IMPLEMENTAÇÃO DE UMA BIBLIOTECA PARA A SOLUÇÃO DO FLUXO DE POTÊNCIA ATRAVÉS DA LINGUAGEM DE MODELAGEM MATEMÁTICA MODELICA

Ricardo Mota Henriques*, Marcelo Aroca Tomim*, Matheus Lacerda Machado*, João Alberto Passos Filho*

*Universidade Federal de Juiz de Fora - Rua José Lourenço Kelmer, s/nº
Faculdade de Engenharia, Sala 4274 - Juiz de Fora - MG - CEP 36.036-330

Emails: ricardo.henriques@ufjf.edu.br, marcelo.tomim@ufjf.edu.br, matheus.machado@engenharia.ufjf.br, joao.passos@ufjf.edu.br

Abstract— In the last decades, the development of computational tools for analysis and simulation of electrical power systems requires the knowledge of structured and object oriented programming languages. In these tools, thorough understanding of data and calculation structures is fundamental for enabling any modification or inclusion of new mathematical models. This knowledge, however, remains usually restricted to the program developers. In this paper, the mathematical modeling language - Modelica - appropriate for hybrid systems modeling through sets of algebraic-differential equations is explored. In order to verify the applicability of the Modelica language to electric power systems, this paper presents the basis for representing the load flow problem in a generic manner. Using tutorial cases, results are compared with the ones obtained with the load flow program Anarede, showing the adequacy of the language to this type of development.

Keywords— Modelica®, Load Flow, Electric Power Systems.

Resumo— Nas últimas décadas o desenvolvimento de ferramentas computacionais de análise e simulação de sistemas elétricos de potência passou a requerer o conhecimento das linguagens estruturadas e das orientadas a objeto. O domínio das estruturas de dados e de cálculo nestes programas é fundamental para alteração ou inclusão de modelos matemáticos dos equipamentos do sistema. No entanto, este conhecimento, normalmente, permanece restrito aos desenvolvedores dos programas. Com essa dificuldade em mente, neste artigo explora-se uma nova linguagem de descrição de modelos matemáticos - Modelica - apropriada para a caracterização de sistemas híbridos, através de conjuntos de equações algébricos-diferenciais. Para comprovar a aplicabilidade da linguagem Modelica em sistemas elétricos de potência, o artigo apresenta uma abordagem genérica para a representação do problema de fluxo de potência. Utilizando casos tutoriais, os resultados obtidos são comparados com os do programa Anarede, comprovando a adequação da linguagem a este tipo de desenvolvimento.

Palavras-chave— Modelica®, Fluxo de Potência, Sistemas Elétricos de Potência.

1 Introdução

Por muitos anos engenheiros eletricistas vem despendendo esforços na modelagem do problema de fluxo de potência para sistemas elétricos de grande porte (SEP), em função da sua importância nos processos relacionados ao planejamento da operação e da expansão ao redor do mundo. Uma série de métodos elaborados especificamente para a solução do problema de fluxo de potência para sistemas de grande porte culminaram no amplamente aceito método de *Newton-Raphson* aliado às técnicas de esparsidade. Essa combinação possibilita a solução do problema de fluxo de potência com convergência quadrática ao mesmo tempo que mantém limitado o uso de memória tanto para armazenamento como para acesso (Tinney & Walker 1967, Tinney & Hart 1967, Monticelli 1983).

Observando a história recente do desenvolvimento de ferramentas de simulação computacional para a área de SEP, sejam de caráter estático (análises de fluxo de potência e curto-circuito) ou dinâmico (análise de transitórios eletromagnéticos e eletromecânicos), retornamos a década de 1970, onde predominou o uso da linguagem FORTRAN. Os desenvolvimentos nesta linguagem marcaram a primeira geração de programas de simulação, dadas a robustez, desempenho e adequação aos problemas típicos do setor de energia. Já na década de 1980, com o crescimento da linguagem C/C++ (trazendo a modelagem orientada a objetos) e a expansão da capacidade de processamento e memória dos computadores, os simuladores da área de SEP expandiram suas funcionalidades, tanto do ponto de vista matemático quanto das funcionalidades para o usuário, através da inclusão de interfaces gráficas que aumentaram as possibilidades de análise e tratamento dos resultados.

Embora as ferramentas computacionais tenham acompanhado a evolução das técnicas de programação e dos computadores, a reutilização do código em futuras aplicações não foi uma característica considerada. Portanto, a inclusão ou alteração de modelos nesses programas acabam dependendo essencialmente dos desenvolvedores dos mesmos. Assim, ao surgir uma demanda por expandir as possibilidades de uma determinada simulação, o suporte ao proprietário da ferramenta deve ser acionado. Via de regra o custo de novos desenvolvimentos é elevado, podendo inviabilizar sua realização. Neste sentido, universidades e centros de pesquisa estudam alternativas que possibilitem a construção de plataformas computacionais abertas, onde seja possível ao próprio engenheiro construir modelos e métodos (Manzoni 2005, Agostini 2002, Dotta 2003).

Com o objetivo de modelar sistemas complexos multi-domínio e de grande porte, mantendo independência de qualquer plataforma de simulação, a linguagem de modelagem matemática Modelica® foi desenvolvida. Modelica® é a especificação de uma lingua-

gem não-proprietária, orientada a objetos, destinada à modelagem genérica de sistemas dinâmicos híbridos (com variáveis contínuas e discretas) através de conjuntos de equações algébrico-diferenciais. Esta linguagem permite a modelagem de sistemas que envolvem múltiplos campos do conhecimento tais como cinemática, eletricidade, eletrônica, hidráulica, termodinâmica, química, controle, etc. Além disso, devido a sua intrínseca orientação a objetos, a linguagem Modelica® também serve como meio de troca de conhecimento entre pesquisadores de diferentes campos do conhecimento. Ou seja, um modelo desenvolvido em linguagem Modelica® pode reter anos de experiência e desenvolvimento em um formato não-proprietário e aberto, que pode ser prontamente disponibilizado ao público em geral (The Modelica Association 2014, Tiller 2001, Fritzson 2004).

O principal objetivo deste artigo é apresentar as bases para a modelagem genérica do problema de fluxo de potência empregando a linguagem Modelica®. A abordagem proposta foi testada na plataforma OpenModelica (The Open Source Modelica Consortium 2018) - OMEdit - para dois sistemas teste e comparados com o programa Anarede (Eletrobras Cepel 2017).

2 Linguagem Modelica

A linguagem Modelica® é uma linguagem universal, não-proprietária, orientada a objetos, para modelagem de sistemas físicos complexos, cujo nome é uma marca registrada da empresa sem fins lucrativos *The Modelica Association*, baseada em Linköping, Suécia (The Modelica Association 2014). A ideia básica da linguagem Modelica® foi estabelecer uma linguagem que expressasse o comportamento de modelos de sistemas de uma ampla gama de áreas em engenharia, sem limitá-los a uma ferramenta particular e/ou comercial (Tiller 2001, The Modelica Association 2014).

A linguagem Modelica® não se tornou apenas uma linguagem para modelagem, mas também de intercâmbio de modelos físicos, tanto para o meio acadêmico como industrial. É especificada por um conjunto de regras que visam traduzir as classes definidas em conjuntos de equações diferenciais, algébricas e discretas, conhecidas como equações algébrico-diferenciais híbridas. Estes conjuntos de equações formam os modelos de simulação de sistemas. Devido ao caráter genérico da linguagem Modelica®, modelos físicos de diversas áreas da engenharia podem ser descritos, tais como: circuitos elétricos, sistemas de tração automotivos, estabilidade de sistemas de potência, dinâmicas de veículos, sistemas hidráulicos e termodinâmicos, entre outros (Tiller 2001, Fritzson 2004).

Um esquema consiste em componentes que são conectados entre si através dos conectores (em inglês, *connectors* ou *ports*). Nesses conectores são descritas as regras de interconexão entre os mesmos. Por exemplo, em um pino elétrico (em inglês, *electrical pin*) são modeladas as leis de *Kirchhoff* que determinam que a soma das correntes é nula e a tensão é a mesma em todas as conexões.

Internamente, um componente é definido através de outro esquema ou através de equações na sintaxe Modelica®. A linguagem Modelica® é uma descrição textual que permite definir todas as equações associadas a um modelo, sejam essas diferenciais, algébricas ou discretas. Conjuntos de modelos podem, também, ser estruturados em bibliotecas (em inglês, *packages*).

É importante destacar que a linguagem Modelica® não surge para competir com as linguagens de programação largamente consolidadas, tais como C/C++ e FORTRAN. Do ponto de vista da ciência de computação, pode-se colocar a linguagem Modelica® em um nível muito mais elevado que qualquer linguagem de programação já citada, dado que essa se restringe a descrição de modelos matemáticos sem preocupação com detalhes acerca de suas soluções numéricas. Como exemplo, pode-se tomar o ciclo de solução de um modelo na plataforma OpenModelica, na qual o modelo descrito através da linguagem de modelagem matemática Modelica® é primeiramente traduzido para a linguagem de programação C/C++, para então ser compilado em linguagem de máquina por intermédio de um compilador específico. Além disso, esse código ainda é vinculado a bibliotecas de alto desempenho para a solução de sistemas de equações algébrico-diferenciais de grande porte. O método padrão de solução desses sistemas resultantes é o DASSL, implementada em FORTRAN na biblioteca DASPK2.0 de domínio público (Petzold 2018). Essa independência da linguagem Modelica® com relação às ferramentas de solução, torna-se um grande atrativo para desenvolvimento de ferramentes de simulação genéricas, extensíveis e de simples manutenção. Um sinal desse interesse é participação de empresas, como SAAB, Ericsson, Siemens, EDF, entre outras, em consórcios de pesquisas e desenvolvimentos nesse sentido (OpenCPS 2015).

Dentro do escopo do presente trabalho a linguagem Modelica® será empregada como ferramenta para a descrição de elementos típicos dos SEP's no problema de fluxo de potência.

3 Fluxo de Carga em Linguagem Modelica

Neste artigo, descreve-se os pontos principais da biblioteca Modelica® desenvolvida para a descrição do problema de fluxo de potência. Neste contexto, foram considerados modelos em regime permanente senoidal para linhas de transmissão, transformadores de dois enrolamentos, bancos de reatores e capacitores, a carga e a geração. O elemento `Barra` também foi modelado, apesar de não caracterizar um elemento físico real, mas um elemento de interface. Outro objetivo do elemento `Barra` foi o cálculo da tensão complexa na sua forma polar.

Em (Monticelli 1983) estão as equações básicas usadas na construção dos elementos modelados com a linguagem Modelica®. Nos subseções a seguir serão apresentados os elementos de forma individual, acompanhados do código Modelica® escrito para concretizar

a existência do elemento dentro do ambiente de simulação, que no caso deste trabalho foi o OMEdit (Open Source Modelica Consortium 2018). Juntos, estes elementos deram origem a um pacote (*package*) que pode ser usado na construção de SEP's, a partir do uso recursivo dos elementos, bastando arrastar de dentro do pacote o elemento a ser utilizado e interligá-lo a outros elementos, até que toda a rede elétrica e sua topologia estejam completas.

A leitura dos dados necessários em cada elemento foi programada no Modelica® de forma que os dados são lidos diretamente de arquivos textos individualizados para cada elemento. Estes arquivos foram gerados a partir do arquivo texto padrão **PWF** do programa Anarede, que contêm todos os dados do SEP. A conversão do arquivo **PWF** para os arquivos individuais dos elementos é feito através de um conversor, implementado em linguagem Python. O conversor recebe o arquivo **PWF** como entrada e gera um arquivo texto para cada elemento do SEP.

A modelagem na linguagem Modelica® dos elementos do pacote para análise do fluxo de potência será explicada de forma sucinta, destacando no código Modelica® de cada elemento as conexões, as equações e a leitura dos dados para o modelo.

3.1 Linha de Transmissão

Segundo (Monticelli 1983), uma linha de transmissão é modelada a partir da Equação 1. A partir das correntes fasoriais injetadas em p.u. do lado p e do lado n (Figura 1), calcula-se o valor das tensões complexas nestes respectivos pontos, também em p.u.. Na Figura 1 nota-se que a linha de transmissão tem dois terminais, onde são injetadas as correntes $\overline{I}_p$ e $\overline{I}_n$.

$$\overline{I}_p = \overline{y}_{pn}\left(\overline{V}_p - \overline{V}_n\right) + \overline{y}_{sh}\overline{V}_p \tag{1a}$$

$$\overline{I}_n = \overline{y}_{pn}\left(\overline{V}_n - \overline{V}_p\right) + \overline{y}_{sh}\overline{V}_n \tag{1b}$$

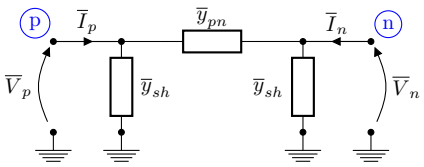

Figura 1: Modelo π de linha de transmissão.

O código Modelica® para a linha de transmissão representada pela Figura 1 e pela Equação 1, foi implementado no OMEdit, que está reproduzido no Código 1. No Código 1 serão destacados os itens fundamentais para a construção deste elemento e também dos demais.

O primeiro destaque está na primeira linha do Código 1, onde foi usado o comando Modelica® **model**. Todo modelo construído em linguagem Modelica® começa com a palavra chave **model**. No caso da linha de transmissão o modelo foi batizado de **TLine**. Todo o código entre a palavra chave **model** até a palavra chave **end** irá determinar o modelo da linha de transmissão.

```
model TLine
 import IC.Files.ReadingFile;
 import IC.Math.Utilities.deg2rad;
 import Modelica.ComplexMath.conj;
 parameter String EqpName;
 parameter Modelica.SIunits.Voltage Vini_From = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini_From");
 parameter Modelica.SIunits.Voltage Vini_To = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini_To");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini_From =
        ReadingFile(PathAndFileName = EqpName, Parmeter = "angleini_From");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini_To = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "angleini_To");
 parameter IC.Units.Percent R = ReadingFile(PathAndFileName = EqpName, Parmeter = "R");
 parameter IC.Units.Percent X = ReadingFile(PathAndFileName = EqpName, Parmeter = "X");
 parameter IC.Units.MegaReactivePower Q = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Q");
 parameter IC.Units.MegaApparentPower Slinha = 100;
 outer IC.Circuit.Basic.BaseData Data;
 Complex Sp(re(quantity = "Power", unit = "MW"), im(quantity = "Power", unit = "Mvar"))
        ;
 Complex Sn(re(quantity = "Power", unit = "MW"), im(quantity = "Power", unit = "Mvar"))
        ;
 IC.Circuit.Interfaces.PositivePin p(v.re(start = Vp_ini_re), v.im(start = Vp_ini_im),
        i.re(start = Ikm_ini_re + IkQ_ini_re), i.im(start = Ikm_ini_im + IkQ_ini_im));
 IC.Circuit.Interfaces.NegativePin n(v.re(start = Vn_ini_re), v.im(start = Vn_ini_im),
        i.re(start = (-Ikm_ini_re) + ImQ_ini_re), i.im(start = (-Ikm_ini_im) +
        ImQ_ini_im));
 protected
  parameter IC.Units.pu Gpn = R * 100 * (Data.Sbase / Slinha) / (R ^ 2 + X ^ 2);
  parameter IC.Units.pu Bpn = -X * 100 * (Data.Sbase / Slinha) / (R ^ 2 + X ^ 2);
  parameter Complex Ypn(re = Gpn, im = Bpn);
  parameter Complex Ysh(re = 0, im = Q / 2 / Data.Sbase);
  parameter Real Vp_ini_re = Vini_From * cos(deg2rad(angleini_From));
  parameter Real Vp_ini_im = Vini_From * sin(deg2rad(angleini_From));
  parameter Real Vn_ini_re = Vini_To * cos(deg2rad(angleini_To));
  parameter Real Vn_ini_im = Vini_To * sin(deg2rad(angleini_To));
  parameter Real Ikm_ini_re = Gpn * (Vp_ini_re - Vn_ini_re) - Bpn * (Vp_ini_im -
         Vn_ini_im);
  parameter Real Ikm_ini_im = Gpn * (Vp_ini_im - Vn_ini_im) + Bpn * (Vp_ini_re -
         Vn_ini_re);
  parameter Real IkQ_ini_re = -Q / 2 / Data.Sbase * Vp_ini_im;
  parameter Real IkQ_ini_im = Q / 2 / Data.Sbase * Vp_ini_re;
  parameter Real ImQ_ini_re = -Q / 2 / Data.Sbase * Vn_ini_im;
  parameter Real ImQ_ini_im = Q / 2 / Data.Sbase * Vn_ini_re;
 algorithm
  assert(Q >= 0, "Line charging must be equal or more than zero");
  p.i := Ypn * (p.v - n.v) + Ysh * p.v;
  n.i := (-Ypn * (p.v - n.v)) + Ysh * n.v;
  if Data.calcflow then
   Sp := p.v * conj(p.i) * Data.Sbase;
   Sn := n.v * conj(n.i) * Data.Sbase;
  else
   Sp := Complex(0, 0);
   Sn := Complex(0, 0);
  end if;
end TLine;
```

Código 1: Modelo de linha de transmissão.

O segundo destaque no Código 1 está na parte inferior e começa com a palavra chave **algorithm** (em outros elementos a palavra chave será **equation**, que tem função similar). Esta seção do modelo do elemento determina quem são as equações que o Modelica® deve resolver quando a linha de transmissão for utilizada. Deve-se observar que esta parte do Código 1 é a transcrição das equações na Equação 1. Cabe ressaltar que alguns elementos destas equações são parâmetros lidos via arquivos e outros são variáveis do modelo que serão calculadas. No caso da linha de transmissão, as admitâncias são parâmetros. Já as tensões e correntes são variáveis. Portanto, o número de equações e incógnitas deve ser igual para que a solução do problema seja possível e determinada. No caso das tensões e correntes, as letras **p/n** precedem o **i** da corrente e o **v** da tensão. Estes **p/n** são os chamados *eletrical pin's* e sua declaração determinam os pontos de interface elétrica do modelo com outros modelos. Na linha de transmissão, por haver o *pin* **p** e o *pin* **n**, haverá dois terminais de conexão, conforme pode ser comprovado ao observar a Figura 1.

Acima da palavra chave **algorithm** estão a declaração de parâmetros do modelo (palavra chave **parameter**), a definição das variáveis das equações (palavra chave **Complex**), a definição dos *eletrical pin's* (palavras chave **PositivePin/NegativePin**) e a

importação de outras bibliotecas e funções utilizadas no modelo (palavra chave **import**).

A importação dos parâmetros a partir da leitura de arquivos texto é feita através da função **ReadingFile**, permitindo a alteração dos valores dos parâmetros sem a necessidade de alteração do Código 1 ou de qualquer outro modelo desenvolvido de forma semelhante, como será visto nos outros elementos.

Deve-se esclarecer que a diferença entre as palavras chave **algorithm** e **equation** nos modelos apresentados aqui está ligada a forma de ordenação das equações no processo de solução das variáveis. Mais detalhes estão em (Fritzson 2004). No Código 1 também podem ser notados cálculos envolvendo parâmetros (palavra chave **parameter**). Estes cálculos são realizados tanto para inicializar valores em variáveis quanto para conversão de unidades.

3.2 Transformador de Dois Enrolamentos

O modelo do transformador de dois enrolamentos e suas equacões estão em (Monticelli 1983) e dão origem à Figura 2. Como na Subseção 3.1, das correntes fasoriais injetadas em p.u. do lado p e do lado n (Figura 2), calcula-se o valor das tensões complexas nestes respectivos lados, também em p.u.. Observando a Figura 2, constata-se que o transformador de dois enrolamentos terá dois terminais, onde são injetadas as correntes $\overline{I}_p$ e $\overline{I}_n$.

```
model TrafoConstTap
 import IC.Files.ReadingFile;
 import IC.Math.Utilities.deg2rad;
 import Modelica.ComplexMath.conj;
 outer IC.Circuit.Basic.BaseData Data;
 parameter String EqpName;
 parameter Modelica.SIunits.Voltage Vini_From = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini_From");
 parameter Modelica.SIunits.Voltage Vini_To = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini_To");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini_From =
        ReadingFile(PathAndFileName = EqpName, Parmeter = "angleini_From");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini_To = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "angleini_To");
 parameter IC.Units.Percent R = ReadingFile(PathAndFileName = EqpName, Parmeter = "R");
 parameter IC.Units.Percent X = ReadingFile(PathAndFileName = EqpName, Parmeter = "X");
 parameter IC.Units.pu a = ReadingFile(PathAndFileName = EqpName, Parmeter = "a");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg phi = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "phi");
 parameter IC.Units.MegaApparentPower Strafo = 100;
 Complex Sp(re(quantity = "Power", unit = "MW"), im(quantity = "Power", unit = "Mvar"))
        ;
 Complex Sn(re(quantity = "Power", unit = "MW"), im(quantity = "Power", unit = "Mvar"))
        ;
 IC.Circuit.Interfaces.PositivePin p(v.re(start = Vp_ini_re), v.im(start = Vp_ini_im),
        i.re(start = Ikm_ini_re + IAB_ini_re), i.im(start = Ikm_ini_im + IAB_ini_im));
 IC.Circuit.Interfaces.NegativePin n(v.re(start = Vn_ini_re), v.im(start = Vn_ini_im),
        i.re(start = (-Ikm_ini_re) + IAC_ini_re), i.im(start = (-Ikm_ini_im) +
        IAC_ini_im));
 protected
  parameter IC.Units.pu Gpn = R * 100 * (Data.Sbase / Strafo) / (R ^ 2 + X ^ 2);
  parameter IC.Units.pu Bpn = -X * 100 * (Data.Sbase / Strafo) / (R ^ 2 + X ^ 2);
  parameter Complex Ypn(re = Gpn, im = Bpn);
  parameter Real Vp_ini_re = Vini_From * cos(deg2rad(angleini_From));
  parameter Real Vp_ini_im = Vini_From * sin(deg2rad(angleini_From));
  parameter Real Vn_ini_re = Vini_To * cos(deg2rad(angleini_To));
  parameter Real Vn_ini_im = Vini_To * sin(deg2rad(angleini_To));
  parameter Real Ikm_ini_re = Gpn / a * (Vp_ini_re - Vn_ini_re) - Bpn / a * (Vp_ini_im
         - Vn_ini_im);
  parameter Real Ikm_ini_im = Gpn / a * (Vp_ini_im - Vn_ini_im) + Bpn / a * (Vp_ini_re
         - Vn_ini_re);
  parameter Real IAB_ini_re = 1 / a ^ 2 * (Gpn * Vp_ini_re - Bpn * Vp_ini_im);
  parameter Real IAB_ini_im = 1 / a ^ 2 * (Gpn * Vp_ini_im + Bpn * Vp_ini_re);
  parameter Real IAC_ini_re = Gpn * Vn_ini_re - Bpn * Vn_ini_im;
  parameter Real IAC_ini_im = Gpn * Vn_ini_im + Bpn * Vn_ini_re;
  parameter Complex aephi(re = a * cos(deg2rad(-phi)), im = a * sin(deg2rad(-phi)));
 algorithm
  p.i := (1 / a) ^ 2 * Ypn * p.v + (-1 / conj(aephi)) * Ypn * n.v;
  n.i := (-1 / aephi) * Ypn * p.v + Ypn * n.v;
  if Data.calcflow then
   Sp := p.v * conj(p.i) * Data.Sbase;
   Sn := n.v * conj(n.i) * Data.Sbase;
  else
   Sp := Complex(0, 0);
   Sn := Complex(0, 0);
  end if;
end TrafoConstTap;
```

Código 2: Modelo de transformador de dois enrolamentos com *tap* constante.

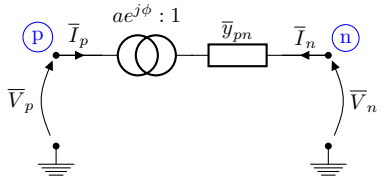

Figura 2: Modelo do transformador de dois enrolamentos com *tap* constante.

O código Modelica® para o transformador de dois enrolamentos representado pela Figura 2 e pela Equação 2, foi implementado no OMEDit, onde o Código 2 traz a implementação realizada. Ao observar o Código 2 nota-se que sua estrutura é muito similar à apresentada na Subseção 3.1, a menos das equações do modelo matemático.

$$\overline{I}_p = \left(\frac{1}{a}\right)^2 \overline{y}_{pn}\overline{V}_p - \left(\frac{1}{ae^{-j\phi}}\right)\overline{y}_{pn}\overline{V}_n \tag{2a}$$

$$\overline{I}_n = -\left(\frac{1}{ae^{j\phi}}\right)\overline{y}_{pn}\overline{V}_p + \overline{y}_{pn}\overline{V}_n \tag{2b}$$

3.3 Bancos de Reatores e Capacitores

O modelo para os bancos de reatores e capacitores *shunt* é mais simples se comparado aos modelos das linhas e dos transformadores. Os capacitores são representados por admitâncias com susceptâncias positivas e os reatores representados por admitâncias com susceptâncias negativas. Assim a Equação 3 pode representar tanto os reatores quanto os capacitores. A corrente no banco é função da tensão no terminal onde estes são conectados.

$$\overline{I}_p = \overline{y}_p\overline{V}_p \tag{3}$$

```
model Shunt_Capacitor
 import IC.Files.ReadingFile;
 import IC.Math.Utilities.deg2rad;
 import Modelica.Constants.pi;
 outer IC.Circuit.Basic.BaseData Data;
 parameter String EqpName;
 parameter Real status = ReadingFile(PathAndFileName = EqpName, Parmeter = "status");
 parameter Modelica.SIunits.Voltage Vini = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "angleini");
 extends IC.Circuit.Interfaces.Shunt(v.re(start = V_re), v.im(start = V_im), i.re(
        start = I_re), i.im(start = I_im));
 parameter IC.Units.MegaReactivePower NominalPower = ReadingFile(PathAndFileName =
        EqpName, Parmeter = "NominalPower");
 protected
  parameter Complex y(re = 0, im = NominalPower / Data.Sbase);
  parameter Real V_re = Vini * cos(deg2rad(angleini));
  parameter Real V_im = Vini * sin(deg2rad(angleini));
  parameter Real I_mod = NominalPower / Data.Sbase * Vini;
  parameter Real I_ang = deg2rad(angleini) + pi;
  parameter Real I_re = I_mod * cos(I_ang);
  parameter Real I_im = I_mod * sin(I_ang);
 equation
  if status == 1 then
   i = y * v;
  else
   i = Complex(0, 0);
  end if;
  assert(NominalPower >= 0, "Nominal Power must be greater than zero");
end Shunt_Capacitor;
```

Código 3: Modelo de banco de capacitores.

A Figura 3 indica que os bancos, ao contrário das linhas e transformadores, tem apenas um terminal, onde a corrente $\overline{I}_p$ depende do tipo de banco *shunt* utilizado. Observando o Código 3 e o Código 4 pode-se constatar que há apenas um terminal de conexão

(palavra chave **extends**). A diferença básica entre os códigos se resume ao sinal da susceptância da admitância $\overline{y}_p$. Os demais aspectos do código são comuns à Subseção 3.1.

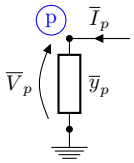

Figura 3: Modelo dos bancos reatores e capacitores.

```
model ReactorBank
 import Modelica.Utilities.Files.exist;
 import IC.Files.ReadingFile;
 parameter String EqpName;
 parameter Real status = ReadingFile(PathAndFileName = EqpName, Parmeter = "status");
 import IC.Math.Utilities.deg2rad;
 import Modelica.Constants.pi;
 parameter Modelica.SIunits.Voltage Vini = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "angleini");
 extends IC.Circuit.Interfaces.Shunt(v.re(start = V_re), v.im(start = V_im), i.re(
        start = I_re), i.im(start = I_im));
 parameter IC.Units.MegaReactivePower NominalPower = ReadingFile(PathAndFileName =
        EqpName, Parmeter = "NominalPower");
 outer IC.Circuit.Basic.BaseData Data;
 protected
   parameter Complex y(re = 0, im = NominalPower / Data.Sbase);
   parameter Real V_re = Vini * cos(deg2rad(angleini));
   parameter Real V_im = Vini * sin(deg2rad(angleini));
   parameter Real I_mod = NominalPower / Data.Sbase * Vini;
   parameter Real I_ang = deg2rad(angleini) - pi;
   parameter Real I_re = I_mod * cos(I_ang);
   parameter Real I_im = I_mod * sin(I_ang);
 equation
   if status == 1 then
     i = y * v;
   else
     i = Complex(0, 0);
   end if;
   assert(NominalPower <= 0, "Nominal Power must be less than zero");
end ReactorBank;
```

Código 4: Modelo de banco de reatores.

3.4 *Carga Tipo Potência Constante*

```
model Load_PQ
  extends IC.Circuit.Interfaces.Shunt(v.re(start = V_re), v.im(start = V_im), i.re(
        start = I_re), i.im(start = I_im));
  import IC.Math.Utilities.deg2rad;
  import IC.Files.ReadingFile;
  import Modelica.ComplexMath.conj;
  import Modelica.Math.atan2;
  outer IC.Circuit.Basic.BaseData Data;
  parameter String EqpName;
  parameter Modelica.SIunits.Voltage Vini = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Vini");
  parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini = ReadingFile(
        PathAndFileName = EqpName, Parmeter = "angleini");
  parameter Real status = ReadingFile(PathAndFileName = EqpName, Parmeter = "status");
  parameter IC.Units.MegaActivePower P = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "P");
  parameter IC.Units.MegaReactivePower Q = ReadingFile(PathAndFileName = EqpName,
        Parmeter = "Q");
  protected
   parameter Complex S(re = P / Data.Sbase, im = Q / Data.Sbase);
   parameter Real V_re = Vini * cos(deg2rad(angleini));
   parameter Real V_im = Vini * sin(deg2rad(angleini));
   parameter Real I_mod = sqrt((P / Data.Sbase) ^ 2 + (Q / Data.Sbase) ^ 2) / Vini;
   parameter Real I_ang = deg2rad(angleini) - atan2(Q, P);
   parameter Real I_re = I_mod * cos(I_ang);
   parameter Real I_im = I_mod * sin(I_ang);
  equation
   if status == 1 then
     S = v * conj(i);
   else
     i = Complex(0, 0);
   end if;
end Load_PQ;
```

Código 5: Modelo de carga de potência aparente constante.

O modelo para as cargas na configuração potência constante se assemelha aos modelos dos bancos de reatores e capacitores no tocante ao fato de que possuem apenas um terminal de interface. Porém sua equação é modelada em termos da potência aparente da carga (Equação 4), cujos valores das potências ativa e reativa em p.u. devem ser constantes. São calculadas a tensão $\overline{V}_p$ e a corrente $\overline{I}_p$ que atendem ao valor de $\overline{S}$ especificado.

$$\overline{S} = \overline{V}_p \overline{I}_p^* \tag{4}$$

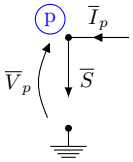

Figura 4: Modelo da carga.

A Figura 4 representa o modelo para carga potência constante, destacando que os sentidos convencionais da tensão e corrente complexas na carga são os considerados positivos. O Código 5 confirma a Equação 4, e a convenção adotada. Os demais aspectos do código permanencem comuns à Subseção 3.1.

3.5 *Geradores Tipo PV e Vθ*

O modelo dos geradores tipo PV e $V\theta$ implementado na linguagem Modelica® tem equações semelhantes as da carga tipo potência constante, pois o gerador é também é representado por uma potência aparente (Figura 5). A primeira diferença é que a potência deve ser injetada e não consumida no terminal, o que justifica a presença do sinal negativo na equação da potência na modelagem dos geradores. Além disso há dois tipos de geradores (PV e $V\theta$), caracterizados pelas variáveis que são especificadas e que impactam no cálculo da potência aparente injetada.

Para o gerador do tipo $V\theta$ as variáveis especificadas são o módulo (V_{esp}) e o ângulo (θ_{esp}) da tensão complexa $\overline{V}_p$ em p.u.. A Equação 5 mostra que o gerador do tipo $V\theta$ calcula a potência aparente $\overline{S}$ e a corrente $\overline{I}_p$ para que as variáveis especificadas sejam atendidas. O Código 6 traz a implementação do gerado do tipo $V\theta$ ou *slack*.

$$\overline{S} = -\overline{V}_p \overline{I}_p^* \tag{5a}$$

$$\overline{V}_p = V_{esp} \underline{/\theta_{esp}} \tag{5b}$$

```
model VoltageSource
 extends IC.Circuit.Interfaces.Shunt;
 import IC.Files.ReadingFile;
 import Modelica.ComplexMath.conj;
 import IC.Math.Utilities.polar2cart;
 parameter String EqpName;
 parameter Real status = ReadingFile(EqpName, Parmeter = "status");
 parameter IC.Units.pu Vmodulo = ReadingFile(EqpName, Parmeter = "Vmodulo");
 parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg Vfase = ReadingFile(
        EqpName, Parmeter = "Vfase");
 outer IC.Circuit.Basic.BaseData Data;
 Complex S;
 equation
  v = polar2cart(Vmodulo, Vfase);
  S = -v * conj(i) * Data.Sbase;
end VoltageSource;
```

Código 6: Modelo do Gerador tipo $V\theta$

Já para o gerador do tipo PV as variáveis especificadas são a potência ativa (P_{esp}) e o módulo da tensão

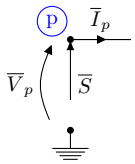

Figura 5: Modelo dos Geradores.

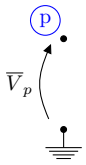

Figura 6: Modelo da Barra.

complexa (V_{esp}). Logo a Equação 6 mostra que o gerador do tipo PV calcula o ângulo de fase de $\overline{V}_p$, a potência reativa Q de $\overline{S}$ e a corrente $\overline{I}_p$ para que as variáveis especificadas sejam atendidas. O Código 7 traz a implementação do gerado do tipo PV.

$$\overline{V_p} = V_{p\_re} + jV_{p\_im} \tag{6a}$$
$$V_{esp}^2 = V_{p\_re}^{\ 2} + V_{p\_im}^{\ 2} \tag{6b}$$
$$P_{esp} + jQ = -\overline{V}_p \overline{I}_p^* \tag{6c}$$

```
model Generator_PV
  extends IC.Circuit.Interfaces.Shunt(v.re(start = V_re), v.im(start = V_im), i.re(
        start = I_re), i.im(start = I_im));
  import IC.Math.Utilities.deg2rad;
  import IC.Files.ReadingFile;
  import Modelica.ComplexMath.conj;
  outer IC.Circuit.Basic.BaseData Data;
  parameter String EqpName;
  parameter Boolean filexist = exist(EqpName);
  parameter String Parmeter;
  parameter Real status = ReadingFile(EqpName, Parmeter = "status");
  parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini = ReadingFile(
        EqpName, Parmeter = "angleini");
  parameter IC.Units.pu Vmodulo = ReadingFile(EqpName, Parmeter = "Vmodulo");
  parameter IC.Units.MegaActivePower P = ReadingFile(EqpName, Parmeter = "P");
  parameter IC.Units.MegaReactivePower Qini = ReadingFile(EqpName, Parmeter = "Qini");
  IC.Units.MegaReactivePower Q;
  protected
   parameter Real V_re = Vmodulo * cos(deg2rad(angleini));
   parameter Real V_im = Vmodulo * sin(deg2rad(angleini));
   parameter Real I_mod = sqrt((P / Data.Sbase) ^ 2 + (-Qini / Data.Sbase) ^ 2) /
         Vmodulo;
   parameter Real I_ang = deg2rad(angleini) - atan2(-Qini, P);
   parameter Real I_re = I_mod * cos(I_ang);
   parameter Real I_im = I_mod * sin(I_ang);
  equation
   if status == 1 then
    v.re ^ 2 + v.im ^ 2 = Vmodulo ^ 2;
    Complex(P / Data.Sbase, Q / Data.Sbase) = -v * conj(i);
   else
    i = Complex(0, 0);
    Q = 0;
   end if;
end Generator_PV;
```

Código 7: Modelo do Gerador tipo PV

3.6 Barra

O modelo de barramento ou simplesmente barra foi gerado com o objetivo de agilizar a captação dos valores das tensões nodais do SEP, construído com os modelos descritos anteriormente. Como pode ser observado na Figura 6, o modelo consiste apenas de um terminal para conectar os outros modelos.

$$\overline{V_p} = V_{p\_re} + jV_{p\_im} = V_p \angle \theta_p \tag{7a}$$
$$V_p = \sqrt{V_{p\_re}^{\ 2} + V_{p\_im}^{\ 2}} \tag{7b}$$
$$\theta_p = \arctan\left(\frac{V_{p\_im}}{V_{p\_re}}\right) \tag{7c}$$
$$\overline{I_p} = 0 \tag{7d}$$

A corrente no terminal p da Figura 6, pela lei de *Kirchhoff* das correntes, é igual a zero. O objetivo básico e único deste modelo é apenas obter o valor do módulo e do ângulo da tensão complexa $\overline{V_p}$. O Código 8 representa a implementação deste modelo, cujas equações são as descritas na Equação 7.

```
model BusBar
  import IC.Files.ReadingFile;
  import IC.Math.Utilities.deg2rad;
  parameter String EqpName;
  parameter Real status = ReadingFile(EqpName, Parmeter = "status");
  IC.Circuit.Interfaces.PositivePin p;
  parameter Modelica.SIunits.Voltage Vini = ReadingFile(EqpName, Parmeter = "Vini");
  parameter Modelica.SIunits.Conversions.NonSIunits.Angle_deg angleini = ReadingFile(
        EqpName, Parmeter = "angleini");
  Modelica.SIunits.Voltage V(start = Vini, min = 0.6, max = 1.5);
  Modelica.SIunits.Angle angle(start = deg2rad(angleini));
  algorithm
   V := sqrt(p.v.re ^ 2 + p.v.im ^ 2);
   angle := atan2(p.v.im, p.v.re);
   p.i := Complex(0);
end BusBar;
```

Código 8: Modelo da Barra.

4 Resultados

Para testar o pacote com os elementos de SEP's desenvolvidos, foram montados dois sistemas tutoriais: IEEE 14 barras e New England 39 barras. Os resultados da solução do fluxo de potência com a plataforma OpenModelica (The Open Source Modelica Consortium 2018) - OMEdit - serão analisados e comparados com os do programa Anarede (Eletrobras Cepel), ferramenta homologada nos Procedimentos de Rede (ONS 2009) para estudos elétricos no Sistema Interligado Nacional Brasileiro (SIN).

Importante salientar que na plataforma OpenModelica - OMEdit - cada modelo (palavra chave **`model`**) tem um elemento gráfico ajustavel associado, que o representa com o número de terminais determinado no código. Na interface gráfica da plataforma é possível arrastar e soltar os modelos aqui apresentados para área de trabalho. A Figura 7 mostra duas barras, um gerador tipo $V\Theta$, um transformador e uma carga arrastadas e conectadas dentro da área de trabalho. Na Figura 7 os pequenos quadrados azuis cheios representam um terminal do tipo *eletrical pin* (palavra chave **`PositivePin`**) e os pequenos quadrados azuis vazios representam outro terminal do tipo *eletrical pin* (palavra chave **`NegativePin`**)

A conexão de modelos na interface gráfica corresponde no Modelica® à palavra chave **`connect`**, que no código a ligação entre modelos. Desta forma, o SEP será um novo modelo (palavra chave **`model`**), que é uma agregação de outros modelos, e as equações do SEP são também uma agregação das equações dos modelos de cada elemento do SEP mais as conexões feitas entre os elementos. O Código 9 representa este sistema exemplo em termos do código Modelica®.

Neste processo de construção do SEP não se trabalha com estruturas de dados, vetores ou matrizes. Ape-

nas os parâmetros de cada modelo devem ser dados e as conexões devem ser realizadas. A solução matemática fica à cargo da ferramenta OMEdit. Para executar a solução na plataforma, o usuário executa a compilação do modelo do SEP, que gera um executável. Este executável, ao ser chamado, realiza a solução e informa os valores das variáveis em cada um dos modelos utilizados do SEP construído. A plataforma OMEdit exibe automaticamente os valores após a solução.

```
model duasbarras
  BusBar Barra1;
  TrafoConstTap Trafo1;
  BusBar Barra2;
  Load_PQ Carga1;
  VoltageSource BarraVTheta1;
  equation
    connect(Trafo1.p, Barra1.p);
    connect(Trafo1.n, Barra2.p);
    connect(Carga1.p, Barra2.p);
    connect(BarraVTheta1.p, Barra1.p);
end duasbarras;
```

Código 9: Modelo de um SEP de 2 Barras.

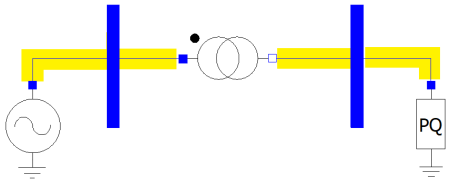

Figura 7: Desenho de um sistema de 2 barras na plataforma OMEdit.

4.1 Sistema IEEE 14 barras

Neste tópico será avaliado o resultado do fluxo de potência para o sistema IEEE 14 barras (Christie 2018). A Tabela 1 traz os resultados das tensões complexas obtidas no OMEdit e no Programa Anarede, para comparação de resultados. A Figura 8 mostra a topologia deste SEP no ambiente OMEdit com os modelos deste trabalho.

Tabela 1: Modelica® × ANAREDE: IEEE 14 barras.

BARRA	TENSÃO (pu)		ÂNGULO (°)		ERRO	
	Modelica	ANAREDE	Modelica	ANAREDE	ΔV	Δθ
1	1,060000	1,060000	00,0000	00,0000	0,00E+00	0,00E+00
2	1,045000	1,045000	-04,9826	-04,9826	0,00E+00	6,00E-07
3	1,010000	1,010000	-12,7251	-12,7251	0,00E+00	2,00E-07
4	1,017670	1,017671	-10,3129	-10,3129	8,65E-07	1,80E-06
5	1,019510	1,019514	-08,7739	-08,7739	3,87E-06	1,58E-05
6	1,070000	1,070000	-14,2209	-14,2209	0,00E+00	4,39E-05
7	1,061520	1,061520	-13,3596	-13,3596	4,51E-07	2,76E-05
8	1,090000	1,090000	-13,3596	-13,3596	0,00E+00	2,76E-05
9	1,055930	1,055932	-14,9385	-14,9385	1,74E-06	1,97E-05
10	1,050980	1,050985	-15,0973	-15,0973	4,65E-06	1,37E-05
11	1,056910	1,056907	-14,7906	-14,7906	3,47E-06	1,96E-05
12	1,055190	1,055189	-15,0756	-15,0756	1,44E-06	1,81E-05
13	1,050380	1,050382	-15,1563	-15,1563	1,72E-06	2,63E-05
14	1,035530	1,035530	-16,0337	-16,0336	3,60E-08	5,82E-05

A Tabela 1 mostra que os resultados obtidos com a plataforma Modelica® e no programa Anarede são iguais. A maior diferença numérica entre Modelica® e Anarede está no ângulo da barra 14, cuja diferença $\Delta\Theta$ é de 5,82E-05, atestando a correção dos modelos elaborados com a linguagem Modelica®.

Os esforços computacionais para o sistema IEEE 14 barras estão na Subseção 4.3, onde na Tabela 3 encontra-se o resumo dos tempos.

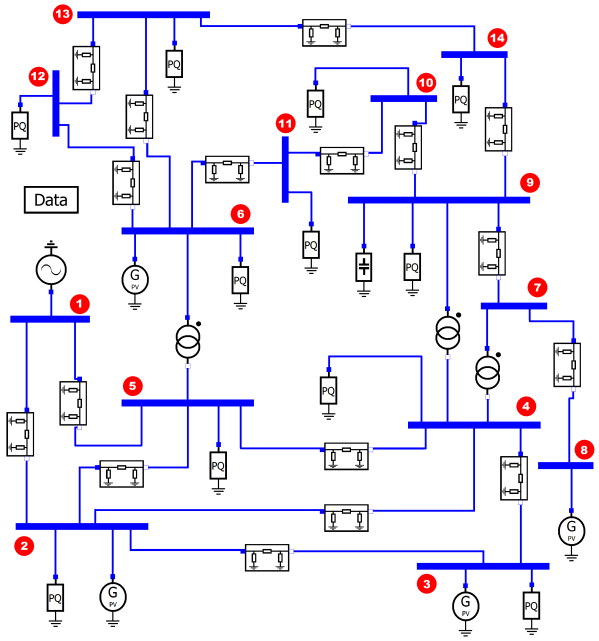

Figura 8: Sistema IEEE 14 barras no OMEdit.

4.2 Sistema New England 39 barras

Agora será avaliado o resultado do fluxo de potência no sistema *New England* 39 barras (PADIYAR 1990). A Tabela 2 exibe os resultados das tensões complexas obtidas com a plataforma Modelica® e o programa Anarede. A Figura 8 mostra a topologia deste SEP no OMEdit.

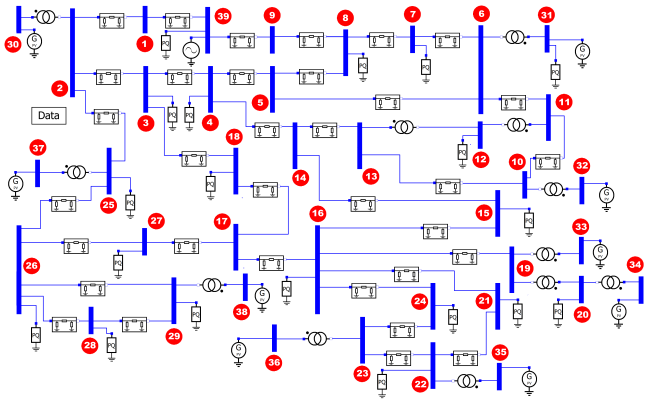

Figura 9: Sistema *New England* 39 barras no OMEdit.

A Tabela 2 atesta que os resultados na plataforma Modelica® e no Anarede são iguais. A maior discrepância entre Modelica® e Anarede está no ângulo da barra 4, cuja diferença $\Delta\Theta$ é de 4,07E-05. Os esforços computacionais serão analisados na Subseção 4.3 a seguir.

4.3 Esforço Computacional

Nesta seção são apresentados e avaliados os tempos computacionais para execução do fluxo de potência nos dois sistemas vistos anteriormente. Para o Modelica®

Tabela 2: Modelica® × ANAREDE: New England.

BARRA	TENSÃO (pu)		ÂNGULO (°)		ERRO	
	Modelica	ANAREDE	Modelica	ANAREDE	ΔV	Δθ
1	1,047650	1,047649	-09,4636	-09,4636	1,40E-06	1,82E-05
2	1,049230	1,049227	-06,9069	-06,9069	2,79E-06	2,14E-05
3	1,030570	1,030575	-09,7523	-09,7522	4,86E-06	1,15E-05
4	1,003860	1,003855	-10,5468	-10,5468	4,82E-06	2,34E-05
5	1,005030	1,005033	-09,3595	-09,3595	2,78E-06	1,50E-05
6	1,007370	1,007373	-08,6573	-08,6573	2,67E-06	1,82E-05
7	0,996707	0,996707	-10,8608	-10,8608	2,72E-07	4,07E-05
8	0,995736	0,995736	-11,3669	-11,3669	2,80E-08	3,29E-05
9	1,028100	1,028096	-11,1910	-11,1910	3,73E-06	2,80E-06
10	1,017010	1,017008	-06,2729	-06,2729	1,73E-06	2,38E-05
11	1,012520	1,012522	-07,0863	-07,0863	2,49E-06	2,84E-05
12	0,999994	0,999994	-07,1020	-07,1020	1,98E-07	2,00E-07
13	1,014190	1,014186	-06,9876	-06,9876	4,45E-06	5,30E-06
14	1,011750	1,011746	-08,6574	-08,6574	3,53E-06	2,26E-05
15	1,015790	1,015794	-09,0759	-09,0759	4,46E-06	2,64E-05
16	1,032260	1,032260	-07,6729	-07,6729	1,48E-07	2,02E-05
17	1,034040	1,034042	-08,6705	-08,6705	2,28E-06	8,90E-06
18	1,031410	1,031409	-09,5106	-09,5106	1,32E-06	8,50E-06
19	1,049850	1,049850	-03,0473	-03,0473	2,84E-07	4,40E-06
20	0,990738	0,990738	-04,4591	-04,4591	2,23E-07	1,30E-06
21	1,03207	1,032072	-05,2673	-05,2673	1,69E-06	4,70E-06
22	1,04992	1,049916	-00,8199	-00,8199	4,18E-06	1,70E-06
23	1,04505	1,045052	-01,0187	-01,0187	2,32E-06	3,00E-07
24	1,03777	1,037765	-07,5532	-07,5532	4,95E-06	1,29E-05
25	1,05778	1,057776	-05,5455	-05,5455	4,15E-06	1,00E-07
26	1,05243	1,052431	-06,8023	-06,8022	8,17E-07	2,20E-05
27	1,03814	1,038139	-08,8128	-08,8128	7,45E-07	3,17E-05
28	1,05058	1,050577	-03,2934	-03,2934	2,77E-06	3,50E-06
29	1,05042	1,050420	-00,5367	-00,5367	1,49E-07	2,00E-07
30	1,04800	1,048000	-04,4894	-04,4894	0,00E+00	2,60E-06
31	0,98200	0,982000	00,1152	00,1152	0,00E+00	7,00E-07
32	0,98300	0,983000	01,7251	01,7251	0,00E+00	2,50E-06
33	0,99700	0,997000	02,1718	02,1718	0,00E+00	1,00E-07
34	1,01200	1,012000	00,7341	00,7341	0,00E+00	1,80E-06
35	1,04900	1,049000	04,1427	04,1427	0,00E+00	1,00E-06
36	1,06400	1,064000	06,8282	06,8283	0,00E+00	3,04E-05
37	1,02800	1,028000	01,2364	01,2364	0,00E+00	3,00E-07
38	1,02700	1,027000	06,5199	06,5199	0,00E+00	2,34E-05
39	1,03000	1,030000	-11,0000	-11,0000	0,00E+00	0,00E+00

foram medidos três tempos: (i) t_1 - tempo de conversão do código Modelica® em código C++, (ii) t_2 - tempo de compilação do código C++ convertido com *GNU Compiler Collection* e (iii) t_3 - tempo de execução do fluxo de potência a partir do executável gerado com o código C++ convertido. Importante destacar que os tempos t_1 e t_2 são gastos uma única vez, durante a construção do sistema elétrico a partir das bibliotecas desenvolvidas. Gerado o executável referente ao sistema elétrico, apenas o tempo t_3 será importante no tocante a execução de outras soluções de fluxo de potência. No caso do programa ANAREDE, apenas o tempo t_3 é aferido, pois não há as etapas (i) e (ii) inerentes ao Modelica®.

Para os dois sistemas apresentados, os tempos de conversão e compilação do fonte C++ convertido são pequenos, conforme mostram t_1 e t_2 da Tabela 3. Em relação a solução do fluxo de potência, o tempo do programa ANAREDE é relativamente menor do que o obtido com o Modelica®. Porém cabe ressaltar que no tempo t_3 do Modelica® está embutido um tempo para geração automática de um arquivo do tipo CSV (*comma-separated values*), com os valores dos módulos das tensões e dos ângulos nodais, o que resulta em um tempo de solução maior do que o realmente ocorrido, em virtude da escrita do arquivo CSV pelo Modelica®.

Tabela 3: Comparação dos tempos computacionais.

SISTEMAS	Modelica®			ANAREDE
	t_1	t_2	t_3	t_3
	(s)			(s)
IEEE 14 Barras	5,6	7,0	0,25	0,012
New England 39 barras	9,7	7,2	0,76	0,015

5 Conclusões

Neste trabalho foram gerados em linguagem Modelica® uma biblioteca de elementos básicos para a análise e simulação de casos de fluxo de potência. Os elementos implementados foram a linha de transmissão, o transformador de dois enrolamentos, os bancos de reatores e capacitores, a carga tipo potência constante, os geradores tipo PV e $V\Theta$ e a Barra.

Com a biblioteca foram construidos dois SEP's tutoriais: IEEE 14 barras e o *New England* 39 barras. Os resultados obtidos com a linguagem Modelica® foram validados com o programa Anarede, que é a ferramenta oficial brasileira de análise em regime permanente.

A forma de modelagem através da linguagem Modelica® se mostrou muito atrativa, em função da capacidade que esta tem de ser usada em problemas matemáticos onde os elementos podem ser representados por um conjunto de equações algébrico-diferenciais, como é o caso dos sistemas elétricos. Os códigos escritos são reduzidos, genéricos e não requerem do desenvolvedor a construção de estruturas de dados ou de ferramentas matemáticas de solução, pois a própria plataforma Modelica® trata da montagem do problema matemático para o sistema modelado.

Nos sistemas tutoriais os tempos medidos de solução do problema são pequenos e não representam desestímulo ao uso do Modelica®. Os tempos de conversão e compilação são gastos uma única vez, sem impacto no tempo de solução, pois os parâmetros dos sistemas são lidos através de arquivos texto e alterações destes parâmetros não irão requerir que o modelo do SEP seja novamente convertido e compilado.

Como trabalhos futuros, outros elementos típicos de SEP serão modelados, tais como SVC (*Static Var Compensator*), CSC (*Controled Series Capacitor*), etc. SEP's de maior porte serão simulados para a análise de fluxo de potência. Outros tipos de simulação também serão alvo de estudos na plataforma Modelica®.

Agradecimentos

Os autores gostariam de agradecer a UFJF, ao CNPq, a CAPES e a FAPEMIG pelo apoio financeiro. Também agradecem à Eletrobras Cepel pelo uso da versão acadêmica do programa ANAREDE.

Referências

Agostini, M. N. (2002), Nova Filosofia para o Projeto de Software para Sistemas de Energia Elétrica usando Modelagem Orientada a Objetos, PhD thesis, Universidade Federal de Santa Catarina (UFSC), Florianópolis.

Christie, R. (2018), 'Power systems test case archive'. **URL:** *http://www.ee.washington.edu/research/pstca/pf14/pg_tca14bus.htm*

Dotta, D. (2003), Modelagem e implementação de aplicações usando uma base computacional orientada

a objetos para sistemas de energia elétrica, Master's thesis, Universidade Federal de Santa Catarina (UFSC), Florianópolis.

Eletrobras Cepel (2017), 'ANAREDE - Análise de Redes Elétricas'.
URL: *http://www.cepel.br/produtos/anarede-analise-de-redes-eletricas.htm*

Fritzson, P. (2004), *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, John Wiley & Sons.

Manzoni, A. (2005), Desenvolvimento de um Sistema Computacional Orientado a Objetos para Sistemas Elétricos de Potência: Aplicação à Simulação Rápida e Análise da Estabilidade de Tensão, PhD thesis, COPPE - Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro.

Monticelli, A. J. (1983), *Fluxo de Carga em Redes de Energia Elétrica*, Editora Edgard Blücher Ltda., São Paulo.

ONS (2009), 'Submódulo 18.2 - Relação dos sistemas e modelos computacionais'.
URL: *http://www.ons.org.br/%2FProcedimentosDeRede%2FM%C3%B3dulo%2018%2FSubm%C3%B3dulo%2018.2%2FSubm%C3%B3dulo%2018.2_Rev_1.0.pdf*

Open Source Modelica Consortium (2018), 'OpenModelica Connection Editor (OMEdit)'.
URL: *https://openmodelica.org/?id=78:omconnectioneditoromedit&catid=10:main-category*

OpenCPS (2015), 'OpenCPS is a multidisciplinary European ITEA3 research project including 18 partners representing universities, research institutes, and industries from Sweden, France, Finland, and Hungary'.
URL: *https://opencps.eu/*

PADIYAR, K. R. (1990), 'Energy function analysis for power system stability', *Electric Machines & Power Systems* **18**(2), 209–210.

Petzold, L. (2018), 'Large Scale Differential Algebraic Equation Solver: DASPK2.0'.
URL: *https://cse.cs.ucsb.edu/software*

The Modelica Association (2014), 'Modelica®- A Unified Object-Oriented Language for Systems Modeling: Language Specification'.
URL: *https://modelica.org/documents/ModelicaSpec33Revision1.pdf*

The Open Source Modelica Consortium (2018), 'The OpenModelica Modeling, Simulation, and Development Environment'.
URL: *https://www.openmodelica.org/*

Tiller, M. M. (2001), *Introduction to Physical Modeling with Modelica*, Kluwer Academic Publishers.

Tinney, W. F. & Hart, C. E. (1967), 'Power Flow Solution by Newton's Method', *IEEE Transactions on Power Apparatus and Systems* **PAS-86**(11), 1449–1460.

Tinney, W. & Walker, J. (1967), 'Direct solutions of sparse network equations by optimally ordered triangular factorization', *Proceedings of the IEEE* **55**(11), 1801–1809.