Gabriel Santos Chiquini

Simulador de circuitos para dispositivos Android

Brasil

2019

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC nº 1204/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG):

Nome completo do autor:

Título do trabalho:

2. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF do TCCG.

Gabriel Santos Chiquini
(Nome completo do autor)[2]

Ciente e de acordo:

(Nome completo do orientador)[2] Data: 17 / 12 / 19

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Versão abril de 2018

[2] As assinaturas devem ser originais sendo assinadas no próprio documento, imagens coladas não serão aceitas.

Gabriel Santos Chiquini

Simulador de circuitos para dispositivos Android

Aplicativo para simulação de circuitos resistivos destinado a dispositivos Android

Universidade Federal de Goiás – UFG

Escola de Engenharia Elétrica, Mecânica e de Computação

Engenharia de Computação

Orientador: Geyverson Teixeira de Paula

Brasil
2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Chiquini, Gabriel
Simulador de circuitos para dispositivos Android [manuscrito] / Gabriel Chiquini. - 2019.
16 f.: il.

Orientador: Prof. Dr. Geyverson Teixeira de Paula.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2019.
Bibliografia.

1. Circuitos elétricos. 2. Simulação. 3. Análise nodal. 4. Aplicativo Android. I. Teixeira de Paula, Geyverson, orient. II. Título.

CDU 621.3

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
COORDENAÇÃO DE ESTÁGIO E PROJETO FINAL

ATA DE AVALIAÇÃO DE PROJETO FINAL

CURSO

( ) Eng. Elétrica ( ) Eng. Mecânica (X) Eng. de Computação
( ) Projeto Final 1 (X) Projeto Final II

AVALIAÇÃO DE PROJETO FINAL

Título do projeto: SIMULADOR DE CIRCUITOS PARA DISPOSITIVOS ANDROID

BANCA AVALIADORA

Membro 1: Prof. Dr. Geyverson Teixeira de Paula

Membro 2: Prof. Dr. Flávio Geraldo Coelho Rocha

Membro 3: Prof. Dr. Carlos Galvão Pinheiro Júnior

ESTUDANTES

Matrícula	Nome
201400931	GABRIEL SANTOS CHIQUINI

NOTAS

Matrícula	Membro 1				Membro 2				Membro 3				Média
	NPT	NTE	NAA	NF	NPT	NTE	NAA	NF	NPT	NTE	NAA	NF	
201400931	10	10	10	10	10,0	10,0	10,0	10,0	10,0	10,0	10,0	10,0	10,0

NPT – Nota plano de trabalho; NTE – Nota do trabalho escrito; NAA – Nota de apresentação e arguição

Para Eng. Elétrica, Mecânica e PFC2 da Eng. Da Computação: NF = 0,1 x NPT + 0,45xNTE + 0,45 x NAA

Para PFC1 da Eng. Da Computação: NF = 0,3 x NPT + 0,7 x NAA

Goiânia, 13 de dezembro de 2019.

Membro 1

Membro 2

Membro 3

Simulador de circuitos para dispositivos Android

Gabriel Chiquini

***Resumo*—O seguinte projeto é um aplicativo para dispositivos Android capaz de simular circuitos elétricos com resistores e fontes de corrente e tensão independentes, em corrente contínua. Como as opções de simuladores para dispositivos móveis é limitada, o objetivo deste trabalho foi atender estudantes de ensino médio e superior para auxiliar o aprendizado sobre análise de circuitos com um aplicativo de uso livre. Nele o usuário pode construir uma representação gráfica do circuito real, com os elementos acima mencionados à disposição. É apresentada a análise nodal modificada como método de análise do circuito. Esta técnica modela matematicamente o circuito como um sistema linear e para isso, será mostrado o método de Gauss para resolvê-lo. Também é apresentada a construção da interface de usuário do aplicativo, que foi desenvolvida com o uso de HTML5 e CSS.**

***Palavras-chave*—circuitos elétricos, análise nodal, simulação, aplicativo Android.**

***Abstract*—The following project is an app for Android devices capable of simulating electrical circuits with resistors and independent current and voltage sources in direct current. As the options for mobile simulators are limited, the purpose of this paper was to serve high school and college students to assist with circuit analysis learning with a free-to-use application. In it the user can construct a graphical representation of the actual circuit, with the above mentioned elements available. Modified nodal analysis is presented as a circuit analysis method. This technique mathematically models the circuit as a linear system and for that, the Gauss method to solve linear systems it will be shown. Also presented is the construction of the application's user interface, which was developed using HTML5 and CSS.**

Keywords—electrical circuits, nodal analysis, simulation, Android app.

I. Introdução

A Utilização de ferramentas que simulam o funcionamento de circuitos elétricos é importante para analisar como um projeto elétrico/eletrônico se comportaria caso estivesse montado. Isso possibilita estudantes e profissionais desenvolverem circuitos fora de um laboratório e com todas os elementos necessários à disposição, além disso, não há o risco de uma montagem errada danificar peças ou de problemas de conexão entre elementos. Simuladores mais robustos podem simular até problemas dos elementos reais, como variação de atributos em função da temperatura. Muitas vezes é difícil fazer medidas em circuitos reais porque instrumentos de medida podem causar interferências e podem ser caros e para medir muitos nós do circuito demandaria muito tempo e esforço. A simulação também é importante no ensino de circuitos elétricos, os alunos podem utilizar-se dos simuladores para entender como cada componente do circuito está se comportando e para ajudá-los na verificação de seus cálculos. A seguir apresentamos um breve histórico dos simuladores mais populares.

A. SPICE Simulator

O SPICE é um simulador para circuitos CC lineares e não lineares e CA lineares, é uma sigla para *Simulation Program with Integrated Circuit Emphasis*, foi desenvolvido inicialmente na Universidade da Califónia (Berkeley), por Donald Pederson e Laurence Nagel em 1973 e evoluiu do simulador CANCER. Ele tem o código aberto mas nunca foi ativamente mantido [1]. Inicialmente utilizava análise nodal para realizar a simulação, mas em sua segunda versão, chamada SPICE2, passou a utilizar a análise nodal modificada, que solucionou problemas encontrados na primeira versão do SPICE [2]. Em casos de circuitos não lineares e transientes, são feitas várias iterações para se chegar ao resultado. Sua utilização é um pouco complexa porque ele funciona apenas em modo texto, recebe um arquivo de entrada com um título, a descrição dos elementos do circuito e os dados da simulação desejada. A saída é exibida no terminal assim que a simulação é concluída. A estrutura do texto da saída varia conforme o tipo da simulação. Por causa da dificuldade de aprendizado e utilização, alguns simuladores criam uma interface gráfica em que usuário desenha o circuito e a aplicação utiliza o SPICE para calcular os resultados.

B. PSpice

PSpice é uma versão modificada e comercial do SPICE2, criada pela MicroSim, atualmente OrCAD e lançada em 1984. Foi o primeiro simulador de circuitos distribuído para computadores pessoais [3]. Inicialmente utilizava a mesma sintaxe do SPICE para entrada e saída de dados. Após algumas versões, lançou um módulo para as formas de onda resultantes graficamente e posteriormente o sistema *PSpice Schematics*, que permite ao usuário montar o circuito graficamente. Internamente é muito semelhante ao SPICE e usa os mesmos princípios de resolução [2]. É o software de simulação mais conhecido e completo, há uma versão limitada destinada a estudantes, porém seu código é fechado e foi desenvolvido apenas para o sistema operacional Windows.

C. Qucs

Qucs é a sigla para *Quite Universal Circuit Simulator*, um simulador para desktops com interface de usuário que suporta a entrada gráfica de circuitos. É gratuito pois é um software livre e ainda em desenvolvimento. A sua primeira versão é de 2003, foi lançado inicialmente para Linux, porém atualmente suporta Windows, macOS e sistemas baseados no BSD. Suporta simulação de corrente contínua e alternada, inclusive em regime transiente. Assim como o SPICE, Qucs também usa a análise nodal modificada para resolver os circuitos [4]. Ele não possui todas as funcionalidades que o

SPICE e o PSpice, porém para circuitos pequenos pode ser mais prático e fácil de ser utilizado, por possuir uma curva de aprendizado menor. [5]

D. EveryCircuit

EveryCircuit é o simulador mais completo para dispositivos móveis, é desenvolvido e comercializado por MuseMaze, Inc. Ele possui uma interface interativa para a entrada do circuito, e mostra em tempo real a saída do circuito usando animações e gráficos. Permite simular circuitos analógicos e digitais [6]. Como é um programa comercial, não há detalhes de como é implementada a análise do circuito. Ele foi desenvolvido para dispositivos móveis Android e iOS e em computadores pode ser utilizado com o navegador Google Chrome, porém sua versão grátis tem tempo máximo de utilização.

Como apresentado, a maioria dos simuladores ainda são destinados a computadores. As ferramentas comerciais já são capazes de simular a maioria dos circuitos práticos e as alternativas gratuitas também oferecem boas soluções. Já para dispositivos móveis, o EveryCircuit é o único simulador completo. Por isso, o desenvolvimento deste simulador busca suprir a falta de ferramentas grátis destinadas a celulares e *tablets*. A ideia de desenvolver para a plataforma Android é devido a sua maior participação no mercado. Atualmente, o Android é líder de mercado com 76% de participação no mercado de smartphones, segundo dados da StatCounter, de outubro de 2018 a outubro de 2019[7].

II. Análise das tensões de nó

O método de análise das tensões de nó ou análise nodal é um método de inspeção capaz de modelar um circuito resistivo em um sistema de equações lineares. Consiste em encontrar as tensões em cada nó do sistema e com isso é possível obter todos os valores de tensão e corrente sobre cada elemento presente no circuito. Um nó é uma conexão de dois ou mais elementos do circuito, em que não há outro componente conectado entre eles. A análise nodal é a aplicação da lei de Kirchhoff das correntes, que tem a seguinte definição: a soma algébrica das correntes em um nó em qualquer instante é zero [8].

Nesta análise, primeiramente se toma uma nó onde será a tensão de referência do circuito, é comumente chamado de terra. Posteriormente usa-se a Lei de Kirchhoff das correntes em cada um dos outros nós, tomando as correntes que "entram e saem" em cada um, cuja soma é igual a zero. O resultado desta modelagem é um sistema linear de equações com o número de equações igual ao número de nós.

A figura 1 mostra um nó de um circuito em que estão conectados os resistores R1 e R2 e a fonte de corrente I_1 e outros três nós conectados em cada um dos componentes. A tensão do nó principal é V_n e nos terminais dos resistores R1 e R2 não conectados a V_n é V_{R1}, V_{R2}, respectivamente. Para cada ramo do nó, entra ou sai uma corrente e, por convenção, é adotado o sinal positivo para quando a corrente está entrando e negativo para quando está saindo. Para a fonte de corrente, o módulo da corrente que entra no nó é o valor nominal da

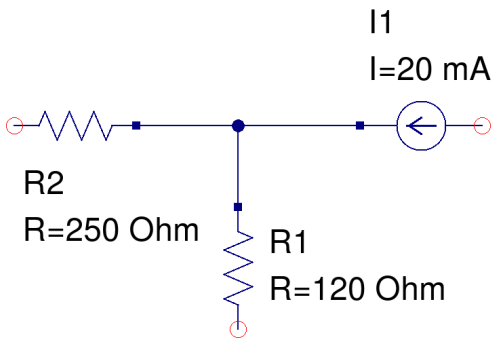

Figura 1: Um nó de um circuito

fonte, ou seja, I_1. Como a corrente da fonte está entrando, ela continua positiva e seu valor é I_1. Já nos resistores, a corrente que passa por eles é calculada pela lei de Ohm, ou seja, é a tensão sobre o resistor dividido pela sua resistência. A tensão sobre cada resistor é a diferença das tensões de nó conectadas em cada um de seus terminais. No resistor R_1, por exemplo, o módulo da corrente é $\frac{V_n - V_{R1}}{R_1}$, mas neste caso o valor é negativo, pois a corrente está saindo do nó.

$$-\frac{V_n - V_{R1}}{R_1} - \frac{V_n - V_{R2}}{R_2} + I_1 = 0 \tag{1}$$

A equação 1 representa a equação do nó da figura 1. Manipulando algebricamente esta equação, obtém-se a forma canônica da equação do sistema linear, isolando V_n, V_{R1} e V_{R2}, porém essa manipulação envolve uma complexidade computacional muito alta, o ideal seria utilizar um método de inspeção que obtêm as equações do sistema de forma completamente numérica.

A. Matriz de condutâncias

A análise nodal de um circuito contendo apenas resistores e fontes de corrente gera matrizes e sistemas de equações de uma forma particular, que pode ser descrita por um algoritmo simples, tornando a modelagem computacional mais rápida e direta. A matriz principal do sistema é chamada de matriz de condutâncias. Este método gera matrizes de um sistema linear no formato $M_G \times M_V = M_i$ no qual M_G é a matriz de condutâncias, M_V são as incógnitas (tensões de nó) e M_i é a matriz de fontes de corrente conectadas em cada nó.

A matriz de condutâncias pode ser descrita por um algoritmo simples, são necessários três passos:

1) Os nós são numerados e a referência é identificada, pois cada linha/coluna da matriz representará um nó.
2) é preenchida a diagonal principal da matriz ($M_{i,i}$), em que cada elemento é formado pela soma das condutâncias conectadas ao nó i, as fontes não são consideradas como condutância.
3) Os outros elementos ($M_{i,j},\ \forall i \neq j$) são formados pela soma das condutâncias conectadas ao nó i e ao nó j ao mesmo tempo, multiplicadas por -1.

Para o circuito da figura 2, com os nós já identificados, a matriz de condutâncias é:

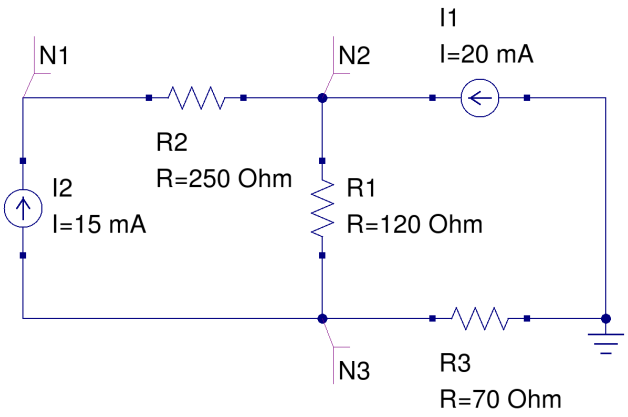

Figura 2: Circuito com quatro nós

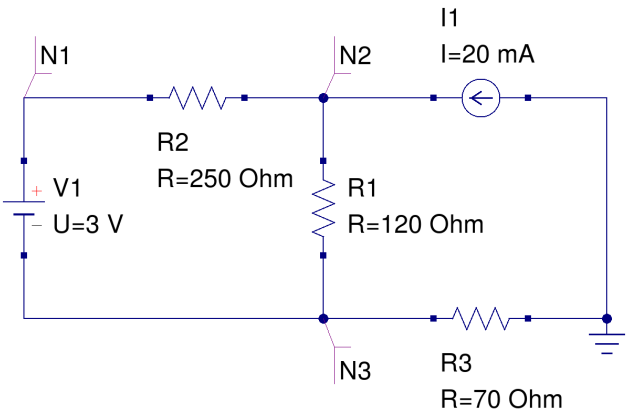

Figura 3: Circuito com fonte de tensão

$$\begin{bmatrix} \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_1} & -\frac{1}{R_1} \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_3} \end{bmatrix} \tag{2}$$

A matriz de fontes é calculada com a soma das correntes das fontes conectadas ao nó. Para o circuito do exemplo a matriz é:

$$\begin{bmatrix} I_2 \\ I_1 \\ -I_1 \end{bmatrix} \tag{3}$$

Com essas duas matrizes, agora pode ser obtida a forma canônica do sistema, que é:

$$\begin{bmatrix} \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_1} & -\frac{1}{R_1} \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_3} \end{bmatrix} \times \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_2 \\ I_1 \\ -I_1 \end{bmatrix} \tag{4}$$

Esta análise é limitada a circuitos contendo apenas resistores e fontes de corrente, porém, comumente são necessários circuitos com fontes de tensão.

O circuito da figura 3 é semelhante ao da figura 2, porém no lugar da fonte I2 foi colocada a fonte de tensão V_1. Para utilizar esse circuito com a análise nodal seria necessário utilizar o conceito de supernó. Um supernó é um nó que possui uma fonte de tensão conectada entre dois nós. No caso da figura 3, o valor da tensão V_1 é $V_3 + 3$. Com isso, pode-se substituir em cada equação onde se usa V_1.

Mais uma vez, isso requer algumas manipulações algébricas, que não são desejadas na modelagem computacional, dificultando a implementação deste método. Ainda assim, ele foi utilizado no simulador CANCER, em conjunto com um outro método de transformação de fontes (equivalente de Norton), visto que àquela época, era a modelagem mais eficiente que se conhecia. [9]

B. Análise nodal modificada

O aumento da pesquisa em circuitos integrados aumentou a necessidade de simulação dos circuitos envolvidos no sistema e como a modelagem nodal ainda era ineficiente, em 1975, foi desenvolvido um método simples para construir matrizes de condutância mesmo com fontes de tensão, que é popularmente conhecido como análise nodal modificada [9]. Este é o método usado em praticamente todos os simuladores (é utilizado em todos que foram citados) por sua eficiência e é capaz de modelar circuitos lineares e não lineares. O método fornece outras técnicas de modelagem para outros dispositivos elétricos como capacitores, indutores e transistores, porém aqui ficaremos limitados aos circuitos resistivos com fontes independentes e de corrente contínua.

Para demonstrar a modelagem, iremos modelar o circuito da figura 3 utilizando-se deste método. Inicialmente montamos a matriz de condutâncias ignorando a presença da fonte no circuito.

$$\begin{bmatrix} \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_1} & -\frac{1}{R_1} \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_3} \end{bmatrix} \tag{5}$$

Para cada fonte de tensão, é adicionada uma linha e uma coluna na matriz, em cada linha é atribuído zero em cada termo, exceto nas colunas que referenciam os nós em que a fonte está conectada. Para a coluna onde está o terminal negativo da fonte é atribuído -1 e para a coluna com o terminal positivo é atribuído 1 e depois uma coluna é adicionada à matriz de forma a manter a simetria. Sobra um termo na diagonal principal, que será preenchido com zero. Então a nova matriz é

$$\begin{bmatrix} \frac{1}{R_2} & -\frac{1}{R_2} & 0 & 1 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_1} & -\frac{1}{R_1} & 0 \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_3} & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} \tag{6}$$

A matriz de fontes é obtida de forma análoga, é adicionado um novo termo a ela que contém o valor nominal da tensão da fonte. A matriz de fontes seria

$$\begin{bmatrix} 0 \\ I_1 \\ 0 \\ V_1 \end{bmatrix} \tag{7}$$

Além desse método fornecer as tensões de nó, ele ainda fornece as correntes que passam pelas fontes de tensão modeladas nas linhas/colunas adicionadas. Estas correntes são as incógnitas que aparecem nas novas equações encontradas, então a nova matriz do sistema é

$$\begin{bmatrix} \frac{1}{R_2} & -\frac{1}{R_2} & 0 & 1 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_1} & -\frac{1}{R_1} & 0 \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_3} & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_{V_1} \end{bmatrix} = \begin{bmatrix} 0 \\ I_1 \\ 0 \\ V_1 \end{bmatrix} \tag{8}$$

e I_{V_1} é a corrente que passa pela fonte de tensão V_1.

C. Análise do algoritmo utilizado

Faremos agora uma análise da complexidade computacional do algoritmo utilizado. Para isso, usaremos a notação Big O, que representa o crescimento do tempo de processamento de acordo com fatores do volume de dados do problema. Esta notação é escrita da seguinte forma: $O(n)$, sendo n o fator em análise. Em nosso problema, haverá dois fatores, n e m, que serão, respectivamente, o número de nós e o número de elementos do circuito. Esta análise é assintótica, então consideramos que os fatores tendem a infinito. Com isso, só será necessário considerar o termo de maior crescimento, utilizando os princípios de limites. Por exemplo, uma complexidade de $O(n^2 + 10n)$ será igual a $O(n^2)$, visto que no infinito a contribuição de $10n$ seria ínfima.

Após o usuário descrever o circuito na interface, o circuito é inicialmente modelado com duas listas, uma contendo os elementos do sistema e seus respectivos terminais e uma outra lista contendo outras listas, uma para cada nó e em cada uma destas há a definição de quais terminais estão conectados neste nó, com exceção do nó da referência, que é excluído antes da análise iniciar.

Primeiramente montaremos a matriz de condutâncias, definindo a diagonal principal, para isso, em cada nó, iteramos sobre o terminal conectado, adicionando o valor de cada resistência encontrada em um acumulador para no final obter o termo da matriz, o que gera uma complexidade de $O(n \times m)$ para o pior caso. Para o pior caso, consideramos um nó com todos os elementos do circuito conectados a ele.

Os outros elementos da matriz são inseridos verificando quais resistores estão conectados ao mesmo tempo em dois nós, ou seja, para o resistor conectado ao nó, procuramos se o seu outro terminal está conectado a outro nó, o esforço computacional é mais alto neste caso, gerando uma complexidade $O(n^2 \times m)$.

Para as fontes de tensão, é feita uma varredura em cada nó de forma a montar uma lista com os nós em que os terminais negativo e positivo das fontes de tensão estão, para depois adicionar as equações necessárias. E chegamos a uma complexidade computacional $O(n \times m)$. Ao final do algoritmo temos uma complexidade de $O(n^2 \times m + 2n \times m)$. A notação Big O tem uma propriedade que diz que o termo de maior crescimento determina a ordem da função. Aplicando este princípio, temos que a complexidade final da matriz de condutâncias é: $O(n^2 \times m)$.

Como a matriz de fontes é menor, sua complexidade também é mais simples, novamente iteramos sobre os nós buscando as fontes de corrente e de tensão. Quando a fonte é de corrente, seu valor é somado ou subtraído de um acumulador para aquele nó, de acordo com a direção da corrente. Quando a fonte é de tensão, é necessário adicionar uma nova linha a esta matriz contendo o valor nominal da tensão da fonte. Este processo gera uma complexidade de $O(n \times m)$, então, somando o esforço computacional dos dois processos e aplicando a propriedade do termo de maior crescimento a complexidade ainda será $O(n^2 \times m)$.

III. Resolução do sistema

A análise do circuito gera um sistema linear de equações, portanto para encontrar a resposta é necessário que ele seja resolvido. Assim como na modelagem do circuito, há diversas maneiras de se resolver um sistema linear. Alguns métodos são simples e não serão explorados, por causa do seu custo computacional alto. São eles:

- Regra de Cramer: Este método utiliza determinantes para se calcular a solução do sistema, o problema deste método é que calcular determinantes para matrizes grandes é muito complexo, portanto não é eficiente.
- Método da comparação: Envolve operações algébricas fáceis para serem resolvidas manualmente, porém, computacionalmente falando, é uma implementação muito difícil, principalmente com o crescimento do número de equações.
- Método da substituição: Assim como o método da comparação, envolve operações algébricas e com sistemas grandes, as operações com polinômios poderiam causar problemas com o ponto flutuante.

A. Eliminação de Gauss

A eliminação de Gauss, ou método do escalonamento, consiste em fazer operações lineares de soma e multiplicação nas linhas nas matrizes até obter uma matriz triangular superior e depois fazer substituições nas equações para encontrar os valores das respostas [10]. Para explicar passo a passo o funcionamento do método, iremos resolver o sistema anteriormente obtido. Uma matriz triangular superior é aquela em que todos os elementos abaixo da diagonal principal são zero. A forma numérica do sistema é a seguinte:

$$\begin{bmatrix} 0.004 & -0.004 & 0 & 1 & : & 0 \\ -0.004 & 0.0123 & -0.0083 & 0 & : & 0.02 \\ 0 & -0.0083 & 0.0226 & -1 & : & 0 \\ 1 & 0 & -1 & 0 & : & 3 \end{bmatrix} \quad (9)$$

Esta matriz é a matriz aumentada do sistema, que é formada pela matriz A unida lateralmente com a matriz B. Os valores foram truncados em quatro casas decimais para ilustrar o problema que o arredondamento pode causar ao resultado final do sistema, visto que esse problema ocorre nas operações com ponto flutuante.

Para obtermos a matriz triangular superior, começaremos zerando todos os elementos da primeira coluna a partir da segunda linha, para isso, calculamos um coeficiente que será o negativo do elemento que desejamos zerar, divido pelo termo da diagonal. Porém, a pergunta que vem em mente é "e se um termo da diagonal principal for zero?". Se isto acontecesse, não seria possível realizar a divisão pelo termo da diagonal. No caso apresentado temos o último elemento da matriz como 0, mas ele não afetará a análise. Quando houver necessidade de eliminar os zeros da diagonal principal, será aplicado uma técnica chamada pivoteamento parcial.

O pivoteamento parcial é simplesmente uma substituição de linhas abaixo de um elemento que possui o valor nulo, gerando uma nova matriz. A cada iteração da etapa de eliminação, observa-se o valor do termo na diagonal principal e compara-se seu módulo com os demais termos na mesma coluna nas linhas abaixo, encontrado o maior módulo, troca-se as linhas de posição no sistema, a atual com o termo de maior módulo. Além de evitar divisões por zero, este método auxilia a evitar problemas de arredondamento, então ele é aplicado mesmo se o termo não for zero. Então, aplicando a ideia ao sistema teremos que a nova matriz aumentada, que é

$$\begin{bmatrix} 1 & 0 & -1 & 0 & : & 3 \\ -0.004 & 0.0123 & -0.0083 & 0 & : & 0.02 \\ 0 & -0.0083 & 0.0226 & -1 & : & 0 \\ 0.004 & -0.004 & 0 & 1 & : & 0 \end{bmatrix} \quad (10)$$

Após o pivoteamento, faremos o escalonamento da matriz, iniciando da primeira linha. O coeficiente da linha será $-\frac{-0.004}{1}$, feito isso, multiplicamos este coeficiente pela primeira linha e somamos com a segunda, isto irá zerar o termo -0.004 e irá alterar os outros termos da linha. Após fazer isso em todas as linhas, teremos uma outra matriz

$$\begin{bmatrix} 1 & 0 & -1 & 0 & : & 3 \\ 0 & 0.0123 & -0.0123 & 0 & : & 0.032 \\ 0 & -0.0083 & 0.0226 & -1 & : & 0 \\ 0 & -0.004 & 0.004 & 1 & : & -0.012 \end{bmatrix} \quad (11)$$

Nesta primeira iteração, todos os termos abaixo da diagonal foram zerados, agora faremos isso para as outras colunas, que resultará na seguinte matriz

$$\begin{bmatrix} 1 & 0 & -1 & 0 & : & 3 \\ 0 & 0.0123 & -0.0123 & 0 & : & 0.032 \\ 0 & 0 & 0.0143 & -1 & : & 0.0216 \\ 0 & 0 & 0 & 1 & : & -0.0016 \end{bmatrix} \quad (12)$$

Essa é a matriz escalonada do sistema, agora basta realizar as substituições regressivamente na matriz para obter cada resposta. O primeiro é obtido com $1x_4 = -0.0016$, que é -0.0016, o segundo é obtido com o valor de $x4$, $0.0143x_3 - (-0.0016) = 0.0216$ e temos que $x_3 = 1.3986$. De forma análoga, $0.0123x_2 - 0.0123 \times 1.3986 = 0.032$, $x2 = 4.0000$ e $x_1 - 1.3986 = 3$ e $x_1 = 4.3986$. A solução geral do sistema é:

$$\begin{bmatrix} 4.3986 \\ 4.0000 \\ 1.3986 \\ -0.0016 \end{bmatrix} \quad (13)$$

Esta solução está próxima à solução exata, porém houve problemas de arredondamento e estes problemas aumentam conforme o tamanho do sistema, a solução mais próxima da exata é

$$\begin{bmatrix} 4.3986 \\ 4.0002 \\ 1.3986 \\ -0.0015 \end{bmatrix} \quad (14)$$

Analisando passo-a-passo a complexidade deste algoritmo, temos:

1) Pivoteamento: Temos um loop para cada linha abaixo da atual. Serão $n \times ((n-1)+(n-2)+...+1)$, operações, isto é, uma progressão aritmética regressiva de razão 1. Utilizando a fórmula a soma de uma PA, teremos um total de $\frac{n^2-n}{2}$. O que leva a uma complexidade de $O(n^2)$.
2) Eliminação: É o maior processo, temos dois loops da mesma forma do anterior, porém em cada linha, teremos um outro loop para recalcular a linha. Este processo tem um total de $2n^3/3$ operações [11] e uma complexidade de $O(n^3)$.
3) Substituição regressiva: São realizados o mesmo número de operações que o pivoteamento, portanto, tem a complexidade de $O(n^2)$.

A complexidade dele será de $O(n^3)$, porém para fins de comparação, será utilizado o total de operações que o algoritmo realiza, que neste caso é $\frac{2n^3}{3} + n^2$.

B. Fatoração LU

A Fatoração LU é um método de decomposição de matrizes que pode ser utilizado para encontrar a solução de um sistema linear. A fatoração decompõe a matriz A em duas matrizes LU. As matrizes L e U são matrizes triangulares inferior e superior, respectivamente. Em vez de um sistema $Ax = B$, teremos $LUx = B$. A matriz U é muito semelhante à matriz obtida através da eliminação de Gauss, já a matriz L é obtida com os coeficientes de linha utilizados no escalonamento [10]. Para explicar o processo, resolveremos o sistema anterior com este método. As mesmas regras de pivoteamento se aplicam a fatoração LU. A matriz inicial L é a matriz identidade e a matriz U é:

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ -0.004 & 0.0123 & -0.0083 & 0 \\ 0 & -0.0083 & 0.0226 & -1 \\ 0.004 & -0.004 & 0 & 1 \end{bmatrix} \quad (15)$$

que é a matriz A do sistema anterior antes do escalonamento. Diferentemente do método de Gauss, na fatoração LU, usa-se apenas a matriz principal na etapa de eliminação. Os coeficientes de linha da primeira coluna são -0.004, 0, e 0.004. Estes valores serão utilizados para zerar os elementos na matriz U e farão parte da primeira coluna matriz L. Nesta etapa as Matrizes LU serão:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.004 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.004 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0.0123 & -0.0123 & 0 \\ 0 & -0.0083 & 0.0226 & -1 \\ 0 & -0.004 & 0.004 & 1 \end{bmatrix} \tag{16}$$

Após aplicar a regra para todas as colunas, teremos as matrizes L e U desta maneira:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.004 & 1 & 0 & 0 \\ 0 & -0.6748 & 1 & 0 \\ 0.004 & -0.3252 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0.0123 & -0.0123 & 0 \\ 0 & 0 & 0.0143 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

Com as duas matrizes, podemos resolver o sistema em dois passos. Primeiramente, vamos ter $Ly = b$ e depois teremos $Ux = y$. Deve ser feita a mesma permutação em b das linhas que foram alteradas pivoteamento. $Ly = b$ será:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.004 & 1 & 0 & 0 \\ 0 & -0.6748 & 1 & 0 \\ 0.004 & -0.3252 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 0.02 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

Como a matriz L é uma matriz triangular superior, o vetor y pode ser encontrado utilizando substituição progressiva, de forma similar à ultima etapa da eliminação gaussiana, com isso teremos que y é:

$$\begin{bmatrix} 3.0000 \\ 0.0320 \\ 0.0216 \\ -0.0016 \end{bmatrix} \tag{19}$$

Nota-se que este vetor é a última coluna da matriz estendida na Eliminação de Gauss, então adicionando o vetor como coluna na matriz teremos uma matriz estendida igual a matriz escalonada. Então, realizando os mesmos passos, teríamos a solução do sistema.

A complexidade deste algoritmo é próxima da eliminação gaussiana, porém há uma substituição progressiva que realiza $n^2/2$ operações. As etapas de pivoteamento, eliminação e substituição realizam o mesmo número de operações, então o total será $\frac{2n^3}{3} + 3\frac{n^2}{2}$. Na notação Big O, a complexidade será a mesma, de $O(n^3)$, então em sistemas grandes, a diferença da execução pode ser pouco perceptível.

Mesmo sendo um pouco menos eficiente, a razão de se utilizar este algoritmo é que a matriz U pode ser reutilizada caso seja necessário calcular outro sistema com a mesma matriz A e uma matriz B diferente. Em aplicações práticas, isto seria útil se houvesse uma análise em transiente de uma fonte variante no tempo. Neste caso, a matriz A se manteria inalterada durante toda a análise. Assim, apenas seria necessário fazer as duas últimas substituições. O que resultaria em apenas n^2 operações. Como não houve nenhuma implementação que reutilizaria a matriz A, o método escolhido foi a eliminação de Gauss.

Em alguns circuitos, poderiam haver operações de soma e subtração que causariam certos problemas no ponto flutuante, por exemplo se houvesse um resistor de 1 MΩ e um outro de 1 mΩ, durante as várias operações do escalonamento, poderia haver somas entre eles e, considerando a limitada precisão do ponto flutuante, o valor do menor resistor poderia ser simplesmente ignorado devido ao truncamento de casas decimais.

Outro problema que poderia ocorrer seria se um valor muito baixo estivesse em uma diagonal, ao calcular o coeficiente de linha, ele seria um número muito alto e poderia causar mais erros de arredondamento nas operações desta linha. Portanto, para evitar problemas como este e oferecer a máxima precisão possível, foi utilizado uma implementação de "Big Number", isto é, um número de precisão arbitrária, que não usa o limite de bits do hardware. Na implementação utilizada, o numéro de dígitos foi de 64 dígitos, na base 10, um grande aumento de precisão comparado ao *double* utilizado no JavaScript, que poderia representar um número inteiro de até 15 casas sem arredondamentos [12]. É muito improvável que uma precisão maior que a adotada seja necessária em fins práticos e não houve perda de performance, os circuitos testados ainda são resolvidos em milissegundos. Para a aplicação desenvolvida neste trabalho, que é voltada para dispositivos móveis (celular e tablet), não é esperado que o usuário simule sistemas grandes, visto que se trata de uma aplicação mais difícil de ser utilizada comparado a um simulador para um computador de mesa.

IV. Interface de usuário

Uma interface de usuário é um mecanismo de interação do usuário com a aplicação desenvolvida. Existem interfaces textuais e interfaces gráficas. O SPICE usa uma interface puramente textual para a entrada e saída de dados. Outros simuladores, como o Qucs usam interfaces gráficas, onde o usuário monta os elementos no circuito de forma semelhante ao que seria feito num circuito integrado, com isso, a curva de aprendizado para utilizar o aplicativo é menor e é mais próximo da realidade. Como este simulador deverá funcionar em dispositivos móveis, a interface foi pensada para ser utilizada numa tela sensível ao toque e significantemente menor, sem teclado e mouse a disposição. Por isso, a interface de foi o ponto de maior atenção no aplicativo.

A interface foi desenvolvida utilizando HTML, CSS e TypeScript. HTML é a sigla para Hypertext Markup Language, e é a linguagem padrão de exibição em páginas *web*. É um texto estruturado e não uma linguagem de programação, a estrutura da página é definida com estruturas chamadas *tags*, que possuem o conteúdo exibido e um conjunto de atributos, para uso do desenvolvedor. Cada *tag* tem seu comportamento na exibição na página, a *tag* `<p>` define um parágrafo, já a *tag* `<ul>` define uma lista de elementos. Cada *tag* forma um nó HTML e os nós podem ser aninhados, formando estruturas mais complexas [13].

O CSS também é uma linguagem estruturada que permite a definição de estilos para os nós HTML. Por padrão, cada nó

HTML tem um estilo de exibição na página, porém normalmente o padrão não é desejável, então o CSS permite definir propriedades como alinhamento, cor, fonte, entre outros. Ele é baseado em regras, cada uma é composta por um *selector*, que é um conjunto de elementos em que a regra será aplicada e suas propriedades, que são definidos numa sintaxe de pares chave valor. Assim como o HTML, é apenas texto estruturado, não possui estruturas de controle presentes em uma linguagem de programação [14].

O TypeScript é uma linguagem desenvolvida pela Microsoft, que adiciona tipagem estática ao JavaScript, ele é apenas *transpilado* para JavaScript e não é executado de fato [15]. Já o JavaScript é a linguagem de programação embutida nos navegadores. É interpretada, de tipagem fraca e dinâmica, que permite a manipulação do HTML e CSS de forma programática. Mesmo que seja embutido no *browser*, existem interpretadores que executam JavaScript em diversos ambientes. É uma linguagem orientada a eventos, o que torna simples gerenciar a interação do usuário com a página [16].

Uma página HTML com muitas interações é difícil de ser gerenciada utilizando as tecnologias embutidas no navegador, portanto para gerenciar a interface, optamos por utilizar o React. Ele é uma biblioteca do Facebook focada em gerenciamento de estados e que permite transformar uma página em componentes, simplificando a manutenção e dá a possibilidade do desenvolvedor escrever HTML de forma dinâmica [17].

Como os nós HTML ocupam bastante memória e são renderizados lentamente, utilizamos canvas, uma tecnologia do HTML5, que permite desenhar gráficos em duas e três dimensões diretamente na página HTML, utilizando apenas um nó HTML, então os gráficos são diretamente renderizados no dispositivo e podem ter aceleração de *hardware*. O canvas é uma tecnologia difícil de ser manipulada, pois envolve muitos conceitos de computação gráfica, então, para abstrair esta complexidade, foi utilizado uma biblioteca, Konva. Konva é uma biblioteca construída especificamente para canvas, que facilita o desenho de objetos e permite movimentá-los, assim como a criação de camadas e filtros de imagem. Ela também manipula eventos que simplificam a interação do usuário e permite a criação de animações diretamente no canvas [18].

A aplicação é feita com base nas tecnologias *web*, então para permitir a compatibilidade como um aplicativo nativo para Android, utilizamos o *framework* Cordova, que transforma uma aplicação *web* em aplicação nativa. Embora o Cordova seja multiplataforma, o aplicativo foi compilado apenas para Android. Ele também pode ser utilizado como uma página HTML estática, mas o foco do trabalho foi em desenvolver uma aplicação *mobile*, visto que para computadores existem outras soluções gratuitas e mais completas.

A figura 4 mostra a tela inicial do simulador, composta de duas seções, o menu, na parte superior e a área onde circuito é construído. O menu é composto pelas seguintes funções, da esquerda para a direita:

- Fonte de corrente: adiciona uma nova fonte de corrente independente ao circuito. O valor da corrente da fonte é exibido logo acima do componente, cujo padrão é $1A$.
- Fonte de tensão: adiciona uma nova fonte de tensão independente. Permite alterar o valor da tensão, cujo padrão é $5V$.
- Resistor: adiciona um resistor ao circuito. O valor da resistência é ajustável, sendo 1000Ω o padrão.
- Terra: adiciona o elemento da referência de tensão ao circuito. É obrigatório adicioná-lo pois a análise nodal depende do nó de referência.
- Ocultar menu: ao tocar neste ícone, o menu superior irá desaparecer, aumentando a área do circuito, para melhor visualização. O botão ocultar e o simular ficarão suspensos
- Simular: ao tocar no botão, o aplicativo usará a representação do circuito para calcular as tensões de nó do circuito;

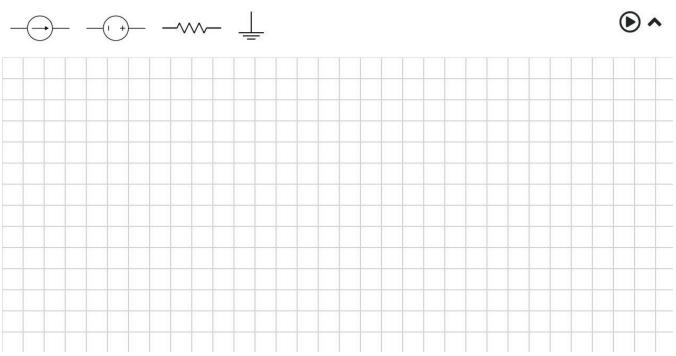

Figura 4

Na área do circuito, há um quadriculado, para auxiliar o usuário no alinhamento dos elementos.

Para demonstrar o uso da interface, iremos simular o exemplo da figura 2 e depois editá-lo para simular o circuito da figura 3.

Começaremos adicionando uma fonte de corrente ao circuito, para adicioná-la, basta tocar no primeiro elemento do menu, que simboliza esta fonte. Internamente, a interface armazena uma lista de elementos, com suas propriedades e terminais relacionados e uma lista de nós, em cada nó há uma lista com as referências de cada terminal. Estas listas serão atualizadas de acordo com eventos gerados pela interação do usuário.

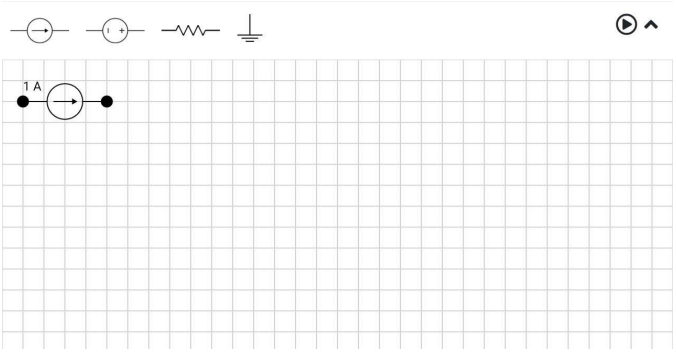

Figura 5: Elemento recém adicionado

Como mostrado na figura 5, a fonte é adicionada ao canto superior esquerdo da área do circuito. Ela é criada com valor de corrente de 1A, por isso iremos alterar esse valor e

alterar sua posição, pois todos elementos são adicionados nesta posição, o que deixaria um elemento sobreposto sobre a fonte. Para alterar a posição do elemento basta fazer o movimento de “arrastar", o aplicativo só permitirá o elemento ser movido para um lugar em que os terminais fiquem alinhados com o fundo quadriculado, ajudando o posicionamento. Ao adicionar uma fonte ao circuito, a lista de elementos é atualizada, adicionando um elemento.

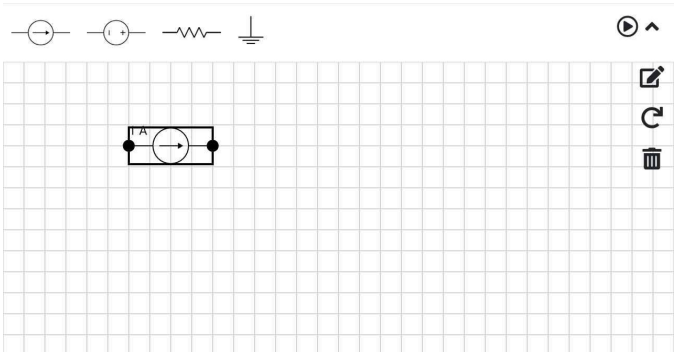

Figura 6: Elemento selecionado

Para alterar o o valor das propriedades de um elemento, é necessário selecioná-lo. A figura 6 mostra a fonte após ser movida e selecionada. Quando um elemento é selecionado, é mostrado na lateral direita um menu com três opções, de editar, rotacionar e de excluir.

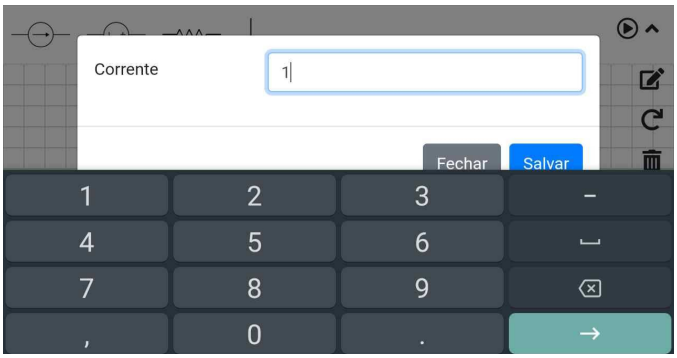

Figura 7: Tela de edição

Após tocar no ícone de editar, é apresentada a tela da figura 7, que permite alterar o valor da propridade do elemento, escolheremos o valor $0,015$ para a corrente, de acordo com o exemplo anterior. Ao realizar esta ação, a fonte é buscado na lista de elementos e o valor de sua corrente é atualizada. Faremos o mesmo processo para todos os elementos, até adicionar todos os componentes necessários.

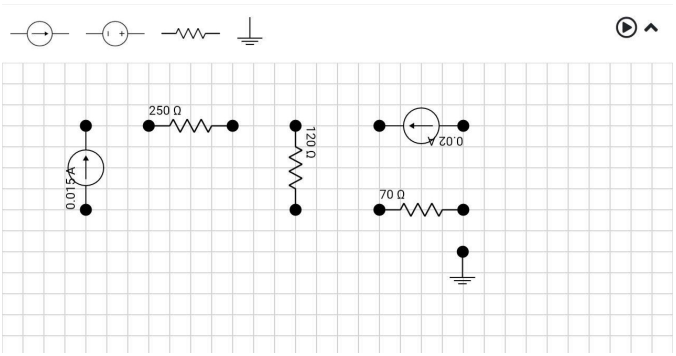

Figura 8: Circuito com os elementos necessários

Após adicionar todos os elementos, o circuito será o apresentado na figura 8. Porém, os elementos ainda não estão conectados, para isso, basta tocar no terminal de um elemento, este terminal aparecerá marcado com a cor branca, e então em um terminal de outro elemento, isso fará com que uma linha reta seja desenhada entre eles. Quando o usuário conecta dois terminais, a lista de nós é atualizada, criando um nó caso não exista conexão anterior entre os terminais em questão, ou então, os terminais selecionados são adicionados ao nó existente. Para facilitar a experiência, foi adicionado um nível de tolerância para facilitar a seleção, porque tocar exatamente no terminal seria complicado por causa do tamanho do dedo do usuário, por conta disso, elementos muito próximos podem ter problemas para serem conectados. Para cancelar a seleção, basta tocar novamente no terminal selecionado.

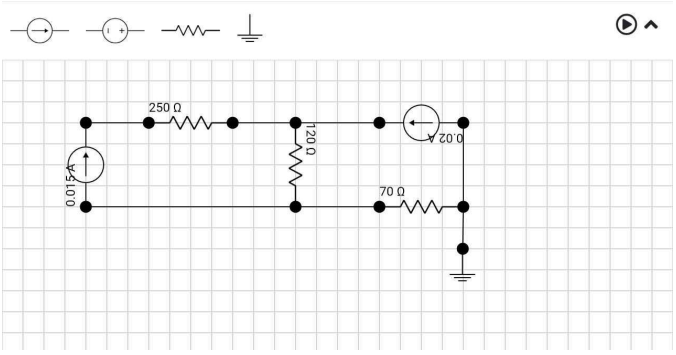

Figura 9: Circuito completo

A figura 9 mostra o circuito desejado. Para calcular as tensões de nó basta clicar no ícone do menu superior à direita semelhante ao botão *play*. Ao fazer isso, o simulador faz uma breve validação se não há terminais sem conexão ou se o terra não foi inserido. Após a validação, a simulação é feita utilizando a lista de nós e a lista de elementos para gerar as matrizes do sistema.

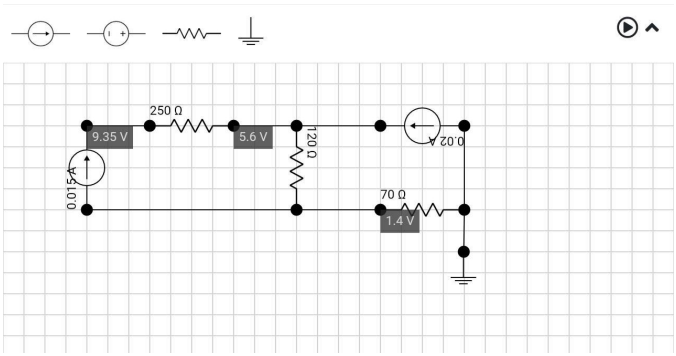

Figura 10: Resultado da simulação

Ao tocar no botão de simulação, as tensões são apresentadas próximo a um terminal de cada nó, como mostra a figura 10. Agora para exemplificar o fluxo de edição do circuito, vamos alterar a fonte de $0.015A$ para uma fonte de tensão de $3V$, igual ao exemplo da figura 3. Para excluir um elemento, basta selecioná-lo e tocar no ícone de excluir, como mostrado na figura 6.

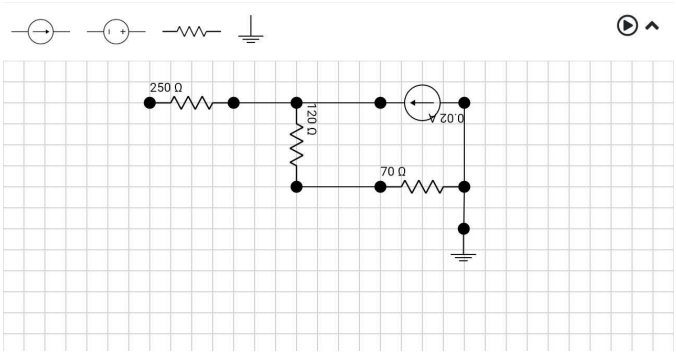

Figura 11: Fonte excluída

Ao excluir a fonte, como mostrado na figura 11, os fios que faziam as conexões entre a fonte e outros nós, também são excluídos, então a fonte é excluído da lista de elementos seus terminais são removidos dos nós que fazem parte. Se o usuário tentar simular o circuito neste estado, apareceria uma mensagem contendo "O circuito possui elementos flutuando", porque o resistor de 250Ω e o de 120Ω estão com um terminal sem conexões. Montaremos então o circuito final com a fonte de tensão.

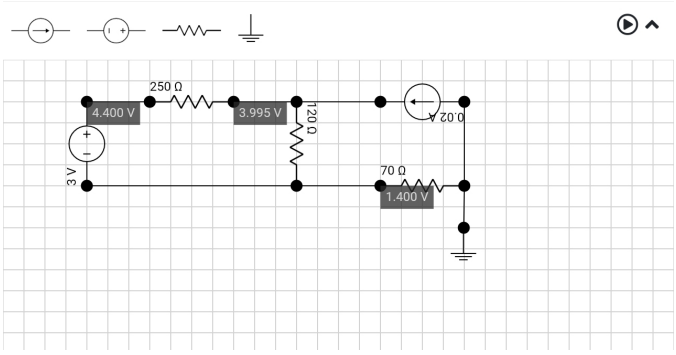

Figura 12: Resultado da segunda simulação

A figura 12 mostra o resultado da simulação resolvida anteriormente. Devido ao uso do *big number*, os valores das tensões de $4.4V$ e $1.4V$ ficaram com valores próximos ao exato.

A. Simulador

V. Estruturação do código

Nesta seção será aprensentada uma breve documentação do código, explicando como a implementação do aplicativo foi estruturada, com auxílio de diagramas UML. UML é a sigla para *"Unified Modeling Language"*, que significa linguagem de modelagem unificada, sua especificação atual é um padrão ISO e contempla 13 diagramas [19]. Seus diagramas são utilizados para documentar de um projeto de software, tem uma linguagem gráfica de fácil entendimento para que seja legível e clara para os desenvolvedores e outros envolvidos no projeto. Como é um aplicativo com poucos recursos, por simplicidade, apresentaremos apenas os seguintes diagramas:

- Diagrama de classe: este diagrama mostra os atributos e métodos das classes contidas no programa sem se preocupar com a forma que são utilizadas. Foi necessário algumas adaptações no diagrama, pois a forma que o TypeScript/JavaScript lida com classes é um pouco diferente de outras linguagens focadas em orientação a objetos, como C# e Java. A definição do diagrama de classe não contempla módulos da forma que o JavaScript implementa [20], portanto para contornar esta limitação, foi utilizada uma classe com métodos estáticos no lugar, cuja semântica é muito semelhante. Outro ponto é que interfaces podem ser utilizadas para determinar estruturas de objetos que não são instâncias de classes, em outras linguagens, não seria possível ter uma instância de uma interface.
- Diagrama de sequência: diferente do diagrama de classes, este mostra a interação das classes durante a execução do programa. Representa a ordem temporal da comunicação entre os objetos, mas sem se preocupar com sua estrutura, visto que elas foram previamente descritas no diagrama anterior.

Inicialmente, o projeto foi dividido em duas partes para melhorar a organização do código. Todo o código que se refere a interface e interação com o usuário foi separado do código que realizava a simulação, a primeira parte será chamada interface e a segunda, simulador. Na interface, temos o projeto com componentes React e uma representação simples do circuito. No simulador, temos o circuito melhor estruturado e funções para realizar a simulação de fato. A interface utiliza o simulador como uma biblioteca, que expõe apenas uma função, em que ela envia a representação do circuito e tem as tensões de nó como resposta.

Inicialmente explicaremos o diagrama de classes do simulador, para melhorar a legibilidade, o diagrama está separado em três partes, unidos por uma classe em comum, *Circuit*. Nos próximos diagramas, ela será exibida sem os atributos para facilitar a leitura. O primeiro diagrama, mostrado na figura 13 mostra a estruturação das classes principais, que contém

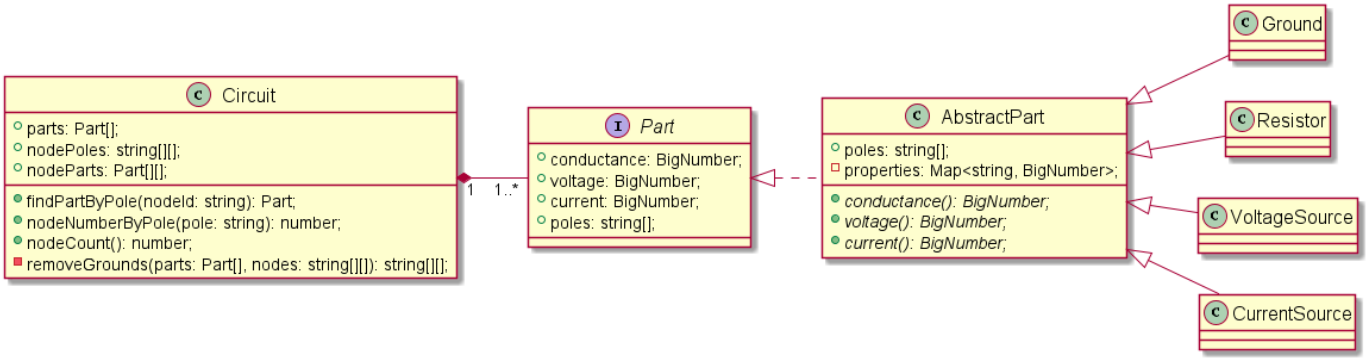

Figura 13: Classes principais

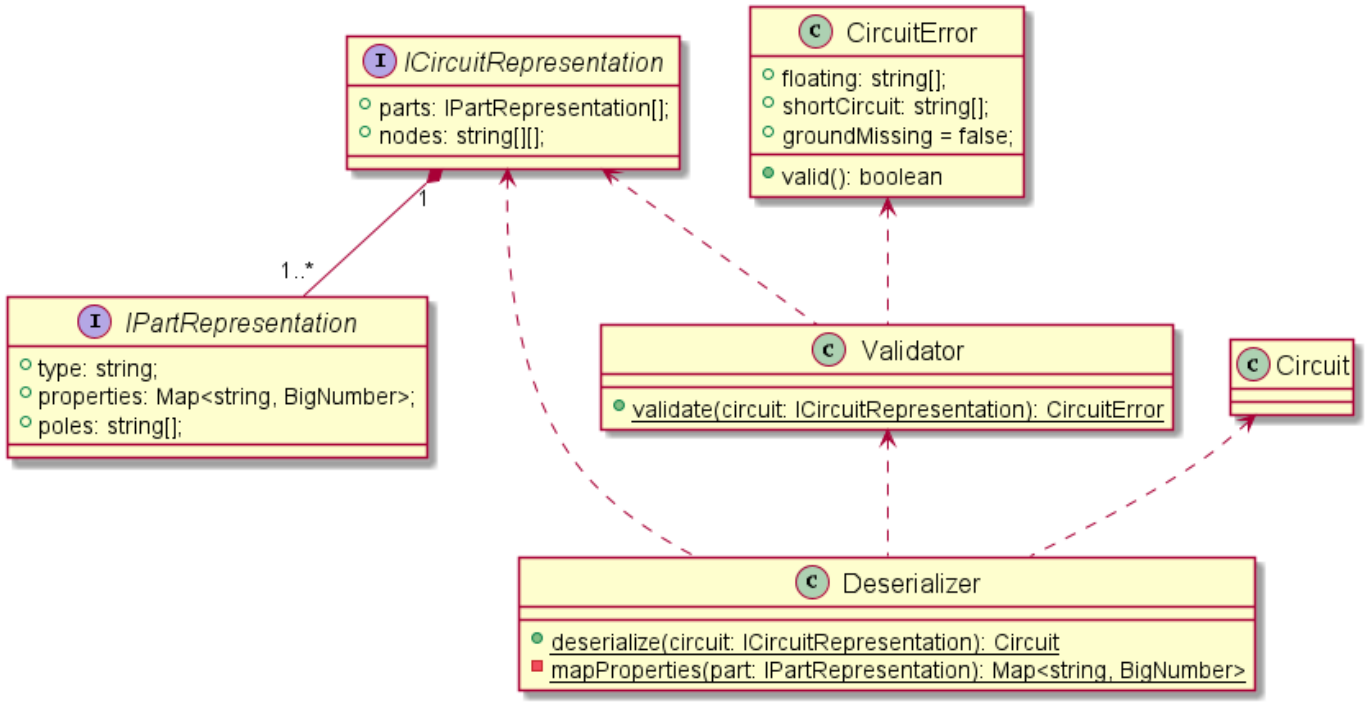

Figura 14: Leitura da representação

todas as propriedades necessárias para realizar os cálculos. As classes contidas são:

- *Part* representa um elemento do circuito, ela possui as propriedades da Lei de Ohm, tensão, resistência e corrente. Num elemento como o resistor, que possui apenas resistência, os outros atributos serão zero. Foi desenvolvido desta maneira para simplificar o mapeamento das propriedades ao montar a matriz e poderia ser utilizado em outros elementos além dos utilizados. Como pode-se perceber no diagrama, os terminais são referenciados por *strings*. Esta decisão parte da facilidade de serializar referenciar as *strings* nos nós.
- *AbstractPart* possui implementações comuns a todos as suas classes derivadas, basicamente todas as classes derivadas apenas armazenam as propriedades para serem lidas pelo simulador, com exceção do terra, que é removido da lista.
- *Circuit* é a classe que contém todas as informações do circuito. A lista de elementos é a propriedade *parts*, cuja estrutura já foi comentada. A lista de nós é a propriedade *nodePoles*, que é uma matriz de *strings*, cada linha da matriz se refere a um nó e as colunas são as referências dos nós que estão nos elementos. A propriedade *nodeParts* é uma matriz que indica que um nó está contido em um elemento, é útil para evitar muitas iterações para descobrir qual elemento um dado terminal se refere. Os métodos *findPartByPole, nodeNumberByPole* e *nodeCount* são utilizados para evitar repetição de código em outras classes, pois são frequentemente executados. O método *removeGround* procura o nó com o terra e o remove do circuito. Como os nós com terra não entram na análise de nenhuma matriz, ele já é removido no início da análise, para simplificar o algoritmo.

O diagrama da figura 14 mostra as classes que transformam a representação passada pela interface em uma instância da classe *Circuit* e fazem a validação do circuito. Suas classes

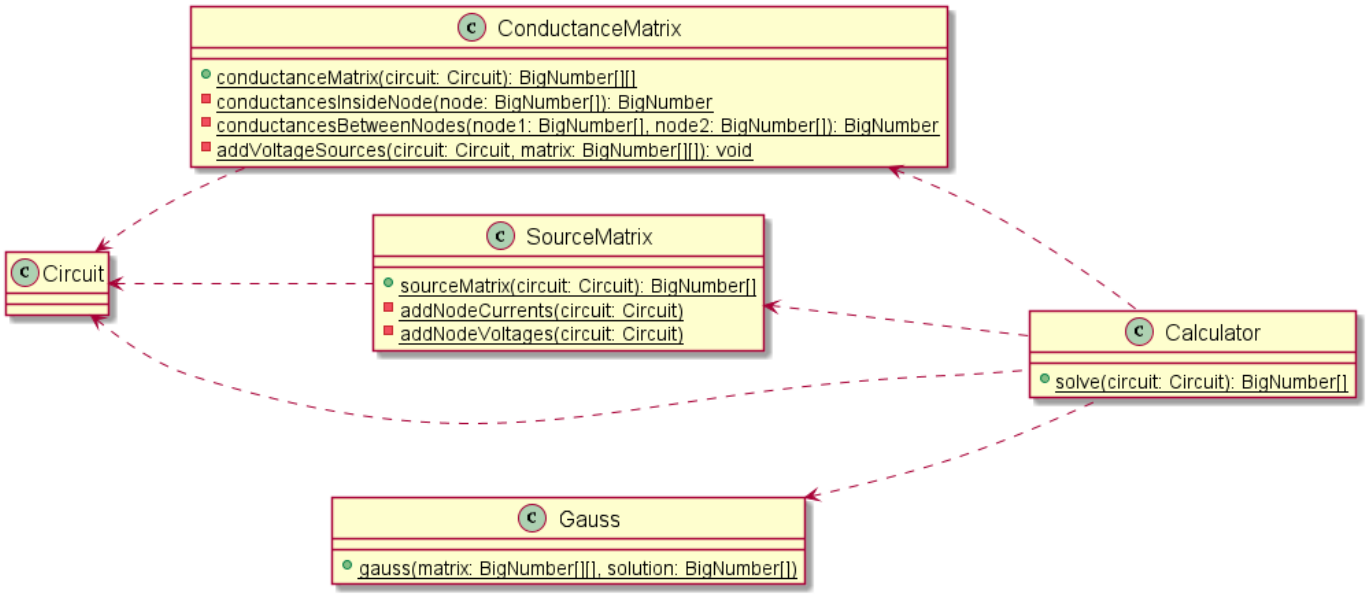

Figura 15: Diagrama de classes da análise nodal

são:

- As interfaces *IPartRepresentation* e *ICircuitRepresentation* tem estrutura que reflete ao que a interface informa ao simulador. *IPartRepresentation* tem uma propriedade não presente na classe *IPart*, que indica qual elemento do circuito aquele objeto se refere, no simulador o tipo do elemento é referenciado pelo nome da classe.
- A classe *Validator* possui um método que valida a entrada do circuito, retorna uma instância de *CircuitError*.
- A classe *CircuitError* contém o método *valid* para simplicar a leitura dos erros e possui outras propriedades que contém os terminais que estão com algum problema, separados por tipo de erro. A validação é capaz de detectar um terminal não conectado a nenhum outro, uma fonte em curto-circuito ou a falta do elemento terra.
- A classe *Deserializer* recebe a representação da interface e transforma numa instância da classe *Circuit*. O método *deserialize* é responsável por chamar a validação e mapear as propriedades para instâncias de classes herdadas de *Part*, através do método mapProperties.

O diagrama da figura 15 mostra a representação dos módulos referentes ao simulador do circuito, onde a análise dos nós acontece. Neste processo, é gerado as matrizes do sistema linear e ele é resolvido. Estão indicados como classes pela limitação pela especificação da UML, mas são módulos JavaScript e possuem as funções indicadas. Todos os módulos trabalham com o objeto *Circuit*.

- A função *solve* no módulo *Calculator* abstrai a complexidade da resolução do circuito, é o ponto de entrada do simulador e responsável por delegar as responsabilidades as outras.
- O módulo *ConductanceMatrix* exporta a função *conductanceMatrix* resposável por gerar a matriz de condutâncias. A função *conductancesInsideNode* encontra a soma das condutâncias conectadas a um nó do circuito, utilizado para montar a diagonal principal da matriz. A função *conductancesBetweenNodes* calcula a soma das condutâncias entre dois nós, que é usado para montar os outros elementos da matriz. Por último, a função *addVoltageSources* adiciona as linhas relacionadas às fontes de tensão na matriz de condutâncias, verificando em quais nós a fonte está conectada.
- O módulo *SourceMatrix* exporta *sourceMatrix* que monta a matriz de fontes. A função *addNodeCurrents* procura os nós com fontes de corrente, soma as fontes conectadas as cada um e preenche a matriz de fontes. A função *addNodeVoltages* adiciona uma linha para cada fonte de tensão no circuito, com o valor nominal de tensão da fonte.
- O módulo *Gauss* realiza a resolução do sistema linear, faz os passos de pivoteamento, escalonamento e substituição, recebe as matrizes de condutâncias e de fontes e retorna as tensões de nó com a corrente que passa por cada fonte de tensão.

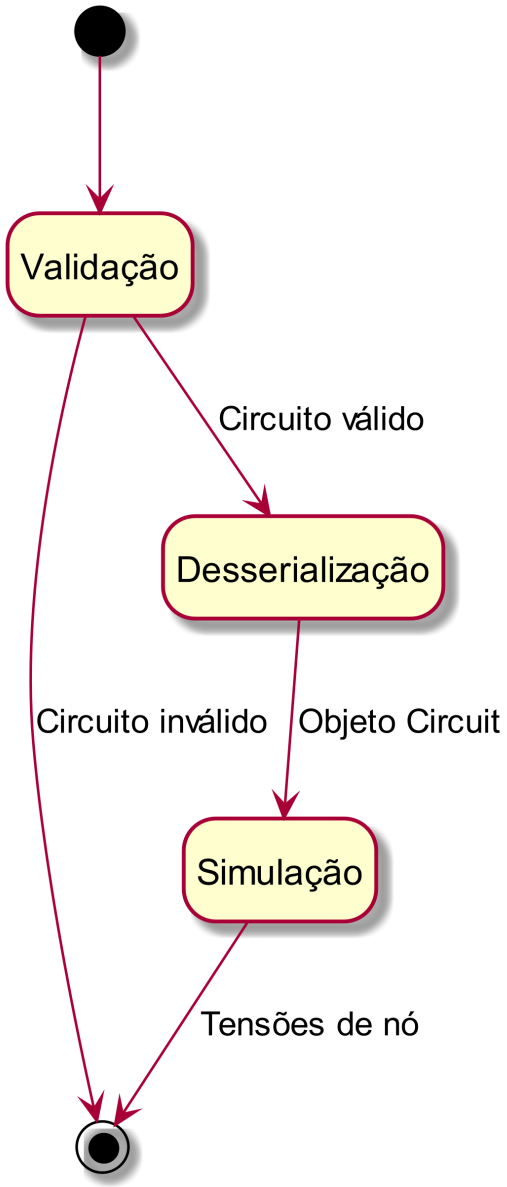

Figura 16: Diagrama de estados de todo simulador

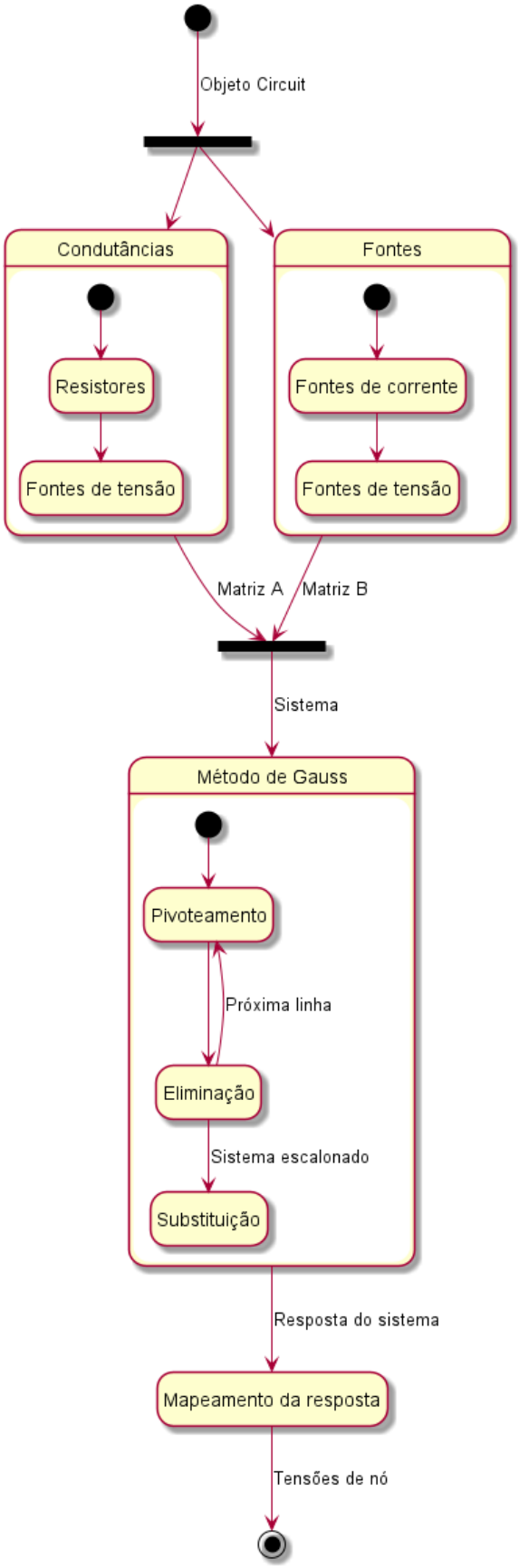

Figura 17: Diagrama de estados da análise nodal

Apresentaremos agora dois diagramas de estados da aplicação, mostrando o fluxo da simulação do circuito. O primeiro diagrama, na figura 16, mostra uma visão de todo o processo de simulação, explicaremos este processo em passos.

1) O ponto de entrada do simulador é o momento que a interface invoca a função exportada pela biblioteca do simulador, neste primeiro momento é feita a validação do circuito montado pelo usuário. Se foi encontrado algum erro na validação, uma exceção é lançada com o objeto *CircuitError*, finalizando prematuramente o processo de simulação. Caso o circuito informado seja válido, passamos para a etapa de desserialização.
2) Nesta etapa, a representação dos elementos é mapeada para uma classe do simulador, como indicado na figura 15 e suas propriedades são definidas. Com todos os elementos mapeados a classe Circuit é instanciada e neste momento, é feito um preprocessamento para preencher as listas da classe para facilitar as buscas de elementos e de nós. Quaisquer nós com o terra são removidos da lista neste momento, pois não devem fazer parte da análise.
3) Por fim, é feito os cálculos da simulação, que explicaremos a seguir em outro diagrama, retornando as tensões de nó à interface.

Como a análise nodal é a parte de maior interesse deste trabalho, o diagrama de estados da figura 17 mostra cada passo do processamento.

1) A análise começa chamando a função *solve*, com o objeto *Circuit*.
2) Primeiramente a matriz de condutâncias é montada. A lista *nodeParts* é percorrida em cada nó para identificar quais são os resistores ali conectados e também é feita uma análise de quais nós cada resistor está conectado, para preencher a primeira fase da matriz de condutâncias. Para as fontes de tensão, percorremos a lista *parts* para identificar os nós conectados e o sentido da fonte.
3) Para montar a matriz de fontes, iteramos sobre a lista *nodeParts*, somando ou subtraindo as fontes de corrente presentes em cada nó, dependendo do sentido da corrente naquele ponto. Depois disso, o resto da matriz é preenchida com os valores nominais das fontes de tensão. Neste passo, toma-se o cuidado de adicionar as fontes de tensão mesma ordem do passo anterior, para que não haja divergência entre os valores. Ao fim deste passo, temos o sistema pronto para a solução
4) Neste passo realizamos a solução do sistema linear, como explicado anteriormente, uma matriz estendida é montada com as matrizes A e B. Então, iteramos em cada linha da matriz, trocando as linhas abaixo da atual, conforme necessário para o pivoteamento, após o pivoteamento, a eliminação é realizada. Com o sistema escalonado, a substituição regressiva é feita, retornando a resposta do sistema linear.
5) Como a resposta do sistema linear é simplesmente um vetor de números contendo as tensões de nó e as correntes que passam pelas fontes, há um mapeamento para retornar apenas as tensões de nó, com a referência de um terminal conectado a este nó. Este mapeamento é necessário pois a ordem que a interface passa para o simulador pode ser alterada durante a análise, o que facilita também a exibir a tensão de nó na interface.

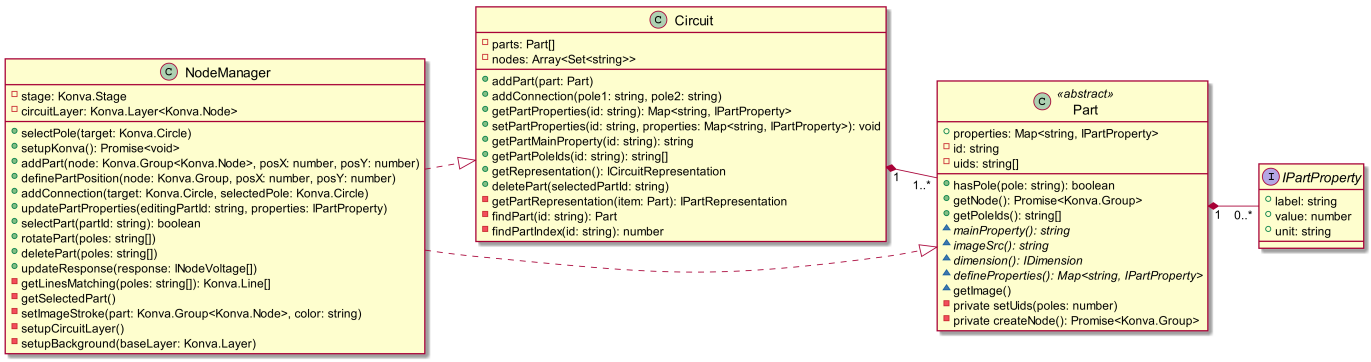

Figura 18: Diagrama de classes do circuito

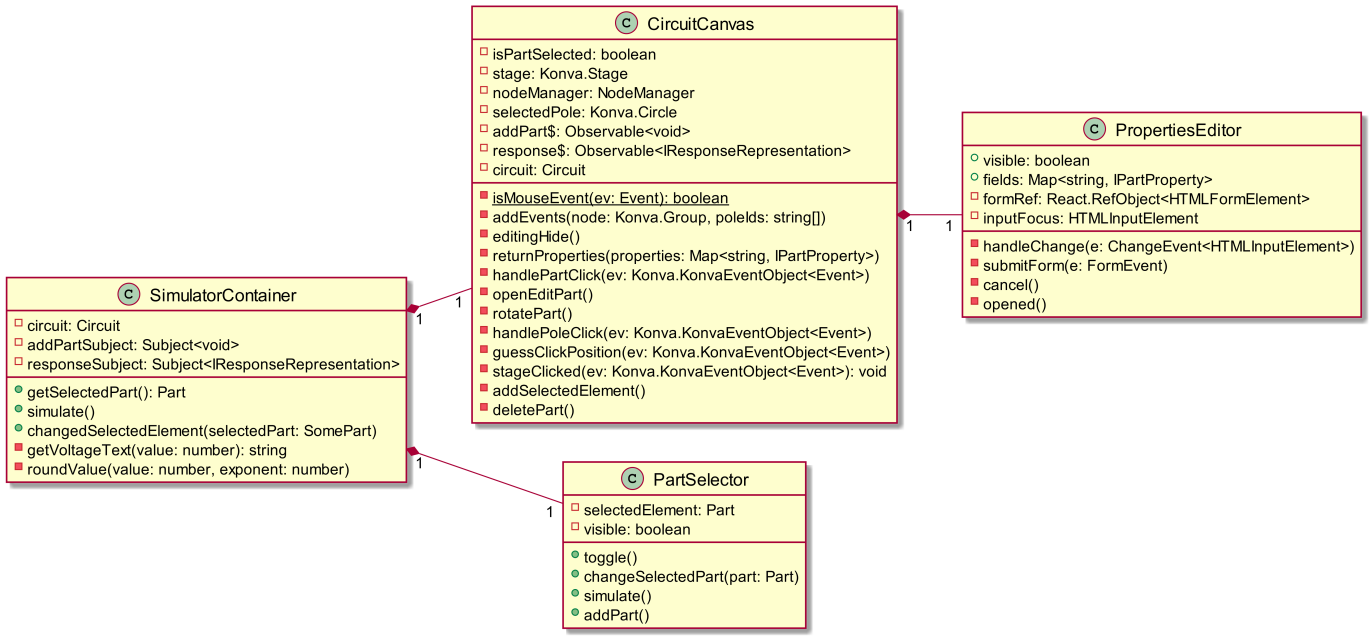

Figura 19: Diagrama de classes React

A. *Interface*

A interface é um projeto construído com o framework React, permite o usuário do aplicativo a montar o circuito. Para esta parte, não será mostrado os diagramas de estados, porque o uso da interface já foi apresentado em detalhes previamente e a combinação de estados é muito grande, porque há basicamente uma tela no aplicativo e ela apresenta muitas opções. Os diagramas de classes também não contemplarão todas as classes, serão mostradas apenas as mais relevantes para demonstrar o funcionamento do aplicativo. Como há muitas interações com o usuário e desenhar o circuito é uma tarefa complexa, as classes possuem muitas propriedades e métodos, então explicaremos apenas os principais.

A figura 18 mostra as classes que armazenam a representação do circuito, este diagrama omite as classes derivadas de *Part*, pois todas contém os mesmos métodos com implementações diferentes.

A interface *IPartProperty* contém as definições de uma propriedade de um elemento, que contempla um nome para exibição, seu valor e a unidade de medida utilizada no circuito.

Cada objeto Part contém um identificador que será utilizado para referenciar o elemento ao manipular o canvas, os identificadores dos terminais e as propriedades do elemento. O método *mainProperty* indica qual é a propriedade que deve ser mostrada ao lado do elemento na área do circuito, atualmente os elementos utilizados possuem apenas uma propriedade, porém, com suporte a novos elementos, pode ser que ele possua mais de uma propriedade. *imageSrc* retorna o arquivo de imagem do elemento atual, para que ele seja renderizado. Todo elemento é criado com um valores padrão para cada propriedade, que é definido pelo método *defineProperties*. Por fim, o método *createNode* é chamado pelo construtor para criar a instância do objeto que será renderizado no *canvas*.

A classe *Circuit* possui apenas as propriedades que armazenam os elementos e os nós do circuito. Note que o *Konva* também usa a nomenclatura *node* para se referir aos objetos renderizados, como o método de *Part*, *createNode* mostra, porém a classe *Circuit* não altera os elementos na interface, apenas trabalha com a representação do circuito. Ela abstrai os comportamentos da classe *Part* para as classes que trabalham

com o circuito, para simplificar o trabalho.

Os métodos *addPart* e *addConection* adicionam elementos na lista de elementos e de nós do circuito. Ao adicionar uma conexão entre dois elementos, é feita uma busca na lista de nós do circuito, se nenhum nó com algum dos dois terminais é encontrado, um novo nó é criado com os dois terminais. O método *getRepresentation* gera uma representação conforme a esperada pelo simulador, as representações dos elementos são geradas pelo método *getPartRepresentation*. Os demais são metódos para simplificar as buscas nas listas e para manipular os elementos do circuito.

NodeManager é uma classe que coordena a organização dos elementos dentro do *stage*, um objeto do framework *Konva* que permite renderizar elementos no *canvas*. Ela trabalha com a adição, remoção, posicionamento e alteração de propriedades dos elementos do circuito. O *Konva* permite que haja sobreposição de camadas, utilizando esta funcionalidade, a camada inferior possui o desenho da grade para posionar os elementos e a superior ficam os elementos do circuito. Com isso, a propriedade *circuitLayer* é a camada onde os elementos são inseridos.

O método *selectPole* altera a cor de um terminal quando ele é selecionado, tornando-o branco. O método *setupKonva*, juntamente com *setupCircuitLayer* e *setupBackground* inicializam a área do circuito, definindo o tamanho e desenhando a grade. O método *addPart* adiciona um elemento na grade do *canvas*, a é necessário que seja indicada a posição porque se a posição que o elemento é adicionado é relativa à parte do *canvas* que o usuário está visualizando. *definePartPosition* é importante para manter o elemento alinhado à grade quando o usuário o está movimentando. *selectPart* marca ou desmarca um elemento como selecionado, *rotatePart* gira o elemento em 90 graus, com relação ao seu terminal esquerdo e *deletePart* apaga o elemento selecionado, as conexões entre os elementos são apagadas com o uso do método *getLinesMatching*, que encontra as linhas que partem daquele terminal. O método *updateResponse* cria um elemento de texto com a tensão de nó e o posiciona próximo a um terminal deste nó.

O diagrama da figura 19 mostra as principais classes que representam as telas do sistema, elas são gerenciadas pelo framework *React* e estendem da classe *React.Component*, este detalhe foi omitido do diagrama para não poluí-lo visualmente. Os métodos que fazem parte do ciclo de vida do componente também não estão representados no diagrama. No React, é comum que uma classes sejam inseridas umas nas outras como módulos da tela, se comunicando através de propriedades passadas pelo componente "pai", quando é necessário uma comunicação proveniente do componente "filho", é necessário que o pai tenha passado uma função como propriedade. Por isso, no diagrama, cada classe está contida em outra, que no final estão contidas em *SimulatorContainer*, que centraliza as informações que são compartilhadas entre os demais componentes. O termo componente denota uma instância de uma classe *React.Component*.

O componente *PropertiesEditor* é responsável por tratar a tela do formulário de edição das propriedades de um elemento do circuito. Ela é mostrada quando a propriedade *visible* é definida. Possui métodos que lidam com o formulário, cuja definição não entraremos em detalhes porque a forma que o *React* trabalha com formulários é um pouco complexa. Ela recebe e retorna a propriedade *fields*, que é alterada quando o usuário toca no botão "Salvar".

O componente *PartSelector* corresponde ao menu superior da tela principal, onde os elementos são exibidos. Tem as propriedades *selectedPart* que indica qual é o último elemento selecionado pelo usuário, para que a interface saiba qual deve ser o elemento inserido no circuito e *visible*, que indica que se o menu está escondido.

O método *toggle* exibe ou esconde o menu, de acordo com seu estado atual. *changeSelectedPart* altera o elemento selecionado e *addPart* notifica o *container* que o usuário deseja adicionar um elemento no circuito. Já o método *simulate* notifica que o usuário tocou no botão de simular o circuito.

Antes de explicar as próximas classes, é necessário explicar o que é *Observable* e *Subject*, estes objetos são parte da biblioteca RxJS, que implementa o padrão de projeto *Observer*. Um *Observable* é um objeto que permite que escuta eventos de forma assíncrona e chama um função quando isto acontece, ele possui a restrição de só ouvir eventos de uma forma particular, um *callback* passado em seu construtor. Já o *Subject* é um objeto híbrido, que permite que um evento seja lançado a qualquer momento em sua fila e permite gerar um *Observable* que escuta esses eventos. A comunicação entre componentes é restrita por funções inseridas nas propriedades, que é de difícil implementação e baixa legibilidade, então o *Observable* é um objeto único que simplica a comunicação.

A classe *CircuitCanvas* é a tela principal do circuito e realiza o tratamento de eventos a respeito do circuito. Inicialmente, a classe *NodeManager* não existia e essa classe era responsável pelos eventos e pelo código referente a renderização, o que a deixou muito extensa. Por isso as duas foram quebradas para evitar a centralização de responsabilidades, mas ainda assim não foi possível delegar todas as funções para *NodeManager*. Por isso ela compartilha algumas propriedades iguais as presentes na outra classe. As demais propriedades são dois *Observables*, *addPart$* e *response$*, cujos eventos indicam que um elemento novo deve ser posicionado no circuito ou que os resultados da simulação devem ser mostrados.

O método *addEvents* cria os eventos que serão escutados pelo *canvas*. *editingHide* trata o cancelamento da edição de um elemento. Os métodos *handlePartClick* e *handlePartClick* tratam os toques/cliques nos elementos e nos terminais e *guessClickPosition* busca a posição da área selecionada, como a implementação é diferente para um clique de mouse e do toque na tela, há um tratamento feito pelo método *isMouseEvent*. *returnProperties* recebe as propriedades editadas no formulário e chama o método *updatePartProperties* para alterar os valores no circuito. *openEditPart* mostra o formulário de edição para o elemento selecionado. Os demais métodos já tem o comportamento explicado na classe *NodeManager*, já que eles apenas o evento, mas delegam o evento para a outra classe.

A última classe é *SimulatorContainer* que trata a lógica compartilhada entre demais os componentes. Possui a instância de *Circuit* e os *Subjects addPartSubject* e *responseSubject* que passam os eventos recebidos de *PartSelector* para *CircuitCanvas*. O método *changedSelectedElement* notifica *Circuit-*

Canvas que um elemento deve ser adicionado. Seu método *simulate* é chamado pela notificação de *PartSelector*, monta a representação utilizando-se do método *getRepresentation* e chama o simulador. Quando recebe a resposta, usa o método *getVoltageText* e *roundValue* para gerar um texto melhor para exibição, arredondando os valores para três casas decimais e utiliza dos prefixos do sistema internacional de unidades para representar valores menores que zero. Se o circuito informado for inválido, apresenta uma mensagem de alerta de acordo com o erro encontrado.

VI. Conclusão

Este trabalho apresentou um aplicativo para dispositivos Android capaz de simular circuitos resistivos com fontes independentes apenas em corrente contínua. O aplicativo permite o usuário montar uma representação gráfica do circuito de forma simples e intuitiva e obter as tensões em cada nó do circuito como resposta. O circuito para simulação pode conter resistores, fontes de corrente e fontes de tensão.

O objetivo deste aplicativo é atender alunos de ensino médio e superior que necessitam realizar simulações de pequenos circuitos, com uma ferramenta simples de ser utilizada, visto que a maioria dos simuladores é focado em circuitos maiores de aplicação prática. Como é destinada a dispositivos móveis, torna-se fácil a utilização em qualquer caso.

O aplicativo foi desenvolvido como uma aplicação *web*, utilizando HTML, CSS e TypeScript e é capaz de ser utilizada diretamente por um *browser*. Para auxiliar o desenvolvimento da interface, foi utilizado as bibliotecas React e Konva.

Para a inspeção do circuito foi utilizada a técnica da análise nodal modificada, que permite que resistores, fontes de tensão e de corrente sejam modelados em um sistema linear por meio de uma matriz de condutâncias e uma matriz de fontes, formando as matrizes A e B do sistema. A resolução do sistema linear é realizada pelo método de Gauss com pivoteamento parcial.

O aplicativo poderia ser melhorado com a possibilidade de realizar outros tipos simulações além da atual. Poderia haver análises fasoriais e transiente. Para isso, seria necessária a inclusão de outros elementos disponíveis para simulação, tais como fontes variantes no tempo, capacitores e indutores. Também seria interessante a adição de fontes dependentes, dada a sua recorrência em circuitos didáticos.

Referências

[1] L. W. Nagel and D. Pederson, "Spice (simulation program with integrated circuit emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr 1973. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[2] P. W. Tuinenga, *A Guide to Circuit Simulation and Analysis Using PSpice*. Englewood Cliffs, New Jersey, USA: Prentice Hall, 1995.

[3] "About | PSpice," OrCAD, (Acessado em 2 de Novembro de 2019). [Online]. Available: https://www.pspice.com/about

[4] S. Jahn, "Background - Qucs Help," (Acessado em 2 de Novembro de 2019). [Online]. Available: http://qucs.sourceforge.net/tech/node14.html

[5] "Background - Qucs Help," Qucs Team, (Acessado em 2 de Novembro de 2019). [Online]. Available: https://qucs-help.readthedocs.io/en/latest/intro.html

[6] "EveryCircuit - Home," MuseMaze, (Acessado em 2 de Novembro de 2019). [Online]. Available: http://everycircuit.com/

[7] StatCounter, "Mobile Operating System Market Share Worldwide | StatCounter Global Stats," (Acessado em 10 de Novembro de 2019). [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[8] R. C. Dorf, *Introdução aos circuitos elétricos*, 8th ed. Rio de Janeiro, Brasil: LTC, 2012.

[9] C. W. Ho, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, no. 6, 6 1975.

[10] R. L. Burden, *Numerical Analysis*. Boston, MA, USA: Cengage Learning, 1997.

[11] R. W. Farebrother, *Linear Least Squares Computations*. CRC Press, 2009, p. 12.

[12] "Number.MAX_SAFE_INTEGER – JavaScript | MDN," Mozilla, (Acessado em 10 de Novembro de 2019). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER

[13] "HTML basics - Learn web development," Mozilla, (Acessado em 6 de Novembro de 2019). [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[14] "What is CSS? - Learn web development | MDN," Mozilla, (Acessado em 6 de Novembro de 2019). [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

[15] "TypeScript - JavaScript that scales," Microsoft, (Acessado em 6 de Novembro de 2019). [Online]. Available: https://www.typescriptlang.org/index.html

[16] "What is JavaScript? - Learn web development | MDN," Mozilla, (Acessado em 17 de Novembro de 2019). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

[17] "React – Uma biblioteca JavaScript para criar interfaces de usuário," Facebook, (Acessado em 6 de Novembro de 2019). [Online]. Available: https://pt-br.reactjs.org/

[18] A. Lavrenov, "Documentation | Konva - JavaScript 2d canvas library," (Acessado em 6 de Novembro de 2019). [Online]. Available: https://konvajs.org/docs/

[19] R. Pressman, *Engenharia de Software*. AMGH, 2011, pp. 727–742.

[20] "JavaScript modules - JavaScript | MDN," Mozilla, (Acessado em 6 de Novembro de 2019). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules