeto, foram definidas algumas configurações de ambiente para fornecer uniformidade ao código base. Para o desenvolvimento da ferramenta, foram utilizados *plugins* e configurações do editor de código empregado, o Visual Studio Code (MICROSOFT, 2020), que foi escolhido devido a sua popularidade e facilidade de uso. Os *plugins* e configurações utilizadas foram as seguintes:

Prettier (PRETTIER, 2020), ESLint (ESLINT, 2020), stylelint (STYLELINT, 2020), EditorConfig (EDITORCONFIG, 2020). Prettier é utilizado para formatar o código sem a necessidade de muitas configurações. ESLint é utilizado para detectar possíveis erros do JavaScript. O stylelint é utilizado para detectar possíveis erros do CSS. E o EditorConfig é utilizado para deixar uniforme o espaçamento dos arquivos.

Como citado anteriormente, a escolha da ferramenta para tornar o desenvolvimento mais flexível foi o Vue. O principal fator pela sua escolha foi a sua baixa curva de aprendizado em relação às outras ferramentas. Sua baixa curva de aprendizado se dá pelo fato de não ser necessário aprender outras técnicas além da base da Web: HTML, CSS e JavaScript, e a própria ferramenta. Já as outras opções necessitam aprender outras técnicas ou mudar a forma de como é utilizado a base da Web.

Com o Vue é possível dividir funcionalidades em componentes e a partir disso reutilizá-los por toda a aplicação. Os elementos que compõem a ferramenta foram organizados como pode ser visto na Figura 26. Cada ícone azul representa uma pasta que possui outros arquivos ou pastas. A pasta *assets* (ativos) contém os recursos do projeto, tais como imagens. A pasta *components* contém os componentes utilizados para compor as diferentes páginas ou outros componentes de ordem mais alta, em que cada componente representa uma possível funcionalidade. A pasta *pages* contém as páginas que o usuário pode acessar. A pasta router contém as rotas das páginas que o usuário pode acessar. A pasta *store* contém os possíveis estados compartilhados na aplicação, tais como a rede. A pasta *styles* contém as regras de CSS que se aplicam a toda aplicação. A pasta *util* contém funções utilitárias que podem ser utilizadas por toda aplicação, tais como o uso de banco de dados local do navegador.

Figura 26 – Estrutura do código base do projeto



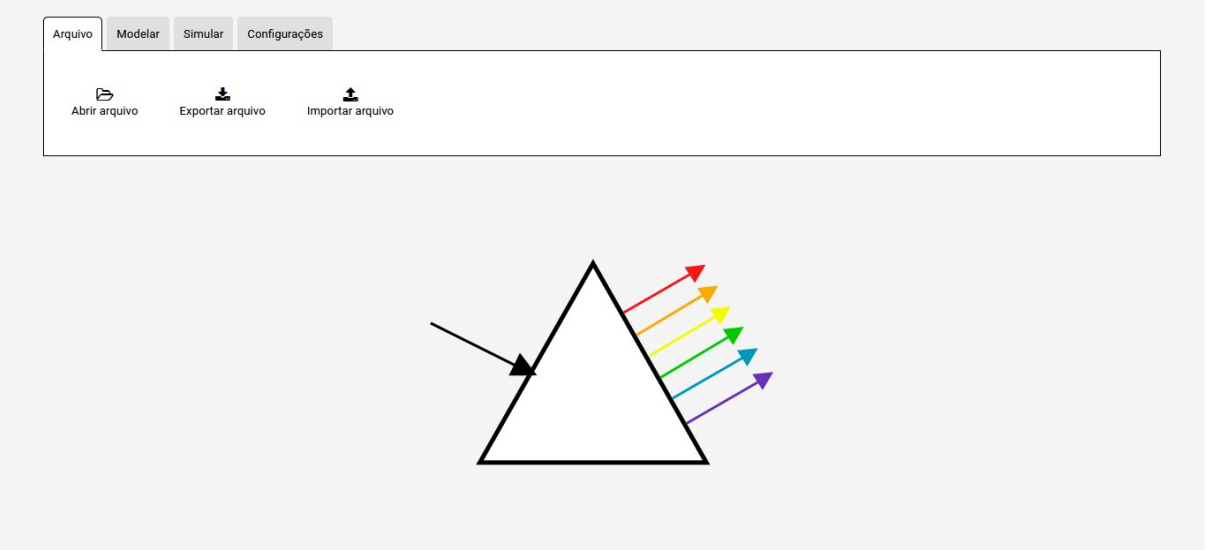
Fonte: Autor (2020)

Optou-se por não projetar uma versão para dispositivos móveis da ferramenta. Essa decisão foi tomada levando em consideração que pode não ser tão simples o processo de modelar e simular uma rede em uma tela de resolução baixa, e também pelo fato dessa ação de modelar não ser uma ação cotidiana que o usuário queira realizar durante o seu dia, já que teoricamente ele modelará em situações específicas. Dessa forma, optou-se por não ajustar o *design* para dispositivos móveis, que de outra forma adicionaria complexidade ao projeto do *software*. Uma versão móvel da ferramenta pode ser elaborada como trabalho futuro.

Após a codificação, foi possível obter um software capaz de modelar e simular Redes de Petri. O *software* é capaz de cumprir os requisitos essenciais para a modelagem e simulação de uma Rede de Petri. Para demonstrar esse resultado, é apresentada uma listagem de figuras, demonstrando as telas criadas, e descrições de como o *design* gerado se relaciona com o protótipo idealizado e também alguns detalhes importantes para entender os detalhes da ferramenta.

Na Figura 27 é possível observar a tela inicial que o usuário verá ao entrar na aplicação. Nela são demonstradas as principais funcionalidades relacionadas ao arquivo PRISMA, tais como abrir, importar e exportar, assim como, também é apresentada a logo da aplicação, caso não tenha nenhuma rede em criação. O arquivo PRISMA gerado pela aplicação possui como extensão “.prisma” para tornar o processo de manuseio de arquivos mais intuitivo, mas, apesar disso, a sua estrutura é de um arquivo JSON, por ser um formato padrão de armazenamento e transferência de informações e também é o formato gerado pela biblioteca JointJS (utilizada internamente pela ferramenta). Essa troca de extensão só foi realizada para evitar a leitura incorreta de outros arquivos JSON que não são relacionados à ferramenta. Decidiu-se remover a opção de salvar arquivo da aba de Arquivo, e transferi-la para a aba de Modelar (como será possível observar em figuras posteriores), pois foi notado que o processo de alternar entre abas para salvar a Rede acabaria se tornando uma tarefa que poderia dificultar a interação do usuário com a ferramenta, pôr a opção salvar o estado da rede está muito relacionado à modelagem.

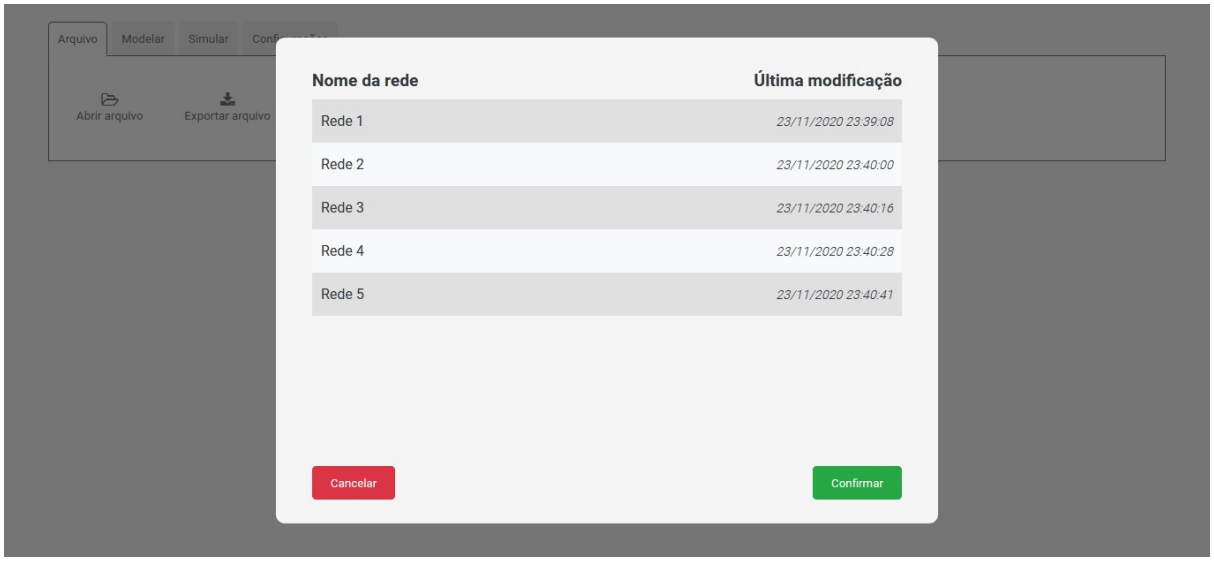
Figura 27 – Tela inicial desenvolvida



Fonte: Autor (2020)

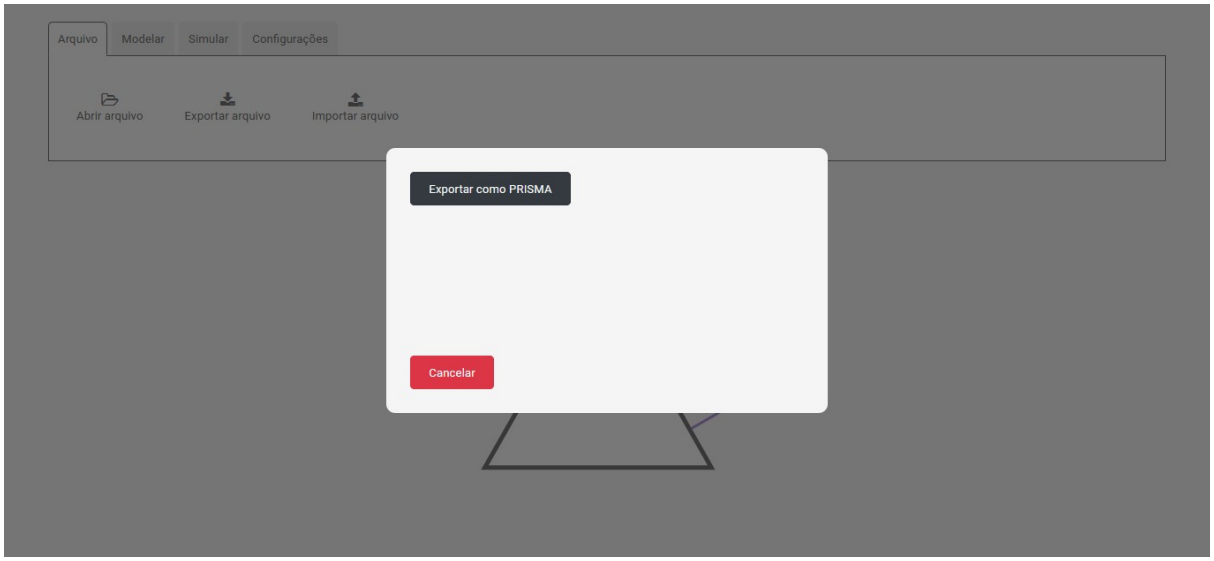
Como citado, as funcionalidades do arquivo são: abrir, exportar e importar. Abrir arquivo está relacionado a ação de abrir uma rede salva localmente no navegador do usuário. Exportar arquivo está relacionado em transformar a rede em um arquivo PRISMA e permitir o *download* do arquivo no computador do usuário. Já importar arquivos está relacionado em ler arquivos PRISMA do computador do usuário e transferir para o navegador. Essas funcionalidades podem ser vistas na Figura 28, Figura 29 e Figura 30, respectivamente. As redes são salvas localmente no navegador do usuário em vez de um servidor próprio. Essa decisão foi tomada para focar o trabalho na parte visual e funcional do projeto. No entanto, a Rede é salva de maneira permanente, dessa forma, o usuário poderá desligar a sua máquina e quando voltar a Rede estará no estado que ele deixou. Ainda com relação a salvar localmente na máquina do usuário, foi analisado uma integração com o GitHub, em que seria possível salvar a rede nos arquivos do GitHub do usuário, mas após análises de como é realizada essa integração, foi decidido não realizar, pois seria necessário requisitar permissões de acesso a informações privadas do usuário, por esse fator de privacidade foi decidido não realizar essa integração com o GitHub.

Figura 28 – Tela desenvolvida de abrir arquivo



Fonte: Autor (2020)

Figura 29 – Tela desenvolvida de exportar arquivo



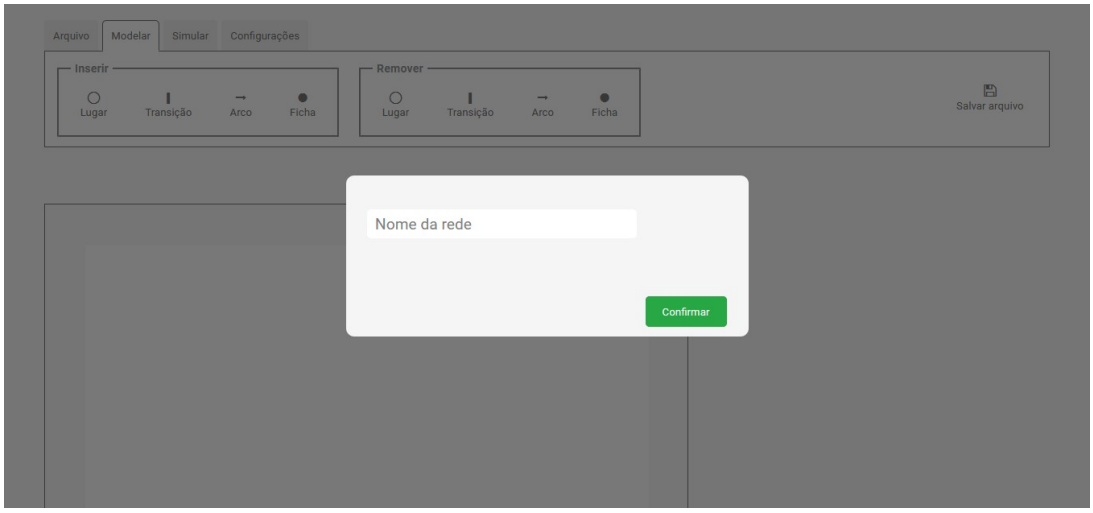
Fonte: Autor (2020)

Figura 30 – Tela desenvolvida de importar arquivo

Fonte: Autor (2020)

Na aba de Modelar foi gasta a maior parte do tempo na codificação. Nesta aba se concentra a maior parte da lógica da aplicação, devido ao processo de criação da rede envolver mais interações entre estados e componentes. Se o usuário decidir abrir a aba de Modelar sem ter aberto um arquivo existente, aparecerá uma *modal* (*popup*) que não fechará até que o usuário digite o nome da Rede. Essa decisão foi tomada, pois não existem outras oportunidades para o usuário indicar o nome da Rede, já que não existe uma etapa de criação de Rede, e toda Rede deve estar vinculada a um nome para tornar possível o processo de salvar. Essa *modal* pode ser observada na Figura 31.

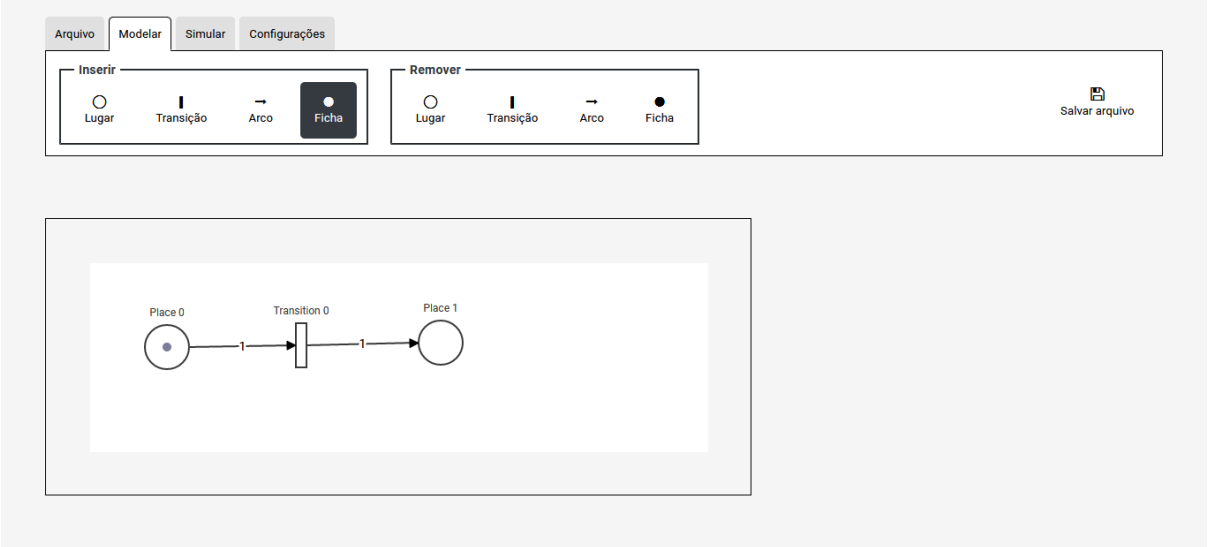
Figura 31 – Tela desenvolvida de criar Rede



Fonte: Autor (2020)

Com o nome definido, é possível começar a criar e editar a Rede. Para tornar isso possível, são fornecidas funcionalidades de criar e remover: lugares, transições, arcos e fichas. Só é possível estar em uma dessas funcionalidades em um determinado instante, e o sistema indicará em qual funcionalidade você está através de uma indicação visual. Também é possível editar o tamanho da rede, para dessa forma poder comportar todos os objetos necessários. Esse processo é possível observar na Figura 32.

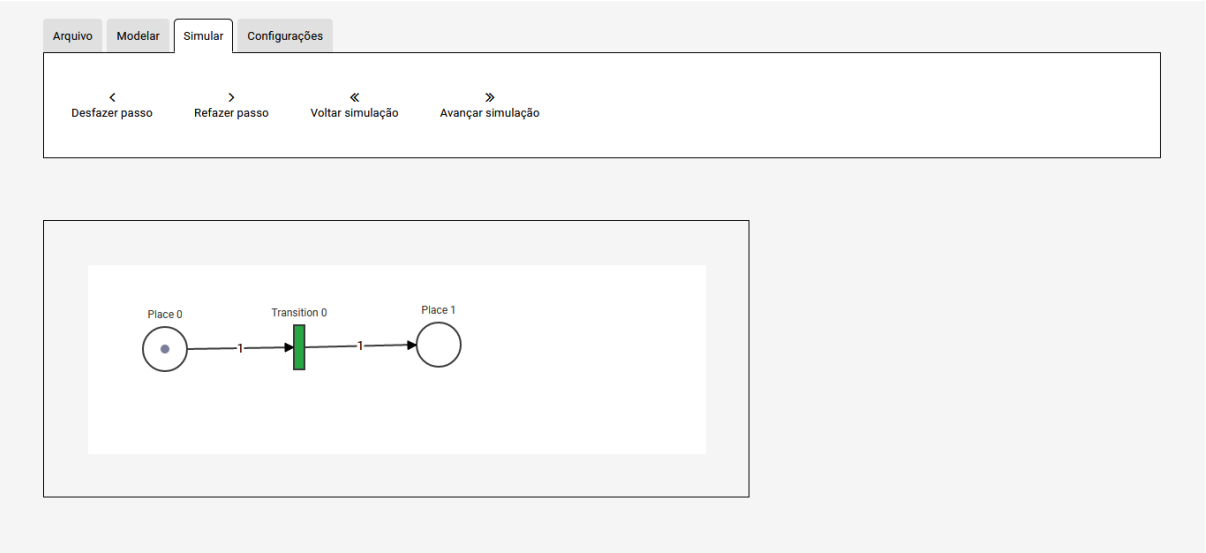
Figura 32 – Tela desenvolvida de modelar Rede



Fonte: Autor (2020)

Após definida a estrutura da rede, é possível simular os estados dela na aba Simular. Para isso são fornecidas funcionalidades de avançar o estado da rede através da interação direta, desfazer o passo feito e refazer o passo desfeito, em que um passo significa uma alteração de estado da rede. Também é permitido avançar a Rede em um número de passos definidos, em que será disparado as transições de forma aleatória, e é possível voltar a Rede em um número de passos definidos. Essas funcionalidades podem ser observadas na Figura 33 e Figura 34. Na Figura 34, pode ser vista tanto a tela de avançar a simulação quanto voltar a simulação.

Figura 33 – Tela desenvolvida de simular Rede



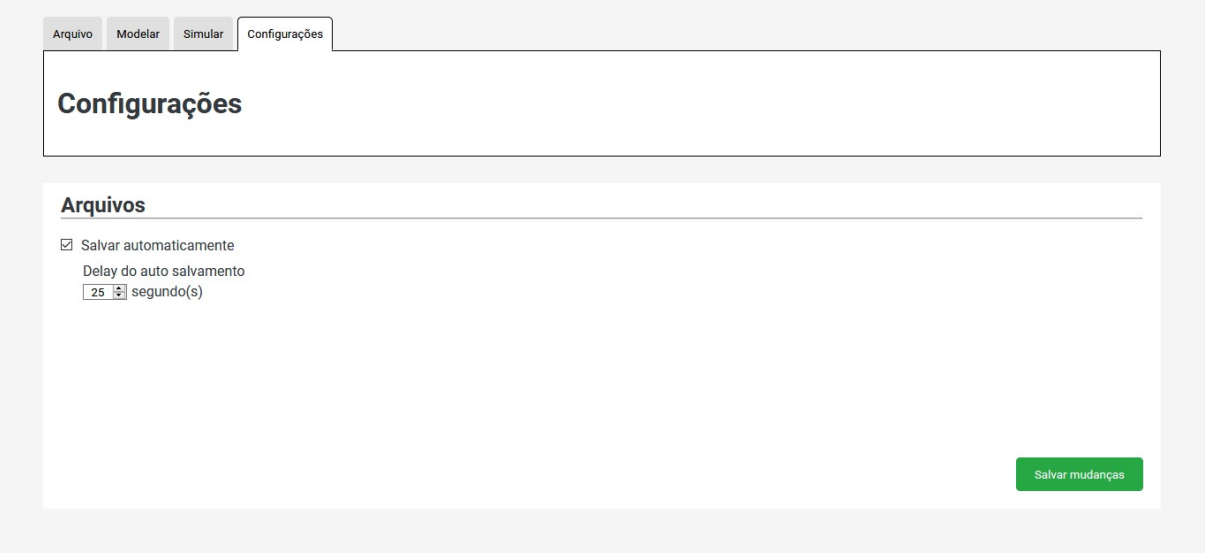
Fonte: Autor (2020)

Figura 34 – Tela desenvolvida de avançar/voltar simulação

Fonte: Autor (2020)

Por fim, na Figura 35 é demonstrada a aba de Configurações, em que foi decidido separar as configurações por seções em vez de opções, como indicado na Figura 20.

Figura 35 – Tela desenvolvida de configurações



Fonte: Autor (2020)

6 CONCLUSÃO

Este trabalho teve como seu foco o desenvolvimento de uma ferramenta capaz de modelar e simular Redes de Petri como uma alternativa ao *software* PIPE. Para o desenvolvimento da ferramenta, foram seguidas técnicas e processos definidos, tais como uma metodologia de *software*, ferramentas para gerenciar requisitos e o progresso do desenvolvimento. Também foram analisados outros *softwares* na área de modelagem e simulação de Rede de Petri para descobrir pontos positivos que poderiam ser integrados no resultado final, assim como, foram analisados detalhes do *software* PIPE. Além das análises, foi utilizada a técnica de prototipagem de médio nível para gerar um artefato capaz de servir como base da interface para o desenvolvimento final da ferramenta, assim como, foram utilizadas ferramentas para garantir a qualidade e uniformidade do código base.

Com todas essas etapas concluídas, foi possível obter uma ferramenta capaz de modelar e simular Redes de Petri, com diferenciais de facilidade de acesso e simplicidade de interface. A facilidade de acesso se dá pelo fato de só ser necessário o uso do navegador para usar a ferramenta. A simplicidade da interface se dá através da separação de funcionalidades por temas semelhantes, indicação visual mais precisa do que uma funcionalidade faz e um melhor *feedback* visual.

Como trabalhos futuros, pretende-se adicionar outras funcionalidades à ferramenta, tais como: exportar a rede como PNG, importar arquivos XML e exportar arquivo XML, formato gerado pelo PIPE. A ferramenta pode servir como base para outros projetos de outros alunos da instituição, a citar: o desenvolvimento de um servidor para os usuários da ferramenta, análises da UI e UX da ferramenta, o desenvolvimento de um protótipo de alto nível mais detalhado e com outras funcionalidades e a evolução da ferramenta com outras funcionalidades presentes no PIPE e que não estão no PRISMA.

REFERÊNCIAS

BALSAMIQ. **Balsamiq**. Disponível em: https://balsamiq.com/. Acesso em: 21 nov. 2020.

BARBOSA, S; SILVA, B. **Interação Humano-Computador**. Elsevier Brasil. 2020.

BONETE, P.; LLADO, C. M.; PUIJANER, R.; KNOTTENBELT, W. J. **PIPE v2.5: A Petri Net Tool for Performance Modelling**. Proc. 23rd Latin American Conference on Informatics (CLEI 2007), San Jose, Costa Rica, October 2007.

CARDOSO, Janette; VALLETE, Robert. **Redes de Petri**. Editora da UFSC, 1997.

COLOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos: conceitos e projeto**. 5ª ed. Porto Alegre: Bookman, 2013.

DEITEL, H.; DEITEL, P.; STEINBUHLER, K. **Sistemas Operacionais**. 3ª ed. São Paulo: Prentice Hall, 2005.

DROPBOX. **Dropbox**. Disponível em: https://www.dropbox.com. Acesso em: 26 jul. 2020.

EDITORCONFIG. **EditorConfig**. Disponível em: https://editorconfig.org/. Acesso em: 23 nov. 2020.

ELIXIR. **Elixir**. Disponível em: https://elixir-lang.org/. Acesso em: 22 jul. 2020.

ESLINT. **ESLint**. Disponível em: https://eslint.org/. Acesso em: 23 nov. 2020.

FREITAS, Henrique Cota de; ALVES, Marco Antonio Zanata; NAVAUX, Philippe Olivier Alexandre. **NoC e NUCA: Conceitos e Tendências para Arquiteturas de Processadores Many-Core**. 2009.

GITHUB. **GitHub**. Disponível em: https://github.com/. Acesso em: 15 nov. 2020.

HAMBURG, Universität. **Petri Nets Tools Database Quick Overview**. Disponível em: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html. Acesso em: 23 jul. 2020.

HPETRISIM. **HPetriSim**. Disponível em: https://github.com/Uzuul23/HPetriSim. Acesso em: 08 nov. 2020.

JOINTJS. **JointJS**. Disponível em: https://github.com/clientIO/joint. Acesso em: 08 nov. 2020.

MICROSOFT. **Visual Studio Code**. Disponível em: https://code.visualstudio.com/. Acesso em: 23 nov. 2020.

NIELSEN, JAKOB. **Usability Engineering**. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 1993.

PRETTIER. **Prettier**. Disponível em: https://prettier.io/. Acesso em: 23 nov. 2020.

RAINER JR, R. K.; CEGIELSKY, C. G. **Introdução a Sistemas de Informação**. 3ª ed. Rio de Janeiro: Elsevier, 2012.

ROSEN, K. H. **Matemática Discreta e Suas Aplicações**. 6ª ed. Rio de Janeiro: McGraw-Hill, 2009.

SPIVEY, J. M. **Understanding Z: a specification language and its formal semantics**. Cambridge University Press, 2008.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 8ª ed. Rio de Janeiro: Prentice Hall, 2010.

STYLELINT. **Stylelint**. Disponível em: https://stylelint.io/. Acesso em: 23 nov. 2020.

TANENBAUM, A. S. **Organização Estruturada de Computadores**. 6ª ed. Rio de Janeiro: Prentice Hall, 2013.

TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos: princípios e paradigmas**. 2ª ed. São Paulo: Pearson, 2008.

TANENBAUM, A. S. **Sistemas Operacionais modernos**. 3ª ed. São Paulo: Prentice Hall, 2010.

TATTERSALL, Sarah. **PIPE**. Disponível em: https://github.com/sarahtattersall/PIPE. Acesso em: 15 nov. 2020.

TRELLO. **Método Kanban: Guia detalhado e 5 modelos prontos para usar**. Disponível em: https://blog.trello.com/br/metodo-kanban. Acesso em: 08 nov. 2020.

TRELLO. **Trello**. Disponível em: https://trello.com. Acesso em: 08 nov. 2020.

TOOLS, Iopt. **IOPT Tools**. Disponível em: http://gres.uninova.pt/IOPT-Tools. Acesso em: 08 nov. 2020.

UFERSA, UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO. 2014. **Projeto Pedagógico do Curso de Bacharelado em Tecnologia da Informação**, 2014.

USABILITY.GOV. **Card Sorting**. Disponível em: https://www.usability.gov/how-to-and-tools/methods/card-sorting.html. Acesso em: 08 nov. 2020.

VUE.JS. **Vue.js**. Disponível em: https://vuejs.org/. Acesso em: 08 nov. 2020.