Titre: Title:	High-Level Programming Methods for the Simulation of Power System Transients
Auteur: Author:	Alireza Masoom
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Masoom, A. (2021). High-Level Programming Methods for the Simulation of Power System Transients [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/9924/

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

High-level programming methods for the simulation of power system transients

ALIREZA MASOOM

Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Génie électrique

Décembre 2021

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée:

High-level programming methods for the simulation of power system transients

présentée par **Alireza MASOOM**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Houshang KARIMI, président
Jean MAHSEREDJIAN, membre et directeur de recherche
Adrien GUIRONNET, membre et codirecteur de recherche
Tarek OULD-BACHIR, membre et codirecteur de recherche
Ilhan KOCAR, membre
George STEFOPOULOS, membre externe

DEDICATION

Dedicated to my beloved parents, wife, and my son

ACKNOWLEDGEMENTS

Many friendly people made this work possible, and to whom I want to express my most heartfelt appreciation.

I want to express my deepest gratitude to my supervisor Prof. Jean Mahseredjian for his guidance, questioning attitude, encouragement, patience, support, and especially friendship. His innovative, broad knowledge and real-life experience in the power systems discipline helped me establish a solid understanding and vision. I learned a lot from him about how to think differently and be creative. I believe this is the most important asset that one can gain from a Ph.D. program.

Special thanks go to Professor Tarek Ould-Bachir for his help, his advice, and his selfless availability. Also, I would like to express my great appreciations to Mr. Adrien Guironnet and his R & D team from “Le réseau de transport de l'électricité (RTE)” for their technical supports.

I want to mention my lab colleagues with whom I shared the lab: Reza Hassani, Sadegh Rahimi Pourdanjani, Maryam Torabi Milani, Amir Sadati, Nazak Soleimanpour, Willy Arnaud Nzale Mimbe, Antoine Stepanov, Hossein Chalangar, Danial Jafarigiv, and Reza Pourramezan.

Finally, the Greatest appreciation is reserved for my wife for her patience, support, and encouragement. Thank you all, and I sincerely apologize to anyone I have left out or omitted inadvertently.

RÉSUMÉ

Modelica est un langage orienté objet conçu pour la modélisation des systèmes cyber-physiques à partir de systèmes d'équations. Le langage permet le développement de bibliothèques de composants facilement composables et réutilisables. Modelica s'appuie sur une représentation standard qui permet une compréhension commune et précise de modèle. Modelica est entièrement compatible avec l'interface de maquette fonctionnelle (FMI). L'FMI est une norme industrielle qui permet de combiner les modules de code de simulation (FMU) provenant de n'importe quel outil de modélisation. La norme a été largement utilisé pour l'échange de modèles et la cosimulation. La tendance à utiliser Modelica se développe de plus en plus dans la modélisation des systèmes électriques et la normalisation des modèles. Par exemple, il a déjà été utilisé pour unifier les modèles des réseaux électriques dans domaine de phaseur dans le cadre du projet iTesla.

Les outils classiques de modélisation en régime transitoires électromagnétiques (EMT), par exemple EMTP®, sont souvent écrits dans des langages impératifs, FORTRAN ou C, ce qui est bien adapté aux calculs numériques avec une architecture fermée, dont le modèle et le solveur sont intégrés. Des tentatives ont également été faites pour développer un outil de modélisation en régime transitoire en utilisant le langage de haut niveau MATLAB. Bien que cette approche élève le niveau d'abstraction, il reste que la modélisation se focalise sur les méthodes numériques.

La principale contribution de ce travail est de construire une librairie EMT basée sur Modelica (soi-disant MSEMT) y compris les modèles avancés de machine synchrone, la ligne de transport (les modèles WB et CP), les charges statiques, les modèles non linéaire (l'arc, le parafoudre), etc.

D'autre part, la librairie s'utilise dans Dynaωo (l'environnement hybride Modelica/C++). Dynaωo est un outil développé par RTE, pour la simulation des phénomènes transitoires. Il permet de bénéficier des avantages de ces deux langages et de contourner les problèmes existants de Modelica en simulation.

Préliminairement, les lignes de transport ont été modélisée dans Modelica ; ensuite, le circuit de IEEE 13-bus, y compris des lignes non transposées et des charges déséquilibrées, a été utilisée pour la validation. Ensuite, les réseaux d'IEEE 118-bus et d'IEEE 39-bus sont utilisés pour vérifier les résultats et comparer les performances de simulation. Dans les tests, on également compare la

performance de Modelica avec SPS. Dans tous les cas, Il est démontré que les résultats des simulations obtenus par Modelica sont identiques à EMTP® (logiciel de référence.). Au niveau de performance, dans la plupart des cas, EMTP® surpasse le Modelica. Toutefois, lorsque les transitoires à très haute fréquence résultant de la foudre ou de la simulation de circuits électroniques de puissance à haute fréquence sont traités, le solveur à pas variable dans Modelica montre une meilleure performance que l'EMTP®. De plus, Modelica montre une meilleure performance que Simscape Electrical Specialized Power System (SPS) dans tous les cas ; l'écart entre les deux simulateurs augmente avec le nombre d'éléments non linéaires.

Par ailleurs, traitement des discontinuités et la modélisation des non-linéarités, qui sont des enjeux majeurs dans les modélisations en régime transitoire, sont des contributions de la thèse. Dans le volet, les modèles d'arc, inductance non linéaire, parafoudre ont été élaboré. Les résultats démontrent une précision parfaite et bonne performance.

ABSTRACT

Modelica is an *object-oriented* and *equation-based* language. The advent of high-level languages and their intrinsic features created a new paradigm for modeling and simulation to focus on the equations instead of solutions. Consequently, there is a motivation for employing a modeling environment where the solvers can be selective, and equations representing the model are expressed declaratively and in high-level formalism. Modelica is fully compatible with FMI, a standard widely used for dynamic model exchange and co-simulation via FMU, a combination of XML files, binaries, and C code.

Classical electromagnetic transient (EMT) simulation tools, i.e., EMTP®, are often written in traditional imperative languages, i.e., FORTRAN or C, which is well suited for numeric computations with a closed architecture, whose model and solver are tightly integrated. Although this approach yields a good performance, there are limitations in power electronic and control system modeling. Besides, many code lines are necessary to satisfy requirements from low-level data management. Efforts have also been made to develop an EMT-type simulator using the higher-level MATLAB language in an open-source-code approach. Although this approach elevates the abstraction level, the models must be programmed using given numerical methods.

Modelica as a standardized language for modeling physical systems has previously been considered to unify the electric power models in the phasor domain study of electrical grids in the iTesla project. The tendency to use Modelica is increasingly growing in power system modeling and standardization of models. Moreover, the language leads to having a consistent model exchange among EMT-type simulation tools.

The main contribution of this proposal is to develop a Modelica-based EMT-detailed library. The library includes advanced linear and nonlinear models such as transmission lines, synchronous generators (including magnetic saturation), static loads, controllers, arc models, surge arresters, etc. The models are constructed according to the EMTP® models and validated with the software one by one. The library is used in Dynaωo, which is a Modelica/C++ hybrid environment. Dynaωo helps to benefit from the advantages of both languages and skirting the existing problems of Modelica in time-domain simulation.

As a preliminary case, the transmission line (constant parameter and wideband models) was modeled in Modelica; then, the IEEE 13-bus platform, which consists of untransposed lines and unbalanced loads, was used for validation. The IEEE 39-bus and IEEE 118-bus networks are used to verify results and compare simulation performance in the following steps. In all cases, it is demonstrated that the results obtained from Modelica are identical to the EMTP®. Regarding the performance, in most cases, EMTP® outperforms the Modelica. However, when very high-frequency transients resulted from lightning or simulation of high-frequency power electronic circuits are addressed, the variable step solver in Modelica shows a better performance than EMTP®. In all cases, Modelica has a better performance than Simscape Electrical Specialized Power System (SPS) in MATLAB; the performance gap between the two simulators increases with the number of nonlinear elements.

Moreover, the thesis's contribution is to explore discontinuity handling and modeling of nonlinearities, which are significant issues in EMT simulations. In this regard, the models of arc, nonlinear inductance, surge arrester have been addressed. The results demonstrate perfect accuracy and high performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

BE Backward Euler

CDA Critical Damping Adjustment

CP Constant Parameter

CSSL Continuous System Simulation Languages

DAE Differential-Algebraic Equation

EMT Electromagnetic Transient

EMTP Electromagnetic Transient Programme

FMI Functional Mock-up Interface

FMU Functional Mock-up Unit

GUI Graphical User Interface

IVP Initial Value Problem

MANA Modified Augmented Nodal Analysis

MSEMT Modelica Simulator of Electromagnetic Transients

ODE Ordinary Differential Equation

OMC OpenModelica Compiler

OOP Object-Oriented Programming

RTE Réseau de transport d'électricité

SM Synchronous Machine

STC Saturation Transformer component

SUNDIALS SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers

TL Transmission Line

TR Transformer

TRV Transient Recovery Voltage

TRAP Trapezoidal

UML Universal Modeling Language

ULM Universal Line Model

WB Wideband

CHAPTER 1 INTRODUCTION

Circuit modeling and simulation are well recognized in electrical engineering. Simulation of an electric circuit includes 1) mathematical expression of the circuit elements (i.e., modeling), 2) formulation of the circuit equations, and 3) methods for the solution of these equations. The thesis concerns, for the first time, the problems, challenges and limitations of *EMT-detailed* modeling and formulation of electrical network equations in a high-level computer language, i.e., Modelica.

The rest of the chapter is organized as follows:

- First, we give the background of modeling and simulation in Section 1.1. We speak about computational software paradigms together with an overview of equation-based object-oriented languages in Section 1.2 and Section 1.3.
- In Section 1.4, Modelica language is outlined briefly.
- We give an overview of EMT simulation goals, approaches, and the main simulation tools in Section 1.5.
- We discuss the problem area and state the research motivations in Section 1.6, the research method used, and the challenges in Section 1.7.
- Finally, the scientific contribution is described in Section 1.8.

1.1 Modeling and Simulation

Modeling is an interesting subject of research in computer science as well as in most disciplines of engineering. Recently, languages that can support modeling in specific domains such as power systems have been considered.

It should be distinguished between modeling and simulations. Simulation is an experiment accomplished on a model, while Modeling means “the process of organizing knowledge about a given system" [1]. Modeling is a solution to create a virtual image of a real-world system, while in simulation, it is possible to experiment with the virtual representation under a wide range of conditions to see how it behaves.

In this thesis, the term *model* means a mathematical representation describing the dynamic properties of a continuous-time or discrete-time power electric system. This thesis primarily concern is the EMT modeling and simulation with the high-level language of Modelica.

Designing Continuous System Simulation Languages (CSSL) [1] is not new and goes back to 1967. The CSSL are all based on *state-space* descriptions where the underlying mathematical description is an ordinary differential equation (ODE) [2]. General-purpose simulation tools, e.g., Simulink [4], using *block* diagrams and/or causal connections. In the block diagram approach, it is possible to graphically model ODEs, and a software tool is then used for performing the numerical simulation.

ODEs can describe many physical systems in the *explicit* form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{1-1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the unknown state vector to be solved for, $\mathbf{u} \in \mathbb{R}^m$ the vector of input signals, and t the independent variable representing time.

In actual electric power systems, the algebraic equations describe the system relying on conservative energy laws, e.g., KVL and KCL [5][6]. The differential-algebraic equations (DAEs) are generalizations of ODEs such that certain algebraic equations constrain the system's dynamical behavior. In DAEs, one or more derivatives of dependent variables are not present in the equations [7]. A model is called “consistent” if the number of DAE system equations is equal to the number of model variables. The consistency of the model is a necessary condition for DAE solvability.

An electrical network can be defined by a set of *implicit* DAEs [5]. A general implicit nonlinear form of DAEs is given by:

$$\mathbf{F}(\dot{\mathbf{x}}(t), \mathbf{x}(t), t) = \mathbf{0} \tag{1-2}$$

where $\mathbf{x}(t) = (\mathrm{x}_1(t), \mathrm{x}_2(t), \dots, \mathrm{x}_n(t))$ is a vector of dependent variables, t denotes the independent variable of time. **F** with the dimension of *n* is assumed to be sufficiently smooth. The main distinction between DAEs and ODEs is that the Jacobian matrix $\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}$ is non-singular [2].

Often the solution of a DAE system depends on the derivatives of the input signal and not just the signal itself, as in the case of ODEs [8]. That is why they can not be solved directly; the best way to solve a high index DAE problem is to convert it to a lower index system first. Applying analytical differentiations as needed to the given system and eliminating the algebraic equations will yield an *explicit* ODE system in the form of (1-1). The number of differentiations required for this transformation is called the index of the DAE. As such, ODEs have an index 0. DAE-index 0 contains neither algebraic loops nor structural singularities. An *index–1 DAE* contains algebraic loops but no structural singularities.

DAEs play a key role in *nonlinear* circuit modeling, mainly because of the chance to automatically set up the circuit equations in semi-explicit index-1 DAE form (1-3).

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \boldsymbol{z}) \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}, \boldsymbol{z}) \end{aligned} \tag{1-3}$$

where $\frac{d\mathbf{g}}{d\mathbf{z}}$ is non-singular.

Some DAEs can easily be converted to ODE forms by simple sorting algorithms, whereas others contain big *algebraic loops* or even *structural singularities*.

As an example of linear DAE, consider the equation (1-4) given by:

$$\begin{aligned} \dot{y}_1 &= y_2 \\ y_1 &= g(x) \end{aligned} \tag{1-4}$$

Differentiating the algebraic equation with respect to *x* once, we obtain:

$$\dot{y}_1 = y_2 = \dot{g}(x) \tag{1-5}$$

another differentiation to obtain an ODE:

$$\dot{y}_2 = \ddot{g}(x) \tag{1-6}$$

so this equation has a differentiation index = 2 because two differentiations of g(x) were needed to obtain an ODE.

$$\dot{y}_1 = y_2$$
$$y_1 = g(x) \tag{1-7}$$

Equation (1-8) shows a DAE index 0, because it can be converted to ODE form (1-9) without differentiation.

$$\dot{x} = -x + y$$
$$x^2 + y^2 = 10 \tag{1-8}$$

$$\dot{x} = -x + \sqrt{10 - x^2} \tag{1-9}$$

It should be noted that initial conditions shall be consistent for variables x and y such that the algebraic equation is satisfied, e.g., $x(0) = 3$, $y(0) = 1$.

1.2 Programming Paradigms

Figure 1.1 shows the classification of programming paradigms. In computer science, programming languages are generally characterized into two approaches.

Imperative languages such as Fortran, C, Pascal, Java, MATLAB, or Python in which the concentration is on the solution of a problem, the statements, and control flows are specified and executed in sequential order (so-called *procedural* languages), i.e., algorithms. In other words, the computer is treated as a device that obeys orders. Everything that is computed must have every detail of that computation spelled out. The *procedural approach* divides the tasks a program is supposed to perform into smaller sub-tasks, which are individually described in the code. This results in programming *modules* that can also be used in other programs. The use of imperative languages limits the applicability and extensibility of models. Moreover, in simulation applications, the numerical solution is often tightly integrated into the models.

Declarative languages are the so-called *logic* programming or very high-level languages. The programmer indicates what goals are to be accomplished but not how specific methods are applied to attain those goals. This paradigm focuses on the *declaration* of *computation logic* rather than assignments and describing the control flow in detail [9]. The distinction is that the computer

system must determine *how* the result is to be computed. The syntax of declarative languages is remarkably different from that of the imperative languages. The main notion of this paradigm is that there is a straightforward way to understand each statement and code, and the solution does not depend on how the statement might be used for solving a problem.

Declarative languages encounter severe problems of *execution efficiency*. If the list of declarative codes is long, the number of code manipulations required for causalization is enormous.

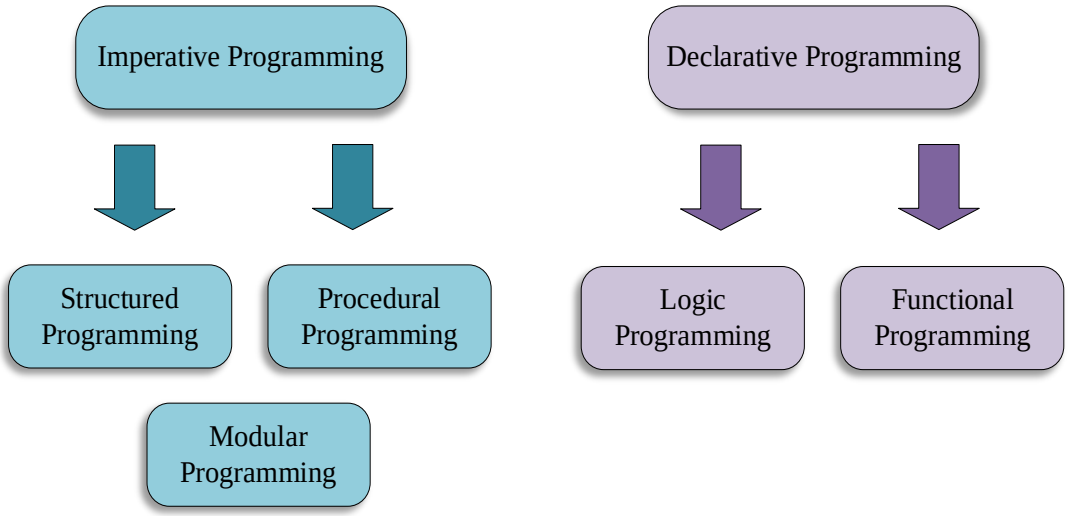


Figure 1.1 Programming paradigms [9]

1.3 Equation-Based Object-Oriented Languages

1.3.1 Equation-based Modeling

Earlier in this section, we introduced programming paradigms. In *equation-based* modeling [10], [11], the modeler only describes the model or system in terms of *DAEs*. Equations describe relations between components of a model. This is the compiler's duty to sort the equations in a solvable order. This procedure is called causalization. This is an excellent property for CSSLs because the modeler can focus his energy on the actual model creation without worrying about the underlying simulation engine.

1.3.2 Object-Oriented Programming

In the 1960s, Simula [12], the first object-oriented language, was designed with the initial purpose of discrete event-based modeling and simulation.

The Object-Oriented Programming (OOP) is viewed as a structuring concept that is used to handle the complexity of large system descriptions. The computer scientists labeled the main features of OOP as *encapsulation, inheritance,* and *polymorphism.*

Encapsulation is the most vital concept in OOP. It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class or package in Modelica. It is primarily used to hide complexity and to build up a hierarchy of sub-models. Encapsulation also enables that model from one paradigm can be transformed into another paradigm.

Inheritance is a feature that allows to import the properties of one class to another class.

Polymorphism simply means that objects with an identical or compatible interface may own different functionalities. In many imperative languages with a nominal *type* system, polymorphism is strongly entangled with inheritance. Modelica owns a structural type system [13], and hence polymorphism is decoupled from inheritance.

From the computer science viewpoint, the essential object-oriented concepts are reflected in Modelica, but due to the declarative character, they are implemented in a different fashion [11].

1.3.3 Mathematical Equations and Acausality

In equation-based languages, the main advantage is that the equations are expressed in an *acausal* form. It means that the causality of how to solve the equations and to sort the equations in terms of knowns and unknowns is not determined during modeling. The procedure is carried out during the simulation. In Modelica, acausality is defined at two abstraction levels:

- equation-level
- level of model connections

As an example of an acausal equation, assume Faraday’s law:

$$\mathrm{v}(t) = \mathrm{L}(t)\ \frac{\mathrm{di}(t)}{\mathrm{dt}} \tag{1-10}$$

where $\mathrm{v}(t)$ is the instantaneous voltage, $\mathrm{L}(t)$ denotes the inductance in the term of time, and $\mathrm{i}(t)$ represents the inductor instantaneous current. In the above equation, "=" defines a relation between the right and left sides of the equation. Depending on which variable is unknown, (1-10) can be translated into four different assignments. The assignment is denoted by ":=".

$$\mathrm{v}(t) := \mathrm{L}(t)\frac{\mathrm{di}(t)}{\mathrm{dt}} \tag{1-11}$$

$$\frac{\mathrm{di}(t)}{\mathrm{dt}} := \frac{\mathrm{v}(t)}{\mathrm{L}(t)} \tag{1-12}$$

$$\mathrm{L}(t) := \frac{\mathrm{v}(t)}{\frac{\mathrm{di}(t)}{\mathrm{dt}}} \tag{1-13}$$

$$\mathrm{v}(t) - \mathrm{L}(t)\frac{\mathrm{di}(t)}{\mathrm{dt}} := 0 \tag{1-14}$$

Therefore, the causality of equation-based models is unspecified during the mathematical representation of the model, and it is determined only when the corresponding system equations are solved based on other variables of system.

The connection between two or more models is also defined as the second level of acausality, for example, a node in an electrical system. This concept refers to *physical modeling* [14][15]. In physical systems, the system is described as a container of functional components that interact by *exchanging energy* through their interface ports. *The physical connection* of components is analogous to connecting real components. Therefore, there exists an exchange of energy between components; there is no *input and output relationship*. In other words, these connection ports are *nondirectional*.

In this method, interfacing variables for connecting the components in a *Physical System* are primarily defined by its *non-flow* and *flow* properties [14][15]. It is not required to specify flow directions and information flow when connecting the components, just as it is not needed to fix this

information when real physical components are interconnected. The number of ports for each model is defined by the number of energy flows it exchanges with other models in the system. For instance, in an electrical system, a resistor has two ports, called pins, e.g. positive and negative pins. Each pin has two variables: voltage, *non-flow*, and current, *flow*. The two variables operate as an interface through the connection of two electrical components. Thus, each component pin is associated with unique types of variables and can be connected only to pins related to the same types of variables.

Therefore, the connections are translated into a set of algebraic equations. These equations, in combination with equations describing the component models, form the DAEs. The DEAs need to be sorted in a solvable order and reformulated in the form of *assignments* [16]. Tarjans algorithm [17] is used to categorize the knowns and unknowns in each equation. This procedure is called matching. During the procedure, it is needed to break the algebraic loops. Algebraic loop or coupled equations is resulted from the conditions that two or more equations are strongly connected, and the system of equations must be solved simultaneously. These equations can be linear or nonlinear algebraic equations. It is required to apply an algebraic solver at each time step to solve the algebraic loops. For nonlinear loops, iterative solution methods, such as Newton's method, must be used [21].

The generated DAEs should be transformed into an *explicit ODE* representation. *Structurally singular* system or *higher–index problem* is another challenge in this procedure [22]. Pantelide's algorithm [23] and dummy-derivatives [24][25] are two main solutions.

Finally, the output of this solution method is the block lower triangular (BLT) matrix [26][27]. There exist packages for solving ODEs such as SUNDIALS [78]. Depending on the selected solver, the equations are discretized and solved.

1.4 Modelica

Several languages have been developed with the common properties of physical modeling using equation systems. Today's state of the art within multidomain physical modeling (e.g., containing mechanical, electrical, hydraulic, thermal, fluid, and control components) is Modelica.

Modelica[1] [14],[28] is a powerful standardized object-oriented declarative equation-based language. The language is widely used to model and simulate continuous and discrete physical systems through *physical connections* [15],[16].

The power system consists of a set of devices that can be described by continuous dynamics as well as discrete events [29]. For example, electrical machines can be modeled using a set of nonlinear ODEs, and transmission lines are molded using the ODEs and delay operators for the calculation of history terms. Control systems, i.e., exciters and governors, are a combination of adders, gains, limiters, and integrator blocks. Discrete events are an essential part of electric systems. The status of a circuit breaker, the tap position of voltage regulating transformers are examples of this type. KCL and KVL are the main conservation laws governing electrical circuits. Object-oriented modeling of an electrical system always leads to implicit DAE descriptions [15]. Modelica is a language specialized in the handling of hybrid DAEs. In this term, hybrid means a combination of continuous and discrete variables [14].

Modelica libraries are built in an object-oriented approach, and every connection can be performed with the corresponding Modelica components, such as interconnection of two capacitors in parallel. This feature often leads to higher index systems.

Modelica has been previously used for power system simulation, especially in the phasor domain. As per the author's knowledge, there is a lack of research on the simulation of EMTs with Modelica.

Several commercial and non-commercial environments support the Modelica language; OpenModelica [30] is the most well-known tool among the non-commercial tools. Dymola [31], Wolfram SystemModeler [32][33], MathModelica [34], MapleSim [35], and JModelica [36] are among the commercial tools designed for Modelica.

More details on the specification of Modelica and formulation of electrical equations in the language will be presented in Chapter 2.

---

[1] Modelica® is a registered trademark of the Modelica Association.

1.5 EMT Modeling and Simulation

Electric power studies are classified into three categories. The first one is the sinusoidal steady-state, in which the components are specified by their equations in phasor domain; thus, a circuit is represented by a system of complex numbers. The Load-flow study usually precedes the steady-state solution. The second category relates to electromechanical transients, which are perturbations of low frequencies related to the interactions between generators and the grid. The last category is electromagnetic transients which deal with fast transients in the range of 0 to 100 MHz or higher. The transients are physically explained by the energy exchange between the electrical components such as inductors and capacitors. The EMT approach contains the vast majority of subjects, such as harmonic studies, overvoltage, switching effects, corona effects, skin effects, distributed-parameter modeling, etc. [37]. In fact, electromechanical transients are also included in EMT-type of studies and can be conducted using EMT-type solution methods. Off-line EMT simulators, the results of which are not in synchronism with a real-time clock, have no computing time constraints and can be made as precise as needed within the available data, models, and related mathematics.

The method by which electric circuit equations are formulated is significant to computer-aided circuit analysis. It directly influences the network computation time, memory allocations, and simulation speed. In modern power system simulators, the systematic formulation of the electric circuit equations can be divided into state variable analysis and modified augmented nodal analysis [48]-[50] (MANA). The summary of each method for "EMT simulation" is explained below:

1.5.1 State Space Analysis

State-space analysis (also known as state variable analysis) is a popular *multi-domain* technique for modeling *physical systems* as vectors of input, output, and state variables related by first-order ODEs [38]. A *linear dynamic system* with p inputs, q outputs and n states can be represented in the implicit matrix form as given by:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad &\Longrightarrow \text{State differential equation} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \quad &\Longrightarrow \text{Output equation} \end{aligned} \tag{1-15}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{u} \in \mathbb{R}^p$ the vector of inputs and $\mathbf{y} \in \mathbb{R}^q$ is the output vector, $\mathbf{A}_{n\times n}$ is the system matrix, $\mathbf{B}_{n\times p}$ denotes the input matrix, $\mathbf{C}_{q\times n}$ is the output matrix and $\mathbf{D}_{q\times p}$ represents the feedthrough matrix. The system is called continuous linear time-invariant (LTI) if, in (1-15), the **A**, **B**, **C**, and **D** matrices are not time-dependent; otherwise, the system is called continuous time-variant.

The main advantages of this method for the LTI systems lie in its lack of overhead when changing numerical integration step size. This is because the discretization of such a system does not affect the contents of state matrices.

One advantage of the state-space method is that it can be easily extended to the analysis of nonlinear systems [39]. The generic form of a state-space model for the representation of a nonlinear system is defined by:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \end{aligned} \tag{1-16}$$

where **f** is for nonlinear state equations and **g** is for output equations.

In electric power systems, the state-space representation gives the option to the modeler to select the integration technique *after* formulating a problem, simplifying the programming of variable time-step integration methods.

The main disadvantage is more longer computing times for formulating the initial set of equations. It is also much inefficient for updating switching devices and nonlinear models. Compared to classic nodal analysis or MANA (discussed in Section1.5.2), there is a dramatic loss of efficiency when solving very large-scale systems. The state-space nodal (SSN) [40] method is a simulation engine based on the combination of state-space and nodal-analysis formulations of circuit equations to alleviate the numerical disadvantages of state-space analysis.

In the following, we introduce two simulation tools working based on the state-space method.

1.5.1.1 Simscape Electrical

Simscape Electrical is a part of the multi-domain Simscape package for modeling and simulating electric systems in the Simulink environment. Simscape is a physical modeling language designed to emulate physical systems [41]. It supports acausal and equation-based modeling. An example of a resistor model in Simscape is shown in Figure 1.2. More explanation of this model can be found in [42].

```
component resistor
% Linear Resistor
% The voltage-current (V-I) relationship for a linear resistor is V=I*R,
% where R is the constant resistance in ohms.
%
% The positive and negative terminals of the resistor are denoted by the
% + and - signs respectively.
nodes
p = foundation.electrical.electrical; % +:left
n = foundation.electrical.electrical; % -:right
end
variables
i = { 0, 'A' }; % Current
v = { 0, 'V' }; % Voltage
end
parameters
R = { 1, 'Ohm' }; % Resistance
end
branches
i : p.i -> n.i;
end
equations
assert(R>0)
v == p.v - n.v;
v == i*R;
end
end
```

Figure 1.2 A Simscape file that implements a linear resistor

1.5.1.2 Specialized Power System Library

The Simscape Electrical Specialized Power System (SPS) is a library in Simscape Electrical developed by Hydro-Quebec to simulate power systems. SPS belongs to the family of physical system modeling and uses a similar block and connection line interface. It employs a state-space block to describe the *linear parts* of a system, e.g., a circuit. Once the simulation begins, the system's topology is analyzed, the linear blocks are separated from nonlinear blocks. Then, the

state-space model (**A**, **B**, **C**, **D** matrices) of the linear part of the circuit is computed. All steady-state calculations and initializations are performed at this stage. Nonlinear models, such as switch devices, motors, and machines, are simulated by current sources driven by the voltages across the nonlinear element terminals. Figure 1.3 represents the interconnections between the different parts of the complete Simulink model. The nonlinear models are connected in feedback between voltage outputs and current inputs of the linear model [43]. It is highlighted that both in Simscape and SPS, nonlinear elements are not represented in the *state-space matrices*.

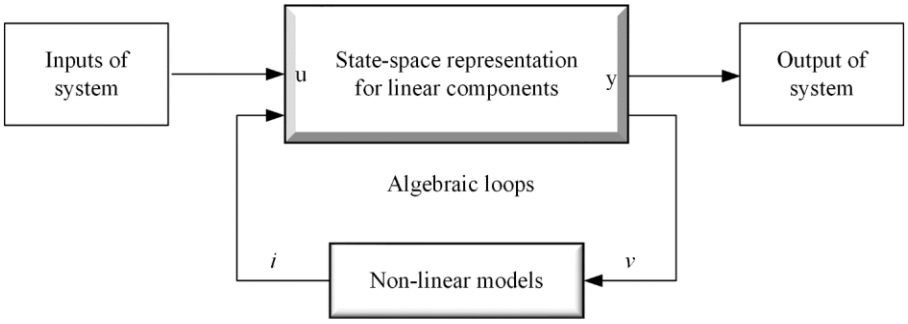


Figure 1.3 Workflow of Simscape electrical [43]

SPS presents some restrictions to circuit system modeling; for example, it is impossible to simulate a capacitor in a loop with an ideal voltage source (as illtreated in Figure 1.4) or connect an inductor in parallel with an ideal current source. It can be mathematically justified that the voltage across the capacitor cannot be chosen as an independent state variable; consequently, it is not possible to write the equations in the linear form of state space. Putting a small resistance (snubber) in series with the capacitor is required to alleviate the problem.

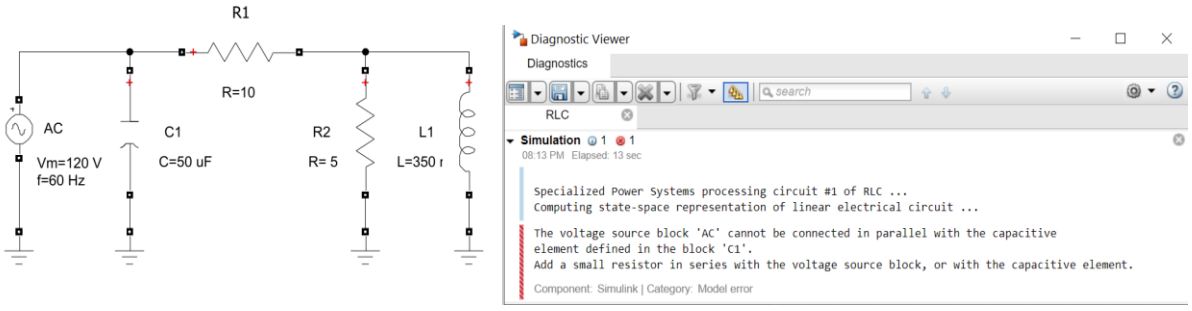


Figure 1.4 The parallel connection of a capacitor and ideal voltage source in the SPS

As other examples of topological constraints in the SPS, it is impossible to connect in series the nonlinear components modeled by a current source to an inductor, or two nonlinear blocks implemented by controlled current source, e.g., a circuit breaker and a nonlinear inductor for inrush current study.

1.5.2 Modified Augmented Nodal Analysis

Nodal analysis (NA), also known as the branch current method, was subsequently introduced as the topological dual of mesh analysis and is identified with Kirchhoff's current law (KCL) in the form of $\mathbf{AI} = \mathbf{0}$, where matrix $\mathbf{A}$ describing the incidence between branches and nodes in the circuit [44]-[46]. The current passing through each branch is written in terms of the circuit node voltages. This method has several limitations, including the inability to process voltage sources and current-dependent circuit elements efficiently.

The backbone of Modified Nodal Analysis (MNA) is the nodal analysis. In this method, the nodal voltage and the current of some branches are determined [47]. Therefore, an electric circuit can be formulated by:

$$\begin{bmatrix} \mathbf{Y} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_I \\ \mathbf{B}_V \end{bmatrix} \tag{1-17}$$

where $\mathbf{Y}$ is the nodal admittance matrix, $\mathbf{S}$ is the augmented matrix for including the connectivity of voltage sources, $\mathbf{V}$ and $\mathbf{I}$ are the unknown nodal voltages and source current vectors, respectively, the vectors $\mathbf{B}_I$ and $\mathbf{B}_V$ are the current and voltage sources.

Modified Augmented Nodal Analysis (MANA) is an extension of MNA to eliminate the limitations of NA and MNA. The solution method in EMTP® is based on MANA. This method offers several advantages [50], [51] over classical nodal analysis and is formulated as below.

$$\begin{bmatrix} \mathbf{Y}_n & \mathbf{V}_c & \mathbf{D}_c & \mathbf{S}_c \\ \mathbf{V}_r & \mathbf{V}_d & \mathbf{D}_{vd} & \mathbf{S}_{vs} \\ \mathbf{D}_r & \mathbf{D}_{vd} & \mathbf{D}_d & \mathbf{S}_{ds} \\ \mathbf{S}_r & \mathbf{S}_{sv} & \mathbf{S}_{sd} & \mathbf{S}_d \end{bmatrix} \begin{bmatrix} \mathbf{v}_n \\ \mathbf{i}_v \\ \mathbf{i}_d \\ \mathbf{i}_s \end{bmatrix} = \begin{bmatrix} \mathbf{i}_n \\ \mathbf{v}_b \\ \mathbf{d}_b \\ \mathbf{s}_b \end{bmatrix} \tag{1-18}$$

where submatrix $\mathbf{Y}_n$ is the linear network admittance of network, and other submatrices represent network elements that can not be modeled as an admittance branch. Equation (1-18) can be represented in the generic forms of $\mathbf{Ax} = \mathbf{b}$. The below examples demonstrate the approach [48], [49].

Assume an ideal voltage source is connected between two nodes, *k*, and *m*. Therefore, the mathematical representation is given by:

$$v_k - v_m = v_{km} \tag{1-19}$$

Equation (1-19) is directly inserted into the main system by placing 1 and −1 in columns *k* and *m*, respectively, of $\mathbf{V}_r$. The source current condition is considered by transposition in the submatrix $\mathbf{V}_c$. Similarly, it is possible to implement the model of an ideal switch using the submatrices $\mathbf{S}$. Let's assume an ideal switch is between nodes *k* and *m*. when the switch is closed, then $v_k - v_m = 0$ and the switch is open $i_{km} = 0$ and the corresponding diagonal cell in $\mathbf{S}_d$ is set to 1.

For nonlinear component modeling [50],[51], EMTP® employs an *iterative* solution for nonlinear branches. For this purpose, first, the nonlinear equation is linearized around a candidate solution point to solve the circuit for this solution point to obtain a better result. The procedure is iterated until the pre-defined tolerance is reached. For example, assume a nonlinear resistor defined by $i = f(v)$ as illustrated in Figure 1.5.a. The linearized equation at point (v_0, i_0) is:

$$i = f'(v_0)v + I_{eq} \tag{1-20}$$

The Norton equivalent of (1-20) is shown in Figure 1.5.b.

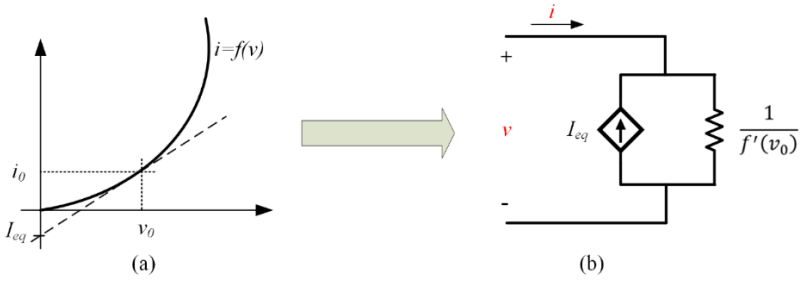


Figure 1.5 (a): An i -v characteristic for a nonlinear resistor. (b): Norton equivalent at the operating point

1.5.2.1 EMTP®

There are presently four EMT-type simulation tools; EMTP® [49], ATP [52], EMTDC (PSCAD) [54] and PowerFactory [55]. The electromagnetic transients program (EMTP®) is a widely used simulation tool. The main idea in these tools is the discretization of components based on a fixed step second-order trapezoidal rule. A Norton equivalent circuit is composed only of a resistance and a current source to represent any circuit element in a power system [45], [53]. EMTP® has great distinguishing features such as handling nonlinear functions, initialization, steady-state studies, frequency scan in phasor domain, implementation of control systems, etc. In EMTP®, the numerical oscillations resulting from the truncation error are suppressed by the Critical Damping Adjustment (CDA) scheme [56].

In the EMTP® solution method, power electronic devices such as thyristors or transistors can be modeled as ideal switches with a resistor to account for losses. Currently, the nonlinear properties of components are modeled with *monotonically increasing* curves [37].

1.5.2.2 XTAP

XTAP (eXpandable Transient Analysis Program) is also an EMT-type simulator. The numerical integration method of XTAP is based on the 2-Stage Diagonally Implicit Runge-Kutta (2S-DIRK) [57] instead of the trapezoidal method [58],[59]. The 2S-DIRK method has a second-order accuracy and is A-stable like the trapezoidal method but does not produce sustained numerical oscillation at discontinuities.

The 2S-DIRK method, besides the increased computation burden, has some limitations such as potential order reduction phenomenon when applied to very stiff and differential-algebraic problems, where the classical order of the method offers a poor indication of occurred numerical error [60].

1.6 Motivation and Objectives

The classical EMT simulators use imperative languages such as Fortran and C. According to the author's knowledge, no EMT simulator has been developed based on declarative languages, and the abilities and advantages of these languages have never been investigated. This research is

conducted to explore the benefits and drawbacks of Modelica as a powerful example of the declarative languages in EMT simulation.

One of the aims of this research is to create and validate a Modelica-based library, the so-called Modelica Simulator of Electromagnetic Transients (MSEMT). The library, which includes the EMT-detailed model of main electrical components, shows the comparative advantages and disadvantages of simulating with Modelica and finally offers solutions for removing the constraints of Modelica-based EMT simulation.

Analysis of simulation accuracy for high-frequency switching devices and examination of event-handling in Modelica are important subjects in Modelica.

Creating sophisticated models such as transmission line (wideband model), synchronous machine (with saturation), machine controls, and ultimately exploring the simultaneous solution of control systems and power networks are the objectives of this reseach thesis.

1.7 Methodology

The methodology used in this thesis is to start by implementing EMT-detailed models in the Modelica language. The advanced linear and nonlinear models such as wideband line model, synchronous generator including saturation, surge arrester, etc., will be examined. Models are programmed on a high-level code and based on model equations. This work is followed by research on new solutions algorithms and modeling approaches. All models are implemented and validated with EMTP®. The second task is to test the performance of the models from the aspects of accuracy and speed in large-scale networks. For this purpose, the performances and results obtained from IEEE 39-bus and IEEE 118-bus test cases will be compared with the EMTP®.

To evaluate the accuracy of different numerical solutions, the relative error between the reference solution trajectory x_{EMTP} and the given numerical solution is calculated using:

$$\varepsilon = \left| \frac{x_{Modelica} - x_{EMTP}}{x_{EMTP}} \right|$$

The errors are calculated and presented for each time point. A base-ten logarithmic scale on the *y*-axis is used to have a better demonstration.

Simulation performance is one of the essential characteristics of a simulator. In our studies, the simulation performance is compared with EMTP® and, if possible, with Simscape Electrical Specialized Power System [43]. It should be noted that the performance Simscape Electrical Specialized Power System is decreased severely with the increase of nonlinear components.

Currently, EMT simulation speed via Modelica-based simulators is not as efficient as EMTP®. On the contrary, the efficiency of modeling, especially for complicated models, is considerably improved. For example, it is demonstrated that we can modify a synchronous machine model modularly to include the magnetic saturation effects.

1.8 Contributions

The following are the contributions of this thesis:

- Development of an EMT-detailed library in Modelica.
- Investigation and analysis of advantages/disadvantages, challenges, and limitations of Modelica for EMT-type simulations.
- Investigation of efficiency and flexibility of modeling with Modelica.
- Investigation of EMT nonlinear models in Modelica from the aspects of accuracy and efficiency.
- Investigation of advantages/disadvantages of variable-step solvers for EMT simulations.
- Investigation of discontinuity handling in Modelica simulators.
- Comparison of simulation speed between Modelica-based simulators and Simscape SPS.
- Developing the Dynaωo for EMT simulations.

The workflow of pure Modelica simulators such as OpenModelica is to compile a model at run-time before launching the simulation itself. This process is an obstacle for large-scale power system simulations. Dynaωo [61], [62] is a hybrid C++/Modelica open-source simulation tool for power systems, initially designed for phasor domain simulation and long-term and short-term stability studies. The simulator engine is changed to allow EMT-type simulations. The Dynaωo strategy for increasing performances for compilation and simulation is to compile non squared Modelica

models individually before simulation. The compiled models are then only instantiated during the simulation. In Dynaωo, like in Modelica, the solver is decoupled from the model. Currently, variable step backward Euler [63] and IDA [64] solvers are available in the simulation suite. The simulation tool, along with some test cases, will be presented in Chapter 6.

1.9 Thesis Outline

The thesis describes the EMT modeling and simulation of electric circuits using the Modelica language. This thesis is composed of seven chapters.

- Chapter 1 gives a literature review on programming languages, electromagnetic transient modeling, and simulation tools. Then, it explains the background motivating this Ph.D. project and summarizes its goals and contributions.
- Chapter 2 is dedicated to studying the Modelica language from mathematical formulations and compilation perspectives, along with examples. In this chapter, an overview of solvers integrated with main Modelica compilers will be given. Then simulation modes, i.e., DAE or ODES modes, are presented as well.
- Implementation of EMT models is the focus of Chapter 3. The coverage of this chapter includes the models for transmission lines, nonlinear inductor, surge arrester, arc models, synchronous machine, control systems, etc.
- Chapter 4 presents the numerical tests and results to compare both the precision and efficiency of the transmission line models in Modelica with the reference software EMTP®. This chapter investigates the accuracy of line models, e.g., constant parameter and wideband line models in the IEEE 13-bus distributed network. The network includes short-length and untransposed overhead transmission lines and underground cables. The
- Chapter 5 investigates the accuracy of developed models in the MSEMT library and the efficiency of simulations through the IEEE39-bus network. Both linear and nonlinear components, along with the control systems, are simulated in this test. Issues of accuracy and stability of variable-step methods are covered.

- Chapter 6 introduces a hybrid C++/Modelica simulation tool, called Dynaωo, to accelerate simulations in Modelica. In this chapter, the accuracy and performance of Dynaωo are compared with OpenModelica and EMTP® through nonlinear and high-frequency transient examples.
- Chapter 7 investigates the Modelica models in the larger scale IEEE 118-bus network. In this test case, the accuracy of the synchronous machine model, including saturation and surge arrester, is validated with EMTP®. The validation of arc models is also examined in Section 7.5.

CHAPTER 2 A REVIEW ON MODELICA AND BASIC CONCEPTS

This section gives an introduction to basic concepts necessary to understand the problems demonstrated in this thesis. This includes some background to Modelica, which provides a quick overview of the circumstances under which Modelica was invented and illustrates the basic motivation and goals behind this language. Additionally, clarifying the adopted models is necessary to appreciate how the models are handled and solved. Finally, quick and basic elementary language constructs are illustrated through electrical examples. For more details, a comprehensive introduction can be found in [14].

2.1 Introduction

Continuous System Simulation Languages (CSSL) are a set of very high-level programming languages aimed to simplify the modeling and simulation of physical systems characterized by ordinary and partial differential equations.

Modelica is a declarative, object-oriented language aimed primarily to model and simulate multi-domain complex systems, such as mechanical, electrical, hydraulic, thermal, and electric power. The first language specification 1.0 [65] was released in September 1997. Since then, the specification has been evolved to version 3.5 [28] with many complex constructs. Modelica Association [66] is responsible for language specification and the Modelica Standard Library (MSL). MSL is a free library containing basic models in various disciplines.

The most important features of Modelica are:

- Modelica language relies on equations. It permits acausal modeling.
- Modelica is a multidomain modeling language; it means models corresponding to *physical objects* from several domains such as electrical, mechanical, etc.
- Modelica is an object-oriented language. It facilitates the *reuse* of components and the evolution of models.
- Modelica supports C-code generation.
- Modelica has a strong construct for creating and connecting components, subcomponents.

- Modelica has interoperability with other languages such as MATLAB, C, Julia [67]-[69], and Python.
- Modelica is *fully* compatible with FMI [70], [71], a standard that allows co-simulation and exchange of dynamic models [72].
- Modelica can be used for model-to-model transformation through a common information model (CIM) [73], [74].

2.2 Object-Oriented Mathematical Modeling in Modelica

Modelica is a declarative language; therefore, the distinction between classes and objects disappears and inheritance is directly copied in the models. It represents a pure mechanism of *type* generation. This means that the type of hierarchy of models is in principle independent from the inheritance. For example, the resistor model extends from a partial model `OnePort`, which includes two variables `v` for voltage and `i` for current. Furthermore, the classes of `p` and `n` of connector `Pin` are public elements of `OnePort`. Since `Resistor` extends from `OnePort`, all elements `v`, `i`, `p`, and `n` are "*copied*" to class `Resistor`.

Modelica offers the features of object-oriented modeling at a higher level of abstraction than the usual object-oriented programming. For instance, it is not required to write code for transporting data between objects using assignment statements. Such code is automatically generated by the Modelica compiler based on the given equations.

2.3 Equation-based Modeling

As already said, Modelica is an equation-based language. Equations are more flexible than assignments since they do not prescribe a specific data flow direction or execution order. This is the key to *physical modeling* and increases the reusability of Modelica classes. In Modelica, connections between components generate equations.

2.4 Symbolic Workflow of Modelica Models

The Modelica Language Specification [28] defines how a Modelica model shall be mapped into a mathematical description as a mixed system of DAE and discrete equations with Real, Integer, and Boolean variables as unknowns.

Let's assume we have built a Modelica model with high-level abstraction in graphical notations packaged in pure mathematical representation; Figure 2.1 outlines a typical compilation and simulation process. It is noted that this procedure might slightly change for different Modelica environments.

- Block 1: In the top-level, the system is modeled with Modelica codes or using the Graphical User Interface (GUI) developed in Modelica. The selected components from the pre-constructed libraries, such as iPSL [112], MSEMT [75], are dragged and dropped on the simulation page. They can be easily connected by linking the pins of components.

- Block 2: The Modelica codes are translated and parsed into a flat Modelica structure. In the Block, firstly, a type checking of models is carried out to make sure that parametrized models conform to the type rules of Modelica. For example, a plus (+) operator cannot have a string and an integer as its left and right operand.

  Secondly, the inherited classes are collapsed hierarchically. For each sub-components of a model, one copy of all equations is generated with distinguished identifiers. For example, if our model is composed of two resistors, `R1` and `R2` whose resistances are 5 and 10 ohms, two equations are generated: `v1=5*i1` and `v2=5*i2`.

  Thirdly, for each connection between two or more nodes, potential variables are set to be equal, and flow variables are summed to zero.

  As a result, the output of this Block is a set of *flat implicit DAEs* in the form of *abstract syntax tree* (AST) consisting of:

  1. Declaration of variables, e.g., `parameter Real v=5`.
  2. Equations in the `equation` section.
  3. Invoking of functions.

4. Algorithms specified in `algorithm` section.
5. `When`, `if` -clauses for triggering discrete-time behavior.

- Block 3: The behavior of a Modelica model is defined in terms of genuine equations, and a Modelica analyzer must assign an equation for each variable as part of the sorting procedure, which also identifies algebraic loops. The idea to process problems with a hundred thousand unknowns is to focus on the structural properties, i.e., which variables are in each equation. Structure Jacobian or *incident matrix* is used for extracting this information. For a system of equations, $\mathbf{J}(\boldsymbol{x}) = \mathbf{0}$, each element *i*, *j*, is zero if x_j does not existe in the expression J_i ; otherwise, it is one.

  The sorting procedure is to sort out unknowns and equations to transform the structure Jacobian to Block Lower Triangular (BLT). The BLT is an approach specialized in permuting the matrix to have non-zero elements of the matrix lower of the main diagonal. A BLT matrix demonstrates the structure of a problem. It decomposes a problem into subproblems, which can be solved in sequence. Each non-scalar block on the diagonal of BLT forms an algebraic loop. All algebraic loops are identified in the sorting procedure in their unique minimal form. The basic algorithm was given by Tarjan [17].

- Block 4: It is imperative to reduce the size of the problem sent to a numerical solver to obtain efficient simulation. The computational cost for solving a system of equations grows rapidly with the number of unknowns because the number of operations is related to the n^3, where *n* is the number of unknowns. A Modelica model has many trivial equations typically in the form of $v_1 = v_2$ or $v_1 = -v_2$ which are the result of connections and object-oriented properties. From the BLT partition, it is relatively straightforward to find unknowns that are constant and can be calculated and substituted at translation. This may considerably reduce the complexity of the problem that has to be solved numerically.

- Block 5: in this stage, the explicit equations are converted to C code for execution.

- Block 6: the selected solver will solve the system of equations during the simulation time.

- Block 7: Output from the simulation process is typically a file (. mat or .csv file) including simulation data for the variables. The data can be later visualized using a GUI.

The process of Block 2 to Block 5 is typically performed at compile-time, and in Block 6, when the model is executed, it is often called the run time process.

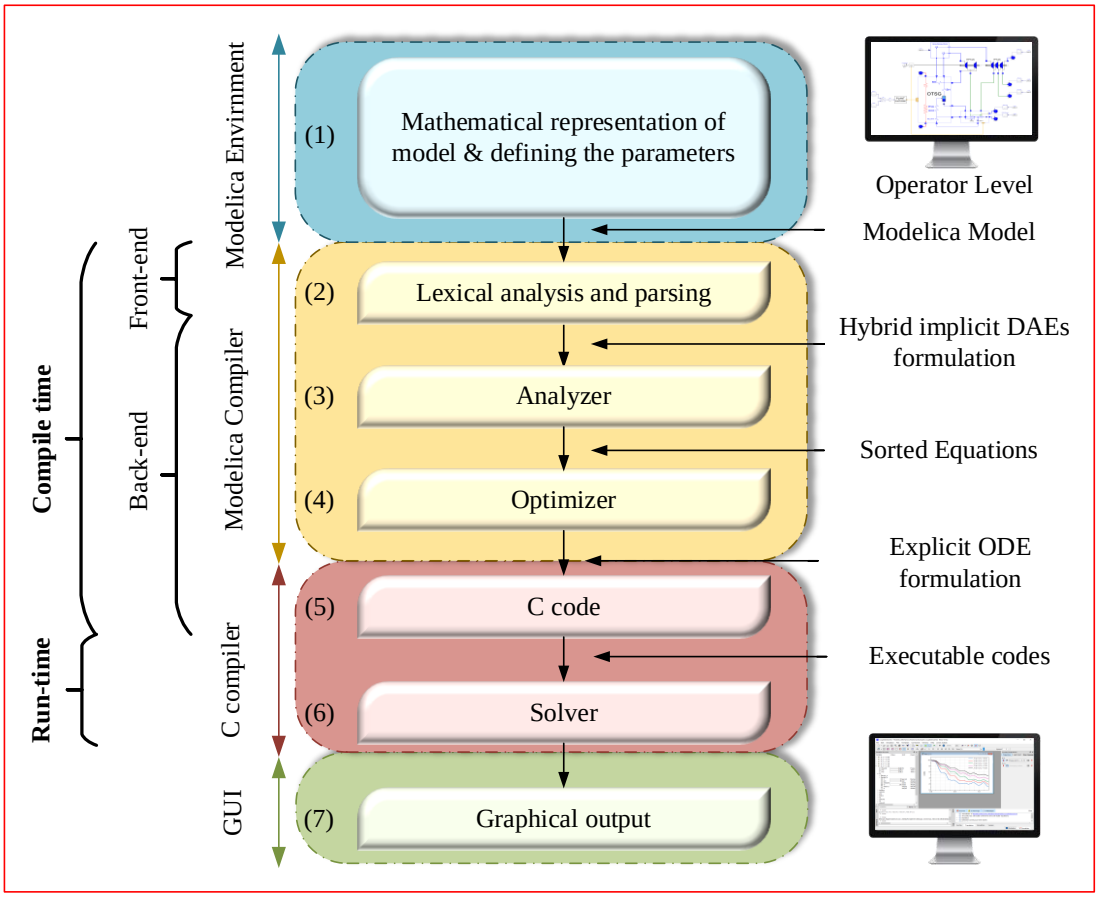


Figure 2.1 Typical workflow of Modelica

2.5 Matching Algorithm

In the previous section, we have discussed the process of causalization of equations in Modelica as one of the time-consuming steps in the compilation. In this section, we would like to have a review of the mathematical aspects of this process.

Graph theory is very popular for symbolically handling the DAEs in Modelica. The first step is to construct the incidence matrix using a directed graph. A directed graph or digraph consists of a set of nodes (called vertices) and directed edges between nodes. An *edge* is defined as an ordered pair of nodes. The bipartite graph can be used to represent the relation between equations and variables in the incidence matrix. Assuming graph **G**, the nodes are divided into two sets, one representing

the rows and the other the columns, such that a row node, i is joined to a column node, j, if and only if a_{ij} is 1 in the incidence matrix [18]. Mathematically, for a set of DAEs, with e_i equations and x_j variables, we can say:

$$\mathbf{G} = \{\mathbf{V}, \mathbf{E}\} \tag{2-1}$$

where, the vertices, **V**, is the set of equations, and is given by:

$$\mathbf{V} = \{e_i\} \quad \forall i \in \mathbb{N} \tag{2-2}$$

and the edge, **E**, is the pairs of relations between equations and variables, is given by:

$$\mathbf{E} = \{(e_i, x_j)\} \quad \forall i, j \in \mathbb{N} \tag{2-3}$$

Now, we should find a matching, i.e., independent edge sets for the constructed bipartite graph **G** using a recursive algorithm. For this purpose. there exist several algorithms in Modelica environments, such as DFS (Depth First Search algorithm.) [17], PR (push-relabel mechanism) [19]. A comparison of appropriate matching algorithms for large-scale DAEs is given in [20].

For example, suppose a system of equations is defined by:

$$\begin{aligned} \mathrm{f}_1(\mathrm{x}_3) &= 0 \\ \mathrm{f}_2(\mathrm{x}_1, \mathrm{x}_2) &= 0 \\ \mathrm{f}_3(\mathrm{x}_2, \mathrm{x}_3) &= 0 \\ \mathrm{f}_4(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_4) &= 0 \end{aligned} \tag{2-4}$$

The vertices **V**, and the edge, **E**, are defined by:

$$\mathbf{V} = \{\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4\} \tag{2-5}$$

$$\mathbf{E} = \{(\mathrm{f}_1, \mathrm{x}_3), (\mathrm{f}_2, \mathrm{x}_1), (\mathrm{f}_2, \mathrm{x}_2), (\mathrm{f}_3, \mathrm{x}_2), (\mathrm{f}_3, \mathrm{x}_3), (\mathrm{f}_4, \mathrm{x}_1), (\mathrm{f}_4, \mathrm{x}_2), (\mathrm{f}_4, \mathrm{x}_3), (\mathrm{f}_4, \mathrm{x}_4)\} \tag{2-6}$$

Figure 2.2 shows the bipartite graph describing the equations (2-4). The general rule for causalization of equations is as below:

- Step 1: The equations which are connected to only one *unknown* variable are identified and causalized. In our example, this is the pair $(\mathrm{f}_1, \mathrm{x}_3)$.

- Step 2: Now, the variable x_3 is known, and it should be identified as a known variable in other equations. This is illustrated by a dashed line.

Now step 1 is repeated by checking the remained acausal equations; we find that f_3 has only "one" unknown variable, e.g., (f_3, x_2). Therefore, the variable x_2 changes to known variable, and we identify it in other equations. By iterating the procedure, x_1 and x_4 are respectively causalized. Each iteration has been marked in Figure 2.2 by a pink number.

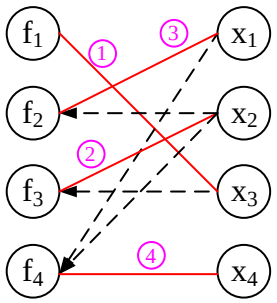


Figure 2.2 Bipartite graph of equation (2-4)

The incident matrix of equations (2-4) is given by:

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array} \tag{2-7}$$

the equations are vertically and horizontally sorted using the algorithm described above, and finally, they are transformed to the BLT matrix, given by:

$$\begin{array}{c} \\ f_1 \\ f_3 \\ f_2 \\ f_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_3 & x_2 & x_1 & x_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array} \tag{2-8}$$

Now the equations are in the form of assignments, and the order of each assignment for solving is known. Matching and causalization for large-scale circuits are not as easy as demonstrated in the example. As earlier mentioned, two challenges are often encountered, algebraic loops and structural singularity. We examine these issues in Section 2.7 and Section 2.8, respectively.

2.6 Analysis of an RLC Circuit

This section intends to demonstrate the modeling and simulation (with OpenModelica workflow) of an RLC circuit. The circuit schematic is illustrated in Figure 2.3. There are five electrical components in this circuit, four passive devices, plus one sinusoidal voltage source. First, the modeling of an inductor is explained, then the procedure is extended to other models, e.g., capacitor, resistor, etc. Finally, we show how the components are connected, and the circuit equations are constructed and causalized. In the simulation procedure, we will demonstrate how these equations are prepared for solving.

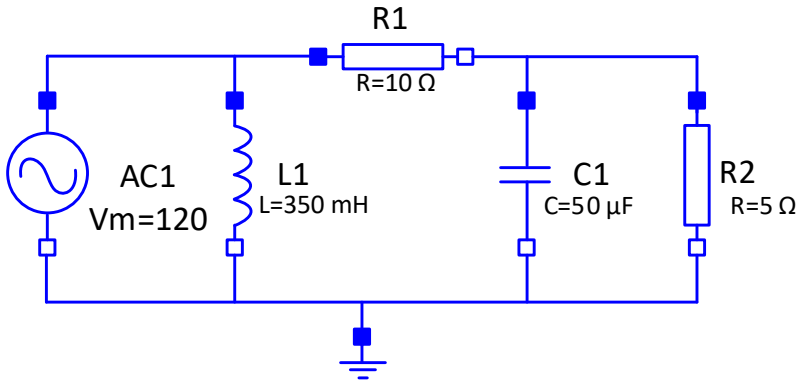


Figure 2.3 Schematic of RLC circuit

2.6.1 Linear Inductor Model

Let us discuss how modeling in Modelica works, using a linear inductor as an example. A linear inductor is a simple electrical component, described by (2-9):

$$v = L\frac{di}{dt} \tag{2-9}$$

where:

v	The voltage *across* the inductor
i	The current *through* the inductor
L	Inductance

From the viewpoint of object-oriented programming, a linear inductor model, denoted by `Inductor`, can be subdivided into several modules with a hierarchal layer. Foremost, it is needed to identify the variables and parameters of the model. In this example, voltage, v, and current, i, are the variables, and the inductance, L, is a parameter of the model. All *variabilities* in Modelica are defined in a specialized class `type`, then package them and recall them when required. Figure 2.4 shows the definition of three types used in the inductor model. For example, we have defined a `type` and named it `Voltage`, then we have declared that `Voltage` is a `Real` value. `quantity` is a description of what the variable represents. In Modelica, variables can have physical units. it is indicated that the `unit` for the `type voltage` is `V`.

```
type Inductance =
Real(quantity="Inductance",
unit ="H");
```

```
type Voltage =
Real(quantity="Voltage",
unit ="V");
```

```
type Current =
Real(quantity="Current",
unit ="A");
```

Figure 2.4 Definition of the types for variables of inductor model

The next step is to define the `pin`. It is the primary component and external communication interface of electrical models. Figure 2.5 shows the implementation of `pin` in Modelica. There is a specific class, i.e., `connector`, for defining the component interfaces. The electrical pin contains two variables, `Voltage` denoted by `v` and `Current` denoted by `i`. We recall that Voltage and Current both are earlier defined in `type`. The prefix `flow` on the second variable indicates that this variable represents a flow quantity, which has special significance for connections. Based on the conservative law of energy, the sum of all flows coming into or out of a specific node is zero. These equations are automatically generated during the parsing of models in Block 1.

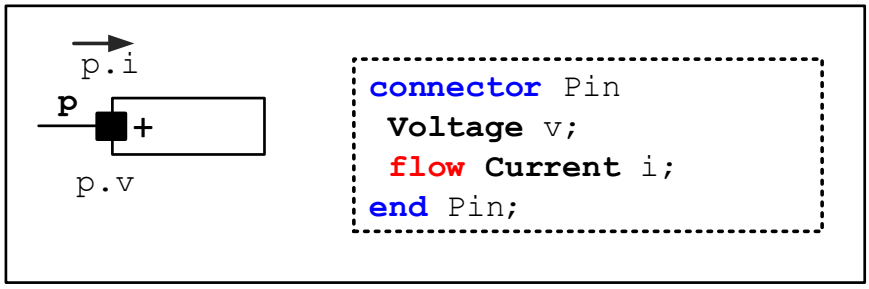


Figure 2.5 Implementation of pin model in Modelica

The next step is to program the partial model `OnePort`, which contains the common properties of a one-port device. Each one-port device is recognized with two pins, i.e., a positive pin denoted by `p` and a negative pin denoted by `n` and a set of equations as illustrated in Figure 2.6. The first equation, `p.v - n.v = v` implies the relationship between the component *non-flow* variables. It defines the voltage drop across the inductor pins as the difference between the pin voltages. The second equation, `p.i + n.i = 0` establishes the relationship between the component flow variables, denoting that the current comes into the device equals the current comes out. It is assumed that the positive direction of current is into the pin throughout the thesis. The third equation is a trivial equation to make the model more understandable.

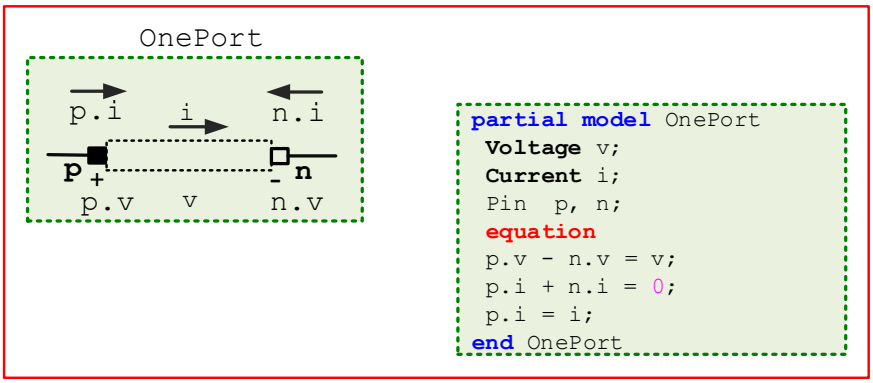


Figure 2.6 Implementation of partial model `OnePort`

Now we can proceed to complete the model, as can be seen in Figure 2.7. `Parameter Inductance L` specifies the inductance value in model `L`. This parameter appears in the block's dialog box generated from the component file and can be modified when building and simulating a model. The comment immediately following the `parameter` declaration, `Inductance`, specifies the type of `L`. The statement `Inductance` determines how the name of the block parameter appears in the dialog box. The properties of partial model `OnePort` are extended into the model using the keyword `extends`.

In the `equation` section, `L * der(i) = v`, describes the operation of a linear inductor based on Faraday's law. It establishes the mathematical relationship between the component non-flow and flow variables, current `i` and voltage `v`, and the parameter `L`.

The operand = used in these equations specifies continuous mathematical equality between the left- and right-hand side expressions. This establishes a symmetric mathematical relationship between the left- and right-hand operands.

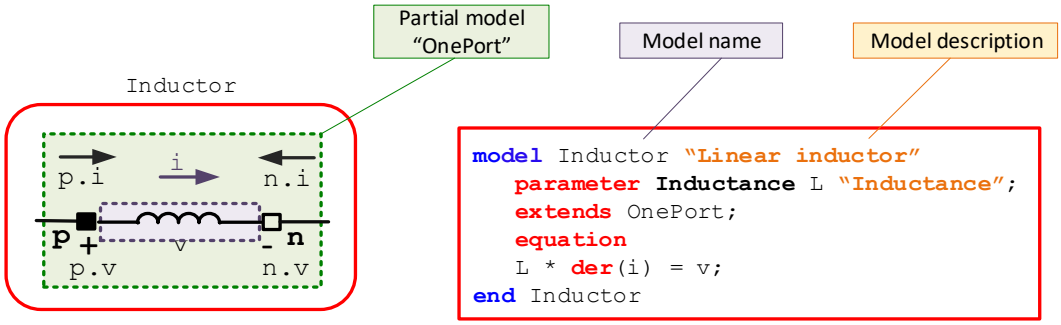


Figure 2.7 Inductor model in Modelica

Figure 2.8 illustrates the resulting graphical user interface (GUI), generated from this component file in OpenModelica. It is possible to group the parameters or create tabs for different types of parameters. For the state variables, we can define initial values in the GUI.

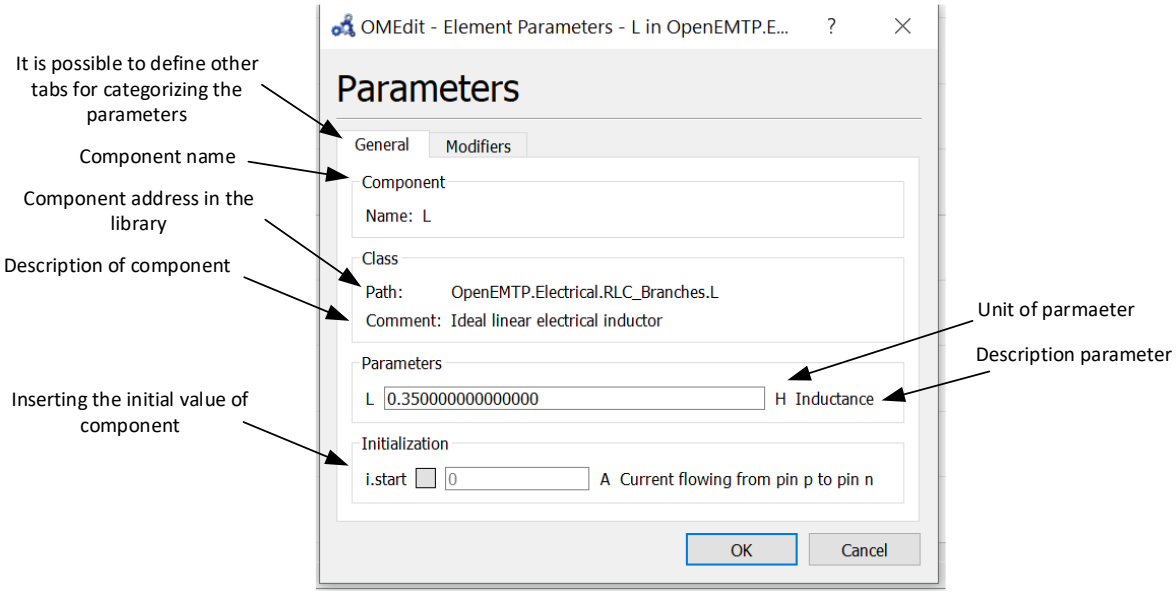


Figure 2.8 Graphical user interface of linear inductor model

It should be emphasized that constructing the models of a resistor or a capacitor is entirely similar to the inductor model by replacing the inductor equation with the appropriate equation, e.g., Ohm's law for the resistor.

2.6.2 Interconnection of Models

Interconnection of models in Modelica language is carried out by the `connect` statement. When the statement is used to connect two models, based on the definition of variables in connecting interface, e.g., `pin`, some equations are constructed based on the following law: The sum-to zero for flow variables and equality of coupling for the non-flow variable.

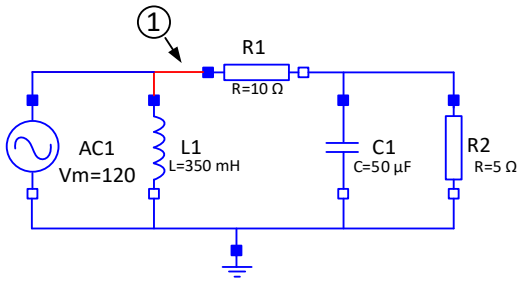
In Figure 2.9, the wire labeled 1 is represented in the model as `connect (L1.p, R1.p)` In the electrical cases, the variables voltage and current are defined as non-flow and flow variables, respectively, in the interface pin. Therefore, Interconnection of the positive pin of resistor (R1), `R1.p`, to the positive pin of the inductor (L1), `L1.p`, by the code

```
connect(L1.p, R1.p);
```

generates two equations automatically.

```
R1.p.v = L1.p.v;
R1.p.i + L1.p.i = 0;
```

These equations are inserted into the set of equations describing the circuit components and shall be solved.




```
model RLC "RLC Example"
RLC_Branches.Ground G;
RLC_Branches.R R1(R = 10);
RLC_Branches.R R2(R = 5);
RLC_Branches.L L1(L = 350e-3);
RLC_Branches.C C1(C = 50e-6);
Sources.CosineVoltage AC(Vm = 120, f = 60);
equation
connect(L1.p, R1.p);
connect(R1.p, AC.p);
connect(R1.n, C1.p);
connect(R1.n, R2.p);
connect(R2.n, G.p);
connect(AC.n, G.p);
connect(L.n, G.p);
connect(C.n, G.p);
end RLC;
```

Figure 2.9 Implementation of RLC circuit with GUIs (left side) and Modelica codes describing the circuit (right side)

2.6.3 Model Compilation

Figure 2.10 shows the parsed models and equations describing the current and voltage at each point for the electrical circuit. For this purpose, the Modelica compiler copies the properties of each model, including its parameters and equations. For the connection point, the required equations which describe the KVL and KCL are generated. For instance, nodes N1 and N2 are defined by three equations, and five equations characterize node N3. The dimension of such automatically generated DAE systems is usually large due to the `connect` statements corresponding to many equations of the form $u = v$ and $u + v = 0$.

The complete set of equations (see Figure 2.10) generated from the RLC circuit consists of 32 DAEs and 32 variables, as well as time and several parameters and constants.

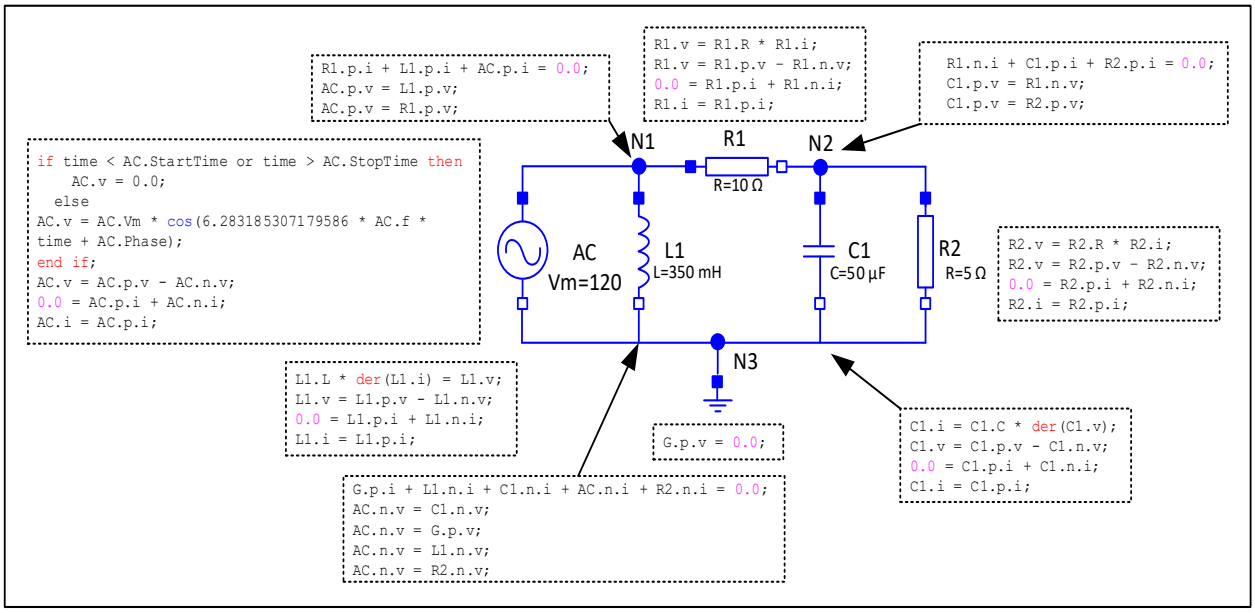


Figure 2.10 Parsed equations of RLC circuit

Table 2.1 presents the 32 variables in the system of equations, of which 30 are algebraic variables since their derivatives do not appear. Two variables, `C1.v` and `L1.i`, are state variables since their derivatives exist in the equations.

Table 2.1 The variables in the RLC circuit model

AC.v	L1.v	R1.v	C1.v	R2.v	G.p.v
AC.i	L1.i	R1.i	C1.i	R2.i	G.p.i
AC.p.v	L1.p.v	R1.p.v	C1.p.v	R2.p.v	
AC.n.v	L1.n.v	R1.n.v	C1.n.v	R2.n.v	
AC.p.i	L1.p.i	R1.p.i	C1.p.i	R2.p.i	
AC.n.i	L1.n.i	R1.n.i	C1.n.i	R2.n.i	

2.6.4 Transformation to State-Space Form

The implicit DAE system described in Figure 2.10 should be further simplified before applying a numerical solver. The next step is to identify the kind of variables in the DAE system. We have the following four groups:

1. All constant and parameters, which are declared with the keyword `constant`, `parameter` are gathered into a vector ***p***. All other constants can be replaced with their values.
2. Variables declared with the input attribute, prefixed by the `input` keyword, are collected into an input vector **u**.
3. Variables whose derivatives appear in the model, the `der()` operator, are grouped in a state vector **x**.
4. All other variables are collected into a vector of algebraic variables, ***y***.

For our simple circuit model these four groups of variables are the following:

p = [`L1.L, R1.R, R2.R, C1.C, AC.f, AC.vm, AC.Phase`]^T

u = [`AC.v`]^T

x = [`L1.i, C1.v`]^T

y = [`L1.v, L1.p.v, L1.n.v, L1.p.i, L1.n.i, R1.v, R1.i, R1.p.v, R1.n.v, R1.p.i, R1.n.i, C1.i, C1.p.v, C1.n.v, C1.p.i, C1.n.i, R2.v, R2.i, R2.p.v, R2.n.v, R2.p.i, R2.n.i, G.p.v, G.p.i`]^T

The system of equations should preferably be in explicit state-space. Derivative of the state vector is computed from the state vector at the current point in time using an iterative numerical solution method for the ordinary differential equations at each iteration step.

2.6.5 Solution Method

Equations illustrated in Figure 2.10 should be causalized as a primary step. There are systematic causalization procedures. For demonstrating how it is done, we come back to our example. First, we must optimize the equations and remove the trivial equations. It helps us to understand the procedure better. Equations (2-10) to (2-18) show the minimal equations describing the system. There are nine equations and unknowns. $f(t)$ is a time-dependent equation representing the sinusoidal voltage source of circuit. *Time* and *state variables* are considered *known* variables, for which no equations need to be found.

In contrast, the inputs of the integrators, `der(L1.i)` and `der(C1.v)`, are unknowns, for which equations must be found. These are the state equations of the state–space description. An iterative numerical solution method is used to find the solution.

$$\mathrm{v}_{L_1} = \boldsymbol{f(t)} \tag{2-10}$$

$$\mathrm{v}_{L_1} = \mathrm{L}_1 d\mathrm{i}_{L1}/dt \tag{2-11}$$

$$\mathrm{v}_{R_1} = \mathrm{R}_1 \mathrm{i}_{R_1} \tag{2-12}$$

$$\boldsymbol{\mathrm{v}_{C_1}} = \mathrm{R}_2 \mathrm{i}_{R_2} \tag{2-13}$$

$$\mathrm{i}_{C_1} = \mathrm{C}_1 \frac{d\mathrm{v}_{C_1}}{dt} \tag{2-14}$$

$$\mathrm{v}_{R_1} = \mathrm{v}_{L_1} - \boldsymbol{\mathrm{v}_{C_1}} \tag{2-15}$$

$$\mathrm{i}_{AC} + \mathrm{i}_{R_1} + \boldsymbol{\mathrm{i}_{L1}} = 0 \tag{2-16}$$

$$-\mathrm{i}_{R_1} + \mathrm{i}_{C_1} + \mathrm{i}_{R_2} = 0 \tag{2-17}$$

$$\mathrm{i}_g - \boldsymbol{\mathrm{i}_{L1}} - \mathrm{i}_{AC} - \mathrm{i}_{C_1} - \mathrm{i}_{R_2} = 0 \tag{2-18}$$

For the first iteration, the known variables, $f(t)$, v_{C_1} and i_{L1}, are colored in green and if no other knowns remained for the corresponding equations, they are causalized. For example, v_{L1} and $\mathrm{i}_{\mathrm{R}_2}$ in (2-10) and (2-13). Then these variables are considered known in other equations and colored in green. The below equations illustrate the procedure. The operand ":=" represents the assignments.

$$\mathrm{v}_{L_1} := f(t) \tag{2-19}$$

$$\mathrm{i}_{R_2} := \mathrm{v}_{\mathrm{C}_1}/\mathrm{R}_2 \tag{2-20}$$

$$\mathbf{v}_{L_1} = L_1 \frac{d\mathrm{i}_{L1}}{dt} \tag{2-21}$$

$$\mathrm{v}_{R_1} = \mathrm{R}_1 \mathrm{i}_{R_1} \tag{2-22}$$

$$\mathrm{i}_{C_1} = \mathrm{C}_1 \frac{d\mathrm{v}_{\mathrm{C}_1}}{dt} \tag{2-23}$$

$$\mathrm{v}_{R_1} = \mathbf{v}_{L_1} - \mathrm{v}_{C_1} \tag{2-24}$$

$$\mathrm{i}_{AC} + \mathrm{i}_{R_1} + \mathrm{i}_{L1} = 0 \tag{2-25}$$

$$-\mathrm{i}_{R_1} + \mathrm{i}_{C_1} + \mathbf{i}_{R_2} = 0 \tag{2-26}$$

$$\mathrm{i}_g - \mathrm{i}_{L1} - \mathrm{i}_{AC} - \mathrm{i}_{\mathrm{C}_1} - \mathrm{i}_{R_2} = 0 \tag{2-27}$$

By looking at the equations (2-21)-(2-27), it is possible to determine the new assignments in the next iteration. In the next iteration, the equations (2-21) and (2-24) are causalized, and the variables $\frac{d\mathrm{i}_{L1}}{dt}$ and v_{R_1}became known; therefore, these two variables are colored and distinguished by bold character to show it is causalized in the new iteration. Equations (2-28)-(2-36) show the product of the procedure.

$$\mathrm{v}_{L_1} := f(t) \tag{2-28}$$

$$\mathrm{i}_{R_2} := \frac{\mathrm{v}_{C_1}}{\mathrm{R}_2} \tag{2-29}$$

$$\frac{d\mathrm{i}_{L1}}{dt} := \frac{v_{L_1}}{L_1} \tag{2-30}$$

$$\mathrm{v}_{R_1} := \mathrm{v}_{L_1} - \mathrm{v}_{C_1} \tag{2-31}$$

$$\mathbf{v}_{R_1} = \mathrm{R}_1 \mathrm{i}_{R_1} \tag{2-32}$$

$$\mathrm{i}_{C_1} = \mathrm{C}_1 \frac{d\mathrm{v}_{C_1}}{dt} \tag{2-33}$$

$$\mathrm{i}_{AC} + \mathrm{i}_{R_1} + \mathrm{i}_{L1} = 0 \tag{2-34}$$

$$-\mathrm{i}_{R_1} + \mathrm{i}_{C_1} + \mathrm{i}_{R_2} = 0 \tag{2-35}$$

$$\mathrm{i}_g - \mathrm{i}_{L1} - \mathrm{i}_{AC} - \mathrm{i}_{C_1} - \mathrm{i}_{R_2} = 0 \tag{2-36}$$

By looking at the above equations, we can find $\mathrm{i}_{\mathrm{R}_1}$ in (2-32), and distinguish it in the remaining acausal equations. Equations (2-37)-(2-45) show the product of this iteration.

$$\mathrm{v}_{L_1} := f(t) \tag{2-37}$$

$$\mathrm{i}_{R_2} := \frac{\mathrm{v}_{C_1}}{\mathrm{R}_2} \tag{2-38}$$

$$\frac{d\mathrm{i}_{L1}}{dt} := \frac{\mathrm{v}_{L_1}}{\mathrm{L}_1} \tag{2-39}$$

$$\mathrm{v}_{R_1} := \mathrm{v}_{L_1} - \mathrm{v}_{C_1} \tag{2-40}$$

$$\mathrm{i}_{R_1} := \frac{\mathrm{v}_{R_1}}{\mathrm{R}_1} \tag{2-41}$$

$$\mathrm{i}_{C_1} = \mathrm{C}_1 \frac{d\mathrm{v}_{C_1}}{dt} \tag{2-42}$$

$$\mathrm{i}_{AC} + \mathbf{i}_{R_1} + \mathrm{i}_{L1} = 0 \tag{2-43}$$

$$-\mathbf{i}_{R_1} + \mathrm{i}_{C_1} + \mathrm{i}_{R_2} = 0 \tag{2-44}$$

$$\mathrm{i}_g - \mathrm{i}_{L1} - \mathrm{i}_{AC} - \mathrm{i}_{C_1} - \mathrm{i}_{R_2} = 0 \tag{2-45}$$

By knowing the variable of i_{R_1}, as can be observed in (2-43) and (2-44), we can compute the i_{AC} and i_{C_1}. These two variables are distinguished as well. The (2-46)-(2-54) show the assignments in this iteration.

$$\mathrm{v}_{L_1} := f(t) \tag{2-46}$$

$$i_{R_2} := \frac{v_{C_1}}{R_2} \tag{2-47}$$

$$\frac{di_{L1}}{dt} := \frac{v_{L_1}}{L_1} \tag{2-48}$$

$$v_{R_1} := v_{L_1} - v_{C_1} \tag{2-49}$$

$$i_{R_1} := \frac{v_{R_1}}{R_1} \tag{2-50}$$

$$i_{AC} := -i_{R_1} - i_{L1} \tag{2-51}$$

$$i_{C_1} := i_{R_1} - i_{R_2} \tag{2-52}$$

$$\mathbf{i}_{C_1} = C_1 \frac{dv_{C_1}}{dt} \tag{2-53}$$

$$i_g - i_{L1} - \mathbf{i}_{AC} - i_{C_1} - i_{R_2} = 0 \tag{2-54}$$

Observing the (2-46)-(2-54)shows that only two equations, (3-53) and (2-54), and two variables, $\frac{dv_{C_1}}{dt}$ and i_g, have remained. There is no priority for them because there is no data dependency between them. Thus, we can causalized them in the last iteration as observed in (2-55)-(2-63).

$$v_{L_1} := f(t) \tag{2-55}$$

$$i_{R_2} := \frac{v_{C_1}}{R_2} \tag{2-56}$$

$$\frac{di_{L1}}{dt} := \frac{v_{L_1}}{L_1} \tag{2-57}$$

$$v_{R_1} := v_{L_1} - v_{C_1} \tag{2-58}$$

$$i_{R_1} := \frac{v_{R_1}}{R_1} \tag{2-59}$$

$$i_{AC} := -i_{R_1} - i_{L1} \tag{2-60}$$

$$i_{C_1} := i_{R_1} - i_{R_2} \tag{2-61}$$

$$\frac{dv_{C_1}}{dt} := \frac{i_{C_1}}{C_1} \tag{2-62}$$

$$i_g := i_{L1} + i_{AC} + i_{C_1} + i_{R_2} \tag{2-63}$$

we arrive at the above set of assignment statements to be computed at each iteration, given values of `C1.v`, `L1.i`, and `t` at the same iteration:

These assignment statements can be converted to code in some programming language, for example, C, and executed with an appropriate ODE solver, usually using implicit schemes. The algebraic transformations and sorting procedure that we somewhat painfully performed on the simple circuit example can be performed automatically and is known as BLT partitioning, converting the equation system coefficient matrix into block lower triangular form (see Figure 2.11).

code #	Assignment	L1.v	R2.i	der(L1.i)	R1.v	R1.i	AC.i	C1.i	der(C1.v)	G.p.i
1	`L1.v := f(t);`	■								
2	`R2.i := C1.v / R2.R;`		■							
3	`der(L1.i) := L1.v / L1.L;`	■		■						
4	`R1.v := L1.v - C1.v;`	■			■					
5	`R1.i := R1.v / R1.R;`				■	■				
6	`AC.i := (-L1.i) - R1.i;`					■	■			
7	`C1.i := R1.i - R2.i;`		■			■		■		
8	`der(C1.v) := C1.i / C1.C;`							■	■	
9	`G.p.i := L1.i - ((-AC.i) - R2.i - C1.i);`		■				■	■		■

Figure 2.11 Block lower triangular for of RLC circuit

The remaining 22 algebraic variables in the equation system of the circuit model that are not part of the minimal 9-variable kernel ODE system solved above can be computed at leisure for those iterations where their values are desired. This is not necessary for solving the kernel ODE system.

There are many algorithms for sorting the equations in data-dependency order and converting the equations to assignment statements. This is possible since all variable values can now be computed in order. One of the most popular algorithms is Tarjan's method, which captures the same information in a graphical data structure called the *structure digraph.* The structure digraph depicts on the left-hand side the equations as a column of nodes. On the right-hand side, the unknowns are

also displayed as a column of nodes. Since the number of equations must always equal the number of unknowns, the two-column vectors are of equal length. A straight line connects an equation with an unknown if that unknown appears in the equation.

The symbolic transformations and reductions of equation systems performed by a real Modelica compiler are much more complicated than shown in this example, including index reduction of equations and tearing of subsystems of equations.

2.7 Example of Algebraic Loop

This section aims to show how an algebraic loop is formed and how it can be eliminated. Let's slightly change the RLC circuit of the previous section and replace the capacitor, C1, with a resistor, R3. The configuration of the new circuit is designed in Figure 2.12. During the procedure of causalization, a strong dependency is observed for voltage and current of R2 and R3, in which it is not possible to specify the priorities for solving them.

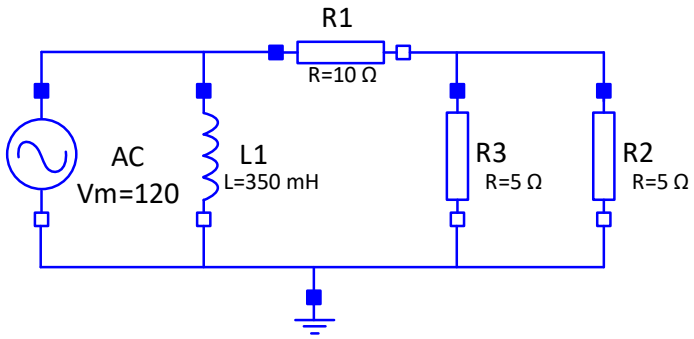


Figure 2.12 RLC circuit with algebraic loop

Equations (2-118)-(2-126) show the main equations of the system (the trivial equations have been removed). As described earlier, the time and state variables ($f(t)$ and i_{L1}) are considered known variables and colored as shown below.

$$v_{L_1} = \boldsymbol{f(t)} \tag{2-64}$$

$$v_{L_1} = L_1 \frac{di_{L_1}}{dt} \tag{2-65}$$

$$v_{R_1} = R_1 i_{R_1} \tag{2-66}$$

$$v_{R_2} = R_2 i_{R_2} \tag{2-67}$$

$$v_{R_2} = R_3 i_{R_3} \tag{2-68}$$

$$v_{R_1} = v_{L_1} - v_{R_3} \tag{2-69}$$

$$i_{AC} + i_{R_1} + \mathbf{i}_{L1} = 0 \tag{2-70}$$

$$-i_{R_1} + i_{R_3} + i_{R_2} = 0 \tag{2-71}$$

$$i_g - \mathbf{i}_{L1} - i_{AC} - i_{R_3} - i_{R_2} = 0 \tag{2-72}$$

The first iteration of causalization is given as below. Now the variable v_{L_1} is known; therefore, we color the other variables which are dependent to v_{L_1}.

$$v_{L_1} := f(t) \tag{2-73}$$

$$\mathbf{v}_{L_1} = L_1 \frac{d i_{L_1}}{dt} \tag{2-74}$$

$$v_{R_1} = R_1 i_{R_1} \tag{2-75}$$

$$v_{R_2} = R_2 i_{R_2} \tag{2-76}$$

$$v_{R_2} = R_3 i_{R_3} \tag{2-77}$$

$$v_{R_1} = \mathbf{v}_{L_1} - v_{R_3} \tag{2-78}$$

$$i_{AC} + i_{R_1} + i_{L1} = 0 \tag{2-79}$$

$$-i_{R_1} + i_{R_3} + i_{R_2} = 0 \tag{2-80}$$

$$i_g - i_{L1} - i_{AC} - i_{R_3} - i_{R_2} = 0 \tag{2-81}$$

The second iteration of causalization is given as below. By looking at the equations (2-84)-(2-90), we see that the causalization algorithm stops because a strong correlation is seen between the variables of these equations.

$$v_{L_1} := f(t) \tag{2-82}$$

$$\frac{d\mathrm{i}_{L_1}}{dt} := \frac{\mathrm{v}_{L_1}}{\mathrm{L}_1} \tag{2-83}$$

$$\mathrm{v}_{R_1} = \mathrm{R}_1 \mathrm{i}_{R_1} \tag{2-84}$$

$$\mathrm{v}_{R_2} = \mathrm{R}_2 \mathrm{i}_{R_2} \tag{2-85}$$

$$\mathrm{v}_{R_2} = \mathrm{R}_3 \mathrm{i}_{R_3} \tag{2-86}$$

$$\mathrm{v}_{R_1} = \mathbf{v}_{L_1} - \mathrm{v}_{R_2} \tag{2-87}$$

$$\mathrm{i}_{AC} + \mathrm{i}_{R_1} + \mathrm{i}_{L1} = 0 \tag{2-88}$$

$$-\mathrm{i}_{R_1} + \mathrm{i}_{R_3} + \mathrm{i}_{R_2} = 0 \tag{2-89}$$

$$\mathrm{i}_g - \mathrm{i}_{L1} - \mathrm{i}_{AC} - \mathrm{i}_{R_3} - \mathrm{i}_{R_2} = 0 \tag{2-90}$$

A glance at the acausal equations gives a clue to us that the equations (2-84)-(2-87) and (2-89) form a system of equations that should be solved simultaneously. These equations can be written as:

$$\begin{pmatrix} 1 & -\mathrm{R}_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -\mathrm{R}_2 & 0 \\ 0 & 0 & 1 & 0 & -\mathrm{R}_3 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{v}_{R_1} \\ \mathrm{i}_{R_1} \\ \mathrm{v}_{R_2} \\ \mathrm{i}_{R_2} \\ \mathrm{i}_{R_3} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathrm{v}_{L_1} \\ 0 \end{pmatrix} \tag{2-91}$$

This is a system composed of five equations and five variables. The tearing algorithm allows us to reduce the size of the system. The generic idea is to assume some variables are known, e.g., i_{R_3}. The variable is called the *tearing variable*, and its selection is a complete NP problem. Using the assumption, remained equations, inner equations, are causalized. Applying the algorithm to (2-91) yields:

$$\begin{aligned} \mathrm{v}_{R_2} &:= \mathrm{R}_3 \mathrm{i}_{R_3} \\ \mathrm{i}_{R_2} &:= \frac{\mathrm{v}_{R_2}}{\mathrm{R}_2} \\ \mathrm{v}_{R_1} &:= \mathbf{v}_{L_1} - \mathrm{v}_{R_2} \end{aligned} \tag{2-92}$$

$$\mathrm{i}_{R_1} := \frac{\mathrm{v}_{R_1}}{\mathrm{R}_1}$$

$$\mathrm{i}_{R_{3,NEW}} = \mathrm{i}_{R_1} - \mathrm{i}_{R_2}$$

where i_{R_3} is an initial guess, and $\mathrm{i}_{R_{3,NEW}}$ is an improved version of that same variable. The following *residual* function can be formulated:

$$\mathrm{F} = \mathrm{i}_{R_{3,NEW}} - \mathrm{i}_{R_3} = 0 \tag{2-93}$$

Equation (2-93) is a scalar linear function in our example. The function is a system of nonlinear equations for the large advanced circuits, where many nonlinear components are used; therefore, a convenient numerical method, algebraic differentiation [76], is used. For example, if we differentiate equation (2-92) with respect to i_{R_3}, we have:

$$\begin{gathered} d\mathrm{v}_{R_2} := \mathrm{R}_3 \\ d\mathrm{i}_{R_2} := \frac{d\mathrm{v}_{R_2}}{\mathrm{R}_2} \\ d\mathrm{v}_{R_1} := d\mathbf{v}_{L_1} - d\mathrm{v}_{R_2} \\ d\mathrm{i}_{R_1} := \frac{d\mathrm{v}_{R_1}}{\mathrm{R}_1} \\ d\mathrm{i}_{R_{3,NEW}} = d\mathrm{i}_{R_1} - d\mathrm{i}_{R_2} \end{gathered} \tag{2-94}$$

and the Jacobian function, J, is defined as:

$$\mathrm{J} = \frac{d\mathrm{F}}{d\mathrm{i}_{R_3}} = d\mathrm{i}_{R_{3,NEW}} - 1 = 0 \tag{2-95}$$

we can calculate $\mathrm{i}_{R_{3,NEW}}$by:

$$\mathrm{i}_{R_{3,NEW}} = \mathrm{i}_{R_3} - \mathrm{J}^{-1}\mathrm{F} \tag{2-96}$$

After finding the solution of the algebraic loop equation, (2-91), equations (2-88), and (2-90) are solved. Figure 2.13 shows the BLT matrix of the circuit. In this matrix, rows 3 to 7 show an

algebraic loop that needs to be solved simultaneously. Since our models are linear, a linear solver shall solve these equations separately, e.g., Gaussian elimination.

code #	Assignment	L1.v	der(L1.i)	R2.i	R3.i	R1.i	R1.v	R2.v	AC.i	G.p.i
1	`L1.v := f(t);`	■								
2	`der(L1.i) := L1.v / L1.L;`	■	■							
3	**`R2.i := R2.v / R2.R;`**			■				■		
4	**`R3.i := R2.v / R3.R;`**				■			■		
5	**`R1.i := R3.i + R2.i;`**			■	■	■				
6	**`R1.v := R1.i * R1.R;`**					■	■			
7	**`L1.v + (-R2.v) + R1.v := 0;`**	■					■	■		
8	`AC.i := (-L1.i) - R1.i;`					■			■	
9	`G.p.i := L1.i - ((-AC.i) - R2.i - R3.i);`			■	■				■	■

Figure 2.13 The assignments and BLT form of RLC circuit with algebraic loop

Now, let's increase the complexity of our example by adding an inductor in parallel with R3. Given the electrical circuit shown in Figure 2.14. Like the previous example, first, we should write the equations of this circuit. At the first iteration, we color the known variables (time, $f(t)$ and state variables, i_{L1}, i_{L2}).

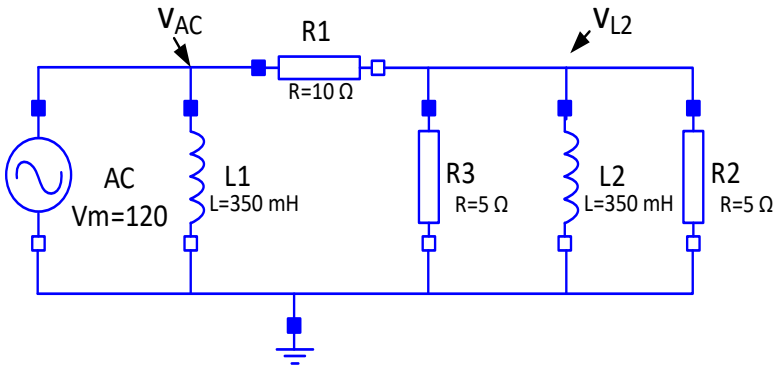


Figure 2.14 RLC circuit with algebraic loop

$$v_{L_1} = \boldsymbol{f(t)} \tag{2-97}$$

$$v_{L_1} = L_1 \frac{di_{L_1}}{dt} \tag{2-98}$$

$$v_{R_1} = R_1 i_{R_1} \tag{2-99}$$

$$v_{L_2} = R_2 i_{R_2} \tag{2-100}$$

$$v_{L_2} = R_3 i_{R_3} \tag{2-101}$$

$$v_{L_2} = L_2 \frac{di_{L_2}}{dt} \tag{2-102}$$

$$v_{R_1} = \mathbf{v}_{L_1} - v_{L_2} \tag{2-103}$$

$$i_{AC} + i_{R_1} + \mathbf{i}_{L1} = 0 \tag{2-104}$$

$$-i_{R_1} + i_{R_3} + i_{R_2} + \mathbf{i}_{L2} = 0 \tag{2-105}$$

$$i_g - \mathbf{i}_{L1} - i_{AC} - \mathbf{i}_{L_2} - i_{R_2} - i_{R_3} = 0 \tag{2-106}$$

In the iteration, the equations (2-97) is causalized for variable v_{L_1}, therefore the variable is colored in as shown below:

$$v_{L_1} := f(t) \tag{2-107}$$

$$\mathbf{v}_{L_1} = L_1 \frac{di_{L_1}}{dt} \tag{2-108}$$

$$v_{R_1} = R_1 i_{R_1} \tag{2-109}$$

$$v_{L_2} = R_2 i_{R_2} \tag{2-110}$$

$$v_{L_2} = R_3 i_{R_3} \tag{2-111}$$

$$v_{L_2} = L_2 \frac{di_{L_2}}{dt} \tag{2-112}$$

$$v_{R_1} = \mathbf{v}_{L_1} - v_{L_2} \tag{2-113}$$

$$i_{AC} + i_{R_1} + i_{L1} = 0 \tag{2-114}$$

$$-\mathrm{i}_{R_1} + \mathrm{i}_{R_3} + \mathrm{i}_{R_2} + \mathrm{i}_{L2} = 0 \tag{2-115}$$

$$\mathrm{i}_g - \mathrm{i}_{L1} - \mathrm{i}_{AC} - \mathrm{i}_{L_2} - \mathrm{i}_{R_2} - \mathrm{i}_{R_3} = 0 \tag{2-116}$$

In this iteration, equation (2-108) can be causalized. By examining the other equations, a strong connection is found between equations (2-109)-(2-111),(2-113), and (2-115). These five equations construct a set of equations with 5 variables. The equation system is formulated as (2-117).

$$\begin{pmatrix} 1 & -\mathrm{R}_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -\mathrm{R}_2 & 0 \\ 0 & 0 & 1 & 0 & -\mathrm{R}_3 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{v}_{R_1} \\ \mathrm{i}_{R_1} \\ \mathrm{v}_{L_2} \\ \mathrm{i}_{R_2} \\ \mathrm{i}_{R_3} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathrm{v}_{L_1} \\ -\mathrm{i}_{L2} \end{pmatrix} \tag{2-117}$$

The tearing algorithm is a general rule and can be applied to linear and nonlinear algebraic equations. The relaxation algorithm or Gaussian elimination can be employed for solving the linear system such as (2-117). The equations (2-114) and (2-116) are causalized, followed by solving (2-117).

2.8 Example of Structural Singularities

In this section, the problem of structural singularities, or so-mathematically called high-index DAE, is introduced. When solving an ODE raised by an electrical system, the problem is to calculate the states when the derivatives are given. Therefore, solving a DAE may also include differentiation. Such a DAE is called high index.

As a standard procedure in Modelica IDEs, the higher index problems are transformed by differentiating equations analytically. The standard algorithm by Pantelides [23] is used to determine how many times each equation must be differentiated. The dummy derivatives [24] method is used in Dymola as well.

It is possible to avoid higher index DAEs by restricting the connection of components and/or using manually differentiated equations for the most common connection structures.

As an example of high index DAE caused by replacing the inductor, L1, with the capacitor, C1, in the RLC circuit example as indicated in Figure 2.15. This circuit leads to a DAE with index 1.

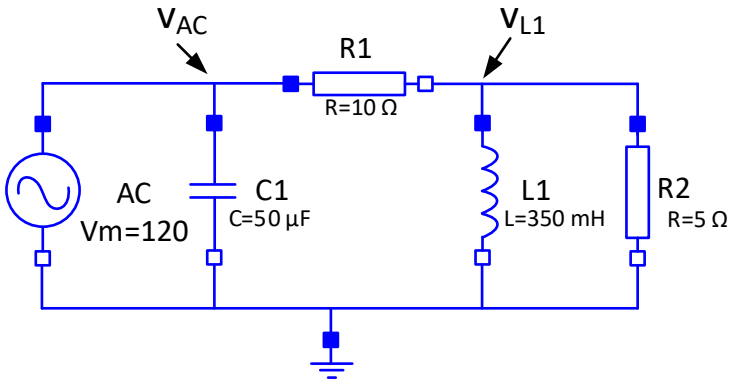


Figure 2.15 RLC circuit with structurally singularity (DAE index 1)

we obtain the following equation by applying KCL and KVL:

$$\mathrm{i}_{AC} = \mathrm{i}_{c1} + \frac{\mathrm{v}_{AC} - \mathrm{v}_{L1}}{\mathrm{R}_1} \tag{2-118}$$

$$\frac{\mathrm{v}_{AC} - \mathrm{v}_{L1}}{\mathrm{R}_1} = \mathrm{i}_{L1} + \frac{\mathrm{v}_{L1}}{\mathrm{R}_2} \tag{2-119}$$

$$\mathrm{v}_{L1} = L_1 \frac{d\mathrm{i}_{L1}}{dt} \tag{2-120}$$

$$\mathrm{i}_{c1} = C_1 \frac{d\mathrm{v}_{C1}}{dt} \tag{2-121}$$

$$\mathrm{v}_{AC} = \mathrm{v}_{C1} \tag{2-122}$$

By inserting (2-120) in (2-119) and (2-121) in (2-118), we obtain the equation in terms of i_{L1} and i_{AC}. The new set of equations are given by:

$$L_1 \left(\frac{1}{\mathrm{R}_1} + \frac{1}{\mathrm{R}_2}\right) \frac{d\mathrm{i}_{L1}}{dt} = \frac{\mathrm{v}_{AC}}{\mathrm{R}_1} - \mathrm{i}_{L1} \tag{2-123}$$

$$\mathrm{i}_{AC} = C_1 \frac{d\mathrm{v}_{C1}}{dt} + \frac{\mathrm{v}_{AC}}{\mathrm{R}_1} - \frac{L_1}{\mathrm{R}_1} \frac{d\mathrm{i}_{L1}}{dt} \tag{2-124}$$

$$\mathrm{v}_{AC} = \mathrm{v}_{C1} \tag{2-125}$$

Equations (2-123)-(2-125) form a DAE system of index 1. For solving (2-124), it is required to symbolically differentiate the constraint equation (2-125) and replace the original constrain with the differentiated one in (2-124). It is recalled v_{C1} is not an independent state, because its value is

dependent to source voltage, which is known. These manipulations yield a new set of ODE system as:

$$L_1\left(\frac{1}{\mathrm{R}_1}+\frac{1}{\mathrm{R}_2}\right)\frac{d\mathrm{i}_{L1}}{dt}=\frac{\mathrm{v}_{AC}}{\mathrm{R}_1}-\mathrm{i}_{L1} \tag{2-126}$$

$$\mathrm{i}_{AC}=C_1\frac{d\mathrm{v}_{AC}}{dt}+\frac{\mathrm{v}_{AC}}{\mathrm{R}_1}-\frac{L_1}{\mathrm{R}_1}\frac{d\mathrm{i}_{L1}}{dt} \tag{2-127}$$

The above algorithm, called Pantelides algorithm, is defined as instead of replacing the constraint equation by its derivative, we add the differentiated constraint equation as an additional equation to the set.

Figure 2.16 shows the BLT matrix defined by OpenModelica. Code lines 4-7 construct a linear algebraic loop.

code #	Assignment	C1.v	der(C1.v)	C1.i	R2.i	R2.v	R1.i	R1.v	der(L1.i)	AC.i	G.p.i
1	C1.v := v(t);	■									
2	der(C1.v) := v'(t);		■								
3	C1.i := C1.C * der(C1.v)		■	■							
4	**R2.v := R2.i * R2.R;**				■	■					
5	**R1.i := L1.i + R2.i;**				■		■				
6	**R1.v := R1.i * R1.R;**						■	■			
7	**C1.v + (-R2.v) - R1.v := 0;**	■				■		■			
8	der(L1.i) := R2.v / L1.L;					■			■		
9	AC.i := (-C1.i) - R1.i;			■			■			■	
10	G.p.i := R2.i - ((-AC.i) - L1.i - C1.i);			■	■					■	■

Figure 2.16: The assignments and BLT form of RLC circuit with structural singularity

2.9 Solver

DAE systems generated from Modelica models are usually stiff and sparse associated with events and discontinuities. Consequently, Modelica simulation environments opt to employ generalized implicit DAE solvers. The BDF-methods are frequently used because of the wider stability region for stiff systems [16]. BDF-methods can be used for solving the DAE of index 1.

Two famous DAE solvers used in the Modelica community are DASSL [71] and IDA from the open-source SUite of Nonlinear and Differential/Algebraic Equation Solvers (SUNDIALS) [78]. In this research, we review the functionality of the IDA solver because it was frequently used for EMT simulation throughout this research. In EMT computations, IDA showed a lower CPU time compared to the DASSL. Since both solvers are using the Backward Differentiation Formula (BDF) method, therefore an overview will be helpful.

2.9.1 BDF-methods

The BDF-methods are a group of *multi-step* methods in the general form of (2-128) in which the function value only will be calculated in the point that is going to be found.

$$\frac{1}{h}\sum_{j=0}^{k}\alpha_j \mathbf{y}_{n-j} = \sum_{j=0}^{k}\beta_j \,\mathbf{f}(\mathrm{x}_{n-j}, \mathbf{y}_{n-j}) \tag{2-128}$$

h refers to the step-size used, α_j and β_j are coefficients for respectively the number of backward steps and the function value at the backward steps. k is called the order of the method and denotes how many backward steps are used. The technique is implicit if $\beta_0 \neq 0$ and if so, it will be necessary to calculate the function value in the point iteratively [2][8]. BDF-method is an implicit multi-step method where $\beta_0 \neq 0$ but $\beta_{1,\dots,k} = 0$. Numerical solvers treating the BDF methods are usually associated with a modified Newton method to solve nonlinear systems at each time step.

The simplest BDF-method is the *implicit Backward Euler* method of order 1 (i.e., BDF 1).

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) \tag{2-129}$$

The backward Euler method is an *A-stable* integration method with *stiff decay*. The latter has an unfavorite impact on high-frequency transient simulations and makes it inappropriate for the EMT computations.

Assuming the semi-explicit form of DAEs as given by:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}, \boldsymbol{z}) \tag{2-130}$$

$$\mathbf{0} = \mathbf{g}(t, \mathbf{y}, \boldsymbol{z}) \tag{2-131}$$

According to the implicit function theorem [79], the equation (2-131) can be reformulated as:

$$\boldsymbol{z} = \bar{\mathbf{g}}(t, \mathbf{y}) \tag{2-132}$$

Therefore, equation (2-130) can be re-written as:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}, \bar{\mathbf{g}}(t, \mathbf{y})) \tag{2-133}$$

Now, if equation (2-133) is discretized using the Backward Euler method, (2-129), we will have:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\, \mathbf{f}(t_n, \mathbf{y}_{\boldsymbol{n}}, \bar{\mathbf{g}}(t_n, \mathbf{y}_{\boldsymbol{n}})) \tag{2-134}$$

In conclusion, Backward Euler is the simplest first-order method, convergent for semi-explicit index 1 DAE.

2.9.2 IDA solver

The IDA solver is designed to address the initial value problem (IVP) for a DAE of the form:

$$\mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}}) = 0, \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \qquad \dot{\mathbf{y}}(t_0) = \dot{\mathbf{y}}_0 \tag{2-135}$$

where $\mathbf{y}$ and $\dot{\boldsymbol{y}}$ are vectors in $\mathbf{R}^N$, t is the independent time variable and $\boldsymbol{y}_0$, $\dot{\boldsymbol{y}}_0$ are given initial values.

The IDA uses the variable-order, variable-coefficient BDF integration method in fixed-leading-coefficient form. The order used in IDA ranges from 1 to 5, with the BDF of order q given by the multistep formula (2-136).

$$\sum_{i=0}^{q} \alpha_{n,i}\, \mathrm{y}_{n-i} = h_n \dot{\mathrm{y}}_n \tag{2-136}$$

Where y_n and $\dot{\mathrm{y}}_n$ are the approximation of $\mathrm{y}(t_n)$ and $\dot{\mathrm{y}}(t_n)$, $h_n = t_n - t_{n-1}$is the step size. The coefficient $\alpha_{n,i}$ are uniquely computed for each q. The insertion of (2-136) in the DAE system (2-135) yields the nonlinear algebraic system (2-137) to be solved at each step.

$$\mathbf{G}(\boldsymbol{y_n}) = \mathbf{F}\left(t_n, \mathbf{y}_n, {h_n}^{-1} \sum_{i=0}^{q} \alpha_{n,i}\, \mathbf{y}_{n-i}\right) = 0 \tag{2-137}$$

The nonlinear system (2-137) is solved with *Newton iteration*. For each Newton correction, this leads to a linear system of the form:

$$\mathbf{J}\left[{\mathbf{y}_n}^{(m+1)} - {\mathbf{y}_n}^{(m)}\right] = -\mathbf{G}(\mathbf{y}_n)^{(m)} \tag{2-138}$$

where ${\mathbf{y}_n}^{(m)}$ is the m^{th} approximation to $\mathbf{y}_n$. Here $\mathbf{J}$ is the Jacobian, where defined as:

$$\mathbf{J} = \frac{\partial \mathbf{G}}{\partial \mathbf{y}} = \frac{\partial \mathbf{F}}{\partial \mathbf{y}} + \alpha \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{y}}} \tag{2-139}$$

The scalar $\alpha = \alpha_{n,0} {h_n}^{-1}$ and it changes whenever the step size or method order changes [80].

For the solution of the linear systems, IDA has two options; first, a *direct family* comprising direct linear solvers for dense matrixes, and second, scaled *preconditioned iterative* (Krylov) linear solvers [80]. For large-scale stiff systems, where direct methods are not feasible, the combination of a BDF and any of the preconditioned Krylov methods (SPGMR, SPBCG, or SPTFQMR) is used [80].

In controlling errors at various levels, IDA uses a weighted root-mean-square norm for all error-like quantities. The multiplicative weights are based on the current solution and the relative and absolute tolerances defined by the user.

$$W_i = [Relative\ TOl\ |y_i| + Absolute\ Tol_i]^{-1} \tag{2-140}$$

When using direct linear solvers, the nonlinear iteration (2-138) is a *Modified* Newton iteration, in the sense that the Jacobian $\mathbf{J}$ is fixed throughout the nonlinear iterations and approximated by:

$$\mathrm{J}_{ij} = \left[\mathrm{F}_i\left(t, \mathrm{y} + \sigma_j e_j, \dot{\mathrm{y}} + \alpha\sigma_j e_j\right) - \mathrm{F}_i(t, \mathrm{y}, \dot{\mathrm{y}})\right]/\sigma_j \tag{2-141}$$

knowing that:

$$\sigma_j = \sqrt{U}\, max\left\{|y_j|, |h\dot{y}_j|, \frac{1}{W_j}\right\} sign(h\dot{y}_j) \tag{2-142}$$

where U is the unit roundoff.

For the case of Krylov methods as the linear solver, the iteration (2-138) is an *Inexact* Newton iteration [82], in which $\mathbf{J}$ is obtained through a matrix-free products $\mathbf{J}v$. These products are approximated by:

$$\mathbf{J}v = [\mathbf{F}(t, \mathbf{y} + \sigma v, \dot{\mathbf{y}} + \alpha\sigma v) - \mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}})]/\sigma \tag{2-143}$$

where $\sigma = 1/\|v\|$.

Solving (2-138) using the Krylov methods rarely converge if preconditioning is not used. Assuming a linear system $\mathbf{Ax} = \mathbf{b}$, it can be preconditioned on the *left* by preconditioning matrix $\mathbf{P}$, using:

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b} \tag{2-144}$$

Then, Krylov methods are applied to (2-144) instead of $\mathbf{Ax} = \mathbf{b}$**.** In IDA, the matrix $\mathbf{P}$ is approximated by:

$$\mathbf{P} \approx \frac{\partial \mathbf{F}}{\partial \mathbf{y}} + \alpha \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{y}}} \tag{2-145}$$

2.9.3 ODE mode

The typical workflow of Modelica is to convert the flattened DAEs to explicit ODEs, as illustrated in Figure 2.17. This procedure requires removing the algebraic loops and index reduction, which are time-consuming and increase the simulation time.

A numerical integration method and linear and non-linear system solvers for the implicit equations are needed to solve the ODE function. Since EMT models give rise to stiff problems, an implicit solver is selected. Applying an implicit solver yield a nonlinear system of equations $\mathbf{G}$ for each time point as:

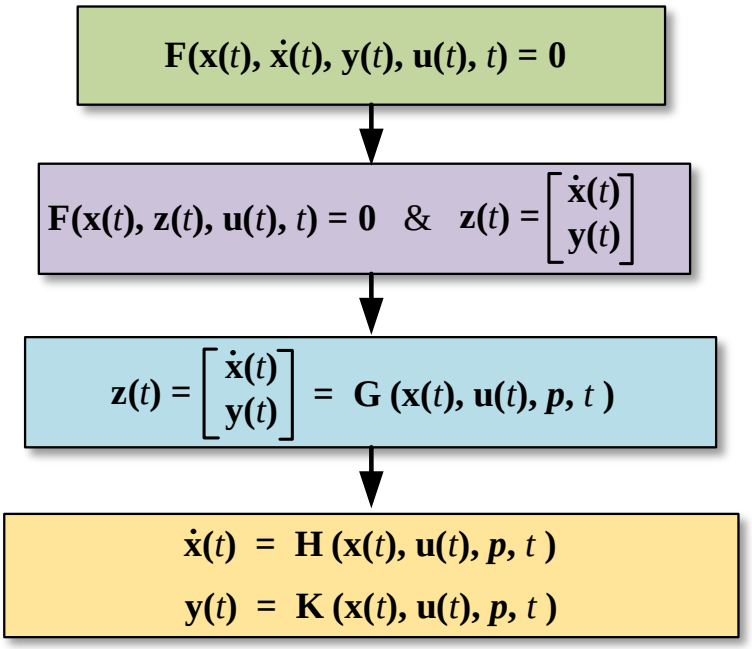


Figure 2.17 The transformation of implicit DAE to explicit ODE

$$\mathbf{x}(t+\Delta t) = \mathbf{G}(\mathbf{x}(t+\Delta t), \mathbf{u}(t), \mathbf{p}, \Delta t, t) \tag{2-146}$$

Newton's method is the principal method for finding the roots of one such system.

$$\mathbf{J}.[\mathbf{x}(t+\Delta t) - \mathbf{x}(t)] = \mathbf{R} \tag{2-147}$$

where **R** denotes the residual form of equation (2-146), and **J** is the corresponding Jacobian matrix.

Numerical computation of Jacobian is the most time-consuming part of solving an ODE. Several methods are implemented in OpenModelica, such as finite-difference approximations, finite difference approximations with coloring, symbolical Jacobian, and symbolical Jacobian with coloring [84]-[86].

For example, in the widely used finite difference method, a numerical approximation of the directional derivative of a vector-valued function **f** is calculated using the formula (2-148).

$$\mathbf{J}_{i,j} = \frac{\partial \mathbf{f_i}}{\partial \mathrm{x}_j} = \frac{f_i\left(\mathrm{x}_j + e_j h_j\right) - f_i(\mathrm{x}_j)}{h_j} \tag{2-148}$$

where h_j is the increment.

2.9.4 DAE mode

As explained in previous sections, engineering applications usually lead to DAEs. Solving the DAEs directly has been considered by mathematicians since several decades ago [2], [7], [8]. For achieving this goal, numerical solvers have been developed in MATLAB based on mass matrix solvers [87]. The main approach is to reduce the differential index and convert the DAEs into semi-explicit (index 1). BDF method, Runge–Kutta [3] methods, RADAU5 [3] can be used for solving the implicit DAEs index 1[2]. The main challenge for handling the DAEs is finding a consistent initial value to satisfy all equations.

The main advantage of solving the DAEs directly remains in skipping of elimination of algebraic loops, tearing, and the generation of symbolic Jacobians. This, in turn, improves the CPU time for large-scale networks drastically [88].

2.9.5 Exploring of Events Handling and Zero Crossings.

Event or discontinuity handling is an important and very common problem for EMT simulations. Many examples of event and discontinuous models exist, such as the opening and closing of a circuit breaker and high-frequency switching of power electronic devices.

In traditional EMT-type simulators, e.g., EMTP®, discontinuities such as current interruption in an RL circuit may lead to numerical oscillations of the trapezoidal rule of integration. Elimination of the oscillations for some models, e.g., machine, is carried out by adding a parallel damping resistance with the RL series circuit [89]. The critical damping adjustment (CDA) method [56] has also been implemented to alleviate the problem. The concept of the CDA method is to use the backward Euler method with halved step for two consecutive time points [89] for some predefined discontinuities.

Event handling in Modelica, i.e., OpenModelica and Dymola, is different and is based on finding the exact time of the event. This feature is ideal for high-frequency switching components with the

penalty of computational cost and higher runtime. Discontinuity handling is possible with step-size control in variable step solvers, i.e., IDA.

Conditional expressions, e.g., `if`, `when`, etc., or built-in functions like `ceil`, `floor`, `div`, etc., generate the *events*. They are categorized as *time events* and *state events*: (1) Time events refer to those discontinuities that involve the time, the built-in global variable that is handled as an input to all models, and we know in advance when it occurs. For example, the closing time of a switch or a thyristor. We know that a thyristor switches on precisely at the firing angle of α at each period. Treatment of the time events is relatively easy, and it is required only to program the timings in the algorithm. (2): sometimes events are generated because of conditional expressions that involve solution variables (e.g., zero-crossing current of a thyristor) and include the discontinuities that we do not know when they occur. The type of discontinuities is called "state events." For example, the opening of the ideal switch occurs when the simulation clock is larger than $T_{opening}$ and the current passing the switch is zero. In the thyristor model, it is not known in advance when the thyristor will open. We just know that it will open when the current passes through zero. For the state event, we know the *event condition* rather than the event time.

For both examples, it is required to implement a root-finding algorithm (zero-crossing function). The zero-crossing function should be monitored continuously during simulation. Since the event time is unknown, it is impossible to reduce the step size to hit them accurately. Instead, we need some sort of iteration (or interpolation) mechanism to locate the event time.

Figure 2.18 demonstrates the typical event handling in the OpenModelica compiler. The simulation starts by finding consistent initial values for the model variables. Event conditions are continuously monitored during the simulation. Variables to be tested for zero-crossing are placed in a vector. When an event is alerted during a time step, it affects the step–size control of the integration algorithm by forcing the simulation to iterate to the earliest zero–crossing within the current integration step. This procedure is called *event iteration*. The iteration process for state events can slow down the simulation. During the event iteration, after handling each event, the entire model is re-evaluated [91].

After the event is treated, the solver needs to find consistent restart values (re-initialization) for the variables of the hybrid DAE model before resuming the integrator for the continuous-time part.

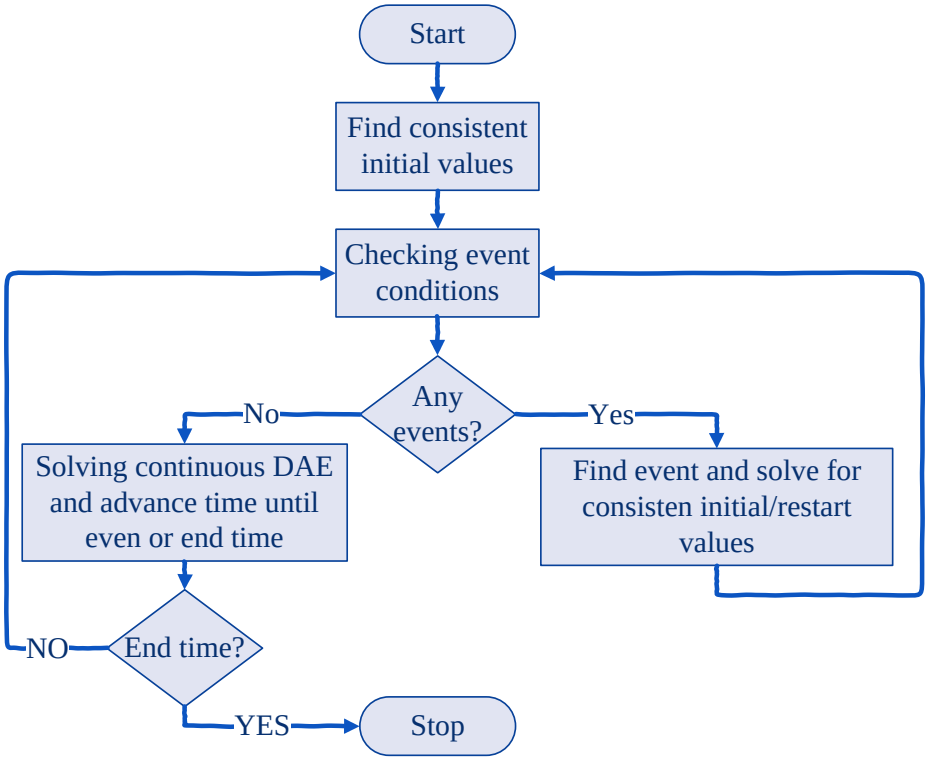


Figure 2.18 Typical event handling algorithm of Hybrid DAE [91]

The algorithm for multi zero-crossing function occurred in a time step is like above. The goal is to reduce the time step in so far as the zero-crossing point is isolated.

For piecewise linear functions $y = f(x)$, where y is used in a state-space model, step-size control should be applied to reduce the step size whenever x passes through the breaking points within an integration step.

To show the discontinuity treatment of Modelica solver (i.e., DASSL) compared with EMTP®, consider the buck-booster converter [92] extracted from EMTP® examples. Figure 2.19 shows the schematic diagram of a converter designed by the GUIs of the MSEMT library. This converter is designed to work in discontinuous mode; therefore, the inductor is completely discharged at the end of the commutation cycle. In this circuit, the pulse generator provides a pulse with a period of 10 μs (switching frequency= 100 kHz) with the duty cycle of 75%. The diode has been modeled with a highly nonlinear resistance. Figure 2.20 shows the voltage and current characteristics of the

diode, which are constructed by 15 piece-wise linear curves. This circuit represents a very stiff system of equations.

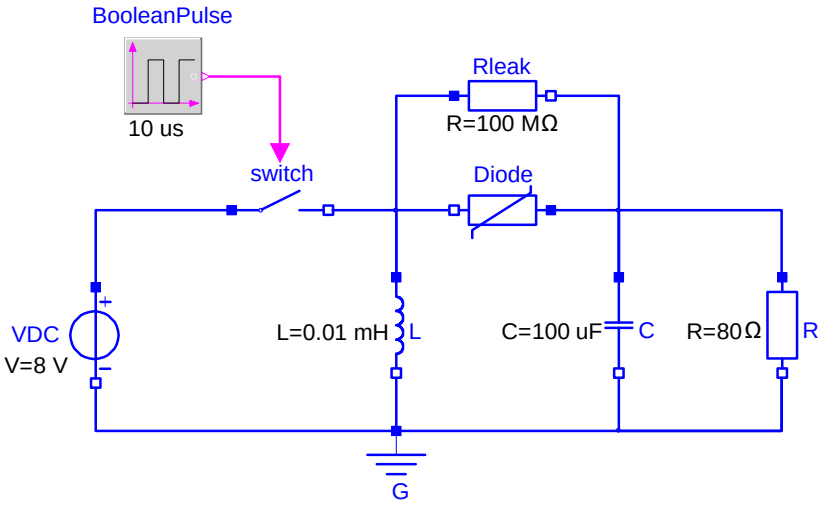


Figure 2.19 Buck-Boost converter for demonstrating simultaneous switching with controls

Table 2.2 shows the parameter of simulation in the two software. It should be noted that the option of "simultaneous switching" has been activated in EMTP®.

Table 2.2 Parameters of simulators and performance comparison

Characteristics	**Modelica (Dymola)**	**EMTP®**	
Solver	DASSL	Trapezoidal/BE	
Δt		Δt: 0.1 μs	Δt: 0.01 μs
Tolerance	1e-6	-	
Δt_{MIN}	8.84e-13 s		
Δt_{MAX}	3.75e-06		
f-evaluations	302 190	-	
J-evaluations	125 976	-	
CPU time (s)	6.34	4.22656	26.31
Number of time-steps	162 894	156 490	1 507 705
CPU-time for one grid interval	0.0422 ms	0.027 ms	17 μs
Performance ratio	1	0.66	4.14

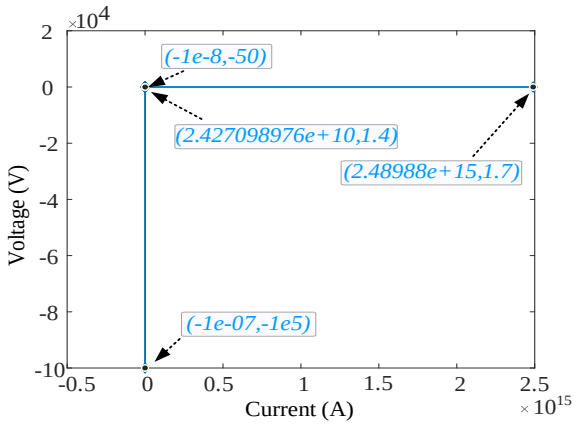


Figure 2.20 The i-v characteristics of the diode

Figure 2.21 shows the inductor current graphs obtained by Modelica and EMTP®. In the zoomed view, a point-to-point comparison of solutions can be observed.

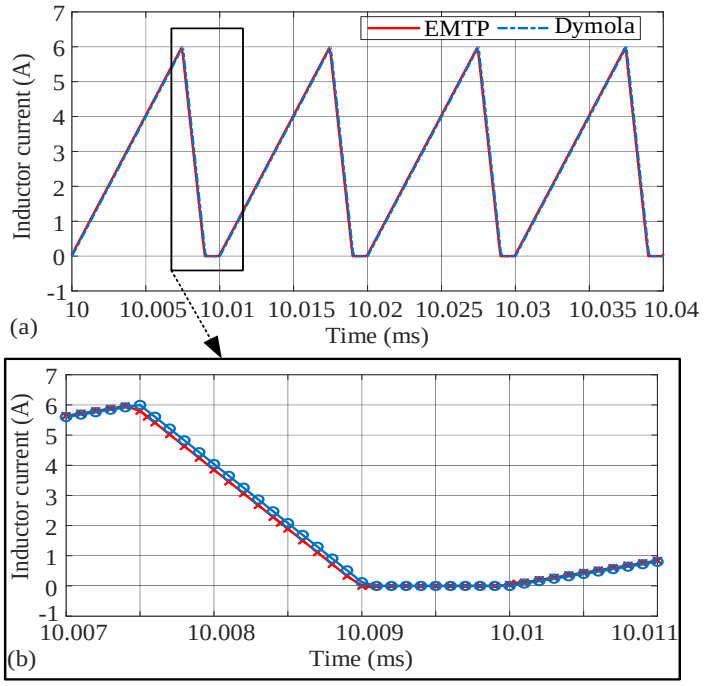


Figure 2.21 (a): Inductor current in the discontinuous mode of Buck-Booster convertor. (b): the close-up view of the inductor current

As it can be seen, both solutions have an excellent agreement even though they have two different solution methods. As we can observe in Table 2.2, the CPU time obtained by Modelica is nearly

close to the one we have from EMTP®. Since two distinct solvers have carried out simulations, comparing the one-grid interval CPU time may be better, introducing a fair comparison.

To show more details on discontinuity handling, let us compare the curves of current passing through the switch accompanied with the pulse waveforms. Figure 2.22.(a) illustrate the Modelica solution for current depicted by blue curve and the pulse waveform distinguished by the red curve. As observed at t = 9.6475 ms, there is a discontinuity, and the pulse waveform changes its status from 1 to 0. The switch, controlled by the pulse generator, changes its "closing" status to open immediately and at the same time point, t = 9.6475 ms. It means Modelica calculates the equations once before the discontinuity is triggered at $t = 9.6475^{-}$ms and once after the discontinuity happens at $t = 9.6475^{+}$ms.

Let's see what EMTP® executes when the clock approaches the discontinuity point. The problem is solved twice in EMTP®: (1) with using simultaneous switching, which is depicted by the blue graph, and (2) without selection of this option plotted by the magenta graph.

Simultaneous switching acts on switches as nonlinear functions and recalculates the network equations without advancing the timepoint. This is distinguishable in Figure 2.22.b. However, EMTP® for the handling of discontinuity at t = 9.6475 ms, gives two timepoints whose the solution with using the simultaneous switching is closer to the Modelica results; one timepoint before the discontinuity is triggered at t = 9.6474 ms and next time point when the discontinuity occurs, at t = 9.6475 ms. Therefore, as it can be observed in Figure 2.22.b., the raise is not fully vertical. This is while we know the abrupt change of switch occurs precisely at the instant of t = 9.6475 ms.

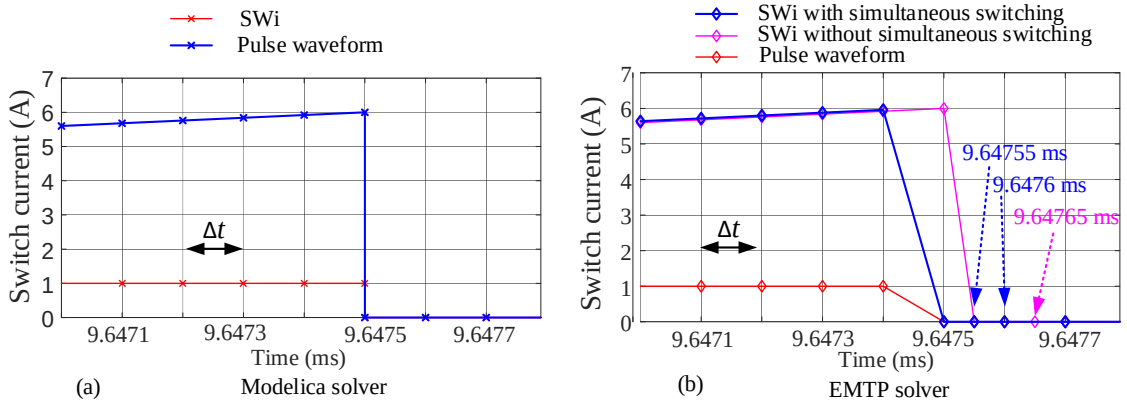


Figure 2.22 Switch current (a): in Modelica (b): in EMTP®

Figure 2.23 sketches the voltage waveform of resistance. The red and magenta curves show the results obtained from EMTP®, respectively, for step sizes of $0.1\mu s$ and $0.01\mu s$. The blue curve shows the results obtained from Modelica. It can be observed that the EMTP®'s results with the step size of $0.01\mu s$ gives the best precision, which is close to Modelica solutions. The CPU time measured in EMTP® for these resolutions is 26.31 s, which gives the ratio 4.14:1 compared to Modelica.

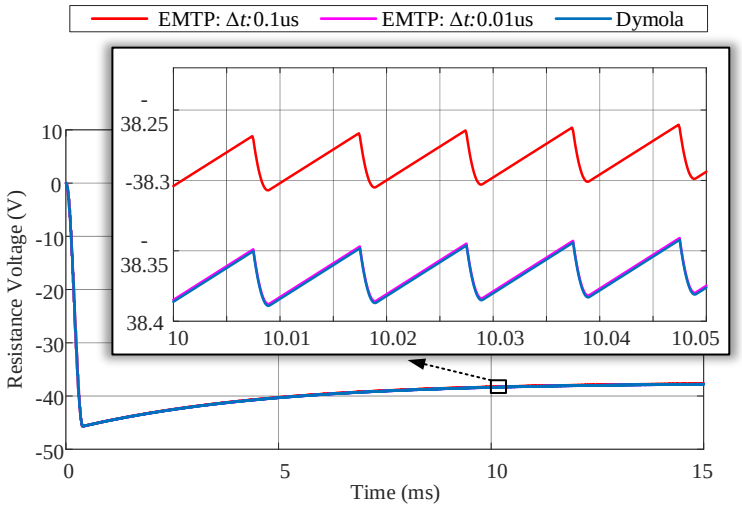


Figure 2.23 The curves of resistance voltage in Modelica and EMTP®

2.10 Exploring of Switch Equation

In Modelica, the status of an ideal switch, that is, zero current in switch-off mode and zero voltage drop in switch-on mode can be defined as:

```
Boolean Switch "Indicates off-state";
If Switch then i=0 else v=0;
```

Modelica compiler translates the above equation into the algebraic form, given by:

$$mi + (1 - m)v = 0 \tag{2-149}$$

where m is a Boolean variable and indicates the switch status, m is "1" when the switch is off, and "0" when the switch status is on. Equation (2-149) can be causalized in two different ways:

$$i = (m - 1)v/m \tag{2-150}$$

$$v = -mi/(1-m) \tag{2-151}$$

Equations (2-150) and (2-151) become undefined (i.e., division by zero) when the switch is closed, i.e., $m = 0$ or open, i.e., $m = 1$, respectively. As such, ideal switch should be implemented by a snubber circuit for some cases.

2.11 Exploring of Control Systems Modeling

In traditional EMT-type simulators, EMTP®, EMTDC, etc., control systems equations are solved separately from the main power electric network. The main drawback is that the circuit and the controls are separately solved, resulting in one time-step delay between the solutions. An iterative method has been proposed and validated in [93] to offer a simultaneous solution for control circuits in EMTP® as well.

As described earlier, the solution method in Modelica is such that the entire electrical network, including the equations describing the electric power system and the ones representing the dynamic behavior of control systems, is formulated in DAE form, then, the system is solved together and simultaneously without any delay between linear and nonlinear components.

The accuracy of Modelica control system models is validated in Chapter 5 and Chapter 7.

2.12 Exploring of Nonlinear Models

The solution method of traditional EMT-type simulators working based on nodal analysis was described in Section 1.5.2. There are usually two main categories for solving nonlinear functions: with solution delays (as in EMTP) and without solution delays. In Modelica's main solvers, IDA and DASSL, a fully iterative method is used, and all solution delays are suppressed by solving all nonlinear functions of a DAE system simultaneously. Typical examples are the surge arrester model, nonlinear inductor, power transformer (STC model), and magnetic saturation of synchronous machine. These models are tested in Chapter 5, Chapter 6, and Chapter 7.

2.13 Interfacing to Other Software

Modelica models can interface with many other languages such as C, Julia, MATLAB, and Python [94]. On the other hand, Modelica models are fully compatible with the FMI standard. This

interface allows model exchange and co-simulation with other simulating tools. FMI allows to export of pre-compiled models, i.e., C-code or binary code, from a tool for import in another tool and vice versa. The FMI standard is Modelica independent. Import and export work both between different Modelica tools or between specific non-Modelica tools.

CHAPTER 3 MSEMT: AN ADVANCED MODELICA LIBRARY FOR ELECTROMAGNETIC TRANSIENT SIMULATIONS

The first step for EMT simulation in Modelica is to create a library with EMT-detailed electrical elements, including the main components such as transmission line, generator, controllers, etc. For this purpose, the Modelica Simulator of Electromagnetic Transient (MSEMT) has been developed to solve practical power system transient problems. In this chapter, mathematical representation and implementation of each model will be provided.

The library is constructed in stand-alone mode and independent from Modelica Standard Library (MSL)[14]. The MSL is a free library developed by the Modelica Association [66] and includes the fundamental components for modeling mechanical, electrical, thermal, fluid, and control systems. In the electrical branch, it consists of the simplified models of some electrical components. The available models in the electrical branch are different from the *EMT-detailed* models developed in the MSEMT. For example, there are no wideband or constant parameter models for transmission lines/cables, surge arrestor, nonlinear inductor, detailed models of the synchronous machine with saturation and control, power transformer (STC model), coupled RL, arc models, exciter, governor, etc. Moreover, the synchronous machine model in the MSL, for example, uses *space phasors* which is less accurate than EMT-type machine modeling applicable to generic unbalanced systems.

All developed models of the MSEMT library comply with the EMTP® models. This is because the results obtained from the library must be validated against the EMTP® software. The following chapters will discuss the results obtained from IEEE 13-Bus, IEEE 39-Bus, or IEEE 118-Bus benchmarks.

3.1 Overview of the MSEMT Library

Figure 3.1 presents an overview of a subset of a simplified implementation of the MSEMT library. This structure is inspired from EMTP®. MSEMT contains two top-level branches: Electrical and NonElectrical. The advanced blocks of electric power components exist in the Electrical branch, making it possible to simulate the electric power benchmarks such as IEEE 118-bus. The NonElectrical branch defines the type of variable, functions, blocks (adder, integrator, etc.),

Boolean algebra (AND, OR, flip-flops, etc.), and consists of icons, symbols, etc. required to have a comprehensive simulation [75].

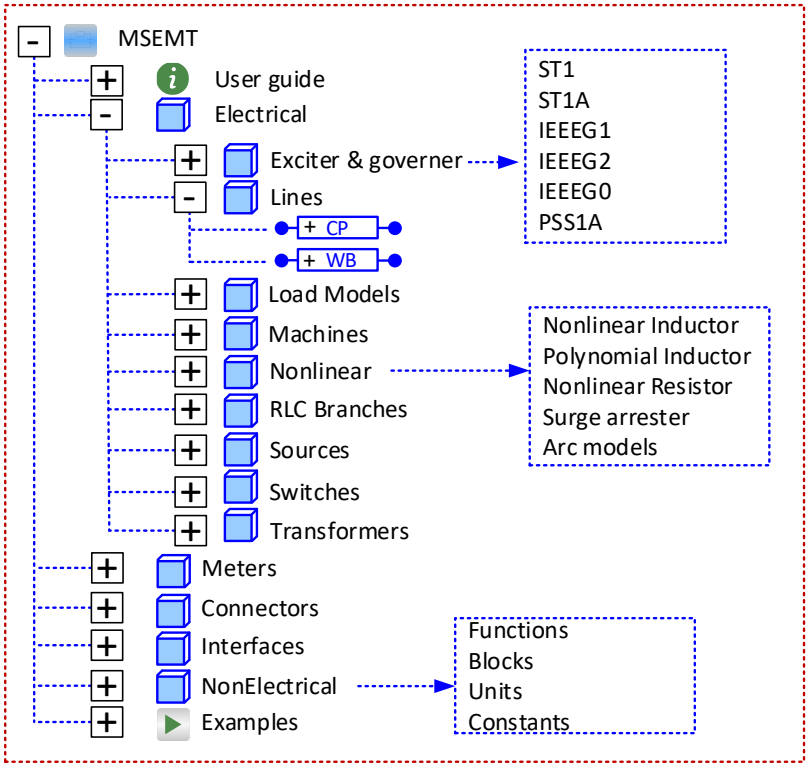


Figure 3.1 Structure of MSEMT library

3.2 Controllers

An excitation control system is a feedback control system that includes the synchronous machine and excitation system. An excitation system is defined as the source of the field current for exciting a synchronous machine.

The developed controllers include the static exciters (types ST1 and ST1A), Governors (types IEEEG1, IEESGO), and the power stabilizer type PSS1A.

The controllers are programmed in block-oriented modeling, as a combination of pre-defined block diagrams, such as adders, multipliers, first-order integrators, lead-lag compensators, etc. These blocks are defined in the branch `MSEMT.NonElectrical.Blocks`.

In Modelica, block diagrams are defined under a specialized class `block`. A `block` is a `class` with fixed causality (data flow direction is known); each part of its interface must have causality equal to `input` or `output`. All variables used in a `block` should be declared by one of the prefixes `input` or `output`, and this is the main restriction of a block,

3.2.1 Exciter ST1

Figure 3.2 illustrates the implementation of the excitation system type ST1 excitation system model, which is a controlled rectifier exciter with a potential source. It means the required DC power is provided through a transformer and rectifier. This device is implemented as described in [95], [96], and complies with the EMTP® model. It allows the modeler to compare the obtained results with EMTP®. As one can see in this diagram, when VREF is not connected, the reference voltage is internally found from the steady-state parameter. Initialization of model is performed from the values obtained from steady-state solution in EMTP®. The parameters of the component are given in Table 3.1.

Table 3.1 The parameters of Exciter ST1

	Name	Description	Type
Data tab	Time constant TB	lead-lag time constant	second
	Time constant TC	lead-lag time constant	second
	Gain KF	excitation control system stabilizer gain	pu
	Time constant TF	excitation control system stabilizer time constant	second
Exciter tab	Gain KA	voltage regulator gain	pu
	Time constant TA	voltage regulator time constant	second
	Rectifier loading factor KC	rectifier loading factor	pu

	Maximum regulator output VImax	maximum regulator voltage input	pu
	Maximum regulator output VImin	minimum regulator voltage input	pu
	Maximum regulator output VRmax	maximum regulator voltage output	pu
	Maximum regulator output VRmin	minimum regulator voltage output	pu

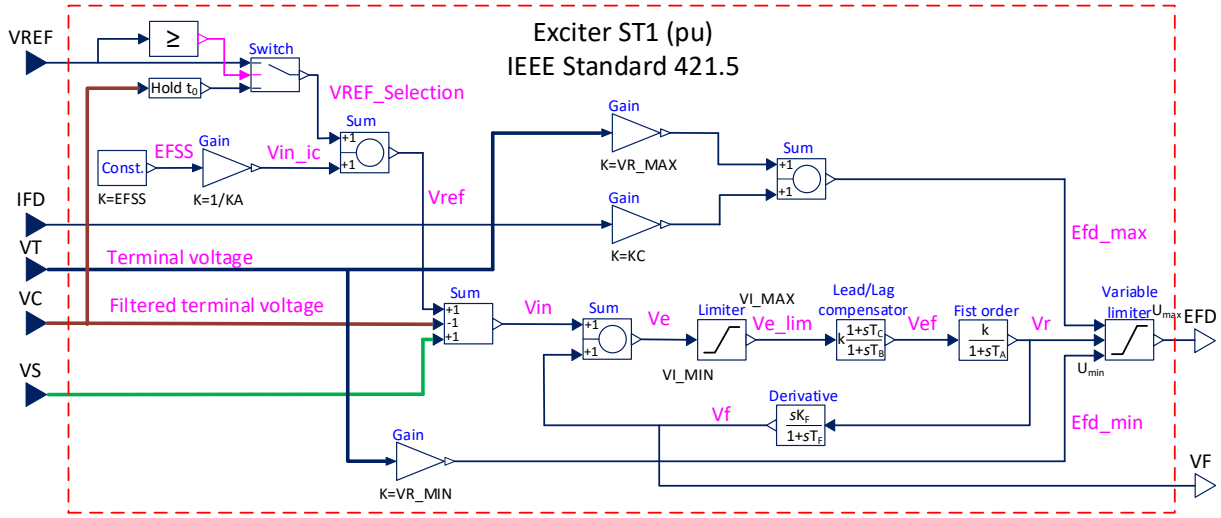


Figure 3.2 Implementation of Exciter ST1 in Modelica

3.2.2 Governor IEEEG1

Figure 3.3 shows the implementation of governor type IEEEG1 and represents a steam governor model. The model is implemented in Modelica as per the definitions and diagram in [97] and in compliance with the EMTP® model. It senses changes in the turbine speed and adjusts the steam

input accordingly. Its response time is generally in the order of seconds. Initialization of model is performed from the values obtained from steady-state solution in EMTP®.

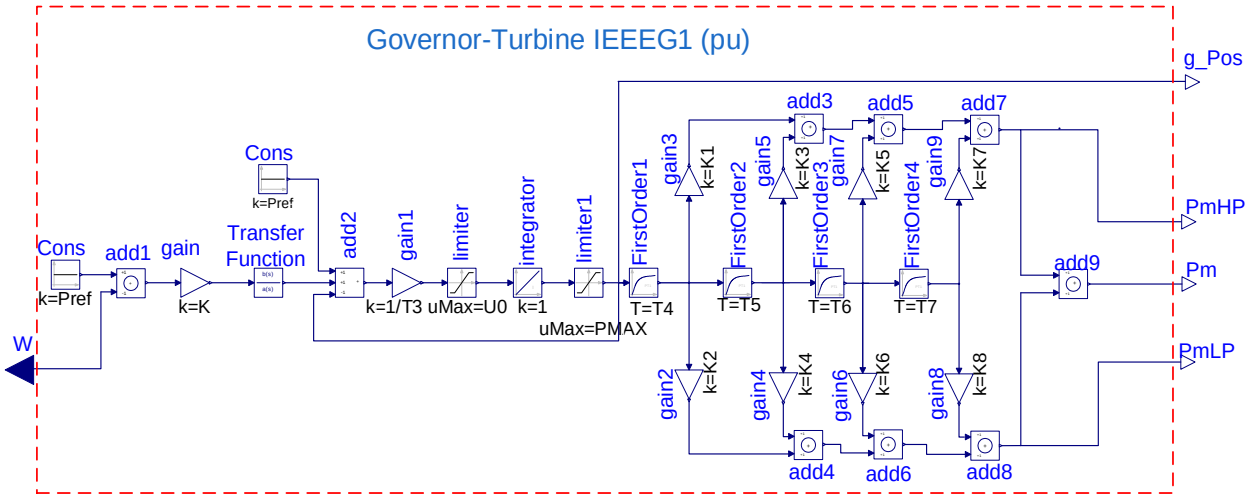


Figure 3.3 Implementation of governor IEEEG1 in Modelica

3.2.3 Governor IEESGO

Figure 3.4 shows the component model of the governor/turbine IEESGO [97] implemented in Modelica and compliance with the EMTP® model. Table 3.2 shows the parameter of the model. The accuracy of this model has been validated in IEEE 118-bus network.

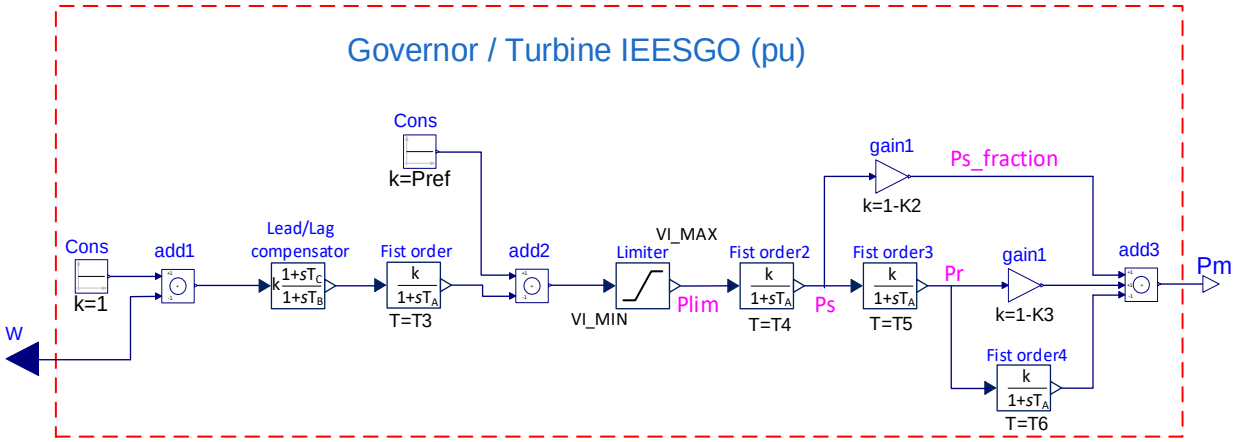


Figure 3.4 Implementation of governor IEESGO in Modelica

Table 3.2 The parameters of Exciter IEESGO

	Name	Description	Type
Governor tab	Governor gain K1	Governor gain	pu
	Lag time constant T1	governor lag time constant	second
	Lead time constant T2	governor lead time constant	second
	Lag time constant T3	governor lag time constant	second
	Maximum power limit PMAX	maximum power limit	pu
	Minimum power limit PMIN	minimum power limit	pu
Turbine tab	Time constant T4	steam flow time constant	second
	Time constant T5	reheater time constant	second
	Time constant T6	IP-LP reheater time constant	second
	Reheater fraction of shaft power K2	reheater fraction of power shaft	pu
	IP-LP fraction of shaft power K3:	IP-LP power fraction	pu

3.3 Transmission Line

The transmission line (TL) is one of the main components of a power system. In EMT studies, two types of models are mainly introduced for transmission lines. These are the PI-section model and the distributed parameter models. The PI-section is a basic model that does not represent propagation delay. The distributed parameter models are more accurate and represent propagation

delay. The models include the constant parameter (CP)-line model, the frequency-dependent (FD)-line model, and the wideband (WB)-line model. The latter is sometimes called the Universal Line Model (ULM).

The CP-line model is the simplest and most efficient one, where the model parameters are frequency independent. The FD-line model [98] evaluates multi-conductor line propagation in the modal domain and considers effects due to frequency dependence of line parameters. However, because modal transformations are approximated by real and constant matrices, its accuracy is best for cases of aerial lines which are continuously transposed.

The WB-line model [99] considers the full-frequency dependency of line parameters and works directly in the phase domain, thus, avoiding simplifying assumptions regarding modal–to–phase transformations.

This section aims at providing a clear and complete description of the theoretical basis for the PI-section, CP-, and WB-line models associated with the implementation of models in Modelica.

3.3.1 PI-section Line Model

In the PI-section model, as depicted in Figure 3.5, mutually coupled RLC elements are used to construct a linear model with a finite number of states. The PI-section model is generally not the best choice for transient solutions because traveling waves on lines can not be reproduced accurately in the model.

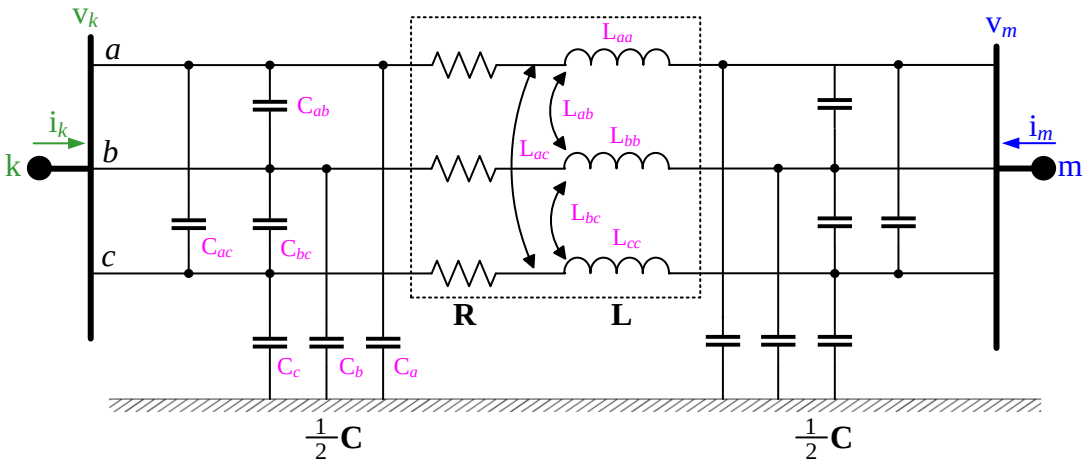


Figure 3.5 Three-phase nominal PI-section model of the transmission line

The linear state-space representation of the PI-section line model is defined by equations (3-1)-(3-3).

$$\mathbf{v}_k - \mathbf{R}\mathbf{i}_{RL} - \mathbf{L}\frac{d\mathbf{i}_{RL}}{dt} = \mathbf{v}_m \tag{3-1}$$

$$\mathbf{i}_k = \frac{\mathbf{C}}{2}\frac{d\mathbf{v}_k}{dt} + \mathbf{i}_{RL} \tag{3-2}$$

$$\mathbf{i}_m = \frac{\mathbf{C}}{2}\frac{d\mathbf{v}_m}{dt} - \mathbf{i}_{RL} \tag{3-3}$$

where **R** , **L** and **C** indicate the total resistance, inductance, and capacitance of transmission line, respectively. $\mathbf{i}_k$ and $\mathbf{i}_m$ are the current vectors in *k*- and *m*- ends respectively, and $\mathbf{i}_{RL}$ denotes the vector of current flowing into the coupled RL branch. The implementation of this model in Modelica is presented in Figure 3.6. The RLC line section parameters can be given either by a 3-by-3 matrix or 2-by-1 matrix representing the balanced transmission line's positive and zero sequences.

```
model PI3ph "Three-phase pI line model"
 parameter MSEMT.NonElectrical.Units.Resistance R[:,:]
 (each displayUnit="Ohm") "R3x3 or R2*1=[R0, R1]";
 parameter MSEMT.NonElectrical.Units.Inductance L[:,:]
 (each displayUnit="mH")  "L3x3 or L2*1=[L0, L1]";
 parameter MSEMT.NonElectrical.Units.Capacitance C[:,:]
 (each displayUnit="uF")  "C3x3 or C2*1=[C0, C1]";
 MSEMT.Connectors.PosPlug Pk ;  // k-end port
 MSEMT.Connectors.negPlug Pm ;  // m-end port
protected
 final parameter Integer iR=size(R,1); // Identification of row dimension of R
 final parameter Integer jR=size(R,2); // Identification of column dimension of R
 parameter Real Rp[:,:] = if iR==jR then R else
         [(2*R[1,2] + R[1,1])/3,      (R[1,1]   - R[1,2])/3,      (R[1,1]   - R[1,2])/3;
          (R[1,1]   - R[1,2])/3       (2*R[1,2] + R[1,1])/3,      (R[1,1]   - R[1,2])/3;
          (R[1,1]   - R[1,2])/3,      (R[1,1]   - R[1,2])/3,      (2*R[1,2] + R[1,1])/3];

 parameter Real Lp[:,:] = if iR==jR then L else
         [(2*L[1,2] + L[1,1])/3,      (L[1,1]   - L[1,2])/3,      (L[1,1]   - L[1,2])/3;
          (L[1,1]   - L[1,2])/3,      (2*L[1,2] + L[1,1])/3,      (L[1,1]   - L[1,2])/3;
          (L[1,1]   - L[1,2])/3       (L[1,1]   - L[1,2])/3,      (2*L[1,2] + L[1,1])/3];

 parameter Real Cp[:,:] = if iR==jR then C else
         [(2*C[1,2] + C[1,1])/3,      (C[1,1]   - C[1,2])/3,      (C[1,1]   - C[1,2])/3;
          (C[1,1]   - C[1,2])/3,      (2*C[1,2] + C[1,1])/3,      (C[1,1]   - C[1,2])/3;
          (C[1,1]   - C[1,2])/3,      (C[1,1]   - C[1,2])/3,      (2*C[1,2] + C[1,1])/3];
 Real  iRL[3];
equation
 Pm.pin.v = Pk.pin.v - Rp * iRL - Lp * der(iRL);
 Pk.pin.i = Cp / 2 * der(Pk.pin.v) + iRL;
 Pm.pin.i = Cp / 2 * der(Pm.pin.v) - iRL;
end PI3ph;
```

Figure 3.6 Implementation of PI-section line model in Modelica

3.3.2 Distributed Parameter Line Model Equations

The distributed parameter line models rely on the traveling wave theory. Figure 3.7 illustrates an *N*-conductor transmission line with the length of $x = \ell$. The frequency-domain equations describing the line at each point x are:

$$\frac{d\mathbf{I}(x,j\omega)}{dx} = -\mathbf{Y}'(j\omega)\mathbf{V}(x,j\omega) \tag{3-4}$$

$$\frac{d\mathbf{V}(x,j\omega)}{dx} = -\mathbf{Z}'(j\omega)\mathbf{I}(x,j\omega) \tag{3-5}$$

where $\mathbf{I}(x,j\omega)$ is the vector of phase currents, $\mathbf{V}(x,j\omega)$ is the vector of line phase voltages, $\mathbf{Z}' = \mathbf{R}'(\omega) + j\omega\mathbf{L}'(\omega)$ is the series impedance matrix in per unit length and $\mathbf{Y}' = \mathbf{G}'(\omega) + j\omega\mathbf{C}'(\omega)$ is the shunt admittance matrix also in per unit length.

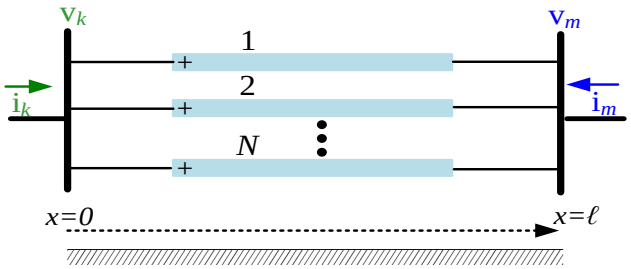


Figure 3.7 *N*-phase transmission line

The differentiation of (3-4) and (3-5) with respect to x leads to below equations.

$$\frac{d^2\mathbf{I}(x,j\omega)}{dx^2} = \mathbf{Y}'(j\omega)\mathbf{Z}'(j\omega)\mathbf{I}(x,j\omega) \tag{3-6}$$

In the same way, the procedure is repeated for (3-5), then we have:

$$\frac{d^2\mathbf{V}(x,j\omega)}{dx^2} = \mathbf{Z}'(j\omega)\mathbf{Y}'(j\omega)\mathbf{V}(x,j\omega) \tag{3-7}$$

The resulting equations are a second-order matrix ODE involving only unknown voltages and currents. Equation (3-8) establishes the general solution of (3-6) and is given by:

$$\mathbf{I}(x,j\omega) = \mathbf{I}_F e^{-\boldsymbol{\Gamma}(j\omega)x} + \mathbf{I}_B e^{\boldsymbol{\Gamma}(j\omega)x} \tag{3-8}$$

where the propagation matrix, $\boldsymbol{\Gamma}(j\omega)$ is:

$$\boldsymbol{\Gamma}(j\omega) = \sqrt{\mathbf{Y}'(j\omega)\mathbf{Z}'(j\omega)} \tag{3-9}$$

Using (3-4) and (3-8), we obtain the general solution (3-10) for equation (3-7). It is given by:

$$\mathbf{V}(x, j\omega) = \mathbf{Y}_c(j\omega)^{-1}\left[\mathbf{I}_F e^{-\boldsymbol{\Gamma}(j\omega)x} - \mathbf{I}_B e^{\boldsymbol{\Gamma}(j\omega)x}\right] \tag{3-10}$$

where $\mathbf{I}_F$ and $\mathbf{I}_B$ are integration constants determined by the line boundary conditions and physically represent the vectors of forward traveling wave (or in the positive x-direction) and backward traveling wave (or negative x-direction). Characteristic admittance matrix, $\mathbf{Y}_c(j\omega)$, is defined as:

$$\mathbf{Y}_c(j\omega) = \boldsymbol{\Gamma}(j\omega)^{-1}\mathbf{Y}'(j\omega) \tag{3-11}$$

If (3-10) is multiplied by $\mathbf{Y}_c(\omega)$ and summed by (3-8), we will have:

$$\mathbf{Y}_c(j\omega)\mathbf{V}(x, j\omega) + \mathbf{I}(x, j\omega) = \mathbf{I}_F e^{-\boldsymbol{\Gamma}(j\omega)x} \tag{3-12}$$

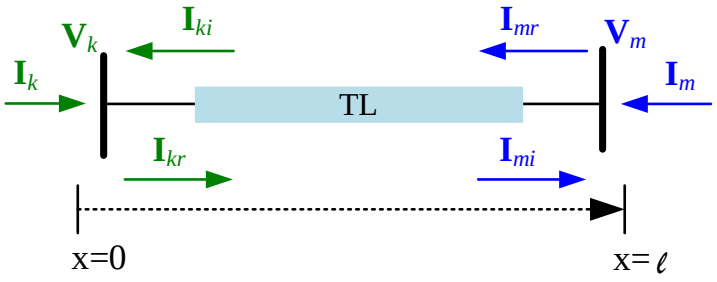


Figure 3.8 Schematic of transmission line with length ℓ and boundary conditions

Figure 3.8 shows the schematic diagram of a transmission line. It is assumed the voltage and current at $x = 0$ are denoted by $\mathbf{V}_k$ and $\mathbf{I}_k$and at $x = \ell$ are represented by $\mathbf{V}_m$ and $\mathbf{I}_m$. It is additionally supposed the direction of the end currents flow into the line. Applying the boundary conditions to (3-12) yields the following equations:

$$\mathbf{I}_F = \mathbf{I}_{kr} \tag{3-13}$$

At node k $x = 0$ $\mathbf{Y}_c\mathbf{V}_k + \mathbf{I}_k = 2\mathbf{I}_{kr}$ (3-14)

$x = \ell$ $\mathbf{Y}_c\mathbf{V}_m - \mathbf{I}_m = 2e^{-\boldsymbol{\Gamma}(j\omega)\ell}\mathbf{I}_{kr}$ (3-15)

$$\mathbf{I}_F = \mathbf{I}_{mr} \tag{3-16}$$

At node m $x = 0$ $\mathbf{Y}_c\mathbf{V}_m + \mathbf{I}_m = 2\mathbf{I}_{mr}$ (3-17)

$x = \ell$ $\mathbf{Y}_c\mathbf{V}_k - \mathbf{I}_k = 2e^{-\boldsymbol{\Gamma}(j\omega)\ell}\mathbf{I}_{mr}$ (3-18)

We can redefine the equations as:

$$\mathbf{Y}_c\mathbf{V}_k - \mathbf{I}_k = 2\mathbf{I}_{ki} \tag{3-19}$$

At node k $$\mathbf{I}_{ki} = \mathbf{H}\mathbf{I}_{mr} \tag{3-20}$$

$$\mathbf{I}_{mr} = \mathbf{I}_{mi} + \mathbf{I}_m \tag{3-21}$$

where the propagation matrix function is defined as:

$$\mathbf{H} = e^{-\boldsymbol{\Gamma}(j\omega)\ell} \tag{3-22}$$

In another way, the vector of phase currents $\mathbf{I}_k$ at *k*-end is related to the vector of phase voltages $\mathbf{V}_k$ and the *incident current* wave $\mathbf{I}_{ki}$ given by (3-19). The incident wave is equal to the *reflected wave* from the *m*-end, $\mathbf{I}_{mr}$, propagated to the *k*-end as represented by (3-20).

Like *k*-end, the equations for the *m*-end are given by:

$$\mathbf{Y}_c\mathbf{V}_m - \mathbf{I}_m = 2\mathbf{I}_{mi} \tag{3-23}$$

At node m $$\mathbf{I}_{mi} = \mathbf{H}\mathbf{I}_{kr} \tag{3-24}$$

$$\mathbf{I}_{kr} = \mathbf{I}_{ki} + \mathbf{I}_k \tag{3-25}$$

Manipulation of the above equations yields the following equations as well.

$$\mathbf{I}_k - \mathbf{Y}_c\mathbf{V}_k = -\mathbf{H}(\mathbf{I}_m + \boldsymbol{Y}_c\mathbf{V}_m) = -\mathbf{H}\mathbf{I}_{m,fw} \tag{3-26}$$

$$\mathbf{I}_m - \mathbf{Y}_c\mathbf{V}_m = -\mathbf{H}(\mathbf{I}_k + \mathbf{Y}_c\mathbf{V}_k) = -\mathbf{H}\mathbf{I}_{k,fw} \tag{3-27}$$

The terms $\mathbf{Y}_c\mathbf{V}_k$ and $\mathbf{Y}_c\mathbf{V}_k$ are considered as *shunt currents* in *m*- and *k*- ends. The vectors of $\mathbf{I}_{m,fw}$ and $\mathbf{I}_{k,fw}$ represent the *forward traveling current-wave* from *m*- and *k*- ends. These equations are valid for both underground cables and aerial TLs.

3.3.3 Constant Parameter Line Model

As earlier mentioned, the CP-line model is the simplest form of the distributed parameter line model with the minimum computational burden. The main challenge in the CP-line model is the computation of delay for voltage and current of each end, with delay value of propagation time.

3.3.3.1 Formulation and theoretical aspects

The CP-line model considers that the TL parameters are not frequency-dependent; consequently, the matrices $\mathbf{Z}'$ and $\mathbf{Y}'$are constant. For the formulation of the CP-line model, first, we drive the equations for the single-phase lossless transmission line, that is $\mathrm{R}' = \mathrm{G}' = 0$. Therefore, we can write the equations (3-26) and (3-27) for single-phase line as:

$$\mathrm{I}_k - \mathrm{Y}_c\mathrm{V}_k = -\mathrm{H}(\mathrm{I}_m + Y_c\mathrm{V}_m) \tag{3-28}$$

$$\mathrm{I}_m - \mathrm{Y}_c\mathrm{V}_m = -\mathrm{H}(\mathrm{I}_k + \mathrm{Y}_c\mathrm{V}_k) \tag{3-29}$$

By multiplying (3-28) and (3-29) by $\mathrm{Y}_c{}^{-1}$ and some mathematical manipulations, we can re-write these equations as below:

$$\mathrm{Z}_c\mathrm{I}_k - \mathrm{V}_k = -\mathrm{H}(\mathrm{Z}_c\mathrm{I}_m + \mathrm{V}_m) \tag{3-30}$$

$$\mathrm{Z}_c\mathrm{I}_m - \mathrm{V}_m = -\mathrm{H}(\mathrm{Z}_c\mathrm{I}_k + \mathrm{V}_k) \tag{3-31}$$

where the characteristic (surge) impedance ($\mathrm{Z}_c = \mathrm{Y}_c{}^{-1}$) and propagation constant are respectively defined as:

$$\mathrm{Z}_c = \sqrt{\frac{\mathrm{L}'}{\mathrm{C}'}} \tag{3-32}$$

$$\mathrm{H} = e^{-j\omega\ell\sqrt{\mathrm{L}'\mathrm{C}'}} \tag{3-33}$$

where L' and C' are the inductance and capacitance of transmission line in unit per length. Therefore, the equations (3-30) and (3-31) are transformed into time domain as given by (3-34)-(3-35).

$$v_k(t) - Z_c i_k(t) = v_m(t-\tau) + Z_c i_m(t-\tau) \tag{3-34}$$

$$v_m(t) - Z_c i_m(t) = v_k(t-\tau) + Z_c i_k(t-\tau) \tag{3-35}$$

knowing that propagation (or travel) time τ is:

$$\tau = \ell\sqrt{L'C'} \tag{3-36}$$

Equations (3-34) and (3-35) can be shown by two Norton equivalents, as illustrated in Figure 3.9. The Norton current sources (called history terms) are defined by:

$$i_k^{hist} = \frac{v_m(t-\tau)}{Z_c} + i_m(t-\tau) \tag{3-37}$$

$$i_m^{hist} = \frac{v_k(t-\tau)}{Z_c} + i_k(t-\tau) \tag{3-38}$$

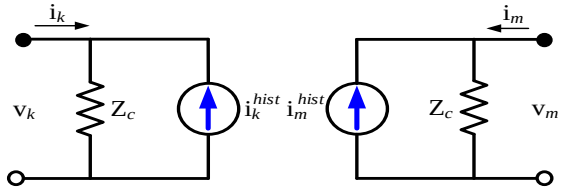


Figure 3.9 Norton equivalent of single-phase lossless CP-line model

For incorporating the losses, the CP-line model is characterized by two lossless line sections, each with a halved propagation time ($\tau/2$) connected in series with lumped resistors $R/4$ at each end as illustrated in Figure 3.10.

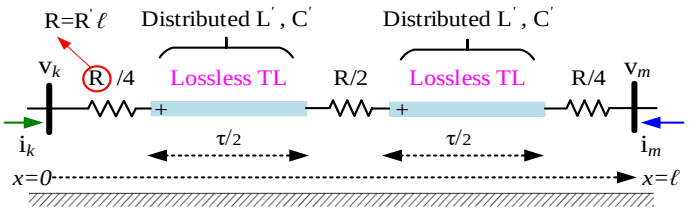


Figure 3.10 Schematic of single-phase CP-line model

In this figure, the prime in R′, L′ and C′ is used to indicate distributed resistance, inductance, and capacitance, i.e., parameters per unit length. The model is accurate as long as R is small compared to the surge impedance. Now let's assume the line is constructed by an *N*-conductor. Modal transformation is used to formulate a multiconductor transmission line to produce diagonal matrices, thereby transforming from *N*-coupled equations in the phase domain to *N*-decoupled equations in the modal domain (mode). The computation of modal matrix for the transposed transmission line is simple, i.e., Clark's transformation matrix, but Eigenvalue analysis is required for the untransposed line. Generally, the transformation matrices for current and voltage are dependent on frequency, and the matrix elements are complex. For the CP-line model, the modal transformation matrix is calculated at a given frequency. The real part of the modal transmission matrix is used for time-domain computations. Each equation is solved for a single-phase line in the modal domain by using modal traveling time and modal surge impedance. The relation between the phase and modal variables are defined as:

$$\mathbf{v}_{phase} = \mathbf{T}_v \mathbf{v}_{mode} \tag{3-39}$$

$$\mathbf{i}_{phase} = \mathbf{T}_i \mathbf{i}_{mode}. \tag{3-40}$$

where $\mathbf{v}_{phase}$and $\mathbf{i}_{phase}$ are the vectors of voltage and current in phase domain, $\mathbf{v}_{mode}$ and $\mathbf{i}_{mode}$ are the same vectors in modal domain. $\mathbf{T}_v$, $\mathbf{T}_i$ are the *N*-by-*N* matrices of model transformation where $\mathbf{T}_i = [\mathbf{T}_v^t]^{-1}$ and *t* indicates the transposition. Applying the technique to the equation (3-34) yields the following modal scalar equations for the *k*-end

$$\mathrm{v}_{k,mode} = \mathrm{Z}_{mdf,mode}(\mathrm{i}_{k,mode} + \mathrm{i}_{k,mode}^{hist}) \tag{3-41}$$

$$\begin{aligned} \mathrm{i}_{k,mode}^{hist}(t) \quad &= +\mathrm{k}_{v1}\mathrm{v}_{k,mode}(t-\tau_{mode}) - \mathrm{k}_{i1}\mathrm{i}_{k,mode}^{hist}(t-\tau_{mode}) \\ &\quad +\mathrm{k}_{v2}\mathrm{v}_{m,mode}(t-\tau_{mode}) - \mathrm{k}_{i2}\mathrm{i}_{m,mode}^{hist}(t-\tau_{mode}) \end{aligned} \tag{3-42}$$

knowing that:

$$\mathrm{k}_{v1} = \frac{1-\mathrm{h}_{mode}}{2}\frac{1+\mathrm{h}_{mode}}{\mathrm{Z}_{mdf,mode}} \tag{3-43}$$

$$\mathrm{k}_{v2} = \frac{1+\mathrm{h}_{mode}}{2}\frac{1+\mathrm{h}_{mode}}{\mathrm{Z}_{mdf,mode}} \tag{3-44}$$

$$k_{i1} = \frac{1 - h_{mode}}{2} h_{mode} \tag{3-45}$$

$$k_{i2} = \frac{1 + h_{mode}}{2} h_{mode} \tag{3-46}$$

$$h_{mode} = \frac{Z_{c,mode} - R_{mode}/4}{Z_{c,mode} + R_{mode}/4} \tag{3-47}$$

$$\tau_{mode} = \ell \sqrt{L'_{mode} . C'_{mode}} \tag{3-48}$$

$$Z_{mdf,mode} = Z_{c,mode} + \frac{R_{mode}}{4}. \tag{3-49}$$

$$Z_{c,mode} = \sqrt{\frac{L'_{mode}}{C'_{mode}}} \tag{3-50}$$

In the above equations, $Z_{mdf,mode}$ is the modified surge impedance, $Z_{c,mode}$ the modal surge impedance for lossless TL, L'_{mode} and C'_{mode} respectively represent the modal inductance and capacitance of TL in per unit length, R_{mode} is the modal resistance of TL and $i^{hist}_{k,mode}(t)$ denotes the modal history current at the *k*-end. τ_{mode} represents the modal traveling time from one end (*k*) to the other end (*m*). Replacing the index *k* to *m* gives the same equations at the *m*-end of TL.

Figure 3.11 shows the time-domain model of an *N*-phase transmission line. As one can see, there is no direct connection between the two terminals, and the voltage and current at one end are seen indirectly, and with time delays, τ_{mode}, at the other through the current sources. The history terms are stored in a *ring buffer*, and hence the maximum traveling time that can be represented is the time-step multiplied by the number of locations in the buffer. Because the time delay is not a multiple of the time-step, the history terms on either side of the actual traveling time are interpolated to give the correct traveling time.

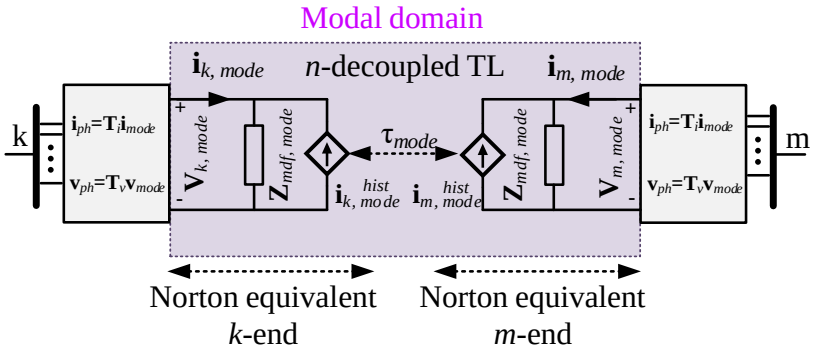


Figure 3.11 Multiconductor transmission line model as two Norton equivalents

3.3.3.2 Implementation in Modelica

The input parameters of the CP-line model are the equivalent modal values of characteristic impedance (`Zc`) and propagation delay (`tau`), modal transformation (`Ti`) for untransposed line, and the length of line (`d`). Figure 3.12 represents the Norton equivalent of CP-line model in Modelica. To decrease the computational burden, only history currents are computed in modal domain, then other calculations are done in the phase domain. Figure 3.13 presents the implementation of the model in Modelica. The codes consist of calculations for Norton equivalent and history currents, as shown in Figure 3.12.

- *Norton equivalent:* the related codes are distinguished by the blue dotted outline and aimed to calculate phase domain voltage and current vectors at each end of the line. The voltage and current at *k*- and *m*- ends are denoted by `Plug_k.pin.v`, `Plug_m.pin.v`, `Plug_k.pin.i`, `Plug_m.pin.i` respectively. The Norton equivalent equations for *k*- and *m*- ends in phase domain are obtained by applying the inverse modal transformation to (3-41):

$$\mathbf{v}_k = \mathbf{Z}_{mdf,phase}(\mathbf{i}_k + \mathbf{i}_k^{hist}) \tag{3-51}$$

$$\mathbf{v}_m = \mathbf{Z}_{mdf,phase}(\mathbf{i}_m + \mathbf{i}_m^{hist}) \tag{3-52}$$

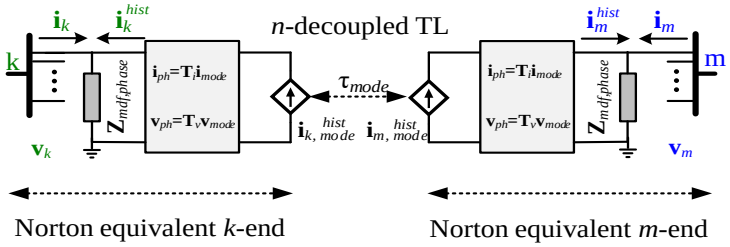


Figure 3.12: Norton equivalent of *N*-conductor CP-line in Modelica

The $N \times N$ Norton resistance matrix, $\mathbf{Z}_{mdf,phase}$, is calculated by (3-53) in phase domain.

$$\mathbf{Z}_{mdf,phase} = \mathbf{T}_i^{-1}\mathbf{Z}_{mdf,mode}\mathbf{T}_i \tag{3-53}$$

$\mathbf{Z}_{mdf,mode}$is a diagonal matrix involving the contribution of line resistance to the characteristic impedance of lossless line; both in modal domain.

- *History Current:* The code for calculating history currents is distinguished by the red dotted outline in Figure 3.13. The history current, `Ik_hist,` and `Im_hist` are computed in the modal domain, then transformed into phase domain. It is noteworthy that the calculation of history terms includes solving an implicit algebraic equation at each time point [107]. In the pieces of code, the Modelica built-in function *delay()*, computes the voltages and history currents by interpolating linearly in a buffer containing past values of these variables.

```
model TL_CP
  parameter Integer m(min = 1) = 3 "Number of phases" ;
  parameter Real Zc[m] "Characteristic impedance{Zc1,Zc2,...,Zcm} in mode" ;
  parameter Real r[m]( each unit = "ohm/km") "{r1,r2,...,rm} in mode" ;
  parameter MSEMT.NonElectrical.Units.Length d( displayUnit="km") "length of line" ;
  parameter MSEMT.NonElectrical.Units.Time tau[m] "tau ={tau1,tau2,...,taum} in mode" ;
  parameter Real Ti[m,m]=MSEMT.NonElectrical.Functions.Clark_Transformation(m);
  //Final Paramters
  final parameter Real R[m]=r*d/1000 ;
  final parameter Real h[m]=(Zc.-R./4)./(Zc.+R./4) ;
  final parameter Real Zmod[m]=(Zc.+R./4) ;
  final parameter Real Zmdf_phase[m,m]=inv(transpose(Ti)) * diagonal(Zmod) * inv(Ti);
  final parameter Real kv1[m] = +((1 .- h)/2).*((1 .+ h)./Zmod) ;
  final parameter Real kv2[m] = +((1 .+ h)/2).*((1 .+ h)./Zmod) ;
  final parameter Real ki1[m] = -((1 .- h)/2).*h ;
  final parameter Real ki2[m] = -((1 .+ h)/2).*h ;
  final parameter Real Tit[m, m] = transpose(Ti) "Ti transposed" ;
  MSEMT.Connectors.PosPlug Plug_k(m = m)
  MSEMT.Connectors.negPlug Plug_m(m = m)
  Real Ik_hist[m],Im_hist[m]; // History current for k-,m-end
  Real Vk_md[m],Vm_md[m];     // Modal Terminal voltage for k-,m-end
  Real dvk_md[m],dvm_md[m];   // Delayed Modal Terminal voltage for k-,m-end
  Real dIk_hist_md[m],dIm_hist_md[m]; // Delayed Modal History current for k-,m-end
  Real Ik_hist_md[m],Im_hist_md[m]; // Modal History current for k-,m-end
equation
// Calculation of Terminal voltage
  Plug_k.pin.v = Zmdf_phase * (Plug_k.pin.i + Ik_hist);
  Plug_m.pin.v = Zmdf_phase * (Plug_m.pin.i + Im_hist);

// Calculation of History current
  for i in 1:m loop
    if time < tau[i] then
      dvk_md[i]  = 0;
      dvm_md[i]  = 0;
      dIk_hist_md[i] = 0;
      dIm_hist_md[i] = 0;
    else
      dvk_md[i]  = delay(Vk_md[i],  tau[i]);
      dvm_md[i]  = delay(Vm_md[i],  tau[i]);
      dIk_hist_md[i] = delay(Ik_hist_md[i], tau[i]);
      dIm_hist_md[i] = delay(Im_hist_md[i], tau[i]);
    end if;
  end for;
  Vk_md = Tit * Plug_k.pin.v;
  Vm_md = Tit * Plug_m.pin.v;
  Ik_hist_md = kv1 .* dvk_md + ki1 .* dIk_hist_md + kv2 .* dvm_md + ki2 .* dIm_hist_md;
  Im_hist_md = kv1 .* dvm_md + ki1 .* dIm_hist_md + kv2 .* dvk_md + ki2 .* dIk_hist_md;
  Ik_hist = Ti  * Ik_hist_md;
  Im_hist = Ti  * Im_hist_md;
end TL_CP;
```

Figure 3.13 Implementation of multiphase CP-line model in Modelica

3.3.4 Wideband Line Model

The WB-line model [99][100] is a phase domain model that gives highly accurate results for aerial lines and underground cables. It considers the full frequency dependency of line/cable parameters.

3.3.4.1 Formulation and theoretical aspects

Suppose the impedance and admittance of the transmission line are functions of frequency; therefore, the time-domain solutions of equation (3-19)-(3-21) are obtained by applying the inverse frequency-domain transformation.

$$\mathbf{i}_k = \mathbf{y}_c * \mathbf{v}_k - 2\mathbf{i}_{ki} \tag{3-54}$$

$$\mathbf{i}_{ki} = \mathbf{h} * \mathbf{i}_{mr} \tag{3-55}$$

$$\mathbf{i}_{mr} = \mathbf{i}_{mi} + \mathbf{i}_m \tag{3-56}$$

In these equations, the symbol $*$ indicates convolution. Like *k*-end, we can write the equations for *m*-end as:

$$\mathbf{i}_m = \mathbf{y}_c * \mathbf{v}_m - 2\mathbf{i}_{mi} \tag{3-57}$$

$$\mathbf{i}_{mi} = \mathbf{h} * \mathbf{i}_{kr} \tag{3-58}$$

$$\mathbf{i}_{kr} = \mathbf{i}_{ki} + \mathbf{i}_k \tag{3-59}$$

Numerical calculation of convolution creates a significant computational burden since it accounts for all history values. The most cost-effective technique is to use the *recursive convolution algorithm*. In this approach, the propagation and the characteristic admittance matrices are directly fitted in phase domain using the vector fitting tool [99]. The approximation of these two matrices in a *partial fraction* form is given by (3-60) and (3-61).

$$\mathbf{Y}_c = \mathbf{G}_0 + \sum_{i=1}^{N_y} \frac{\mathbf{G}_i}{s - q_i} \tag{3-60}$$

$$\mathbf{H} = \sum_{k=1}^{N_g} \sum_{i=1}^{N_h(k)} \frac{\mathbf{R}_{k,i}}{s - p_{k,i}} e^{-s\tau_k} \tag{3-61}$$

where $\mathbf{G}_0$ is a constant residue at the infinite frequency, q_i represents i^{th} pole, $\mathbf{G}_i$ is the corresponding matrix of residues, N_y is the order of fitting, N_g is the number of modes, $N_h(k)$ denotes the number of poles used to fit the k^{th} modal propagation matrix, $p_{k,i}$ is the fitting pole, τ_k is the time delay of the kth mode and $\mathbf{R}_{k,i}$ is the matrix of residues. The state-space form of shunt and incident current for the *k*-end are represented by (3-64) and (3-65). The same equations hold for the *m*-end of the line.

$$\mathbf{I}_{sh,k} = \mathbf{G}_0\mathbf{V}_k + \sum_{i=1}^{N_y} \mathbf{W}_i \tag{3-62}$$

$$\mathbf{W}_i = \frac{\mathbf{G}_i}{s - q_i}\mathbf{V}_k \tag{3-63}$$

and

$$\mathbf{I}_{ki} = \sum_{k=1}^{N_g} \sum_{i=1}^{N_h(k)} \mathbf{X}_{k,i} \tag{3-64}$$

$$\mathbf{X}_{k,i} = \frac{\mathbf{R}_{k,i}}{s - p_{k,i}}\mathbf{I}_{mr}e^{-s\tau_k} \tag{3-65}$$

Therefore, the time-domain solution of (3-64)-(3-65) can be evaluated by applying a fast-recursive algorithm to state space methods.

$$\mathbf{i}_{sh,k} = \mathbf{G}_0\mathbf{v}_k + \sum_{i=1}^{N_y} \mathbf{W}_i \tag{3-66}$$

$$\frac{d\mathbf{w}_i}{dt} = q_i\mathbf{w}_i + \mathbf{G}_i\mathbf{v}_k \tag{3-67}$$

and

$$\mathbf{i}_{ki} = \sum_{k=1}^{N_g} \sum_{i=1}^{N_h(k)} \mathbf{x}_{k,i} \tag{3-68}$$

$$\frac{d\mathbf{x}_{k,i}}{dt} = p_{k,i}\mathbf{x}_{k,i} + \mathbf{R}_{k,i}\mathbf{i}_{mr}(t - \tau_k) \tag{3-69}$$

In EMT-type programs, (3-67) and (3-69) need to be discretized using the trapezoidal integration method. This work sets focus on the evaluation of these equations in time domain using declarative language.

3.3.4.2 Implementation of WB-line model

Figure 3.14 represents the Norton equivalent of WB-line model, which is used for configuration of the model in Modelica. Figure 3.15 illustrates the implementation of the WB-line model in Modelica. The model is composed of two pieces of code. (1) to compute the voltage and current at line terminal, which is distinguished by red dotted frame, and shunt current using (3-66) and (3-67), which is distinguished by blue dotted frame. (2) the codes for calculations of the incident and reflected currents as formulated by (3-68) and (3-69). The yellow dotted frame shows the appropriate code.

The fitting parameters of model are currently calculated by EMTP® and are imported automatically into Modelica readable file.

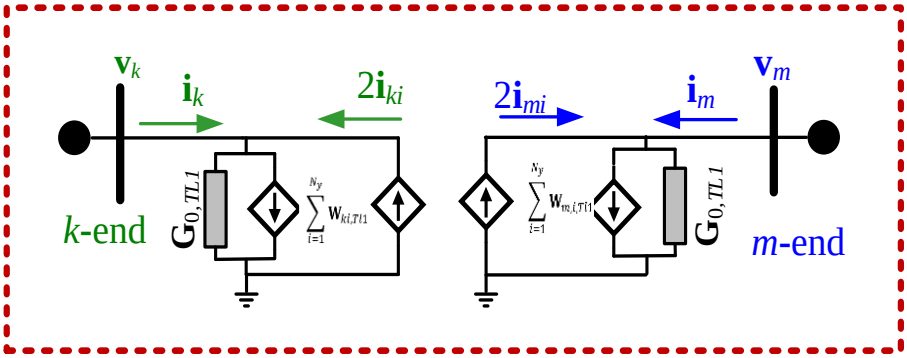


Figure 3.14 Norton equivalent of WB-line model

```
model WideBand
  import MSEMT.NonElectrical.Functions.ComplexMath.real;
  import MSEMT.NonElectrical.Functions.ComplexMath.imag;
  constant Complex j = Complex(0,1);
  parameter Integer m( final min=1) = 3  "Number of phases" ;
  parameter Real G0[:,:] "G0 is a constant matrixi_fitting Yc";
  parameter Real G[:,:,:] "Zeros of fitting Yc" ;
  parameter Real q[:] "poles of fitting Yc" ;
  parameter Real tau[:]  "propagation time" ;
  parameter Complex Pm_H[:,:] "poles of fitting H";
  parameter Complex Rm_H[:,:,:,:] "zeros of fitting H";
  final parameter Integer Ny = size(G, 1) "Order of fitting of Yc" ;
  final parameter Integer Ng=size(Rm_H,1)    "Number of groups" ;
  final parameter Integer No_H=size(Rm_H,2)  "Order of Fitting" ;
// Definition of Pk and Pm terminals
  MSEMT.Connectors.PosPlug Pk(m = m) ;
  MSEMT.Connectors.PosPlug Pm(m = m) ;
// varaibles of shunt current k-side
  Real wk[m, Ny](start=zeros(m,Ny),each fixed=false) ;
  Real sum_wk[m] ;
  Real i_shk[m]  ;
// varaibles of shunt current m-side
  Real wm[m, Ny](start=zeros(m,Ny),each fixed=false) ;
  Real sum_wm[m] ;
  Real i_shm[m] ;
// incident current
 Real i_ki[m],i_mi[m];
 Real i_kr[m] ; // i_kr:  reflected current-wave vector from k-end
 Real i_mr[m] ; // i_mr:  reflected  current-wave vector from m-end
 Real xk_Re[Ng, No_H, m](start=zeros(Ng, No_H, m),each fixed=true) ;  // States convolution of propagation function xR: Real
Part, k-end
 Real xk_Im[Ng, No_H, m](start=zeros(Ng, No_H, m),each fixed=true) ;
// States convolution of propagation function  xI: Imaginary Part, k-end
 Real xm_Re[Ng, No_H, m](start=zeros(Ng, No_H, m),each fixed=true) ;  // States convolution of propagation function xR: Real
Part, m-end
 Real xm_Im[Ng, No_H, m](start=zeros(Ng, No_H, m),each fixed=true) ;
// States convolution of propagation function  xI: Imaginary Part, m-end
 Real i_hkr[m, Ng]   // i_hkr: i_kr with delay
 Real i_hmr[m, Ng]   // i_hmr: i_mr with delay
equation
// Calculation of Termian current
  Pk.pin.i = i_shk - 2 * i_ki;
  Pm.pin.i = i_shm - 2 * i_mi;
  for p in 1:Ny loop
    for k in 1:m loop
      der(wk[k, p]) = q[p] * wk[k, p] + G[p, k, :] * Pk.pin.v;
      der(wm[k, p]) = q[p] * wm[k, p] + G[p, k, :] * Pm.pin.v;
    end for;
  end for;
// sum of all columns for each phase
  for k in 1:m loop
    sum_wk[k] = sum(wk[k, :]);
    sum_wm[k] = sum(wm[k, :]);
  end for;
  i_shk = G0 * Pk.pin.v + sum_wk; // Calculation of shunt current, k-end
  i_shm = G0 * Pm.pin.v + sum_wm; // Calculation of shunt current, m-end
// Calculation of Incident Current
 i_kr = i_ki + Pk.pin.i;
 i_mr = i_mi + Pm.pin.i;
  for k in 1:Ng loop
    if time<tau[k] then
      i_hmr[:, k] =zeros(m);
      i_hkr[:, k] =zeros(m);
      else
      i_hmr[:, k] = delay(i_mr, tau[k]);
      i_hkr[:, k] = delay(i_kr, tau[k]);
    end if;
    for p in 1:No_H loop
      for i in 1:m loop
// calc of History cuurent k-end
    der(xk_Re[k, p, i])=real(Pm_H[k, p])*xk_Re[k, p, i]-imag(Pm_H[k, p])*xk_Im[k, p, i]+real(Rm_H[k, p, i, :])* i_hmr[:, k];
    der(xk_Im[k, p, i])=imag(Pm_H[k, p])*xk_Re[k, p, i]+real(Pm_H[k, p])*xk_Im[k, p, i]+imag(Rm_H[k, p, i, :])* i_hmr[:, k];
// calc of History cuurent m-end
    der(xm_Re[k, p, i])=real(Pm_H[k, p])*xm_Re[k, p, i]-imag(Pm_H[k, p])*xm_Im[k, p, i]+real(Rm_H[k, p, i, :])* i_hkr[:, k];
    der(xm_Im[k, p, i])=imag(Pm_H[k, p])*xm_Re[k, p, i]+real(Pm_H[k, p])*xm_Im[k, p, i]+imag(Rm_H[k, p, i, :])* i_hkr[:, k];
      end for;
    end for;
  end for;
//summation of fitting orders for each phase
  for i in 1:m loop
    i_ki[i] = sum(xk_Re[:, :, i]);
    i_mi[i] = sum(xm_Re[:, :, i]);
  end for;
end WideBand;
```

Figure 3.15 Codes for implementation of WB-line model

3.4 Load Models

The three-phase parallel or series RL load implements a three-phase balanced or unbalanced load as a parallel or series combination of RL elements. At the specified frequency, the load exhibits a constant impedance.

As one can see in Figure 3.16, the parameters of the model are the load nominal line-line voltage (`V`) in kV, the load active and reactive power (`P[3]={Pa, Pb, Pc}` in MW and `Q[3]={Qa, Qb, Qc}` in Mvar respectively) and frequency (`f`) in Hz.

```
model PQLoad "Three-phase parallel/series Yg-connected PQ load "
  import MSEMT.NonElectrical.Constants.pi;
  Boolean ParallelConfig=true;
  parameter Real V(unit = "kV RMSLL") = 25 "Nominal Voltage";
  parameter Real P[3](each unit = "MW") "Active powers {Pa,Pb,Pc}";
  parameter Real Q[3](each unit = "MAVR") "Rective powers {Qa,Qb,Qc}";
  parameter Modelica.SIunits.Frequency f = 60 "Nominal frequency";
protected
 Real R1 = if ParallelConfig then V ^ 2 / 3 / P[1] else  V ^ 2 *P[1]/ 3 / (P[1]^2+Q[1]^2);
 Real R2 = if ParallelConfig then V ^ 2 / 3 / P[2] else  V ^ 2 *P[2]/ 3 / (P[2]^2+Q[2]^2);
 Real R3 = if ParallelConfig then V ^ 2 / 3 / P[3] else  V ^ 2 *P[3]/ 3 / (P[3]^2+Q[3]^2);
 Real L1 = if ParallelConfig then V ^ 2 / 3 / (2 * pi * f * Q[1]) else V ^ 2 *Q[1]/ 3 / (P[1]^2+Q[1]^2)/(2 * pi * f);
 Real L2 = if ParallelConfig then V ^ 2 / 3 / (2 * pi * f * Q[2]) else V ^ 2 *Q[2]/ 3 / (P[2]^2+Q[2]^2)/(2 * pi * f);
 Real L3 = if ParallelConfig then V ^ 2 / 3 / (2 * pi * f * Q[3]) else V ^ 2 *Q[3]/ 3 / (P[3]^2+Q[3]^2)/(2 * pi * f);
 MSEMT.Connectors.PosPlug Pk;
 Real IL[3],IR[3]; // Inductor and resistor current
 Real VL[3],VR[3]; // Inductor and resistor voltage
equation
 {L1,L2,L3}.* der(IL) =VL ;
 {R1,R2,R3}.* IR = VR;
if ParallelConfig then
 Pk.pin.i=IL.+IR;
 Pk.pin.v=VR;
 Pk.pin.v=VL;
 else
 Pk.pin.v=VR+VL;
 Pk.pin.i=IL;
 Pk.pin.i=IR;
 end if;
end PQLoad;
```

Figure 3.16 Implementation of PQ load in Modelica

Similar models for three/single-phase capacitive loads are available in the MSEMT library.

3.5 Synchronous Machine

This section implements the classical dq model of a balanced wye-grounded synchronous machine (SM). Figure 3.17 shows a two-pole SM in which damper winding effects are represented with three damper windings: one on the *d*-axis, *kd*, and two on the *q*-axis, *kq1*, and *kq2*. The *q*-axis is assumed to be leading the *d*-axis by 90 deg, and the direction of the positive stator current is out of the terminals [101].

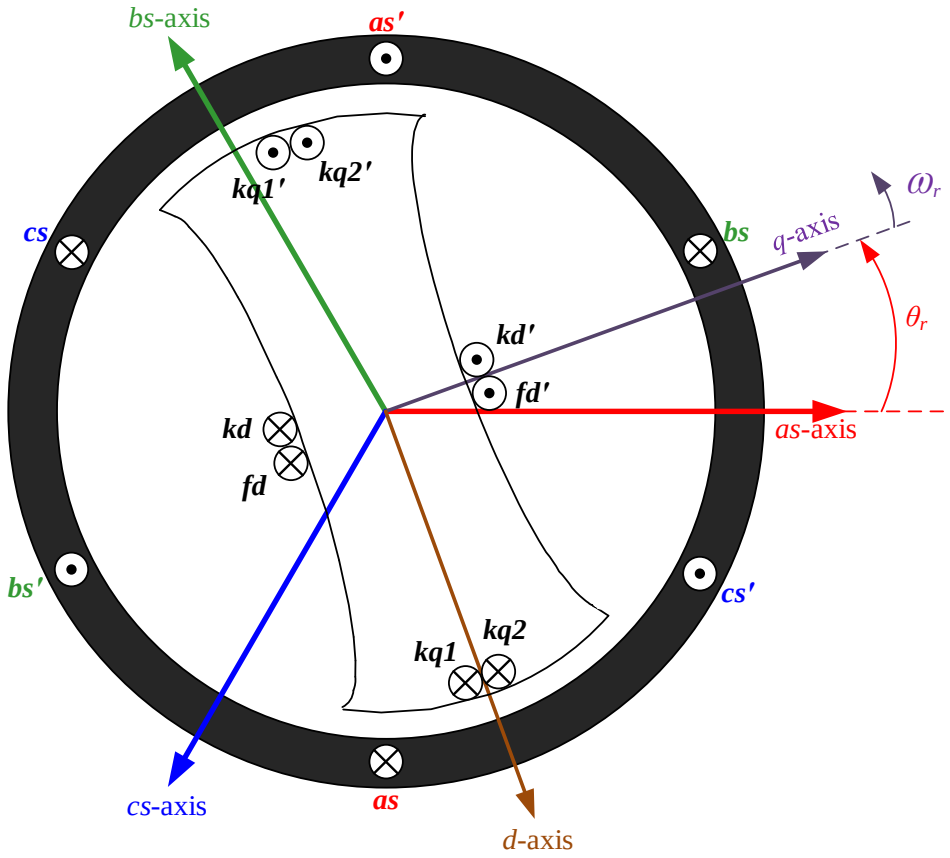


Figure 3.17 Two-pole, three-phase, wye-connected salient-pole synchronous machine

The sixth-order state-space model and the mechanical equations based on a single mass sorted from (3-70) to (3-87) are implemented. In these equations, bold uppercase represents matrices, bold lowercase denotes vector, and operator p is d/dt. The per-unit (pu) electrical equations are expressed in the rotor reference-frame as following:

$$\mathbf{v}_{dq0} = \mathbf{P}(\theta)\mathbf{v}_{abc} \tag{3-70}$$

$$p\boldsymbol{\psi} = \omega_b(\mathbf{A}\boldsymbol{\psi} + \mathbf{u}) \tag{3-71}$$

$$\mathbf{A} = -(\mathbf{R}\mathbf{L}^{-1} + \mathbf{W}) \tag{3-72}$$

$$\mathbf{i} = \mathbf{L}^{-1}\boldsymbol{\psi} \tag{3-73}$$

$$\mathbf{i}_{abc} = \mathbf{P}^{-1}(\theta)\mathbf{i}_{dqz} \tag{3-74}$$

$$\mathbf{i}_{abc,actual} = \mathbf{i}_{abc}.\,\mathrm{I}_{stator,base} \tag{3-75}$$

The mechanical equations in the per-unit form that describes the dynamics of a single mass rotor are given as:

$$\mathrm{T}_e = \psi_d \mathrm{i}_q - \psi_q \mathrm{i}_d \tag{3-76}$$

$$\mathrm{T}_{net} = \mathrm{T}_m - \mathrm{T}_e - \mathrm{D}\Delta\omega_r \tag{3-77}$$

$$\mathrm{T}_m = \mathrm{P}_m/\omega_r \tag{3-78}$$

$$p\Delta\omega = \mathrm{T}_{net}\frac{1}{2\mathrm{H}} \tag{3-79}$$

$$\omega_r = 1 + \Delta\omega \tag{3-80}$$

$$p\Delta\theta = \omega_b \Delta\omega \tag{3-81}$$

$$\theta = \Delta\theta + \omega_b t \tag{3-82}$$

where

$$\mathbf{u} = \left[\mathrm{v}_q, \mathrm{v}_d, \mathrm{v}_{fd}, 0, 0, 0\right]^T \tag{3-83}$$

$$\boldsymbol{\psi} = \left[\psi_q, \psi_d, \psi_{fd}, \psi_{kd}, \psi_{kq1}, \psi_{kq2}\right]^T \tag{3-84}$$

$$\mathbf{i} = \left[\mathrm{i}_q, \mathrm{i}_d, \mathrm{i}_{fd}, \mathrm{i}_{kd}, \mathrm{i}_{kq1}, \mathrm{i}_{kq2}\right]^{\boldsymbol{T}} \tag{3-85}$$

$$\mathbf{i}_{dq0} = \left[-\mathrm{i}_q, -\mathrm{i}_d, 0\right]^{\boldsymbol{T}} \tag{3-86}$$

$$\mathbf{R} = \mathrm{diag}\left(\mathrm{R}_a, \mathrm{R}_a, \mathrm{R}_{fd}, \mathrm{R}_{kd}, \mathrm{R}_{kq1}, \mathrm{R}_{kq2}\right) \tag{3-87}$$

The vector $\mathbf{v}_{abc}$ is the pin voltage, $\mathbf{v}_{dq0}$ is the pin voltage in *dq* frame, $\mathbf{P}(\theta)$ is the Park transformation defined in a specific class function and reused here, vectors **u**, **i**, and **ψ** denote stator and rotor voltages, currents, and flux linkages in *dq* frame and $\mathbf{i}_{abc}$ is the stator current. $\mathbf{W}_{6\times6}$ is the rotor speed-dependent matrix where all elements are zero except $w(1,2) = \omega_r$ and $w(2,1) = -\omega_r$, $\mathbf{L}_{6\times6}$ is a symmetrical matrix of self and mutual inductances in the rotor reference frame, $\mathbf{R}_{6\times6}$ is stator and rotor resistance matrix, Δω is rotor speed deviation in pu, Δθ denotes rotor angle deviation in pu, θ is electrical rotor angle in radians, ω_b is base rotor speed in radians per second and H represents inertia constant in s. The mechanical, electrical torque and the damping factor in pu are denoted respectively by T_m, T_e and D.

3.5.1 Magnetic Saturation

The following assumptions are made for magnetic saturation modeling: The leakage flux saturation and cross saturation are ignored. It means that only the magnetizing inductances, L_{md} and L_{mq} are saturable. The air-gap flux linkage determines magnetic saturation. The sinusoidal distribution of the magnetic field on the pole face is not affected by magnetic saturation.

Since the saturation relationship between the total air-gap flux, $\boldsymbol{\psi}_T$, and the magnetomotive force under loaded conditions is supposed to be the same as at no-load conditions; thus, magnetic saturation of stator and rotor can be modeled by the *no-load saturation curve*, which is characterized by a *piecewise linear* function [102].

Therefore, the mathematical model of saturation is given by [102]:

$$\psi_T = f\left(\psi_{T,us}\right) = f\left(\sqrt{\psi_{md,us}^2 + \psi_{mq,us}^2}\right) \tag{3-88}$$

$$\psi_{md,us} = L_{md,us} i_{md} \tag{3-89}$$

$$i_{md} = i_d + i_{fd} + i_{kd} \tag{3-90}$$

$$\psi_{mq,us} = L_{mq,us} i_{mq} \tag{3-91}$$

$$i_{mq} = i_q + i_{kq1} + i_{kq2} \tag{3-92}$$

where $\psi_{T,us}$ is the total unsaturated air-gap flux, $\psi_{md,us}$ and $\psi_{mq,us}$ are the unsaturated magnetizing flux linkages respectively, $L_{md,us}$ and $L_{mq,us}$ are the unsaturated magnetizing inductances, and i_{md} and i_{mq} are the magnetizing currents. Throughout the thesis, the subscripts *sat* and *us* mean saturated and unsaturated, respectively.

As illustrated in Figure 3.18, the saturated magnetizing flux linkages on *dq* axis ($\psi_{md,sat}$ and $\psi_{md,sat}$) can be adjusted by a ratio of corresponding unsaturated values.

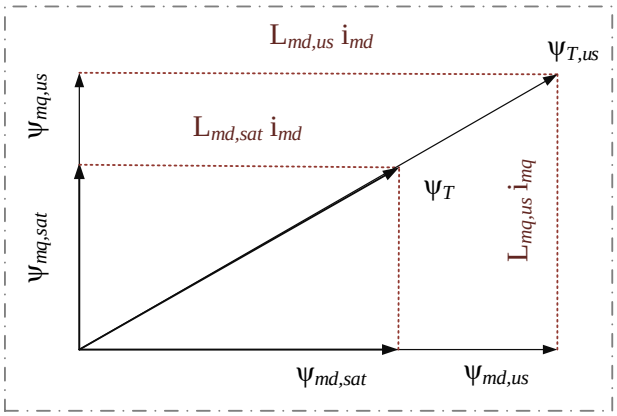


Figure 3.18 Saturated and unsaturated magnetizing flux linkages in *dq* axes of a synchronous machine

In EMTP®, a piecewise-linear function is used for representation of the magnetic saturation as shown in Figure 3.19.

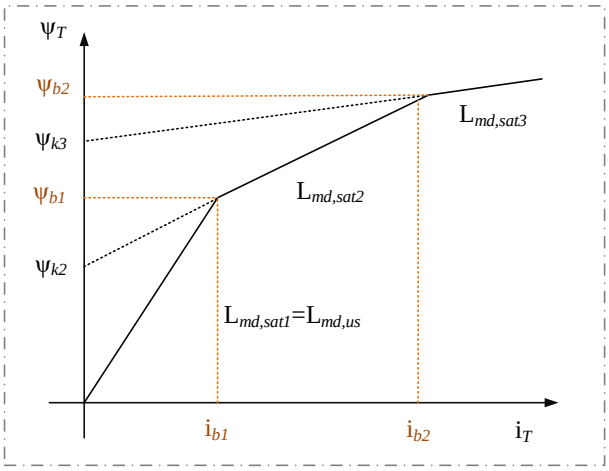


Figure 3.19 Magnetic saturation characteristic (piecewise-linear approximation)

For the j^{th} operating segment, ψ_T is given by:

$$\psi_T = \psi_{kj} + b_j \psi_{Tu} \tag{3-93}$$

$$= \psi_{kj} + b_j L_{md,us} i_T \tag{3-94}$$

$$i_T = \sqrt{i_{md}^2 + \left(\frac{L_{mq,us}}{L_{md,us}}\right)^2 i_{mq}^2} \tag{3-95}$$

$$b_j = \frac{L_{md,satj}}{L_{md,us}} \tag{3-96}$$

where b_j is the saturation factor and ψ_{kj} is the residual flux. The saturated values $L_{md,sat}$ and $L_{mq,sat}$ are computed as:

$$\begin{aligned} L_{md,sat} &= b_j L_{md,us} \\ L_{mq,sat} &= b_j L_{mq,us} \end{aligned} \tag{3-97}$$

For a *salient pole* machine, because of large airgap path along the *q*-axis, it is only required to correct the ψ_{md}; thus:

$$\begin{aligned} L_{md,sat} &= b_j L_{md,us} \\ L_{mq,sat} &= L_{mq,us} \end{aligned} \tag{3-98}$$

Figure 3.20 demonstrates the solution procedure for the electrical equations of SM. In the case of no saturation, the relationship between field current, i_{fd} and terminal voltage v_t, is linear. Therefore, magnetizing inductances in matrix **L** (equation (3-99) are constant ($q_j = d_j = 1$). If saturation is selected, it is required to compute the magnetizing inductances for each time point. The matrix **L** is time-variable ($q_j = d_j = b_j$ for round rotor and $q_j = 0,\ d_j = b_j$ for salient pole machine).

$$\mathbf{L} = \begin{pmatrix} L_{ls} + q_j L_{mq,us} & 0 & 0 & 0 & q_j L_{mq,us} & q_j L_{mq,us} \\ 0 & L_{ls} + d_j L_{md,us} & d_j L_{md,us} & d_j L_{md,us} & 0 & 0 \\ 0 & d_j L_{md,us} & L_{lfd} + d_j L_{md,us} & d_j L_{md,us} & 0 & 0 \\ 0 & d_j L_{md,us} & d_j L_{md,us} & L_{lkd} + d_j L_{md,us} & 0 & 0 \\ q_j L_{mq,us} & 0 & 0 & 0 & L_{lkq1} + q_j L_{mq,us} & q_j L_{mq,us} \\ q_j L_{mq,us} & 0 & 0 & 0 & q_j L_{mq,us} & L_{lkq2} + q_j L_{mq,us} \end{pmatrix} \tag{3-99}$$

3.5.2 Implementation of Synchronous Machine Model in Modelica

Implementation of the SM model in Modelica is based directly on its equations without providing solution procedures, predictions, and supplementary codes as in traditional EMT-type tools. The *dq* model equations are linked to the main network equation with the interface of Park's transformation and provide a simultaneous solution. It is demonstrated that this method is numerically stable and yields the same results as EMTP®.

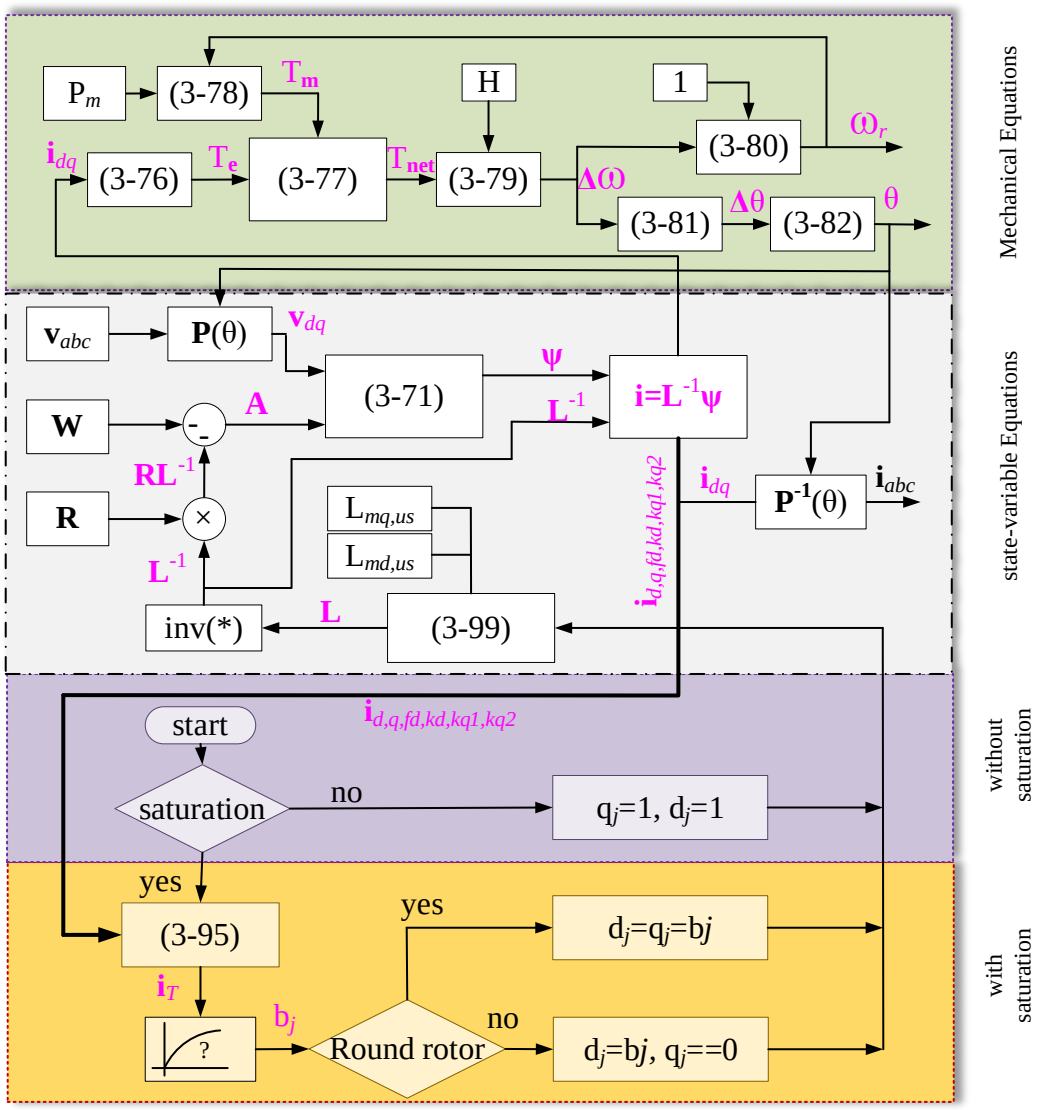


Figure 3.20 Solution procedure of synchronous machine with/without magnetic saturation in Modelica

Figure 3.21 shows the GUI of the synchronous machine model designed in Modelica language. The different parameters of the model are categorized according to their functionalities. Operational parameters are used as input parameters of the model; then, fundamental parameters are computed using the classical method [101] in the Modelica model. The model discussed above has been implemented for the first time in Modelica.

Figure 3.22 illustrates the Modelica codes for the implementation of SM. The terminal voltages are represented by `Pk.pin[1].v`, `Pk.pin[2].v` and `Pk.pin[3].v` for the phases *a*, *b* and *c*

respectively. `P(theta)` represents a pre-defined function for the Park's transformation calculations.

Equation (3-71) is used as a differential equation for the implemented model. The state vector `Phi` represents fluxes, and the input vector `u` the voltages. The system matrix `A` is a time-dependent matrix computed as per (3-72).

The matrix of parameters for representation of saturation, `SD`, is given by a 2-by-*n* matrix, where *n* is the number of points taken from the no-load saturation curve. The first row of this matrix contains the values of field currents (actual value), while the second row contains values of corresponding terminal voltages (per unit). `LinearInterplate(SD1PU, SD2PU, iT)` is a function coded to interpolate the `iT` by the two vectors of field current (`SD1PU`) and voltage (`SD2PU`) which both are calculated in the *non-reciprocal* per unit [103]. The function returns the total flux (`PhiT`) and `Lmdsat,` which is used to calculate coefficient `b` as per (3-96). The stator currents are represented by `Pk.pin[1].i`, `Pk.pin[2].i` and `Pk.pin[3].i` for the phases, *a*, *b* and *c*, respectively [75], [124].

The input pins for field voltage and output pin for field current are based on the non-reciprocal per unit. Figure 3.23 shows the relation between the reciprocal and non-reciprocal per-unit systems. The equations mentioned above are based on L_{md}-based reciprocal per unit system, i.e., the per-unit field current required to produce 1 pu terminal voltage in open-circuit test equals to X_{md}^{pu}. In non-reciprocal per unit 1 pu field current is required to generate 1 pu terminal voltage in the open-circuit test; therefore, the field base current equals to field no-load current.

For the initialization of SM, the initial flux and rotor angle are calculated from given steady-state values of terminal voltage and current. The steady-state voltage and currents at the terminal of SM can be read from EMTP®.

It is observed that the code structure is entered exactly as above equations without any further consideration of order, solution procedures, sequences, and other lower-level details. Moreover, it indicates that only a few code lines are needed to elaborate state-space equations through readily available Modelica functions and constructs. This is a drastic improvement and distinctive advantage over classical coding for performing similar tasks [75].

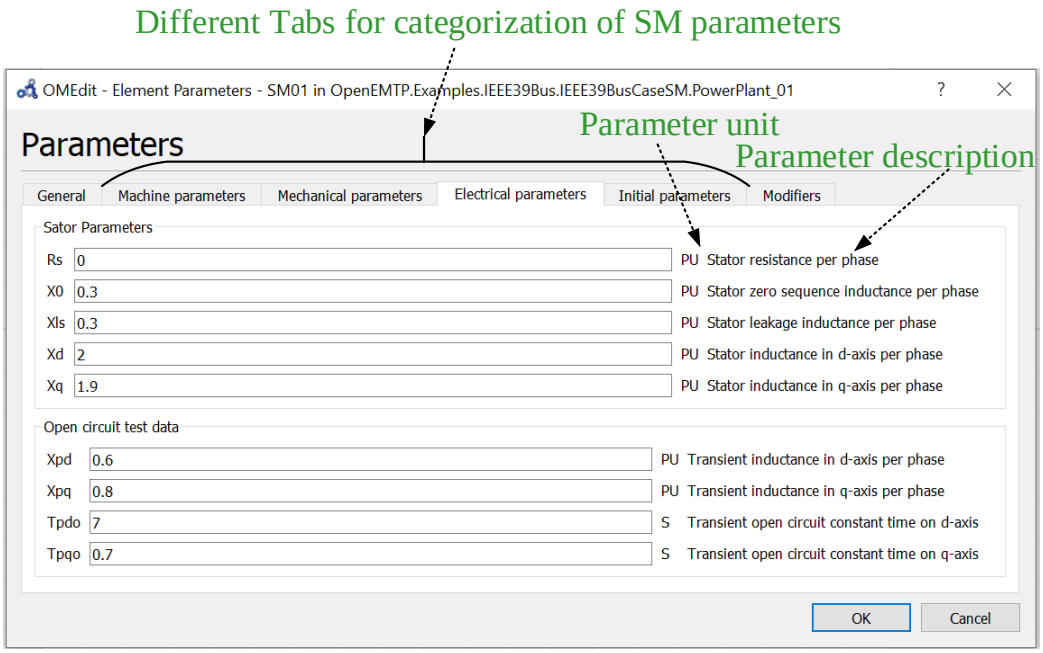


Figure 3.21 The GUI of the synchronous machine model implemented in Modelica

```
model SM "Synchronous Machine 6 order including saturation"
equation
// Conversion of terminal voltage to pu
 vabc= {Pk.pin[1].v,Pk.pin[2].v,Pk.pin[3].v} /Vsbase;
// Conversion from abc frame to dq0 frame
   vdq0  = P(theta)*vabc;
// State space electrical equations
   der(Phi)  = Wb * (A * Phi + u);
   A         = -(R * inv(L) + W);
   i         =  inv(L) * Phi;
// Implementation of magnetic saturation
   imd = Ip[2] + Ip[3] + Ip[4]; //imd = id + ifD + ikd
   imq = Ip[1] + Ip[5] + Ip[6]; //imq = iq + ikq1+ ikq2
   iT  = sqrt(imd^2 + (Lmqus/Lmdus)^2 * imq^2);
  (PhiT,Lmdsat) =  LinearInterpolation(SD1pu,SD2pu, iT);
   b  = Lmdsat / Lmdus;
   if Sauration then
     if RoundRotor then
       q=b;
       d=b;
     else
       q=0;
       d=b;
     end if;
   else
     q=1;
     d=1;
   end if;
    Lq      = Lls   + q * Lmqus;
    Ld      = Lls   + d * Lmdus;
    Lffd    = Llfd  + d * Lmdus;
    Lkdkd   = Llkd  + d * Lmdus;
    Lkq1kq1 = Llkq1 + q * Lmqus;
    Lkq2kq2 = Llkq2 + q * Lmqus;

   L= [  Lq    ,   0    ,    0   ,   0    ,q*Lmqus ,q*Lmqus ;
         0     , Ld     , d*Lmdus, d*Lmdus,   0    ,   0    ;
         0     , d*Lmdus, Lffd   , d*Lmdus,   0    ,   0    ;
         0     , d*Lmdus, d*Lmdus, Lkdkd  ,   0    ,   0    ;
       q*Lmqus ,   0    ,    0   ,   0    ,Lkq1kq1 , q*Lmqus;
       q*Lmqus ,   0    ,    0   ,   0    , q*Lmqus, Lkq2kq2];

 // Conversion from dq0 to abc frame
   iabc          =  inv(P(theta))* idq0;
// Calculations of actual Terminal current
   Pk.pin[1].i  = -iabc[1] * Isbase;
   Pk.pin[2].i  = -iabc[2] * Isbase;
   Pk.pin[3].i  = -iabc[3] * Isbase;
// Mechanical equations
   Te          = Phi[2] * idq0[1] - Phi[1] * idq0[2];
   Tnet        = Tm - Te - D * dw;
   Tm          = Pm_pu / Wr;
   der(dw)     = Tnet * (1 / 2 / H);
   Wr          = 1 + dw;
   der(d_theta)= dw * Wb;
   theta       = d_theta + Wb * time;
// where
// u = {Vq      , Vd      , Vfd   , Vkd   , Vkq1   , Vkq2 }
  u = {vdq0[1], vdq0[2], vfd , 0    , 0     ,  0   };
// Phi = {Phiq  , Phid  , Phifd , Phikd , Phikq1,Phikq2 }
  Phi = {Phi[1], Phi[2], Phi[3], Phi[4], Phi[5] , Phi[6]};
// i = {iq     , id     , ifd   , ikd   , ikq1   , ikq2  }
  i = {i[1]  , i[2]  , i[3] , i[4] , i[5] , i[6] };
// Change of sign due to generating mode
   idq0    = {-i[1], -i[2], 0};
   W[6, 6] = [ 0   , Wr  , 0  , 0  , 0  , 0  ;
              -Wr  , 0   , 0  , 0  , 0  , 0  ;
               0   , 0   , 0  , 0  , 0  , 0  ;
               0   , 0   , 0  , 0  , 0  , 0  ;
               0   , 0   , 0  , 0  , 0  , 0  ;
               0   , 0   , 0  , 0  , 0  , 0  ];
  R[6, 6] = diagonal({Rs, Rs, Rfd, Rkd, Rkq1, Rkq2});
end SM;
```

Figure 3.22 Synchronous machine Modelica codes

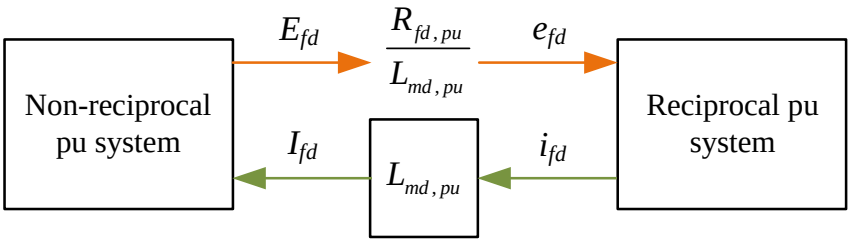


Figure 3.23 Relation between reciprocal and non-reciprocal per unit system

3.6 Nonlinear Component Models

Modeling of nonlinear components is fundamental in the analysis of electromagnetic transients. Phenomena, such as ferro-resonance, harmonic overvoltages, and inrush currents in transformers, require nonlinear inductance and arrester models. The Modelica language allows nonlinear components modeling without any topological or numerical restrictions. The is no limit to mathematically expressions of nonlinearity in polynomial, exponential, piece-wise linear functions, or a combination of them.

3.6.1 Nonlinear Inductor

The characteristics of the nonlinear inductor in EMT-type tools are expressed by monotonically increasing piecewise linear curves. A nonlinear inductor is mathematically defined as:

$$\varphi = f(\mathrm{i}) \tag{3-100}$$

$$\mathrm{v} = \frac{d\varphi}{dt} \tag{3-101}$$

where φ, v and i are the flux, voltage, and current of the nonlinear inductor, respectively, and f represents a piecewise linear function as depicted in Figure 3.24. The Modelica code describing a nonlinear inductor is shown in Figure 3.25.

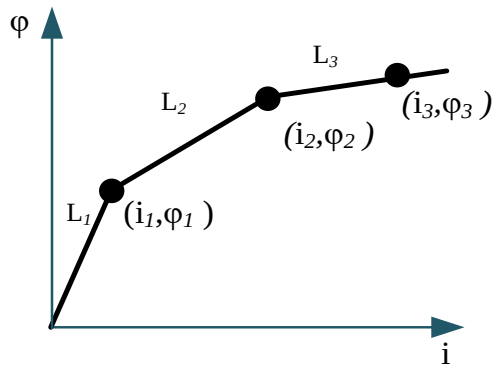


Figure 3.24 Piecewise linear representation of current-flux relation

In a similar procedure to the linear inductor model (see Section 2.6.12.6.1), it is first required to identify and declare the variables. The `import` statement is used to import the predefined mathematical functions, e.g., `sort`, `interpolate`, and physical type of `MagneticFlux`. These functions are available in the MSEMT library in the branch of `NonElectrical`. This spares the developer from having to describe things in the local model constantly. By contrast, the modeler can place definitions in packages and then recall those packages. The keyword `extends` is employed to specify inheritance from the pre-defined partial model `OnePort` into the model. The piecewise linear relationship of current versus flux is represented as a parameter of the model by a 2D array $\mathbf{T}_{N\times 2}$. it is possible to partition the curve in *n*-sections. Flux is a state variable in this model and is declared by the type `MagneticFlux`. Its initial value is set to zero by default, but it is possible to change the initial value in the GUI of the model (see Figure 3.26). The array elements are sorted and arranged in the `protected` section to define a symmetric variable (`i_vec and flux_vec`). Only equations (3-96) and (3-97) are directly expressed in the `equation` section. The function `interpolate(i_vec, flux_vec, i)`, which represents (3-100), interpolates linearly in the vectors (`i_vec, flux_vec`) and returns the value `flux` that corresponds to the `i`. This function defines a causal relation between flux and current of the model. The second equation implies (3-101) and is an acausal relation. As one can see, the modeler has absolutely nothing to do with how the simulation engine will use that model. There is not any input/output orientation. There is no need to define how the model equations are inserted into the main network equations.

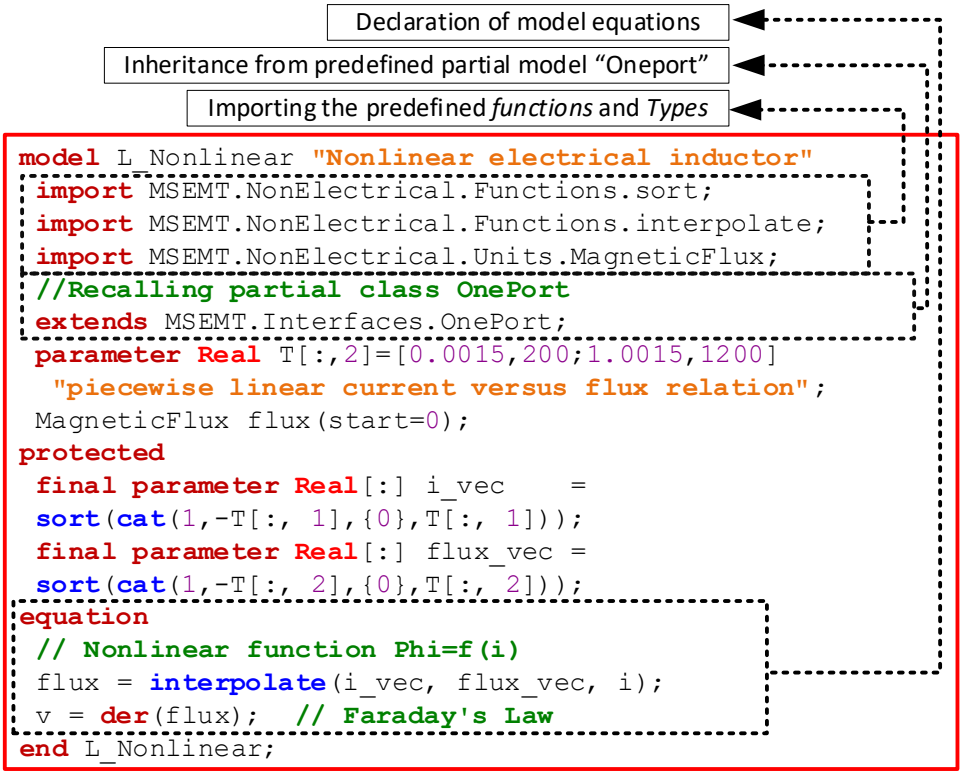


Figure 3.25 Nonlinear inductor model implemented in Modelica

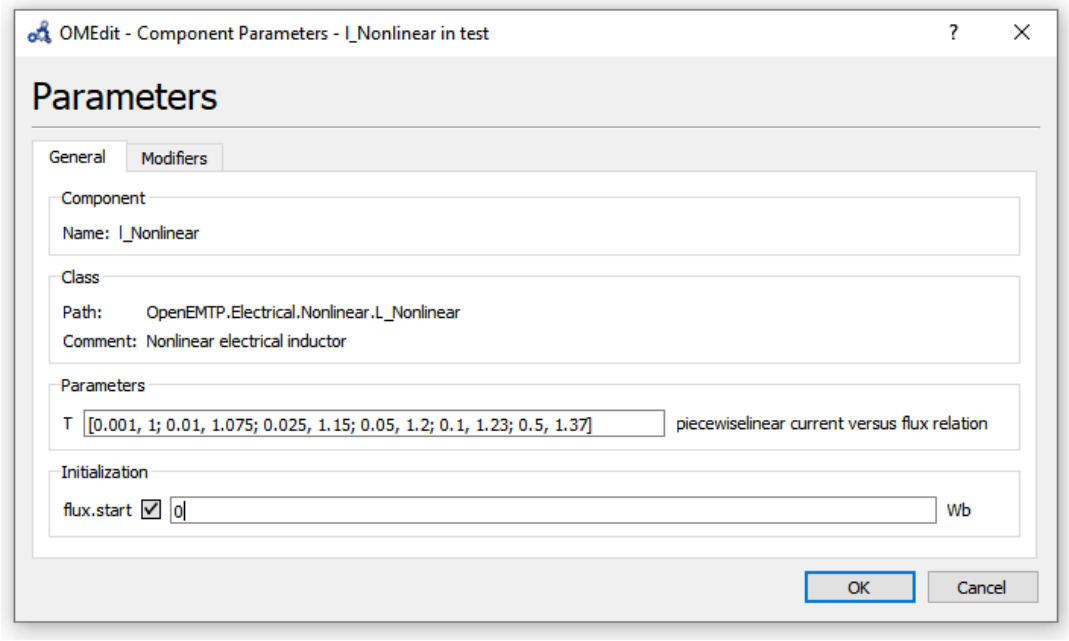


Figure 3.26 GUI of a nonlinear inductor in Modelica simulator

3.6.2 Nonlinear Resistor

Voltage-dependent resistors are widely used for various applications, such as modeling of electronic switches.

3.6.2.1 Piecewise Linear Resistor Model

The nonlinear resistor block represents a time-varying resistor working on a piecewise linear representation of the voltage-current resistance characteristic, as depicted in Figure 3.27. A monotonically increasing voltage-current characteristic specifies the resistance. The first and last voltage points are extended to negative and positive infinity, respectively. Linear interpolation is used between the data points.

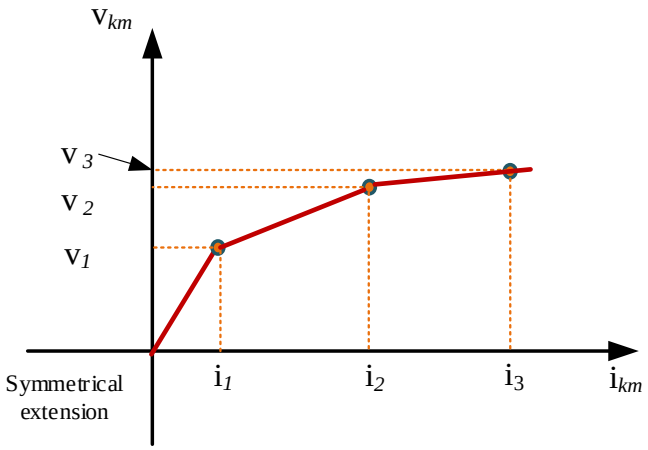


Figure 3.27 The piecewise linear current-voltage characteristics of resistance

Figure 3.28 shows the Modelica implementation of the nonlinear resistance model. As one can see, it is only required to program the function `interpolation` for finding the current passing through the component (`i`) corresponding to its terminal voltage (`v`) using the 2D table `T`. The function `interpolation` has been predefined and is available in the branch of MSEMT>NonElectrical>Functions. It is noted that the relation between current and voltage of the model is causal and is defined by a function. It is once again observed that modeling using high-level codes is very similar to the model's equations. Implementation of one such model in an imperative language such as MATLAB imposes hundreds of lines of code, which is complicated to understand.

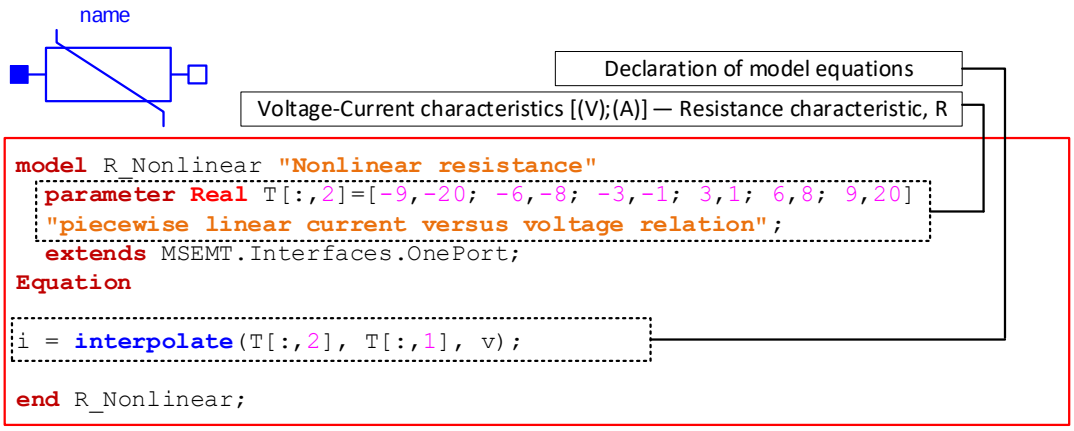


Figure 3.28 Modelica codes for the implementation of the piecewise linear resistor model

3.6.2.2 Polynomial Model

The polynomial nonlinear behavior of the resistor R, is described through the following relation:

$$v_{km} = R_1 i_{km} + R_2 {i_{km}}^2 + R_3 {i_{km}}^3 \qquad (3\text{-}102)$$

where v_{km} and i_{km} denote the voltage and current of the resistor, respectively, and R_1, R_2 and R_3 are the parameters of the model. The model imposes a heavy computational burden during simulation rather than the piecewise linear resistance model, because in the second one, the number of nonlinearities is limited to the finite number of linear sections

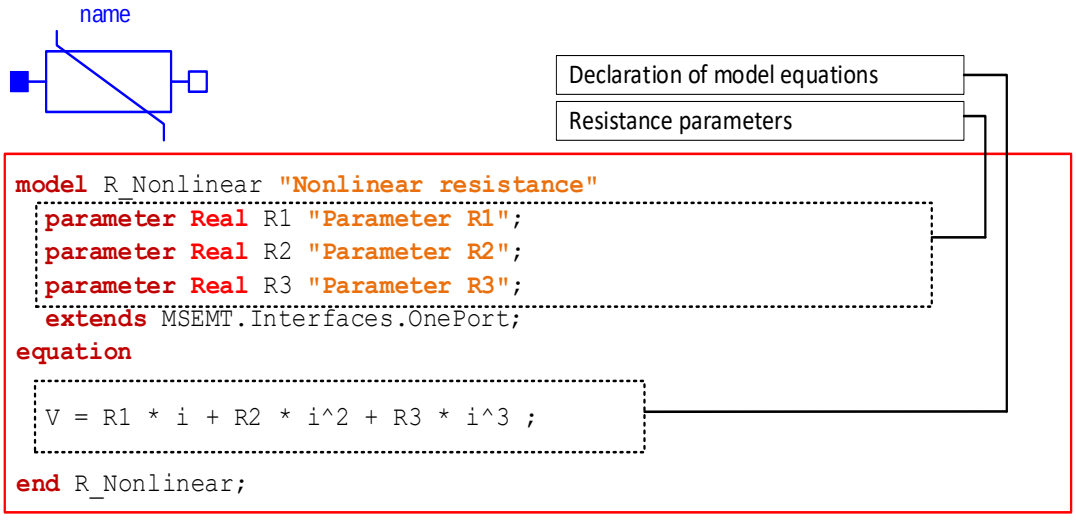


Figure 3.29 Modelica codes for the implementation of the polynomial resistor model

3.6.2.3 Surge Arrester Model

Surge arresters protect the insulation of equipment, e.g., transformers in electrical systems against overvoltage transients caused by lightning or switching surges. The voltage and current characteristic of a gapless metal-oxide surge arrester (ZnO), as illustrated in Figure 3.30, is a severely nonlinear resistor with an *infinite* slope in the normal operation region and an almost *horizontal* slope in the protection region (temporary and lightning overvoltages). The following power function in EMTP® represents the nonlinear resistance:

$$i_{km} = p_j \left(\frac{v_{km}}{V_{ref}} \right)^{q_j} \tag{3-103}$$

where i_{km} and v_{km} are arrester current and voltage, j is the segment number starting at the voltage V_{min_j}, multiplier p_j and exponent q_j are coefficients defined for each V_{min_j} and V_{ref} is the arrester reference voltage. A *linear* function is used for the first segment.

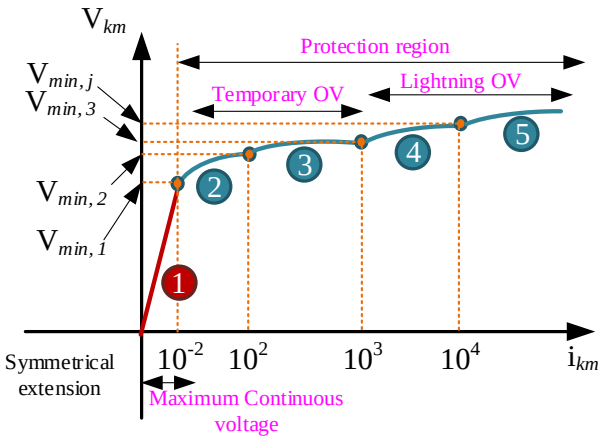


Figure 3.30 Voltage-current characteristic of ZnO surge arrester

The procedure for modeling a nonlinear resistance (arrester function) is similar to the technique used for the nonlinear inductance. Figure 3.31 illustrates the codes for the implementation of the surge arrester. The parameters of p_j, q_j and V_{min_j} are defined by n -by-3 matrix `T`. Figure 3.32 shows the `ExponentialInterpolate()` function defined by the specific class `function`, where the operating voltage is searched for the appropriate segment, j. Then, using (3-103), the value of i_{km} is interpolated. The properties of partial class `OnePort` are inherited to apply the appropriate equations of one-port devices [124]. It is noted that the relation between current and voltage of the model is causal and is defined by a function.

As one can see, the implementation of the model is straightforward, and there are no limitations for connections of this model in arbitrary network conditions.

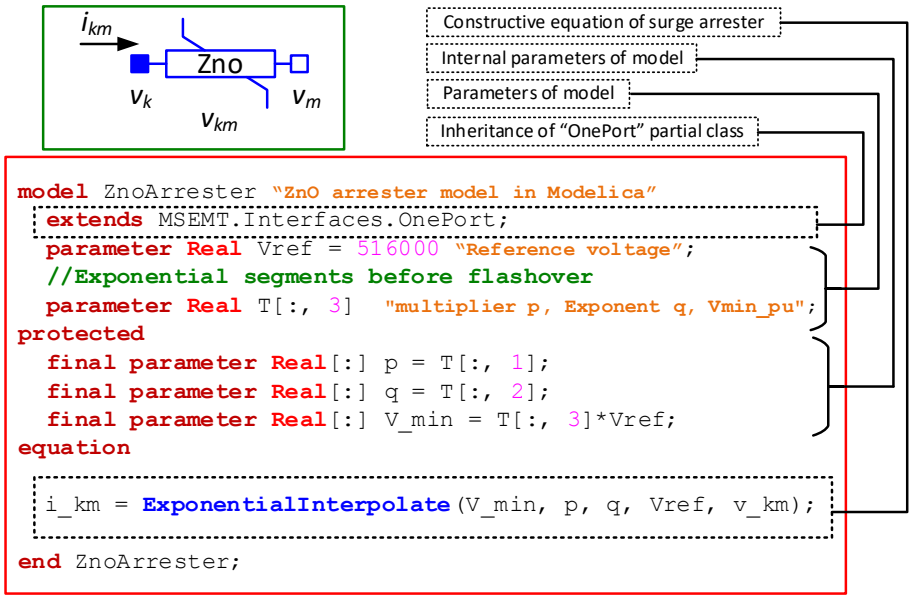


Figure 3.31 Codes used for implementation of the ZnO surge arrester model in Modelica

```
function ExponentialInterpolate "Interpolate exponentially in a vector"
  extends UserGuide.Icons.Function;
  input Real y[:] "Abscissa table vector (strict monotonically increasing values required)"; //Vj
  input Real p[size(y, 1)] "Ordinate table vector";
  input Real q[size(y, 1)] "Ordinate table vector";
  input Real Vref;
  input Real u "Desired abscissa value";
  input Integer iLast=1 "Index used in last search";
  output Real yy "Ordinate value corresponding to xi";
  output Integer iNew=1 "xi is in the interval x[iNew] <= xi < x[iNew+1]";
protected
  Integer i;
  Integer nx=size(y, 1);
  Real yi=abs(u),xi;
  Real x1,y1;
  Real m;

algorithm
  assert(nx > 0, "The table vectors must have at least 1 entry.");
// Search point
// search forward
  if yi >= y[nx] then
    i := nx;
  else
    i := 1;
    while i < nx and yi >= y[i] loop
      i := i + 1;
    end while;
    i := i - 1;
  end if;
// Interpolate
  if i == 0 then     //linear  segment
    x1 := p[1] * (y[1] / Vref) ^ q[1];
    m := y[1] / x1;
    xi := 1 / m * yi;
  else
    xi := p[i] * (yi / Vref) ^ q[i];
  end if;

//i=0
//i>1
  iNew := i;
//symetricalization based on odd function f(-x)=-f(X)
  if u >= 0 then
    yy := xi;
  else
    yy := -xi;
  end if;
end ExponentialInterpolate;
```

Figure 3.32 Implementation of function `exponentialInterpolation`

3.6.2.4 Arc Models

Arc models are mathematically modeled as a time-variant resistance or conductance function of arc current, voltage, and several time-variant parameters representing arc properties. This section introduces the main arc models proposed by Mayr and Cassie [104]. The arc behaves as a nonlinear resistance described by a nonlinear ODE.

The Mayr arc model [104] equation is defined by (3-104).

$$\frac{1}{g_m}\frac{dg_m}{dt} = \frac{1}{\tau_m}\left(\frac{vi}{p_0} - 1\right) = \frac{1}{\tau_m}\left(\frac{i^2}{p_0 g_m} - 1\right) \tag{3-104}$$

where g_m is the conductivity of the Mayr arc model, v and i are the instant values of the arc voltage and current in the circuit breaker, τ_m denotes the time constants of the electric arc and p_0 represents the dissipated power at current passing through zero. The parameters of $\tau_m$and p_0 in the model are determined based on the arc conductance[104]. The constants τ_m and p_0 have impact on arc voltage behavior during current zero-crossing such as extinguishing voltage, transient recovery voltage, and rate of build-up of recovery voltage. The Mayr model shows an increasing arc voltage (the extinction peak) close to the current zero-crossing.

In EMTP®, arc is modeled using the block diagrams approach. The same approach is used in Simscape Specilized Power System library[43] as well. In Modelica, arc is completely modeled through the explicitly expressing of its equation. Figure 3.33 shows the implementation of the Mayr model in Modelica. As one can see, two equations are expressed in `equation` section. the first is arc equation. i.e., (3-104), and other Ohm's law, which descrive the relation between voltage and current in a `OnePort` component.

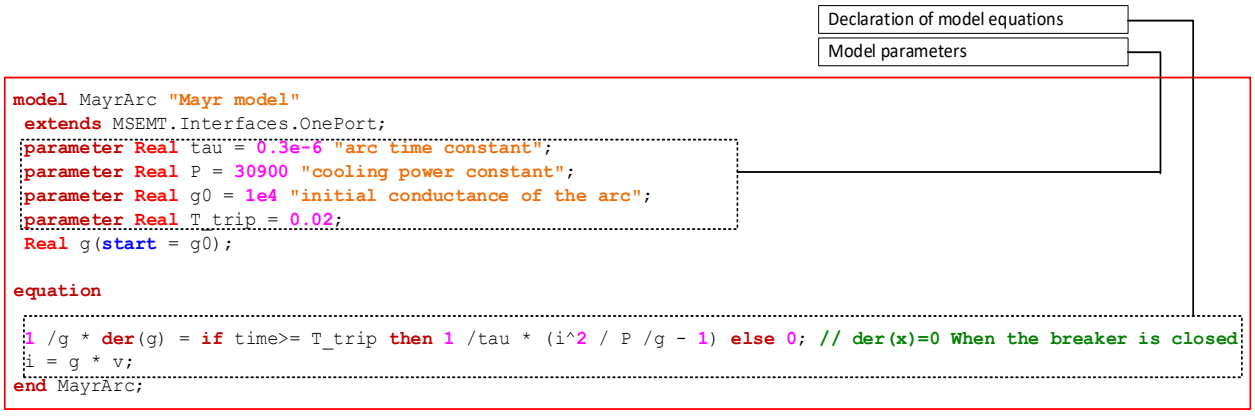


Figure 3.33 Modelica codes of the Mayr arc model

The Cassie arc [104] model equation is defined by (3-105).

$$\frac{1}{g_c}\frac{dg_c}{dt} = \frac{1}{\tau_c}\left(\frac{v^2}{v_0} - 1\right) \tag{3-105}$$

where g_c is the conductivity of the Cassie model, v is the instant values of the arc voltage, τ_c denotes the time constants of the electric arc and v_0 determines the average value of arc voltage. Cassie Model assumes that the arc has a constant temperature being cooled by forced convection. Current density and electric field strength are considered fixed in the model as well [104].

Figure 3.34 illustrates the implementation of the Cassie model in Modelica. As it can be observed, the equation (3-105) is directly implemented in Modelica code.

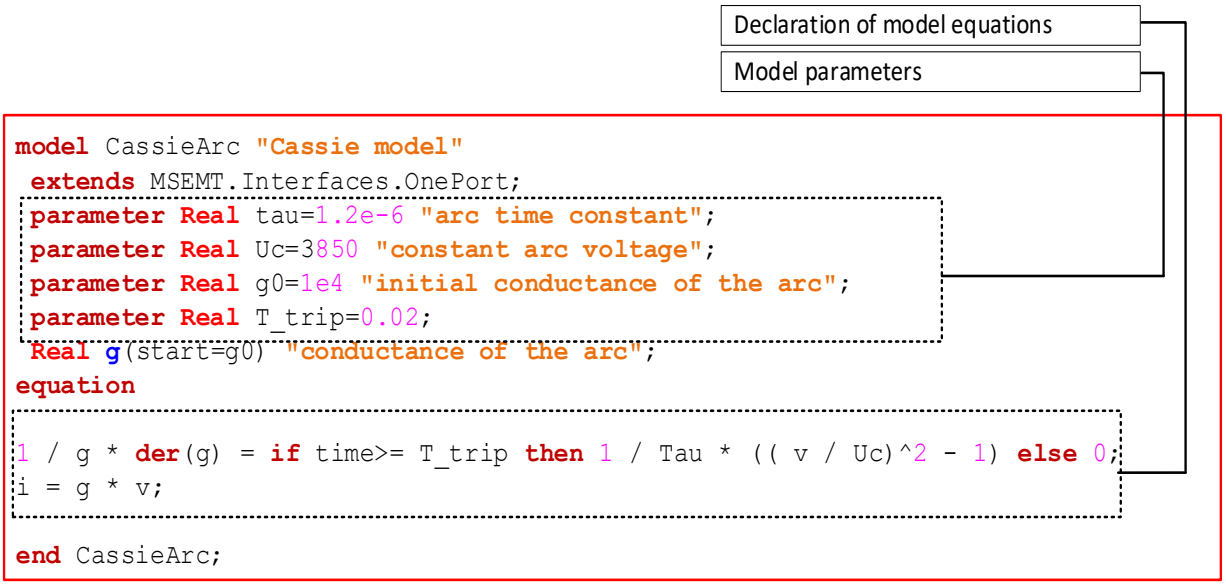


Figure 3.34 Modelica codes of the Cassie arc model

The series combination of both models gives a new model called the Cassie-Mayr model [104], which can be defined by:

$$\frac{1}{g} = \frac{1}{g_c} + \frac{1}{g_m} \tag{3-106}$$

Figure 3.35 shows the Modelica code of Cassie-Mayr arc model. In the piece of code, the arc conductance is reformulated in terms of arc current. The mathematical manipulation has no impact on the results in Modelica. All arc models described in this section will be validated in Section 7.5.

```
model CassieMayrModel"Cassie-Mayr model"
extends OpenEMTP.Interfaces.OnePort;
parameter Real tau_m=0.5e-06 "Mayr arc time constant";
parameter Real tau_c=1e-6 "Cassie arc time constant";
parameter Real P=10.0e+04 "cooling power constant";
parameter Real Uc=2000 "constant arc voltage";
parameter Real g0=5e+07 "initial conductance of the arc";
parameter Real T_trip=28e-3;
Real gc(start=2*g0);  // Conductance of Cassie Model
Real gm(start=2*g0);  // Conductance of Mayr Model
Real g; // Arc conductance
Real r; // Arc resistance
equation
der(gm)=if time>= T_trip then 1 / tau_m * (i^2/P - gm) else 0;
// der(x)=0 When the breaker is closed
der(gc)=if time>= T_trip then 1 / tau_c * ((i / Uc)^ 2/gc - gc) else 0;
// der(x)=0 When the breaker is closed
1 / g = 1 / gc + 1 / gm;
r = 1 / g;
i = g * v ;
end CassieMayrModel;
```

Figure 3.35 Cassie-Mayr arc model in Modelica

3.7 Switches

The modeling of ideal switch is essential for EMT studies. Implementation of the switch with zero resistance in the closing condition in Modelica causes numerical problems. A snubber resistance (R=1e-15 Ω) is considered to eliminate the problem. The operating logic of the ideal switch in the MSEMT library is as below:

1) the switch closes when the simulation time is greater or equal to `Tclosing`.
2) the switch opens when the simulation time is greater or equal to `Topening` and the current passing the switch crosses zero.

Figure 3.36 shows the implementation of the ideal switch model in Modelica using the predefined block diagrams available in MSMET>NonElectrical>Blocks.

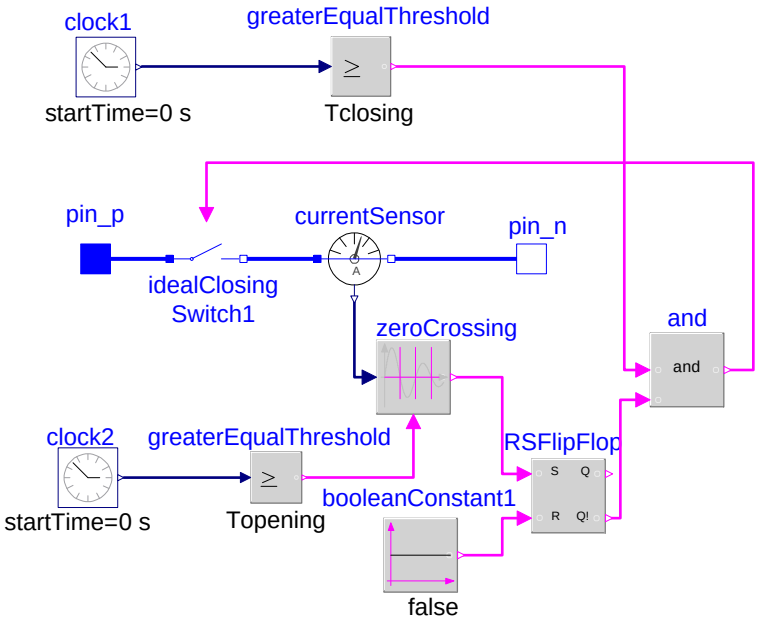


Figure 3.36 Block diagram approach for modeling of the ideal switch

3.8 Transformers

The saturable transformer component (STC) model has two parts. The first part represents the windings resistance and inductance and contains linear elements. The second part models the behavior of the transformer core and is described with nonlinear inductance.

The STC model, which is also known as the star equivalent circuit, is illustrated in Figure 3.37.

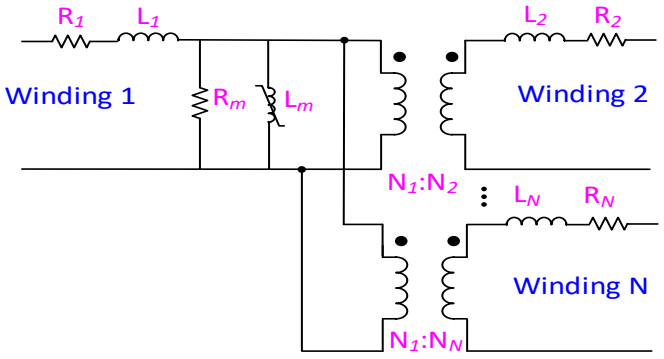
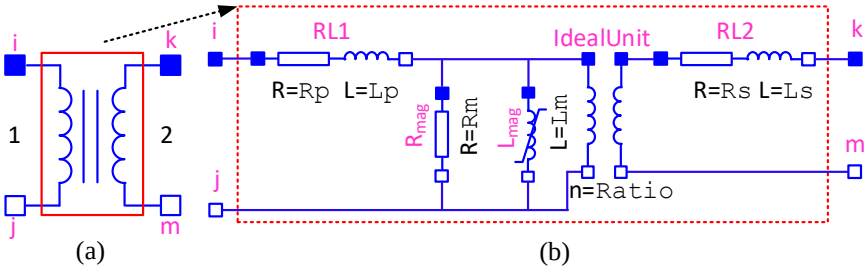
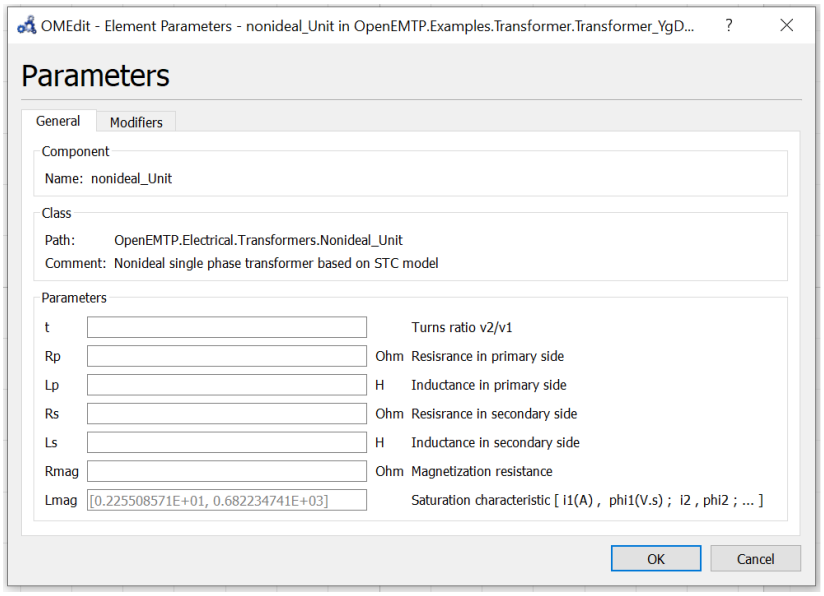


Figure 3.37 Single-phase N-winding STC model

The STC model can be extended to a three-phase or three-winding transformer model. In EMT-type programs, the magnetic saturation is represented by a piecewise linear inductor [104].

The implementation of a single-phase transformer in Modelica, as illustrated in Figure 3.38.(b), is a combination of pre-defined models linked to each other and then packaged to construct a new model. Figure 3.38.(a) illustrates the GUI symbol of the single-phase transformer in Modelica. Figure 3.38.(b) shows the implementation of the model. The model is a combination of pre-defined models linked to each other, then packaged to construct a new model. The input parameters as observed in Figure 3.38.(c) include the RL-branch values (`Rp, Lp, Rs,Ls`), the turn ratio (`Ratio`), and the magnetizing branch (`Rm, Lm`). Nonlinear data is entered for `Lm`.





(c)

Figure 3.38 (a): The icon of STC model in MSEMT library, (b): the sub-model of transformer model, (c): the GUI of transformer model

Figure 3.39 illustrates the Modelica codes describing the transformer components depicted in Figure 3.38. These codes are based on defining the model parameters, recalling the component models of resistance, inductance, nonlinear inductance, and ideal unit transformer, labeling them

appropriately, e.g., R1, R2, L1, etc., and finally assigning the proper parameter to them, e.g. (`final R=Rs`).

```
model Nonideal_Unit "Nonideal single phase transformer based on STC model"
  import MSEMT.NonElectrical.Units.Resistance;
  import MSEMT.NonElectrical.Units.Inductance;
  parameter Real t "Turns ratio v2/v1";
  parameter Resistance Rp "Resisrance in primary side";
  parameter Inductance Lp "Inductance in primary side";
  parameter Resistance Rs "Resisrance in secondary side";
  parameter Inductance Ls "Inductance in secondary side";
  parameter Resistance Rmag "Magnetization resistance ";
  parameter Real Lmag[:, 2] = [0.225508571E+01, 0.682234741E+03] "Saturation characteristic [
i1(A) ,  phi1(V.s) ;  i2 , phi2 ; ... ]";
  MSEMT.Interfaces.PositivePin Pin_i ;
  MSEMT.Interfaces.NegativePin Pin_j ;
  MSEMT.Interfaces.PositivePin Pin_k ;
  MSEMT.Interfaces.NegativePin Pin_m ;
  MSEMT.Electrical.Transformers.IdealUnit IdealUnit1(final g = t) ;
  MSEMT.Electrical.RLC_Branches.R R2(final R = Rs) ;
  MSEMT.Electrical.RLC_Branches.L L2(final L = Ls) ;
  MSEMT.Electrical.RLC_Branches.R R1(final R = Rp) ;
  MSEMT.Electrical.RLC_Branches.L L1(final L = Lp);
  MSEMT.Electrical.RLC_Branches.R Rm(final R = Rmag) ;
  MSEMT.Electrical.Nonlinear.L_Nonlinear Lm(final T = Lmag) ;
equation
  connect(R1.p, Pin_i);
  connect(R1.n, L1.p) ;
  connect(Rm.n, Pin_j);
  connect(Lm.n, Pin_j);
  connect(L1.n, Rm.p) ;
  connect(Lm.p, L1.n) ;
  connect(IdealUnit1.p1, L1.n) ;
  connect(IdealUnit1.n1, Pin_j) ;
  connect(IdealUnit1.p2, R2.p) ;
  connect(L2.p, R2.n) ;
  connect(L2.n, Pin_k) ;
  connect(IdealUnit1.n2, Pin_m);
end Nonideal_Unit;
```

Figure 3.39 Modelica codes for the implementation of single-phase STC model

3.8.1 Three-phase Transformer

The existing three-phase transformer models are based on three single-phase transformers as described in Section 3.8. The winding data are given in per unit. The appropriate conversion method is based on the actual transformer connections as described below:

$$Z_{b1} = \mathrm{V}_{b1}^2/\mathrm{S}_b \tag{3-107}$$

$$Z_{b2} = \mathrm{V}_{b2}^2/\mathrm{S}_b \tag{3-108}$$

$$I_{b1} = 1000S_b/V_{b1} \tag{3-109}$$

Where V_{b1} and V_{b2} are respectively nominal RMS line-to-line voltages on windings 1 and 2 in kV, S_b is the nominal transformer nominal power in MVA and I_{b1} represents the nominal current in winding 1.

3.8.1.1 Delta-Delta Configuration

For this configuration, the actual values are computed using (3-110)-(3-116) with $C_1 = 3$ and $C_2 = 3$. In these equations, C_1 and C_2 are coefficients whose values are defined based on the configuration in winding 1 and 2. These coefficients are 1 and 3, respectively, for wye and delta connections. D_w is a model parameter which defines the ratio of winding impedance on winding 1.

$$C_1 = 3 \tag{3-110}$$

$$C_2 = 3 \tag{3-111}$$

$$R_1 = C_1 R_{pu} Z_{b1} D_w \tag{3-112}$$

$$R_2 = C_2 R_{pu} Z_{b2} (1 - D_w) \tag{3-113}$$

$$L_1 = C_1 X_{pu} Z_{b1} D_w/(2\pi f) \tag{3-114}$$

$$L_2 = C_2 X_{pu} Z_{b2} (1 - D_w)/(2\pi f) \tag{3-115}$$

$$R_m = C_1 R_{m,pu} Z_{b1} \tag{3-116}$$

The actual nonlinear characteristic flux and current are computed using:

$$\varphi_{L,\ nonlinear} = \frac{1000\sqrt{2}\, V_{b1}}{2\pi f} \varphi_{L,\ nonlinear,pu} \tag{3-117}$$

$$I_{L,nonlinear} = \sqrt{\frac{2}{3}}\, I_{b1} I_{mag,pu} \tag{3-118}$$

where f denote the frequency, the ratio is given by:

$$n = V_{b2}/V_{b1} \tag{3-119}$$

3.8.1.2 Configurations DY +30, DY -30, DYg +30 and DYg -30

For these configurations, the actual values are computed with (3-112)-(3-116) with $C_1 = 3$ and $C_2 = 1$. The equations (3-117) and (3-118) remained unchanged, and the ratio is calculated by:

$$n = V_{b2}/\sqrt{3}V_{b1} \tag{3-120}$$

3.8.1.3 Configurations YD +30, YD -30, YgD +30 and YgD -30

For these configurations, the equations (3-112)-(3-116) are repeated with $C_1 = 1$ and $C_2 = 3$. The actual nonlinear characteristic flux and current vectors are computed using:

$$\varphi_{L,\ nonlinear} = \frac{1000\sqrt{2}\,V_{b1}}{\sqrt{3}\,2\pi f}\varphi_{L,\ nonlinear,pu} \tag{3-121}$$

$$I_{L,\ nonlinear} = \sqrt{2}\,I_{b1}I_{mag} \tag{3-122}$$

The ratio is given by:

$$n = \sqrt{3}\,V_{b2}/V_{b1} \tag{3-123}$$

3.8.1.4 Configurations YY and YgYg

For these configurations, the equations (3-112)-(3-116) are repeated with $C_1 = 1$ and $C_2 = 1$. The actual nonlinear characteristic flux and current vectors are computed by (3-121) and (3-122). The ratio is given by:

$$n = V_{b2}/V_{b1} \tag{3-124}$$

Figure 3.40 illustrates the GUI of the three-phase transformer with the configuration of YgD-30. The model parameters are defined similarly to the same model in EMTP®. Figure 3.41 shows the Modelica codes for the implementation of the transformer as described in Section 3.8.1. The first part of the codes is related to the parameters of the model. This part starts with `parameter Real`. The second part computes the actual values of parameters. `Nonideal_Unit_A`, `Nonideal_Unit_B`, and `Nonideal_Unit_C` are the single-phase units respectively defined for phases *a*, *b* and *c* and *reused* in the three-phase model. Reusability of models is one of the

advantages of the object-oriented modeling approach created in Modelica. In the `equation` compartment, the connections of components are defined.

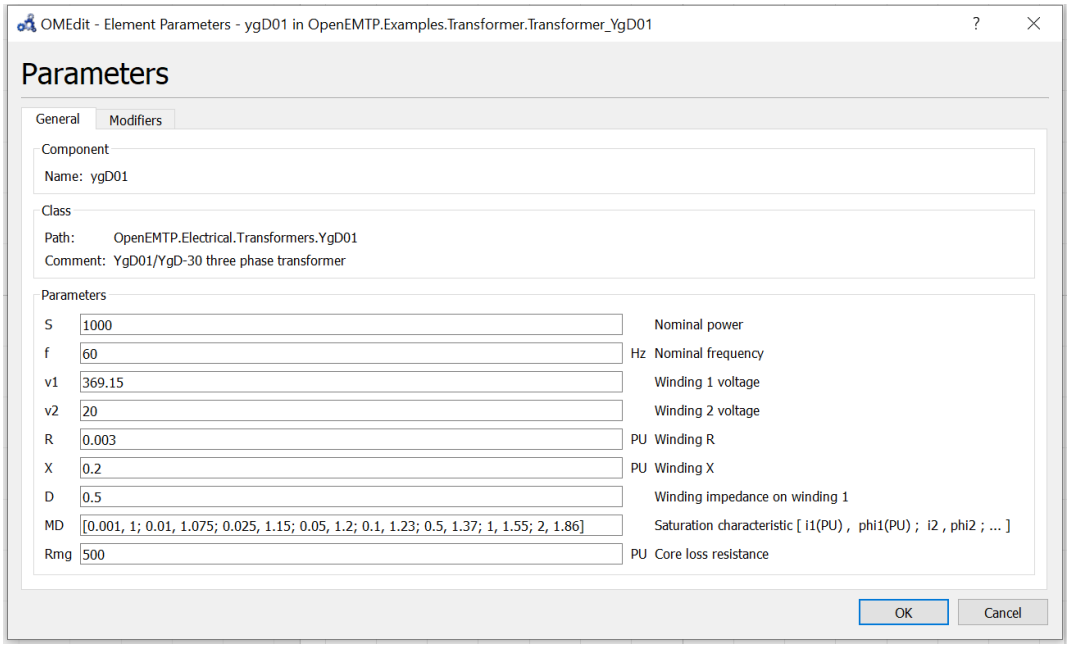


Figure 3.40 The GUI of three-phase transformer type YgD01

```
model YgD01 "YgD01/YgD-30 three phase transformer"
   //3-phase data parameters in PU
  import MSEMT.NonElectrical.Constants.pi;
  parameter Real S(unit = "MVA", start = 200) "Nominal power";
  parameter MSEMT.NonElectrical.Units.Frequency f(start=60) "Nominal frequency";
  parameter Real v1(unit = "kV RMSLL", start = 315) "Winding 1 voltage";
  parameter Real v2(unit = "kV RMSLL", start = 120) "Winding 2 voltage";
  parameter Real R(unit = "PU", start = 0.00375) "Winding R";
  parameter Real X(unit = "PU", start = 0.015) "Winding X";
  parameter Real D(start = 0.9) "Winding impedance on winding 1";
  parameter Real MD[:, 2] = [0.002, 1; 0.01, 1.075; 0.025, 1.15; 0.05, 1.2; 0.1, 1.23; 2., 1.72]
"Saturation characteristic [ i1(PU) ,  phi1(PU) ;  i2 , phi2 ; ... ]";
  parameter Real Rmg(unit = "PU", start = 500) " Core loss resistance";
  final parameter Integer C1 = 1;
  // Y:C1=1 Delta: C1=3
  final parameter Integer C2 = 3;
  // Y:C2=1 Delta: C2=3
  // base parameters
  final parameter Real Z_b1 = v1 ^ 2 / S "Impedance base side 1";
  final parameter Real Z_b2 = v2 ^ 2 / S "Impedance base side 2";
  final parameter Real i_b1 = 1000 * S / (sqrt(3) * v1) "Base current side 1";
    //Actual parameters
  final parameter Real R1 = C1 * R * Z_b1 * D "Resistance on side 1";
  final parameter Real R2 = C2 * R * Z_b2 * (1 - D) "Resistance on side 2";
  final parameter Real L1 = C1 * X * Z_b1 * D / (2 * pi * f) "Inductance on side 1";
  final parameter Real L2 = C2 * X * Z_b2 * (1 - D) / (2 * pi * f) "Inductance on side 2";
  final parameter Real Ratio = (v2 / v1)* sqrt(3);
  final parameter Real Rm = C1 * Rmg * Z_b1 "Magnetizating resistance on side 1";
  final parameter Real im[:] = sqrt(2) * i_b1 * MD[:, 1];
  final parameter Real Phim[:] = (1000 * sqrt(2) * v1 * MD[:, 2]) / (2 * pi * f * sqrt(3));
  final parameter Real MDD[:, 2] = [im, Phim];
  MSEMT.Connectors.PosPlug k ;
  MSEMT.Connectors.PosPlug m ;
  MSEMT.Connectors.PlugToPin_p Ph_a(k=1, m=3) ;
  MSEMT.Connectors.PlugToPin_p Ph_b(k=2, m=3) ;
  MMSEMT.Connectors.PlugToPin_p Ph_c(k=3, m=3) ;
  MSEMT.Connectors.PlugToPin_p Ph_A(k=1, m=3) ;
  MSEMT.Connectors.PlugToPin_p Ph_B(k=2, m=3) ;
  MSEMT.Connectors.PlugToPin_p Ph_C(k=3, m=3) ;
  MSEMT.Electrical.RLC_Branches.Ground ground1 ;
  MSEMT.Electrical.Transformers.Nonideal_Unit Nonideal_Unit_A(final Lmag = MDD,final Lp = L1, final Ls =
L2, final Rmag = Rm, final Rp = R1, final Rs = R2, final t = Ratio)  ;
  MSEMT.Electrical.Transformers.Nonideal_Unit Nonideal_Unit_B(final Lmag = MDD,final Lp = L1, final Ls =
L2, final Rmag = Rm, final Rp = R1, final Rs = R2, final t = Ratio)  ;
  MSEMT.Electrical.Transformers.Nonideal_Unit Nonideal_Unit_C(final Lmag = MDD,final Lp = L1, final Ls =
L2, final Rmag = Rm, final Rp = R1, final Rs = R2, final t = Ratio)  ;
equation
  connect(Nonideal_Unit_A.Pin_i, Ph_a.pin_p) ;
  connect(Ph_a.plug_p, k) ;
  connect(Nonideal_Unit_B.Pin_i, Ph_b.pin_p) ;
  connect(Ph_b.plug_p, k) ;
  connect(Nonideal_Unit_C.Pin_i, Ph_c.pin_p) ;
  connect(Ph_c.plug_p, k) ;
  connect(Nonideal_Unit_C.Pin_j, ground1.p) ;
  connect(Nonideal_Unit_B.Pin_j, ground1.p) ;
  connect(Nonideal_Unit_A.Pin_j, ground1.p) ;
  connect(Nonideal_Unit_A.Pin_k, Ph_A.pin_p) ;
  connect(Ph_A.plug_p, m) ;
  connect(Nonideal_Unit_C.Pin_m, Nonideal_Unit_A.Pin_k) ;
  connect(Nonideal_Unit_A.Pin_m, Nonideal_Unit_B.Pin_k) ;
  connect(Ph_B.pin_p, Nonideal_Unit_B.Pin_k) annotation ;
  connect(m, Ph_B.plug_p) ;
  connect(Nonideal_Unit_B.Pin_m, Nonideal_Unit_C.Pin_k) ;
  connect(Ph_C.pin_p, Nonideal_Unit_C.Pin_k);
  connect(Ph_C.plug_p, m);
end YgD01;
```

Figure 3.41 Modelica codes for transformer type YgD01

3.9 Block Diagrams

Block diagrams are used in the MSEMT library to model control systems, such as exciters and governors for synchronous generators and various other functions required for controlling power systems. The models are frequently used in test cases. Some available models are explained in this section to demonstrate the branch of NonElectrical>Blocks and the simplicity of adding missing control system components in Modelica.

3.9.1 Lead-Lag Compensator

Lead and lag compensators are extensively used in control systems. A lead compensator is to increase the stability or speed of response of a system. A lag compensator aimes to reducethe steady-state error.

A first-order lead compensator $\mathrm{G}(s)$ can be designed using the transfer function given by:

$$\mathrm{G}(s) = K_c \frac{s - z_0}{s - p_0} \tag{3-125}$$

where the magnitude of z_0 is less than the magnitude of p_0.

The equation (3-125) can also be rewritten as:

$$\mathrm{G}(s) = K_c \frac{T_1 s + 1}{T_2 s + 1} \tag{3-126}$$

where T_1 and T_2 are respectively lead and lag time constants.

Figure 3.42 shows the Modelica codes for the implementation of the lead-lag compensator.

```
block LeadLagCompensator
  extends Modelica.Blocks.Interfaces.SISO;
  parameter Real K "Gain";
  parameter MSEMT.NonElectrical.Units.Time T1 "Lead time constant";
  parameter MSEMT.NonElectrical.Units.Time T2 "Lag time constant";
  parameter Real y_start "Output start value"
    annotation (Dialog(group="Initialization"));
  parameter Real x_start=0 "Start value of state variable"
    annotation (Dialog(group="Initialization"));
  MSEMT.NonElectrical.Blocks.RealExpression par1(y=T1);
  MSEMT.NonElectrical.Blocks.RealExpression par2(y=T2);
  MSEMT.NonElectrical.Blocks.TransferFunction TF(
    b={K*T1,K},
    a={T2_dummy,1},
    y_start=y_start,
    initType=Modelica.Blocks.Types.Init.InitialOutput,
    x_start={x_start});
protected
  parameter Modelica.Units.SI.Time T2_dummy=if abs(T1 - T2) < Modelica.Constants.eps
       then 1000 else T2 "Lead time constant";
equation
  if abs(par1.y - par2.y) < Modelica.Constants.eps then
    y = K*u;
  else
    y = TF.y;
  end if;
  connect(TF.u, u);
end LeadLagCompensator;
```

Figure 3.42 Implementation of Lead-Lag Compensator in Modelica

3.9.2 Hold_to

This device captures as output the value presented by the input at $t = 0$. It means for $t > 0$, the output value `y` is calculated as `u(0)`. In the implementation of this block, the Modelica keyword `initial()` is used. This function returns true during the initialization phase and false otherwise.

```
block Hold_t0
  MSEMT.Connectors.RealInput u;
  MSEMT.Connectors.RealOutput y ;
equation
  when { initial()} then
    y = u;
  end when;
end Hold_t0;
```

Figure 3.43 Implementation of Hold_t0

3.9.3 Park's Transformation

Figure 3.44 shows the pieces of code for the implementation of `ParkTransform` block. The rotating frame angular position is defined by the input variable `theta`, in rad. This block implements both *power-variant* and *power-invariant* transformations. In this block, it is possible to align the *d*-axis or *q*-axis to phase-*a*. This function is employed in the synchronous machine model.

```
block ParkTransform "This block implements a rotating reference frame transformation commonly
known as the Park transform"
//The block implements a power invariant a-phase to d-axis alignment
  import PI=MSEMT.NonElectrical.Constants.pi;
  parameter Integer Type=1 "Phase-a axis alignment"
      annotation(Dialog(group="Parameters"),choices(
      choice=1 " d-axis is aligned to phase a",
      choice=2 " q-axis is aligned to phase a"));
  parameter Boolean PowerInvariant=true
      annotation (Evaluate=true, choices(checkBox=true));
  final parameter Real  K1=if PowerInvariant then sqrt(2/3) else 2/3;
  final parameter Real  K2=if PowerInvariant then sqrt(1/2) else 1/3;
  MSEMT.Connectors.RealInput u[3] ;
  MSEMT.Connectors.RealInput theta ;
  //Y={Yd,Yq,Y0}
  MSEMT.Connectors.RealOutput y[3];
  Real T[3,3] "Transformation matrix";
equation
  if     Type==1 then
  T = K1 * [cos(theta), cos(theta-2*PI/3), cos(theta+2*PI/3);
            sin(theta), sin(theta-2*PI/3), sin(theta+2*PI/3);
             K2       ,         K2       ,         K2      ];
  elseif Type==2 then
  T = K1 * [sin(theta), sin(theta-2*PI/3), sin(theta+2*PI/3);
            cos(theta), cos(theta-2*PI/3), cos(theta+2*PI/3);
             K2       ,         K2       ,          K2     ];
  end if;
  y=T*u;

end ParkTransform;
```

Figure 3.44 Implementation of Park's transformation

3.10 Functions

This section includes some functions used in the MSEMT library. Implementation of these functions, however, is simple but necessary to have a comprehensive library.

3.10.1 Clark's Transform

Clark's matrix is used in the modeling of the CP-line for modal transformation. For an *m*-phase balanced transmission line, the matrix is programmed as illustrated in Figure 3.45. This function is used in the CP-line model.

```
function Clark_Transformation "The alpha-beta transformation (also
known as the Clark transformation) for m-phase system"
  extends UserGuide.Icons.Function;
  input Integer m = 3; // "Number of phases"
  output Real[m, m] Ti;
algorithm
  for col in 1:m loop
    for row in 1 : m loop
      if col == 1 then
        Ti[row, col] := 1 / sqrt(m);
      elseif
            ( col < row) then
        Ti[row, col] := 0;
      elseif
            ( col == row) then
        Ti[row, col] := -(col - 1) * (1 / sqrt(col * (col - 1)));
      else
        Ti[row, col] := (1 / sqrt(col * (col - 1)));
      end if;
    end for;
  end for;
end Clark_Transformation;
```

Figure 3.45 Implementation of Clark's transformation

CHAPTER 4 ACCURACY ASSESSMENT OF TRANSMISSION LINE MODELS

In this chapter, two numerical examples are presented for the validation of line/cable models. To show the accuracy of the proposed approach, the EMTP® trapezoidal/backward Euler solver is set as a reference. The 2-norm cumulative relative error is used to measure the error level against the reference software, EMTP®.

4.1 Test Case for Underground Cable

Figure 4.1 shows the layout and electrical circuit of a 3-phase, 6-conductor cable intended for measuring the accuracy of the Modelica wideband model. The length of the cable is 15 km. The cores at the receiving end of the cable are left open, and the shield is grounded at both sides of the line through a resistance of 1 Ω. In this example, the matrix **H** has been approximated by 42 complex poles while 14 real poles are used for fitting of $\mathbf{Y}_c$. The fitting parameters are read from EMTP®. The simulation is carried out using the trapezoidal solver for 50 ms with a time-step of 10 μs in EMTP® and OpenModelica. The closing times of each phase are phase-*a* 0 s, phase-*b* 0.63 ms, and phase-*c* 0.4 ms. The system is energized by a 3-phase, 60 Hz, 169 kV sinusoidal voltage source.

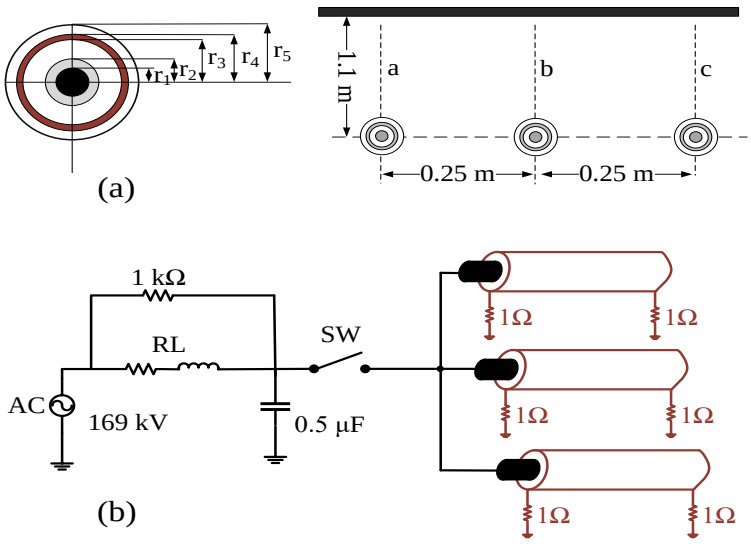


Figure 4.1 Underground cable system, 169 kV, 3-phase, 6-Conductor (a): Physical layout *(*b): Electrical connection diagram

Figure 4.2 shows the core and shield voltages at the receiving end of the cable. A black dashed line is used for EMTP® waveforms, and a solid line is for Modelica results. High-frequency oscillations are observed in the core and sheath voltages because of the source capacitance. The values obtained with the Modelica-based model are perfectly superposed to those obtained with EMTP®. The 2-norm error during the transient state [0,10] ms is 0.9 %. The simulation CPU time is 14.34 s compared to EMTP®, which is 0.57 s.

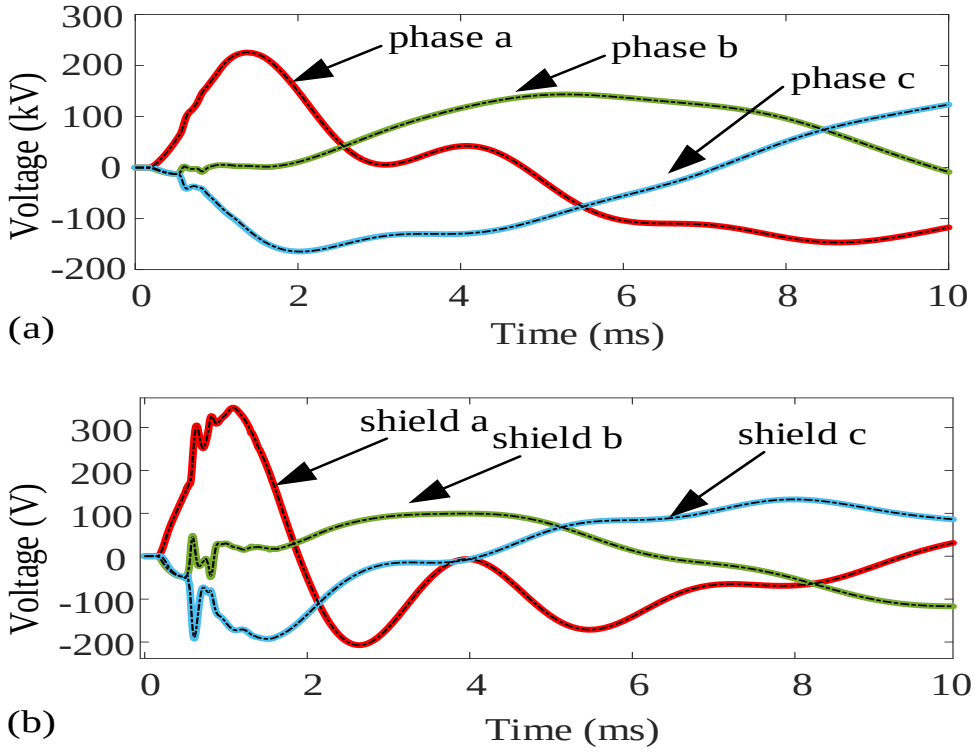


Figure 4.2 Voltage waveforms in receiving-end of WB-line model for (a): 3-core conductor (b) Shield conductors, black dashed line is EMTP®, solid line is Modelica

4.2 Test Case for Aerial Transmission Line

Figure 4.3 shows the IEEE 13-Bus distribution test circuit [106], used to test accuracy and performance with Modelica. The circuit uses the CP-line models. The PQ-loads are modeled with RLC circuits using power-flow results obtained from EMTP®. The distribution network is operating at 4.16 kV. The network contains 9 three-, two- and single-phase, short-length,

untransposed transmission lines, shunt capacitors, a transformer, and unbalanced loads. The mutual effects are assumed between the phases of transmission lines [107].

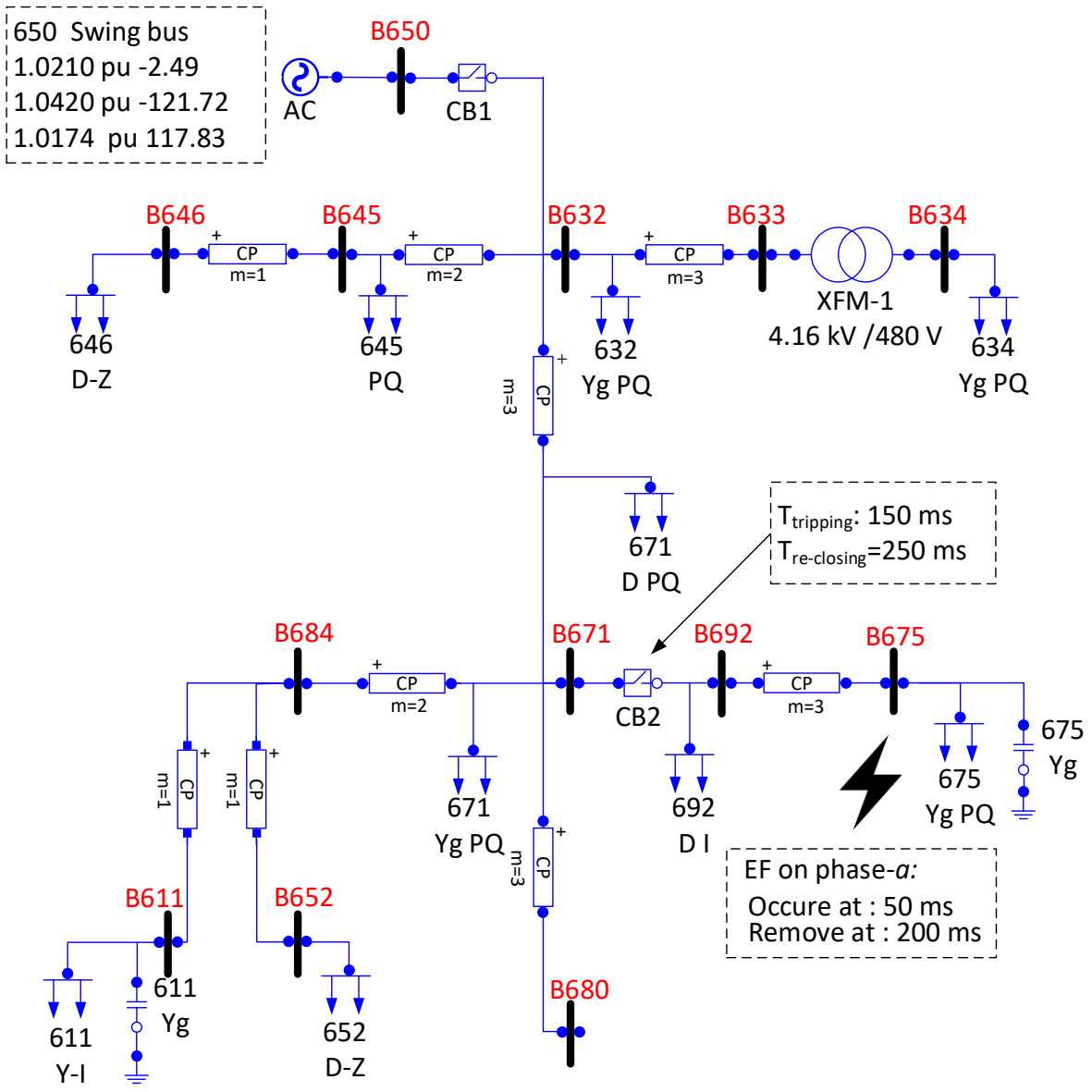


Figure 4.3 Single line diagram of IEEE 13-Node with the CP-line model. Earth fault occurs to the bus B675 at 60 ms, CB2 is opened 100 ms after the fault

The switch CB1 is initially open. The simulation time-step is set to 0.2 μs since it is required to be smaller than the propagation delay of the shortest TL. The simulation period is 200 ms. For the

study of transient performance, a phase *a*-to-ground fault with the resistance of 1 Ω is applied to the bus B675 at 60 ms. After the fault is detected by the protection relays (not simulated here), an opening command is immediately sent to the breaker CB2 at t =160 ms.

In the first scenario, simulation is run using the trapezoidal solver in Modelica and EMTP®; then, the results are graphically and quantitatively compared with EMTP®. Figure 4.4.(a) depicts the voltage waveforms of phases-*a*, -*b*, and -*c* at bus B675 denoted by red, green, and blue curves. The black dashed line represents the EMTP® results. The initial switching leads to distortion on all three phases; then, the system reaches steady-state at t = 35 ms. The plots match perfectly. The voltage profile is 0.99 pu for phase-*a*, 1.05 pu for phase-*b*, and 0.98 pu for phase-*c*. Figure 4.4.(b) presents the voltage waveforms in the interval of fault and post fault. In this case, both curves have an excellent agreement as well [107].

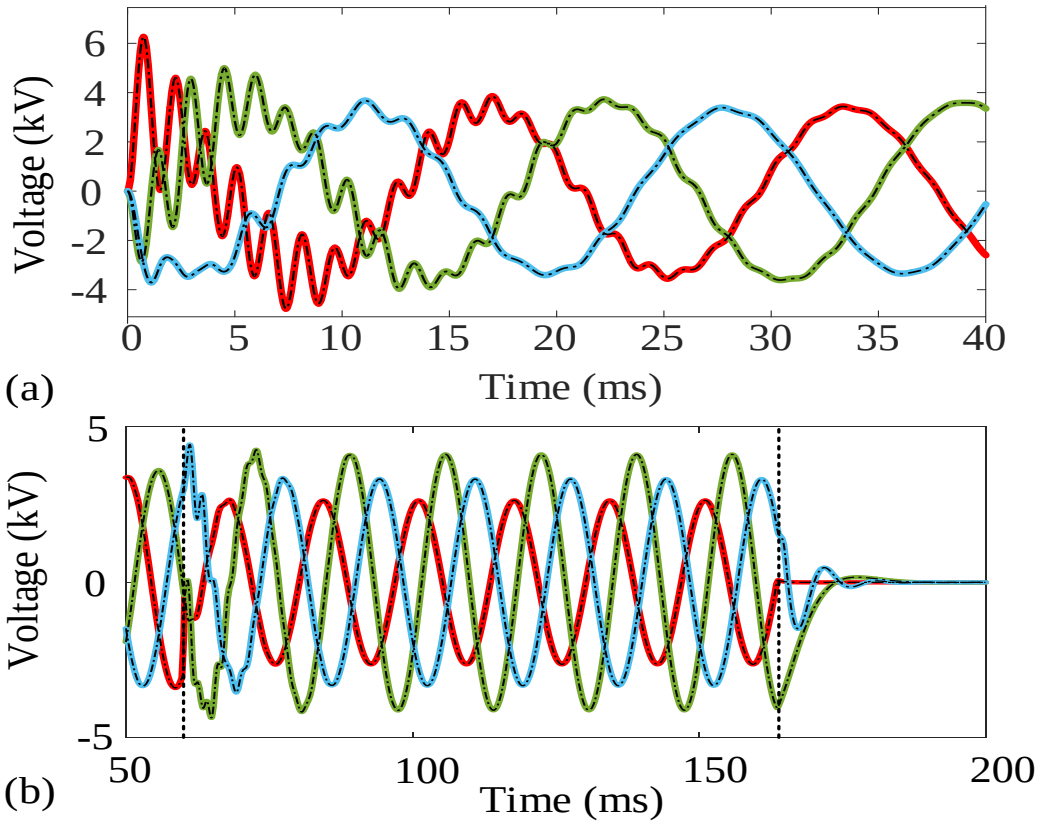


Figure 4.4 (a) Voltage waveforms at Bus 675 at the interval [0, 40] ms, black dashed line is EMTP®, Solid line is Modelica; (b) transient state after the occurrence of fault at 60 ms till the line is tripped at 160 ms

Figure 4.5.(a) illustrates the current waveforms passing through CB2. The maximum current at phase-a reaches 3 kA during the fault. There is a high-frequency transient in the current waveforms of phase-*b* and-*c* at the moment of fault. A high-frequency transient on current waveforms in the interval of [0, 35] ms can be observed in Figure 4.5.(b) as well. The transient components of voltages and currents are due to the capacitance charging of two healthy phases and discharging of faulted phase capacitance. An excellent superposition is also observed in the current curves. To compare with EMTP® results, the 2-norm cumulative relative error is computed for voltage at B675: it is 1.6911e-07%, 0.9552e-07% and 3.2600e-07% for phases-*a*, -*b* and -*c*, respectively [107].

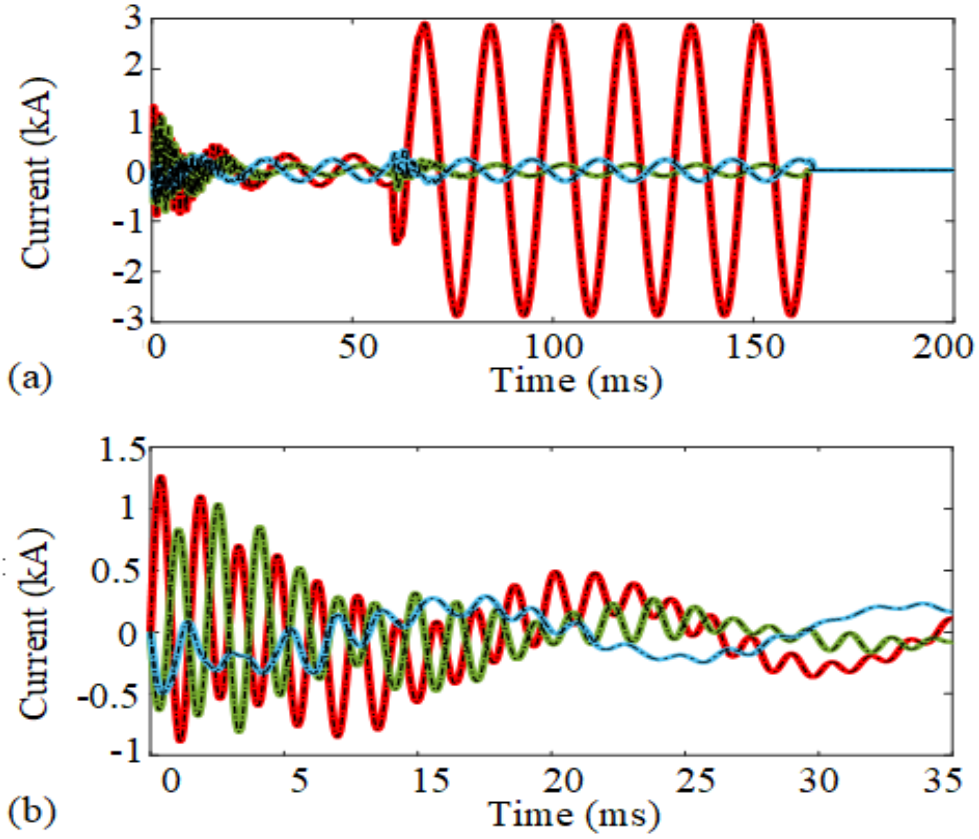


Figure 4.5: (a) Phase current waveforms of CB2, black dashed line is EMTP®, solid line is Modelica; (b) transient state in the interval [0, 35] ms

In the second scenario, the simulation is repeated using the variable-order adaptive time-step solvers: DASSL [71] and IDA [78] with the following settings: initial step size is 0.2 μs, the

maximum step size is 0.02 s, maximum integration order is 2. Due to the strict separation between solvers and models in Modelica, it is easy to switch to another numerical integration method and observe the results. Adaptive solvers use a local truncation error scheme to adapt the current time-step to what is going on in the simulation. During transients, the time-step is decreased to catch the transient phenomena completely while increasing during the steady-state to increase computational performance.-The main advantage of the variable time-step solver is that the user does not have to set the time step for a specific analysis,e.g., electromagnetic or electromechanical transient studies.

Table 4.1shows the 2-norm error computed for voltage signals at B675. As can be observed, the precision of variable-step solvers is very close to the second-order fixed-step trapezoidal solver [107].

The CPU time for DASSL and IDA solvers are 83.680 s and 81.576 s, respectively. The value for EMTP® is 8.715 s. It shall be noted that the CPU time for simulation in the OpenModelica environment includes logging of all variables. All tests are performed on a desktop with Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, 2201 MHz processor, and 32 GB of RAM.

Table 4.1 2-norm cumulative relative error comparison

Phase/Solver	**Phase *a***	**Phase *b***	**Phase *c***
DASSL	1.8790e-08 %	1.1445e-08%	5.8530e-08%
IDA	2.6843e-09 %	1.9535e-10%	6.7485e-09%

4.3 Conclusion

The proposed WB- and CP-line models provide results identical to those from the EMTP®. However, the environment demonstrates several declarative language modeling advantages: models are implemented in very few lines of code, are easily reusable, modifiable, and highly portable. In addition, the possibility to switch from one solver to another is a native feature. Notwithstanding, the conventional EMT-type tools provide faster computation time in the ratio of 10:1 for our initial test cases provided above. Further analysis on this aspect is conducted next.

Further developments of the MSEMT library will help assess and identify the potential challenges for concrete large-scale transient electromagnetic simulations [107].

CHAPTER 5 IEEE 39-BUS TEST CASE

5.1 Introduction

This chapter presents simulation results of the modified IEEE 39-bus benchmark system [108] to validate the accuracy of the proposed models. The same test case is also simulated with EMTP® [49] as reference software. The results are compared with the wideband line models.

Figure 5.1.(a) illustrates the IEEE 39-bus network created in the Modelica using the MESEMT library. The network includes 34 transmission lines (WB-line model), 10 power plants on buses B30-B39, each consists of a synchronous machine, machine controls, and transformer. There are 19 load transformers with static load models. The voltage ratio of load transformers is 345/25 kV, and the winding connection is YgD01. IEEE 39-bus network contains a three-winding 345/300/12.5kV YgYgD grid transformer which connects buses B19 and B20. The primary winding is connected to bus B19, the secondary is coupled to bus B20, and the tertiary is open circuit. The short-circuit data of this transformer is given in Table 5.1.

Table 5.1 IEEE-39 grid transformer data (YgYgD) [108]

Bus	Bus	R12	R13	R23	X12	X13	X23	Tap	S	U1	U2	U3
19	20	0.0022	0.0058	0.0058	0.193	0.292	0.1	1.06	1400	345	300	12.5

The reactive power compensator used in this benchmark is a 92MVA shunt capacitor on bus B24. Load models are based on constant impedance calculated using the voltage obtained by the load-flow solution (Table 2-13 of [108]) of EMTP®. The models of all three-phase transformers consist of single-phase units (STC model). The magnetization branch, including the nonlinear inductor is placed on the high-voltage side. The model uses a piecewise linearly interpolated curve to represent saturation.

Figure 5.1.(b) shows the submodel of PowerPlant 03, which contains a single-mass Wye grounded configuration synchronous machine, the governor IEEEG1 [97], exciter ST1 [96], a step-up

369.15/20 kV,1000MVA transformer with the connection type of YgD01, first-order filter, and block for conversion of terminal voltage from *dq* frame to phase frame.

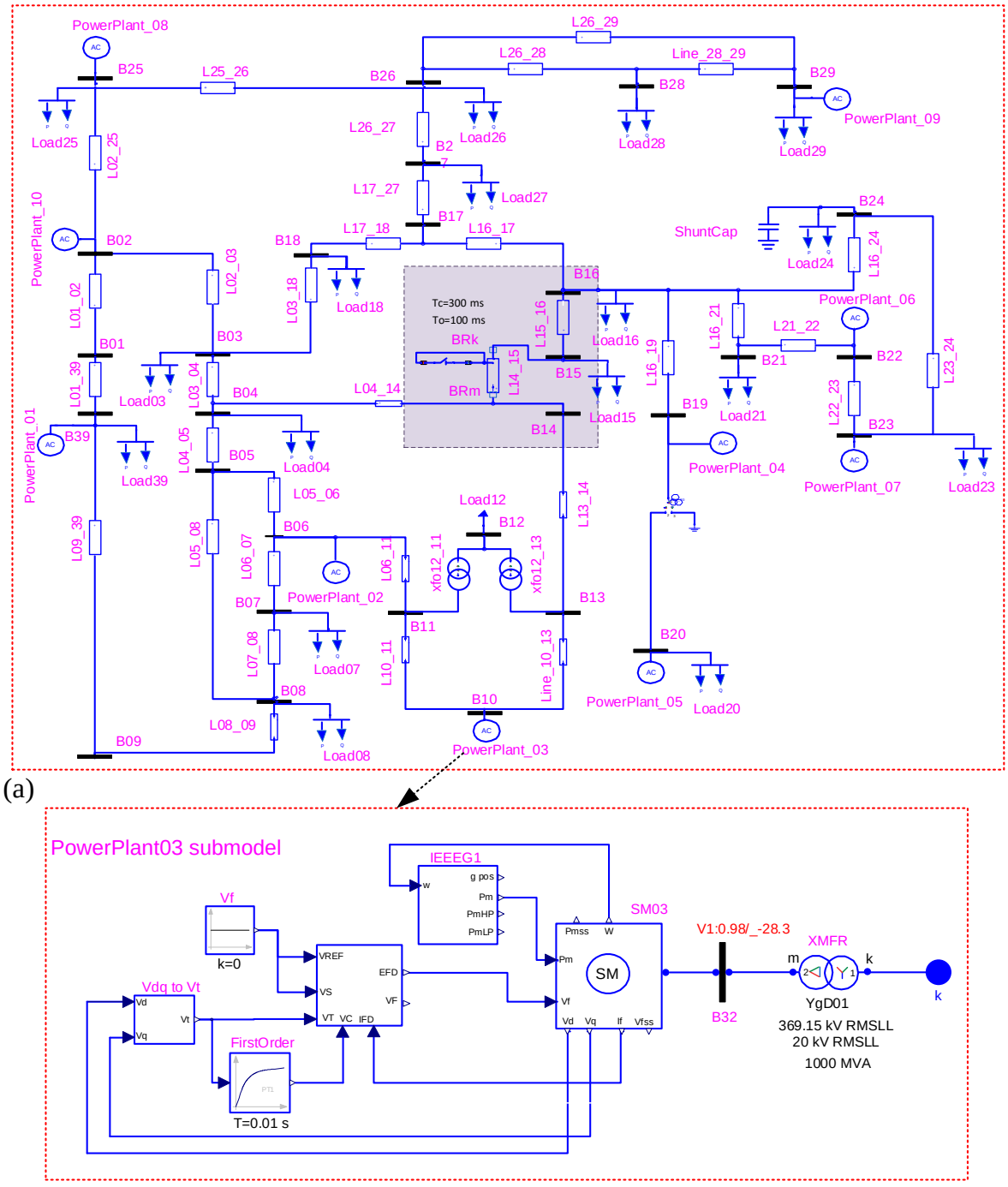


Figure 5.1.(a): IEEE 39-bus network, which is designed in Modelica using the MSEMT library incorporating the WB-line model. (b): the submodel of PowerPlant03

The transient response scenario is illustrated in Figure 5.2.a. A temporary phase-to-phase fault occurs on phases '*a*' and '*b*' of TL_14_15 near B15 at t = 100 ms followed by the isolation of the line at t = 200 ms (i.e., breakers BRm and BRk open simultaneously after 6 cycles). The fault is cleared at t = 300 ms; then, the line is reconnected at t = 450 ms.

Re-energizing the TL introduces high-frequency transient oscillations and allows us to investigate the accuracy of transformer models in nonlinear operation. For this purpose, the curve of flux versus current for load transformer 15, which is located near the faulted bus, is compared with EMTP®.

Numerical tests are performed using the variable-step IDA solver with the tolerance of 1e-6 in OpenModelica[30] and Trapezoidal/Backward Euler integrator in EMTP® with a step size of 25 µs. The Modelica network contains 12 648 DAEs. There is no initialization for the simulations with Modelica.

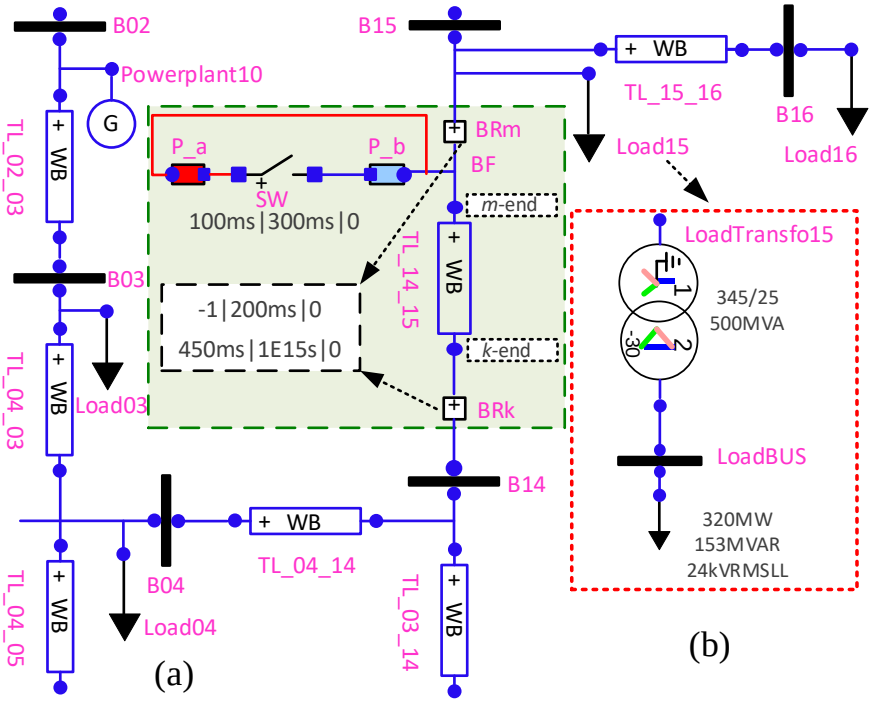


Figure 5.2 a) Schematic of the faulted zone of IEEE 39-bus network created in Modelica GUIs; (b) the sub-circuit of Load15, the circuit contains a three-phase YgD-30 load transformer (STC model) and a constant impedance load model

5.2 IEEE 39-bus Incorporating WB-Line Models

The vector fitting parameters of the WB-line model for transmission lines of IEEE 39-bus network are calculated in EMTP® for 8 decades starting at $f_{min} = 0.1$ Hz. The maximum orders of fitting for the propagation matrix, ($N_i^{\mathbf{H}}$) and admittance matrix ($N_{\mathbf{Y}_c}$), are 7 and 9 respectively [75].

Figure 5.3.(a) shows the simulation results presenting phase voltage waveforms at the *m*-end of TL_14_15. The close-up plot of phase voltages during re-energization of the line after clearing the fault is observed in Figure 5.3.(b) As can be seen from the plots, the results obtained with Modelica match perfectly the EMTP® results [75].

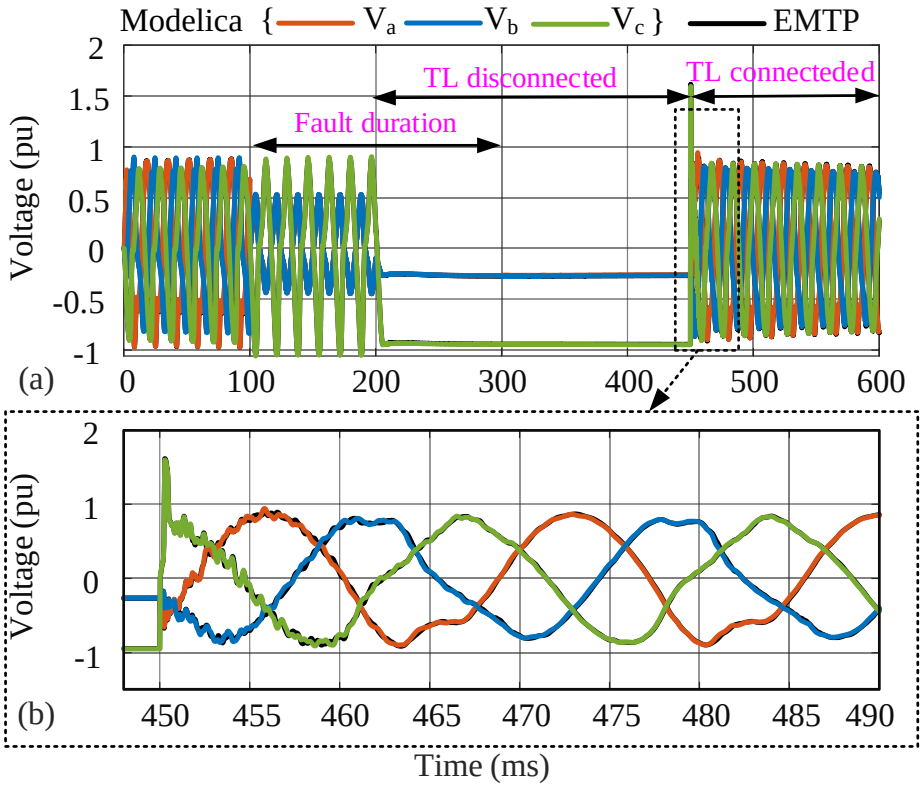


Figure 5.3 (a) voltage waveforms at the m-end of TL_14_15; (b) close-up view after re-energization of TL_14_15

Accuracy assessment is carried out in Figure 5.4 by drawing the graph of relative errors for voltage waveforms in Figure 5.3.(a). The slight difference is justified by the different methods of

discontinuity handling, control system implementation, and numerical accuracy of the solver in each simulation tool.

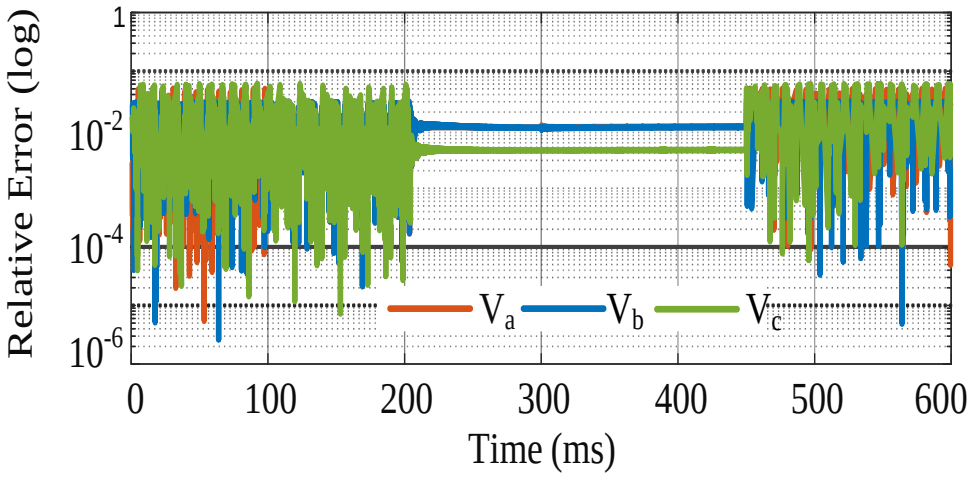


Figure 5.4 Assessment of accuracy for Modelica-based simulation: relative errors of phase voltages at the *m*-end of TL_14_15

5.3 Solution Evaluation for STC Model

The simultaneous solution for nonlinear functions in Modelica can be verified by demonstrating that all solution points remain on the same nonlinear characteristic segments for both simulations. This is carried out in Figure 5.5 for the nonlinear inductor of the LoadTransfo15 (see Figure 5.2.(b)). It is observed that both solutions are precisely on the magnetization curve of LoadTransfo15, and no overshooting or other instabilities are observed in the boundary points of linear segments[75].

In both simulations, the problem is solved through an iterative method, and the need for changing the segment is realized before the last point has been within its improper range. Mathematically speaking, IDA is an adaptive solver, and when simulation reaches a breakpoint (either state event or time event), it reduces the step size; once the last point of the current segment is solved, the segment change is accepted [75].

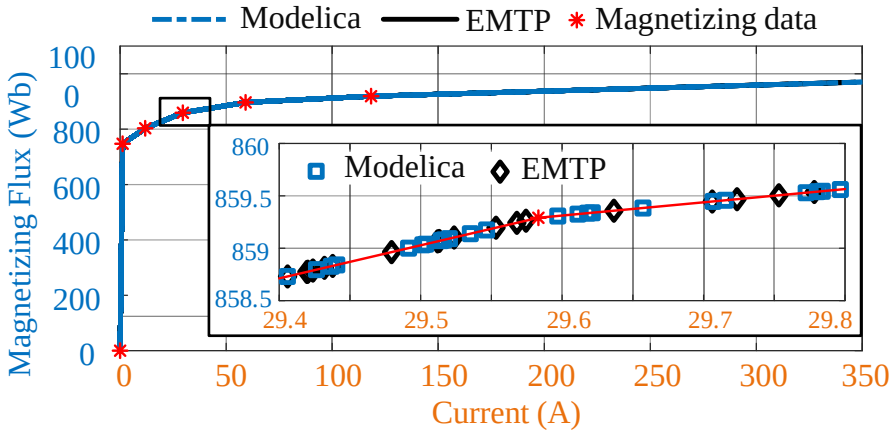


Figure 5.5 Superposition of magnetizing inductance in the LoadTransfo15; zoom-in view of knee-point solutions

5.4 Evaluation of SM Model Accuracy

In this section, the behavior of the proposed SM model in an unbalanced operation is examined. For this purpose, the generator connected to the B32 in the PowerPlant_03 (see Figure 5.1.(b)) is selected as the nearest generator to the fault point. The resulting transients of stator current in phase-a (i_a), is depicted in Figure 5.6. As it can be observed, the transient responses produced by Modelica match with the reference solutions.

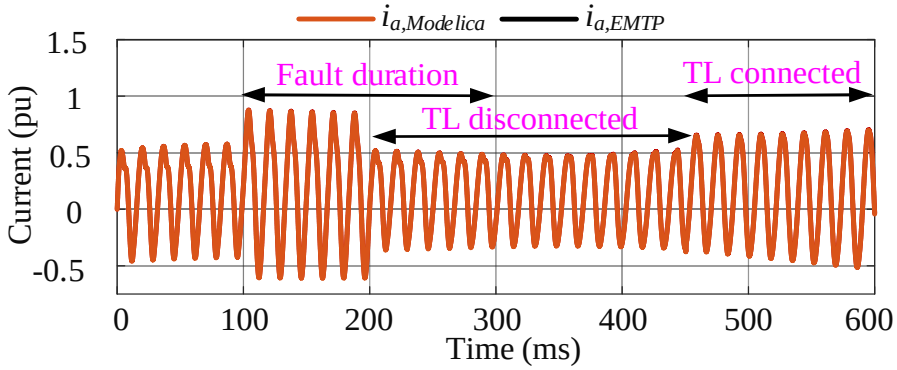


Figure 5.6 Stator current in phase-a (i_a), for the generator in PowerPlant_03

Figure 5.7 illustrates the transient current computed in the damper *kq1*. One can observe that all transients fit the reference solution obtained by EMTP®.

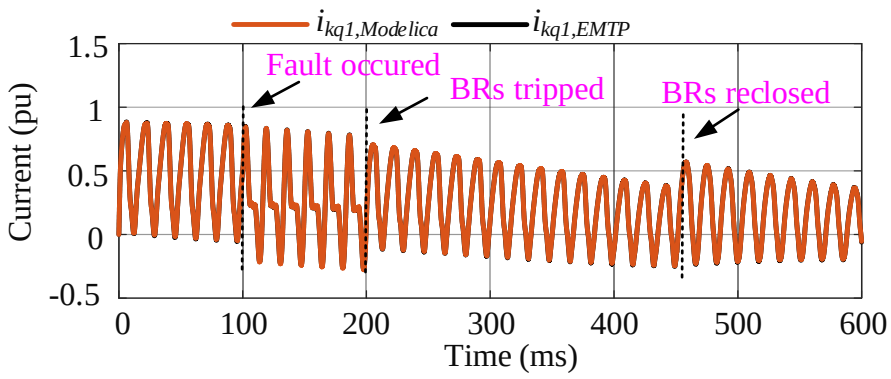


Figure 5.7 Damper winding current, i_{kq1}, for the generator in the PowerPlant_03

5.5 Evaluation of Accuracy for Controllers

Figure 5.8 shows the output of the exciter in the PowerPlant_03 (see Figure 5.1.(b)), which controls the field voltage of the generator connected to the B32. As one can observe, both solutions are indistinguishable.

Similarly, the mechanical power regulated by the governor in PolwerPlant_03 (see Figure 5.1.(b)) is compared in Figure 5.9. It is observed that the governor controls the output power proportionally to rotor speed, and both simulation results are in excellent agreement. Figure 5.10 illustrates the rotor speed of the same generator. Simulation results are perfectly identical.

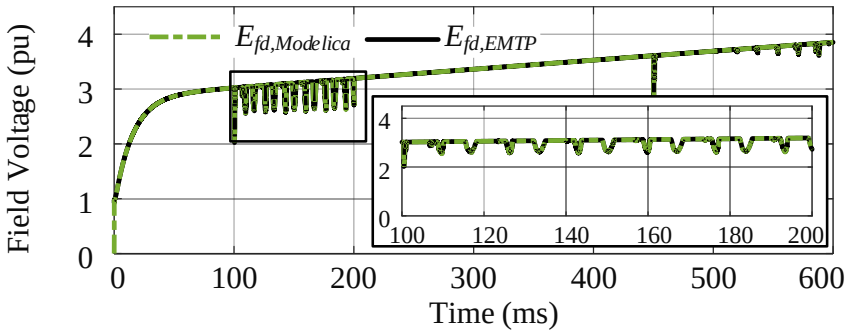


Figure 5.8 Field voltage E_{fd}, regulated by the exciter in PowerPlant_03

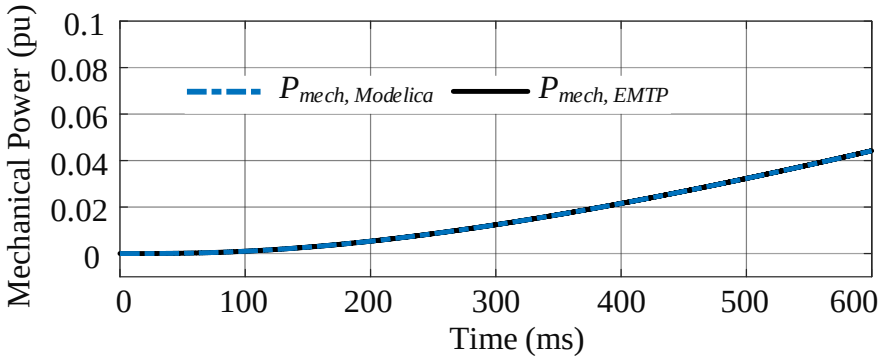


Figure 5.9 Mechanical power P_{mech}, regulated by the governor in PowerPlant_03

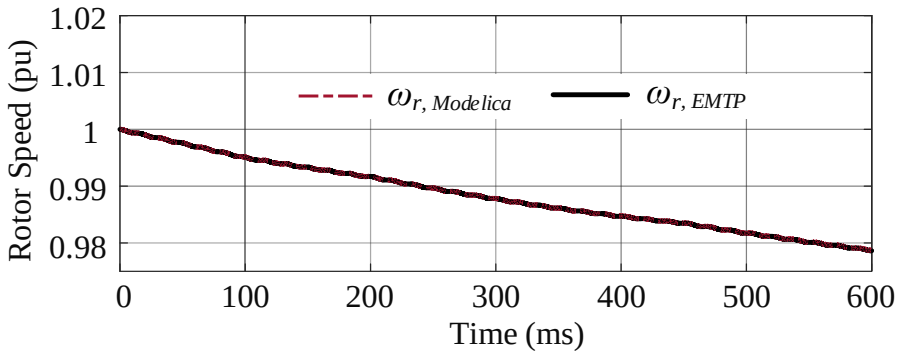


Figure 5.10 Rotor speed ω_r, for the generator in PowerPlant_03

5.6 Runtime Benchmark

The objective is to evaluate the efficiency of the Modelica environment compared with the Simscape Electrical Specialized Power Systems (SPS) [43] library in Simulink (R2020b) and EMTP®. Simulation in Simulink is carried out in discrete mode [43] with Tustin backward Euler solver ($\Delta t = 25\ \mu s$). The variable step solver is ode23tb [109]. It is noted that SPS is comparable to Modelica in its usage of state-space solution of network equations. However, the handling of nonlinearity is different from the Modelica solution method (see Section 1.5.1.2).

For the efficiency of a solver, three parameters are considered: total CPU time, the number of timesteps, and CPU-time for one grid interval (CGI), i.e., CPU time divided by the number of timesteps. Table 5.2 presents the numerical details of the simulation of the IEEE 39-bus network employing the WB-line model. The simulation is repeated with the CP-line model; the solver and

performance details are given in Table 5.3. As one can see, the CPU time of the Modelica environment is not satisfactory compared to EMTP® in both cases; however, the Modelica offers a better runtime than Simulink. Modelica has the lowest CPU time per timestep after EMTP® as well [75].

Although Modelica offers computational speed advantages over existing environments, such as Simscape Electrical (Specialized Power System), its performance is not yet comparable to specialized simulation packages, such as EMTP®.

Table 5.2 Comparison of simulation performance for IEEE 39-bus network using the WB-model

Simulator	**OpenModelica**	**Simulink (SPS)**	**EMTP®**
Solver	IDA	Trap/BE / ode23tb	Trap/BE
Solver type	variable step	discrete / variable step	fixed step
Tolerance	Tol:1e-3	Tol:1e-6	-
Δt	-	25 μs	25 μs
CPU time (s)	9 657	10 620	23.8
Number of time-steps	317 315	24 000	34 741
CGI (ms)	30.43	442.5	0.68

Table 5.3 Comparison of simulation performance for IEEE 39-bus network using CP-model

Simulator	**OpenModelica**	**Simulink (SPS)**	**EMTP®**
Solver	IDA	TBE / ode23tb	Trap/BE
Solver type	variable step	discrete / variable step	fixed step
Tolerance	Tol:1e-6	Tol:1e-6	-
Δt	-	25 μs	25 μs
CPU time (s)	366	1 801	13
Number of time-step s	38 945	24 000	34 704
CGI (ms)	9.39	75.04	0.37

5.7 Conclusion

This work contributed a new approach to the simulation of electromagnetic transients. It is based on the high-level programming environment of Modelica. The new approach is based on modern concepts of programming such as declarative, equation-based, object-oriented paradigms, which are all unified in Modelica.

In this chapter, MSEMT an EMT-detailed library containing linear and nonlinear power electric components was validated. The models yield results identical to those from the EMTP with similar numeral stability and accuracy. It was demonstrated that the proposed models are implemented in a few lines of code, are simply modifiable, expandable, and highly legible. The formulation of models is explicitly based on their true mathematical equations. This achievement has a significant impact on model development efficiency and standards. It is also noted that MSEMT is a powerful environment for power system transients education. Also, Modelica is compatible with the FMI and can be used for co-simulation and model exchange. Although Modelica offers computational speed advantages over existing environments, such as Simscape Electrical (Specialized Power System), its performance is not yet comparable to specialized simulation packages, such as EMTP. Further research is carried out to improve performance.

CHAPTER 6 DYNAωO HYBRID C++/MODELICA SOLUTION

Many techniques have been proposed over the years to accelerate the simulation speed in Modelica simulators, such as using FPGA [115], solver manipulation [116], DAE-mode compilation, power system-specific solvers [88], or efficient Jacobian calculations [117]. Despite these efforts and large improvements, the performance of pure Modelica simulators such as OpenModelica [30], Dymola [31] remains a barrier for industrial applications and large-scale systems [122].

A hybrid C++/Modelica solution called Dynaωo [61], [62], [118] was proposed for simulation in the phasor domain to bypass the limitations encountered with complete Modelica tools while ensuring the advantages of an equation-based approach. Dynaωo is an open-source simulation package primarily designed by RTE for short- and long-term stability analysis. It aims at providing a transparent, flexible, interoperable, and robust simulation tool that could ease collaboration and cooperation in the power system community. This method improves the performances to levels similar to domain-specific simulation tools for phasor-domain simulations [118]. The contribution of this section is to draw the status of Modelica-based EMT simulations using Dynaωo.

6.1 Introduction

The overall goal of the Dynaωo approach is to bypass the limitations of pure Modelica tools for large-scale simulations while keeping the advantages provided by the Modelica approach. It can also be summed up in two main principles: The intent is to use the Modelica language as much as possible for modeling complex elements while sticking to a strict separation between model and solver sides while managing to preserve acceptable performances for industrial use.

To properly understand the design and architecture choices of Dynaωo, it is necessary to recall some characteristics of both the Modelica language and Modelica compilers such as OpenModelica. Modelica has been historically developed for complex but rather small physical problems. Connectivity or graph analysis is difficult and costly to conduct in a pure Modelica approach. Backup solutions using external programming languages, such as C or Fortran, exist but are pretty difficult to connect and integrate into Modelica models. Native generic Modelica tools do both compiling and simulation at run-time. When going to large systems, the compile-time (consisting of different steps such as flattening, sorting, and eventually causalizing the equations –

depending on the compiling mode ODE/DAE) becomes too costly for large-scale simulations. Besides, one should also remember that compiling must be redone even if only parameters are modified. Finally, the generated codes provided by native Modelica compilers remain less efficient and less optimized than manually written codes in a classical programming language. To avoid some of these limitations, Dynaωo uses a hybrid C++/Modelica approach for modeling and a unique method that compiles before run-time partial Modelica models.

Figure 6.1 depicts the structure of Dynaωo. A model can be either directly written in C++ or Modelica. The cunning point in Dynaωo is to temporarily create a square model using fictitious equations for pending connections (typically currents), to be able to compile the models, and then to remove these fictitious equations from the model structure once compiled. It allows compiling models one by one to end up with pre-compiled libraries that are only instantiated at run-time. Moreover, each of these libraries can be used as many times as needed with different parameter values. Once compiled by the OpenModelica compiler, the models are post-processed by Python scripts to provide the same methods and have a single formalism for C++ and Modelica models. The origin of the model is thus entirely transparent for the rest of the tool and the solvers.

Solvers are decoupled from models in Dynaωo. New models can be introduced without further modifications in the solvers, and new solvers can be tested and used without requiring any action on existing models. Moreover, it is straightforward to compare numerical strategies and to observe and analyze the impacts on the results and performances as the modeling side is unchanged. Solvers and models only exchange a finite set of information needed for solving the system. The modeling part notably exposes the following methods to the solving part [61]:

1. the residual functions $\mathbf{f}(t, y, y')$, which are the system equations evaluated at each time step.
2. the Jacobian matrix $\mathbf{J}(t, y, y')$ used for the time-step numerical resolution.
3. the root functions $\mathbf{g}(t, y, y')$, which are used to detect instants of discrete variable changes or mode changes (i.e., a change in the form of an equation from f_1 to f_2, such as a limitation).
4. the mode functions that give the form of an equation at a time t (between f_1 and f_2, for example).

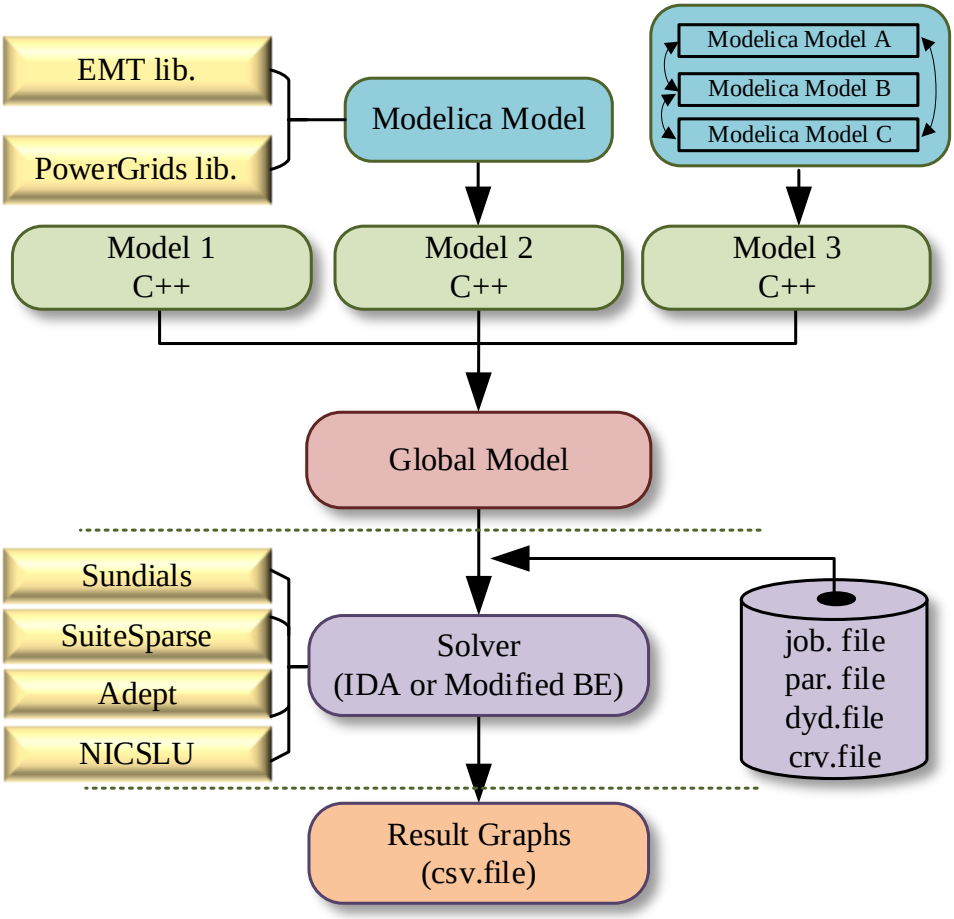


Figure 6.1 Dynaωo structure and exchanges between solvers and models

6.2 Native Models and Solvers

Dynaωo contains a set of models and solvers natively available for any user. The provided models in the Dynaωo library consist of phasor and simplified models for stability analysis, but no EMT model is available. Any solver can be integrated, as long as it contains a few standard methods such as initializing the problem, solving it, or reinitializing it. Currently, two solvers are included in Dynaωo. The first is the Backward Euler integrator with a variable time-step strategy [63], specifically designed for long-term voltage stability simulation.

The second solver is a variable time-step, variable order DAE system solver called IDA [80], a part of the SUNDIALS suite [78]. The integration method in IDA relies on an approximation of the

derivative using the kth order backward differentiation formula (BDF) method given by the multi-step formula (6-1):

$$\sum_{j=0}^{k} \alpha_{n,j} y_{n-j} = h_n \dot{y}_n \tag{6-1}$$

where y_n and $\dot{y}_n$ are the computed approximations to $y(t_n)$ and $\dot{y}(t_n)$, respectively, and the step size is $h_n = t_n - t_{n-1}$. The coefficients $\alpha_{n,j}$ are uniquely determined by the order *k*, and the history of the step sizes. On every step, it chooses the order *k* and step size to control local errors according to user tolerances (relative and absolute): *k* can, in theory, be selected between 1 and 5 but is limited to 1 or 2 in Dynaωo to preserve the A-stability property. Two different LU factorization algorithms, i.e., KLU [119] and NICSLU [120] are coupled with the algebraic solvers. Both have proven [121] efficiency. The IDA has been augmented to include a root-finding feature for event handling while integrating the initial value problem. The scheme is based on checking for sign changes of a set of user-defined functions, $g_i(t, y, \dot{y})$ for each time step. This scheme yields a high precision at the cost of time [78].

6.3 Modifications, Open Questions, and Remaining Challenges for EMT Simulations

To run EMT simulations with Dynaωo, it is necessary to do some modifications in the simulation codes. After adding the EMT library, it is required to enrich the range of Modelica structures in the tool: indeed, some keywords such as "*delay*" or some Modelica functions were not adequately handled by the tool. Once done, a few adjustments also have to be made on the simulation structure and the numerical solver: default values have to be adapted to EMT-type simulations, e.g., time-step minimal values, strategy to reinitialize the solver after an event, or output management. These different changes enable us to compile a large part of the library, and at this stage, no barrier related to the use and support of the Modelica language is identified that could compromise the long-term development of the approach.

Nevertheless, there are still open issues that will need further investigation and research to make definitive statements.

6.4 Simulations and Results

Three case studies have been used to validate the behavior of Dynaωo, enriched by the modifications presented in the last section, in terms of accuracy and performance. The obtained results and the simulation time are compared with the reference software EMTP®–with the Trapezoidal and Backward Euler (BE) method–and a native open-source Modelica tool – OpenModelica. Code generation and simulations were carried out on a laptop with Intel Core i7-6820HQ 2.7 GHz 4 cores - CPU with HT; 62 GB DDR4 main memory; running on Fedora 29 and using OpenModelica 1.14.1 and Dynaωo 1.2. The simulations are performed without initialization.

6.4.1 Case 1: Capacitor Bank Switching

The schematic for a capacitor bank switching in a 230 kV substation designed in OpenModelica using the MSEMT library [75] is presented in Figure 6.2. This case exhibits both low and high natural frequencies. It aims at studying how well the solution method performs for *stiff DAE systems*. The two breakers in Figure 6.2 are initially open. CB1 is closed at t=20 ms, which introduces high-frequency transient oscillations. CB1 is then opened at t=125 ms and recloses at t=175 ms. The capacitor C2 is energized at t=225 ms. The simulation interval is 500 ms with a time-step of 10 µs.

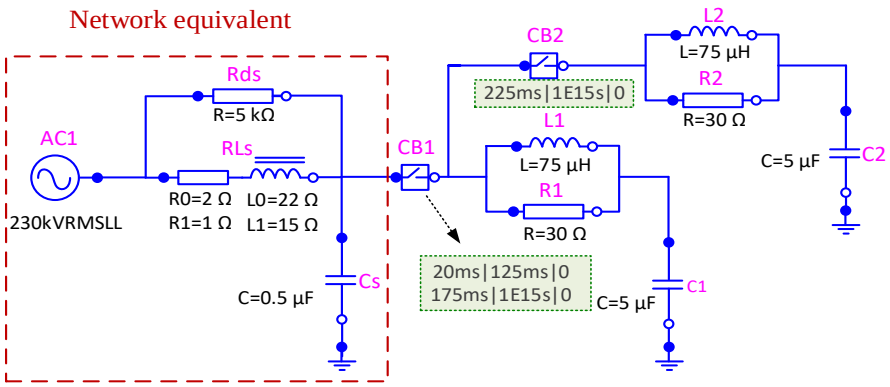


Figure 6.2 Test circuit 1: 2-step back-to-back capacitor banks designed using MSEMT in OpenModelica

Figure 6.3.(a) superimposes the voltage curves at C1 from Dynaωo and EMTP® for the first 300 ms. Close-up views of reclosing of CB1 and closing of CB2 are given in Figure 6.3.(b)-(d). It is

observed that Dynaωo results perfectly match the EMTP® during transients. At each switching, two transient events are observable: low frequency and high-frequency oscillations. For example, energizing C1 causes oscillations with frequencies of 27.26 kHz (it is not observable with th) and 340 Hz (see Figure 6.3.(c)), respectively. At the instant of closing of CB2, the fast transient is 8220 Hz, whereas the slower transient is 246 Hz, as observed in Figure 6.3.(d) and Figure 6.3.(b), respectively. No numerical instability, e.g., numerical oscillations, are identified during the simulation.

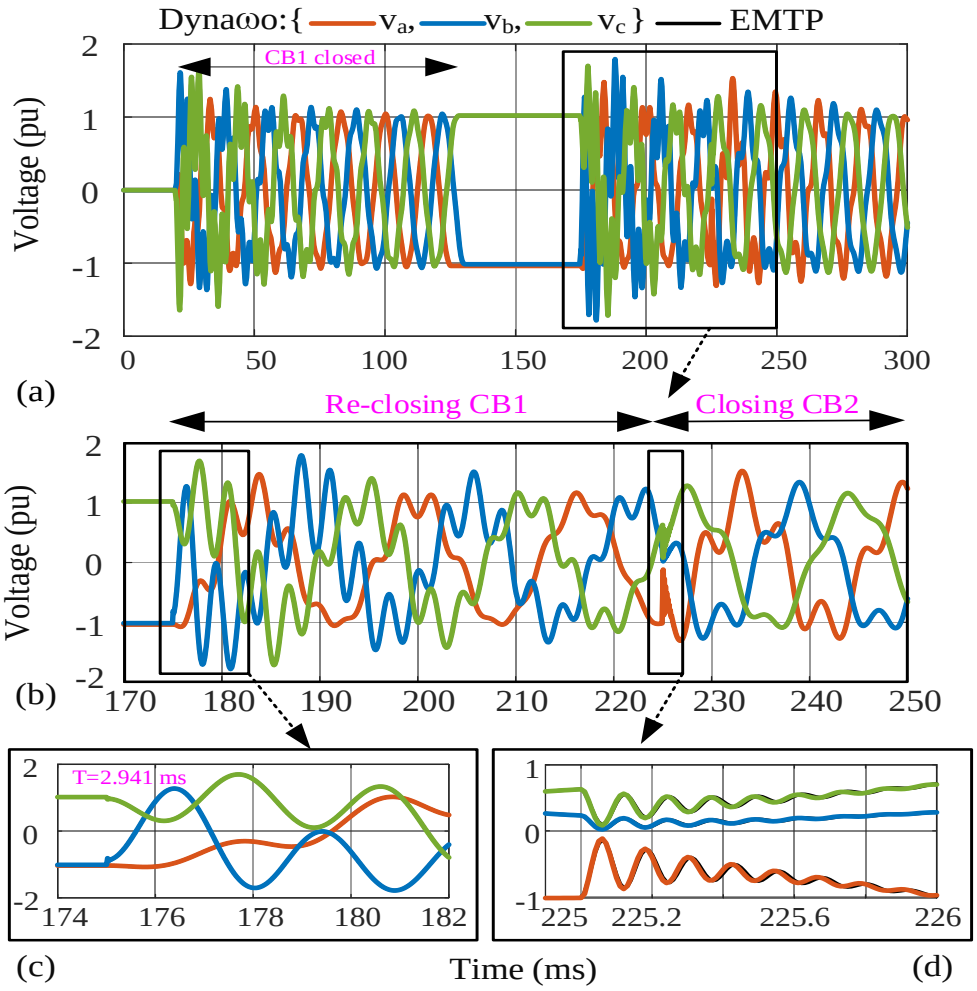


Figure 6.3 (a): Voltage waveforms on C1; Dynaωo solver: IDA, $\Delta t_{max} = 10$ µs, Tol=1e-6; EMTP® solver: Trapezoidal/BE, $\Delta t = 10$ µs. (b): Zoom-in view of voltage curves after reclosing of CB1 and closing CB2. (c): Low-frequency oscillations of 340 Hz. (d): High-frequency oscillations of 8220 Hz due to energization C2

Table 6.1 presents the performances obtained for Dynaωo and OpenModelica when using the IDA solver with the following parameters: initial time-step and maximum time-step is 10 µs, relative and absolute accuracy are 1e-6, and the maximum order is 2. One should also note that IDA has been modified in Dynaωo to introduce a minimum step size: its value is set to 1e-10 s in our case. Results are compared with EMTP® performance obtained with a fixed time-step of 10 µs. The simulations have been run 5 times, and the average computing time is extracted. It shows that the simulation time in both Modelica-based tools is similar, which is logical as the solver properties and the models used are identical. OpenModelica performs better on the pure solving aspects: one possible explanation is the handling of the Jacobian calculation; in Dynaωo, the Jacobian is evaluated using numerical differentiation while it is directly available in the OpenModelica environment. Nevertheless, when adding front-end and back-end times and especially the compilation time, Dynaωo becomes 1.79 times faster than OpenModelica [122].

Table 6.1 Case study 1: Performance comparison

Simulator	**Dynaωo**	**OpenModelica**			**EMTP®**
		Comp.	Sim.	Total (C+S+AP)	
CPU-time(s)	2.34	1.59	2.11	4.21	0.5

Table 6.2 presents the characteristics of the simulations carried out in Dynaωo and OpenModelica, especially the number of time-steps solved, the number of Jacobian evaluations, and the number of residual equations. It confirms that the overall behavior of IDA in OpenModelica and Dynaωo is the same, even if slight differences appear due to the precision chosen for event detection and the equation simplifications in both tools.

Table 6.2 Case study 1: IDA behavior during simulation

Simulator	**Dynaωo**	**OpenModelica**	**EMTP®**
Number of time-steps	90 818	119 749	50 008
J evaluations	2 963	2 963	-
F evaluations	121 481	135 394	-

To further evaluate the possibilities of the simulation tool, the simulations have been relaunched with different sets of parameters. Performances and accuracy sensitivity of results for different tolerances with IDA have been assessed. Table 6.3 shows the performance aspects, while Figure 6.4.(a) focuses on accuracy. This figure depicts the high-frequency oscillations of voltage phase-*a* on C1 during energizing C2. The number of time points, $n_{\Delta t}$, for different solvers is compared in Figure 6.4.(b). It is observed in the curves obtained by the IDA solver, the number of time points varies depending on the rate of changes on the curve, and tolerance; e.g. $n_{\Delta t,\ red} > n_{\Delta t,\ green} > n_{\Delta t,\ blue}$ and also $n_{\Delta t,a} > n_{\Delta t,b}$. The IDA solver with the tolerance of 1e-6 yields the closest results to EMTP® with a time-step of 1 µs whose CPU-time is 3.94 s. Thus, user-defined precision is a key and determining parameter for selecting the step size.

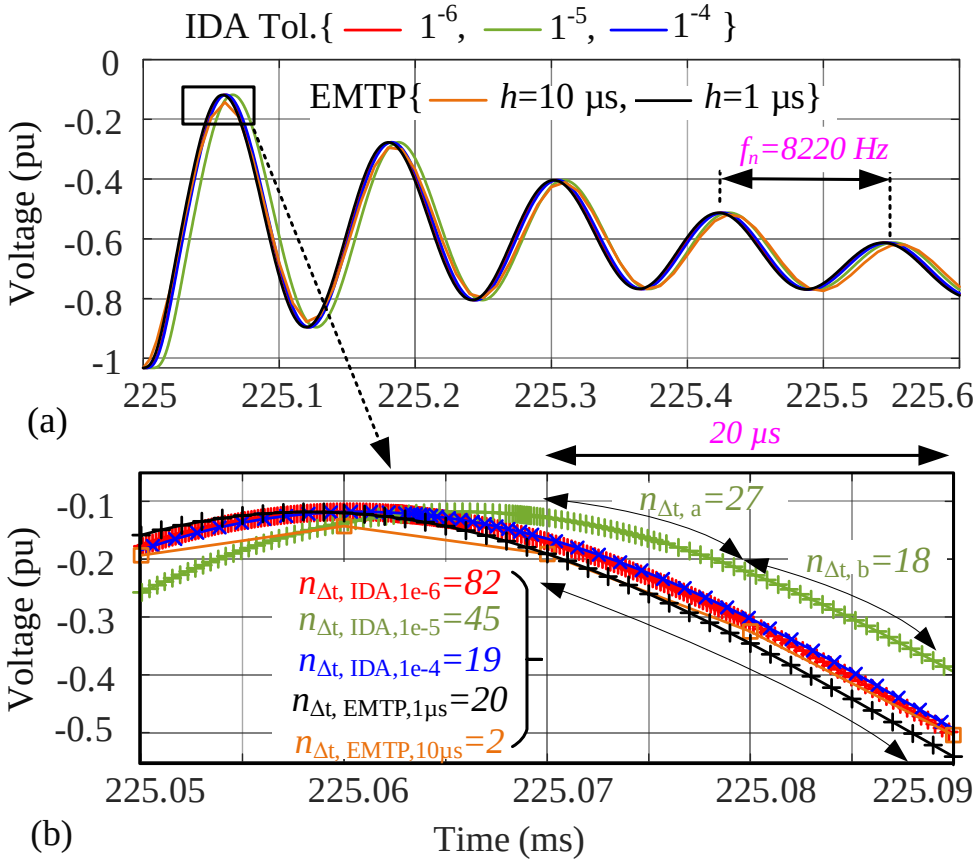


Figure 6.4 (a): Voltage waveforms on C1, phase-*a* at the instant of C2 energization, Dynaωo solver: IDA with different tolerances; EMTP® solver: Trapezoidal/BE, Δt =1 and 10 µs. (b): Comparison of the number of time points within 20 µs

Table 6.3 Performances for different solving strategies

Solver	CPU-time (s)	Gain (compared to IDA, tolerance = 1e-6)
IDA (tolerance = 1e-6)	2.34	1
IDA (tolerance = 1e-5)	1.43	1.63
IDA (tolerance = 1e-4)	1.02	2.29

6.4.2 Case 2: Parallel Transmission Line Switching

Figure 6.5 shows a network equivalent (coupled-RL) feeding a balanced three-phase PQ load of 500 MW and 100 MVAR at 400 kV through two identical parallel lines.

The breaker BR1 is initially open and closes at t=0 s. TLM1 and TLM2 are constant-parameter (CP) line models. In normal conditions, the line breakers are closed. L1 represents a shunt compensator. The load is connected to Bus BOR at t=100 ms. A phase-a-to ground fault with a resistance of 1 Ω is applied to the TML2 at t=200 ms. Immediately after detection of the fault by the protection relays (not simulated here), an opening command is sent to the breakers BRm2 and BRk2 at t=300 ms. Then, the fault is cleared at t=350 ms, and finally, the line breakers are reclosed at 430 ms. The simulation time and time-step are set to 500 ms and 5 µs, respectively.

This scenario aims at validating the accuracy of the *delay* operator developed in Dynaωo and the stability of the solver over discontinuities imposed by several *state events*.

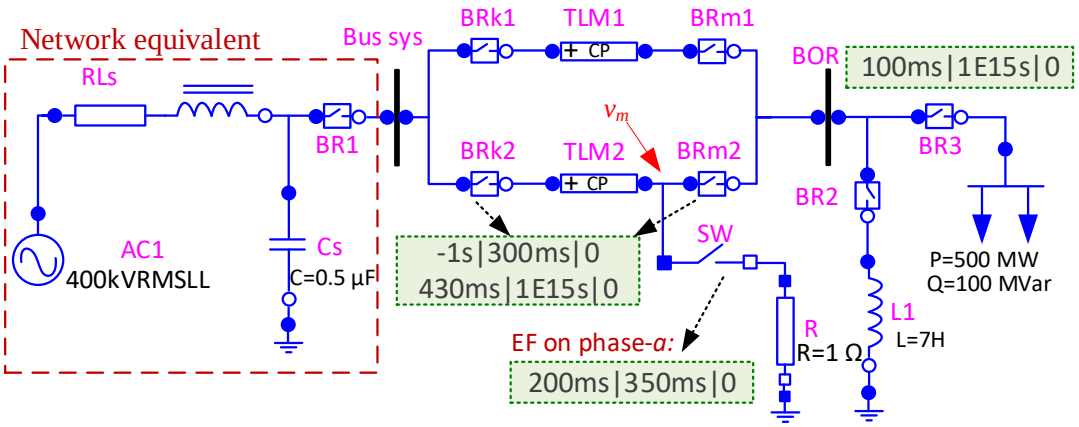


Figure 6.5 Test circuit 2, switching of parallel transmission lines (CP-line model)

Figure 6.6 depicts the voltage waveforms at the *m*-end of TLM2. The black curves represent EMTP® results. It is observed that both curves are in excellent agreement.

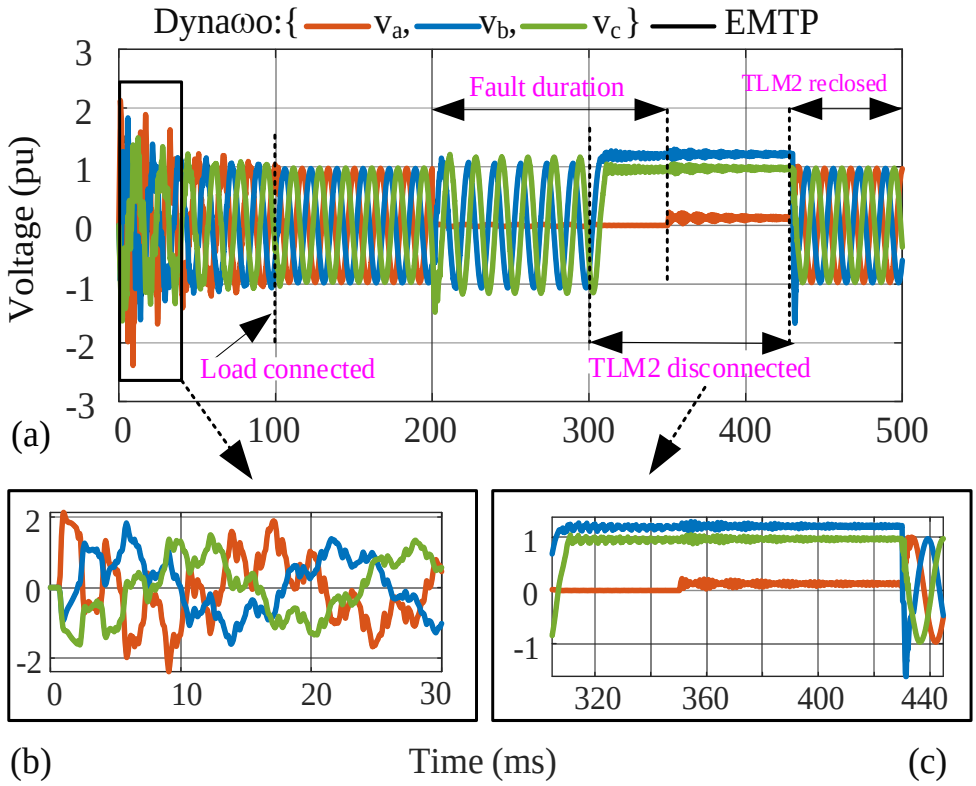


Figure 6.6 (a): Voltage waveforms at the *m*-end of TLM2; Dynaωo solver: IDA, $\Delta t_{max} = 5$ µs, Tol=1e-6; EMTP® solver: Trapezoidal/BE, $\Delta t = 5$ µs. (b): The close-up view of the energization of the line. (c): The zoom-in view of voltage at the *m*-end of TLM2 when disconnected from both sides

Figure 6.7.(a) illustrates the current waveforms passing through the *m*-end of TLM2. Figure 6.7.(b) zoom in the transients after disconnecting the line. It shows the impact of traveling waves in phase-*a* and repeats nearly at each 2τ. The current continues oscillating and decreasing- due to the resistances of line and fault-until the SW is opened. Figure 6.7.(c) shows the transients at the instant of re-energizing TLM2. One can observe that the results match the EMTP® curves fully.

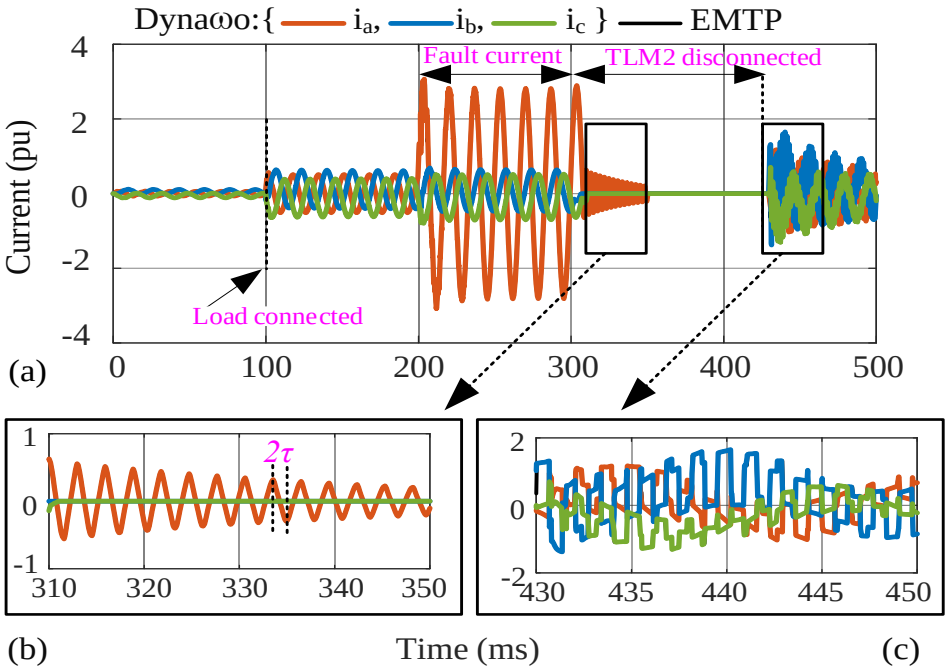


Figure 6.7 (a): Current waveforms at the *m*-end of TLM2. (b): The zoom-in view of current at the *m*-end of TLM2 after disconnecting the line. (c): The zoom-in view of current at the *m*-end of TLM2 at the instant of energizing the line

Similar to Case 1, Table 6.4 reports the performances obtained for Dynaωo and OpenModelica when using the IDA solver with the following parameters: initial time-step and maximum time-step is 5 µs, relative and absolute accuracies are 1e-6, and the maximum order is 2. The same network is simulated with EMTP® with the time step of 5 µs. One can see that Dynaωo presents an overall better performance of simulations compared to OpenModelica. In this case, the use of a variable time-step solver and the number of Jacobian evaluations, 16,042, are the most penalizing points. It is noted that $n_{\Delta t,\ Dyna\omega o} = 209{,}871$ and $n_{\Delta t,\ EMTP} = 100{,}010$.

Table 6.4 Case study 2: Performance comparison

Simulator	**Dynaωo**	**OpenModelica**			**EMTP®**
		Comp.	Sim.	Total (C+S)	
CPU-time (s)	18.74	5.31	13.6	19.46	1.6

6.4.3 Case 3: Nonlinear Circuit of Surge Arrester

This case study aims to examine the behavior of Dynaωo for the simulation of nonlinear components during very fast transients. The solution of nonlinear systems is accomplished with Newton iterations in Dynaωo and EMTP® solvers.

Figure 6.8 shows the frequency-dependent model proposed by the IEEE W.G. 3.4.11[123] for surge arrester modeling. The model represents the arrester as two *highly nonlinear resistors*, ZnO1 and ZnO2, separated by an R-L filter. For slow front surges, the R-L filter is negligible. Thus, ZnO1 and ZnO2 are effectively connected in parallel. For fast-front surges, the impedance of this filter becomes more important and causes a current distribution between the two nonlinear branches.

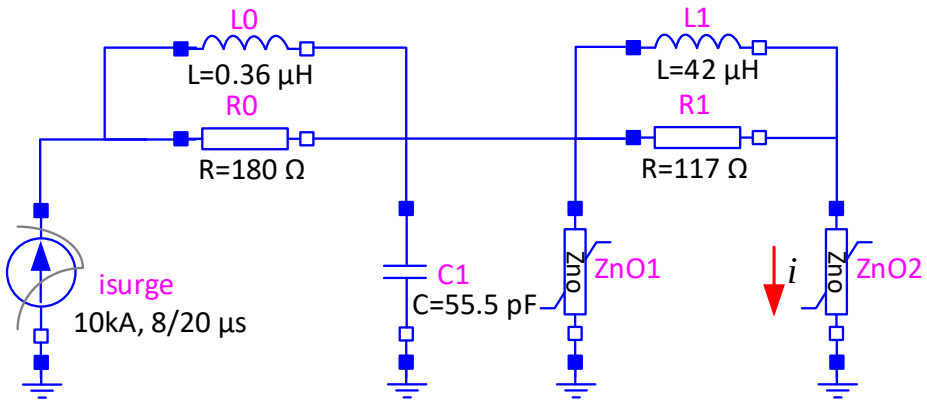


Figure 6.8 Test circuit 3; modeling of an Ohio-Brass ZnO Arrester for a 330 kV Network, MCOV=209 kV, d=1.8 m, n=1

Simulation is run for 300 µs with $\Delta t_{max} = 10\,ns$, $Tol = 10^{-6}$ in Dynaωo and $\Delta t = 10\,ns$ in EMTP®. Figure 6.9 illustrates the voltage and current waveforms of ZnO2 compared with EMTP®. The graphs are fully superimposed. Figure 6.10 shows the solution points on the non-linear characteristic curve of ZnO2. The solution points are not superimposed but are on the same slope. The solutions always remain on the actual nonlinear segments; no overshooting is observed. There are no numerical oscillations and instability. The simulation time for different simulators is presented in Table 6.5. IDA solves the system with the total number of 31 790 solution points while in a fixed-step solver, e.g., Trapezoidal/BE $n_{\Delta t} = 30{,}019$.

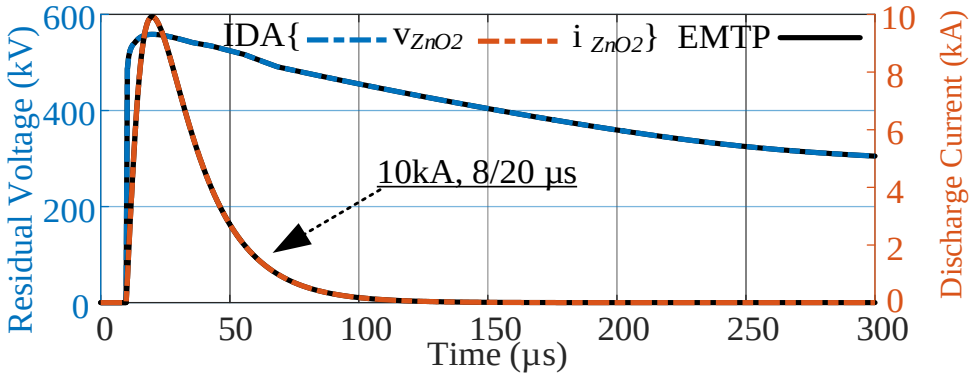


Figure 6.9 Residual voltage and discharge current curves in ZnO2. Dynaωo solver: IDA, $\Delta t_{max} = 10ns$, Tol=1e-6; EMTP®: Trapezoidal/BE, $\Delta t = 10ns$

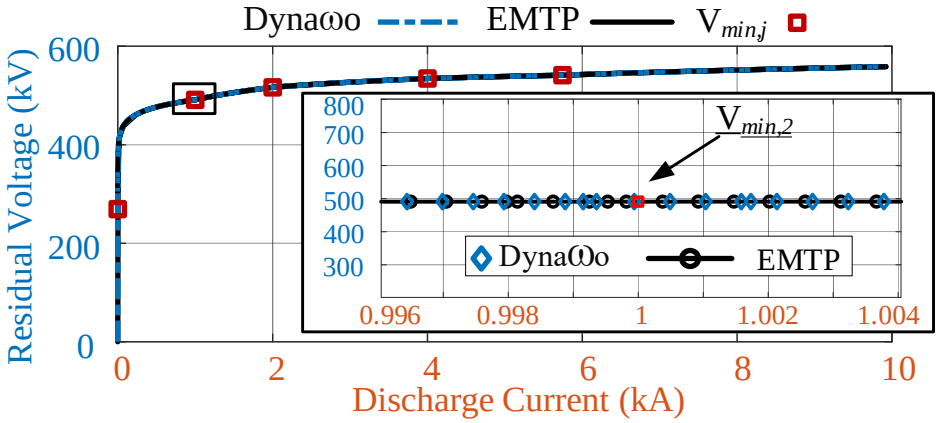


Figure 6.10 Voltage vs. current curve of ZnO2; Zoom-in view: comparison of solution points in the nonlinear segment 2

Table 6.5 Case study 3: Performance comparison

Simulator	**Dynaωo**	**OpenModelica**			**EMTP®**
		Comp.	Sim.	Total (C+S)	
CPU-time (s)	0.19	0.02	0.15	0.17	0.17

6.5 Conclusions

Modelica is a powerful modeling language for power system simulation based on describing the models by implicit DAEs. This chapter demonstrated a hybrid approach to EMT simulations using Modelica and C++. The new approach contributes to improving the run-time of EMT-type simulation in Modelica.

The method is based on modern programming concepts such as declarative, equation-based, object-oriented paradigms, where all unified in Modelica. The improved approach has been validated in terms of accuracy and solution speed using EMTP®. The results show that the obtained performance is better than pure Modelica tools, e.g., OpenModelica. The obtained results for all three cases also confirm the numerical stability of IDA for stiff systems, notably including components with nonlinear characteristics.

The advantages of Dynaωo are not in numerical performance when compared to EMTP® but in high-level modeling capabilities. However, it is shown that performance improvements are possible, and further research is being conducted on this aspect.

CHAPTER 7 ELECTROMAGNETIC TRANSIENT MODELING OF LARGE POWER NETWORKS WITH MODELICA

7.1 Introduction

This chapter is designed based on two purposes, first analysis of electromagnetic transient simulations in a large network. For this goal, the IEEE 118-bus network is proposed. The second purpose is to focus on the accuracy and performance of nonlinear models, including the synchronous machines with magnetic saturation (see Section 3.5), surge arrester (see Section 3.6.2.3), and, finally, arc models (see Section 3.6.2.4).

The IEEE 118-bus benchmark [109] contains the following models: synchronous generators (including saturation model) with controls, transformers, transmission lines, nonlinear inductances, and nonlinear surge arresters. The basic models, such as resistance, inductance, and advanced models, e.g., various models of transmission line, loads, saturable transformers, synchronous machine (without saturation), Controls (machine controls), etc., were already presented in previous works [107], [122]. This work focuses on the synchronous machine model with saturation and the nonlinear arrester.

This chapter presents simulation results of the modified IEEE 118-bus benchmark [108], which is used to validate the accuracy of the proposed models. The same test case is also simulated with EMTP®. The results are compared using the PI-section models for transmission lines. Figure 7.1.(a) shows the schematic diagram of the IEEE 118-bus network designed using the developed MSEMT library in Modelica. A user-friendly graphical user interface (GUI) with an illustrative icon is designed for each component model for entering the parameters and drawing networks quickly. The physical connection of components is carried out by interconnecting the terminals of appropriate components.

The IEEE 118-bus circuit consists of 54 generating units with controls (a few power plants contain more than one SM; the total number of SMs is 69), 177 transmission lines (RL coupled), 9 three-winding grid transformers, 145 two-winding transformers (91 Yd1-connected load-serving transformers+ 54 generator transformers), and 91 three-phase loads. The voltage levels are 345kV transmission, 138kV sub-transmission, 25kV distribution, and {20, 15, 10.5} kV generation. The lines are modeled using PI sections. It was impossible to use constant parameter line models with

propagation delay because of the very high computational cost of the Modelica built-in *delay* operator. Simulation in both simulators, i.e., EMTP® and Modelica, start with zero initial states. The network includes 519 nonlinear inductances and 1909 RLC elements. All SMs use a single-mass Wye-grounded model, including the normalized saturation characteristics. The models of all three-phase transformers consist of single-phase units. The nonlinear magnetization branch is placed on the high-voltage side. The model uses a piecewise linearly interpolated curve defined by 8 points to represent saturation. All loads are represented by a constant impedance model.

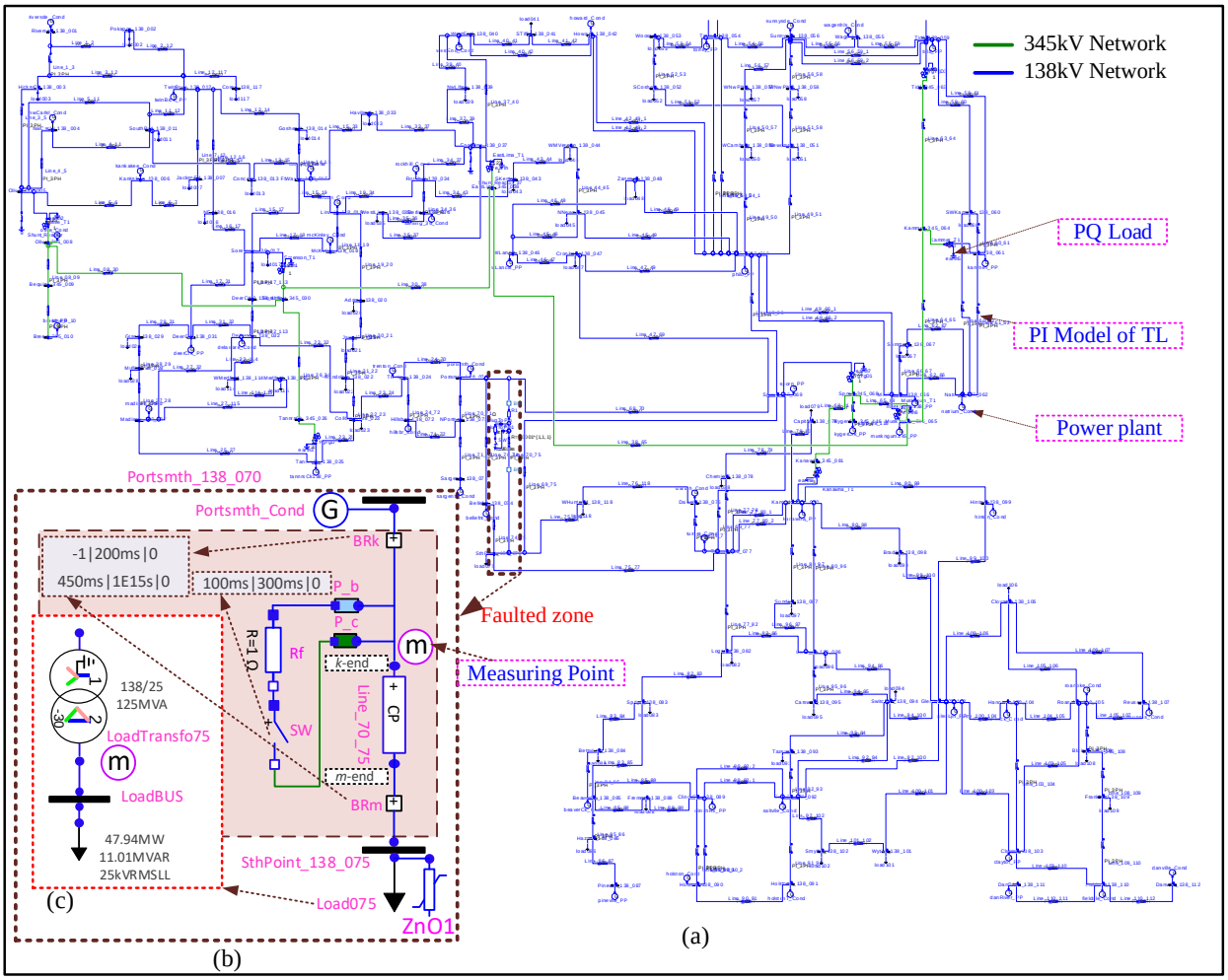


Figure 7.1 (a): IEEE 118-bus Network including 177 PI-section models of TL sketched using the Modelica GUI. (b): the faulty zone; a phase-*b*-to-phase-*c* fault at *k*-end of Line_70_75. The powerplant "Portsmth_Cond" is selected for validation of SM with saturation in Case 2, Surge arrester ZnO1 is inserted in the circuit only for Case 3. (c): the sub-circuit of Load75 including a saturable transformer model and constant-impedance model of load

7.2 Case 1: Phase-to-Phase Fault Analysis

For creating a transient disturbance, (see Figure 7.1.(b)), a temporary phase-to-phase fault with a fault resistance of 1Ω is applied on the phases '*b*' and '*c*' of "Line_70_75" at *t* =100 ms followed by the isolation of the line at *t*= 200 ms (i.e., the breakers BRm and BRk open simultaneously after 6 cycles). The fault is cleared at *t* = 300 ms; then, the line is reconnected at *t* = 450 ms.

Re-energizing the TL introduces high-frequency transient oscillations and allows to investigate the accuracy of transformer models in nonlinear regions. For this purpose, the curve of flux versus current for LoadTransfo75, which is located near the faulty line, is compared with EMTP®.

Numerical tests are performed using the variable-step DASSL solver [77] in ODE mode with the tolerance of 1e-3 and the maximum integration order of 5 in Dymola 2021x. In EMTP®, Trapezoidal/backward Euler integrator with the step sizes of 1 μs and 5 μs is employed. The simulation time is 500 ms. The network model in Modelica contains 96308 acausal DAEs. The total number of network nodes and the size of the main system of equations in EMTP® are 2533 and 3773, respectively.

Figure 7.2.(a) depicts the voltage waveforms of phases-*a*,-*b* and-*c* at the *k*-end of Line_70_75 obtained by the two simulators with different precisions. An excellent agreement is observed between the results. Figure 7.2.(b) shows the simulation results for phases-*b* and-*c* in the interval of [300, 310] ms, i.e., after the fault is removed. The results produced by Modelica models are almost identical to EMTP® when step size is 1 μs (black curve), while the high-frequency transient oscillations (*f*=1820 Hz) are not captured by EMTP® when $\Delta t = 5$ μs (blue curve). Figure 7.2.(c) depicts the curves of voltage after the re-energization of TL. The consistent results between Modelica and EMTP® are observed in this period once more. The close-up view of the phase a voltage waveform at the instant of closing the breakers BRm and BRk shows that Modelica voltage waveform rises precisely at *t* = 450 ms while in EMTP®, it goes up the next time point. The close-up illustrates the discontinuity treatment discrepancies between the two simulators. This is an important issue for the simulation of circuits with high-frequency switching.

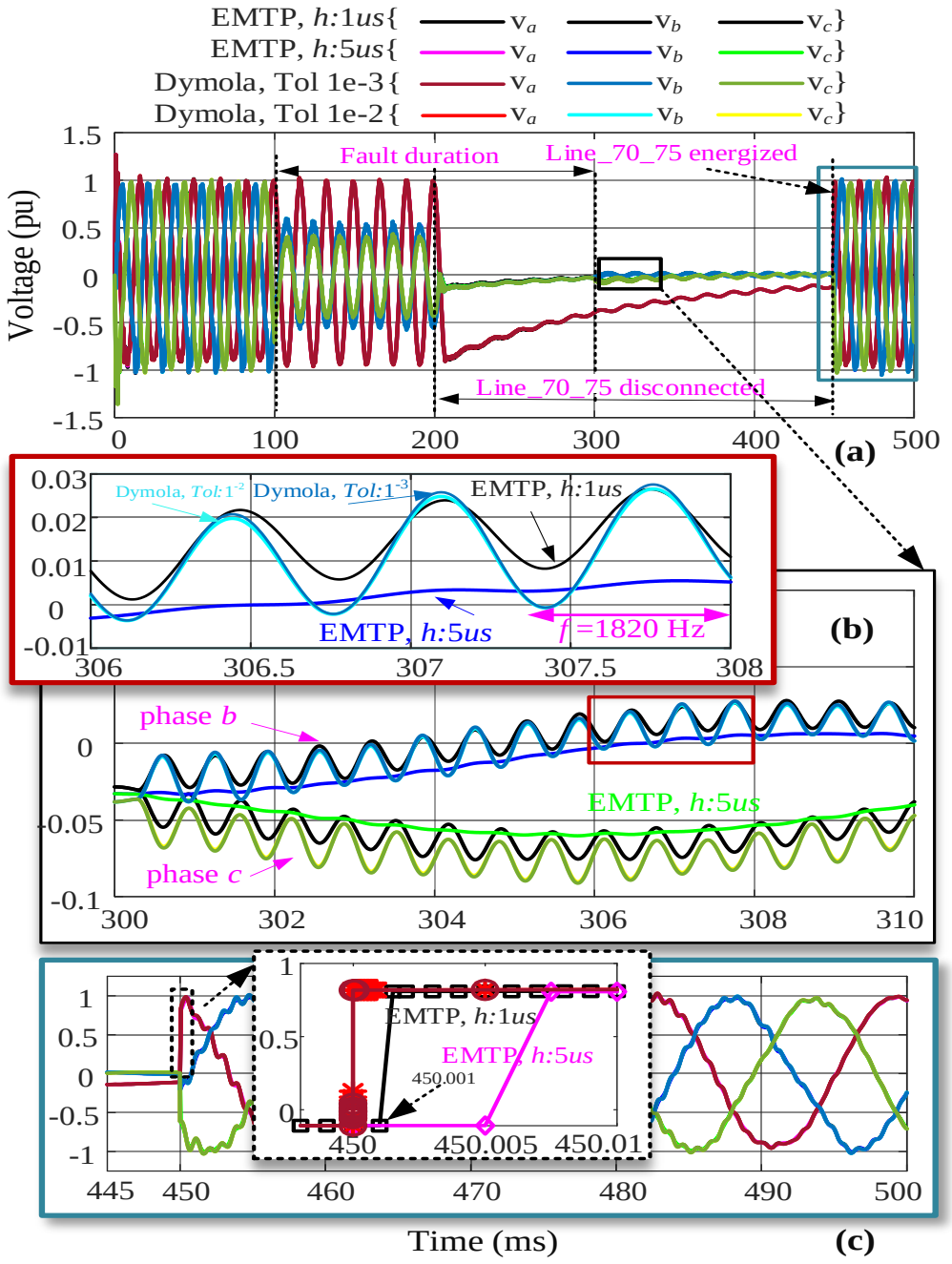


Figure 7.2. (a): Voltage waveforms of phases-*a*, -*b* and -*c* at the *k*-end of Line_70_75; (b): comparison of results for the phases-*b* and-*c* for different solvers' parameters. (c): voltage waveforms after re-energization of Line_70_75; the close-up at the instant of closing the breakers BRk and BRm

For validating the accuracy of nonlinear components, the magnetization branch curve of LoadTransfo75 (see Figure 7.1.(c)) is examined in Figure 7.3. Once again, the results obtained by the two models show an excellent agreement, and transformer operating points (depicted by the red dashed line) move on the transformer current-flux characteristics (distinguished by the solid red line). The iterative solution allows reproducing the nonlinear function accurately in both tools.

The number of nonlinear components and control closed loops has a significant impact on the accuracy and speed of simulation. For example, simulation of the same network, that is IEEE 118-bus, jams in Simscape Electrical Specialized Power Systems (SPS) package [43], which is comparable to the Modelica environment (both are based on state-space modeling approach). This package is based on the state-space representation of the linear network in a loop with external current sources denoting the nonlinear components. In Modelica, nonlinear functions are solved simultaneously through iterative methods, which gives the most accurate results.

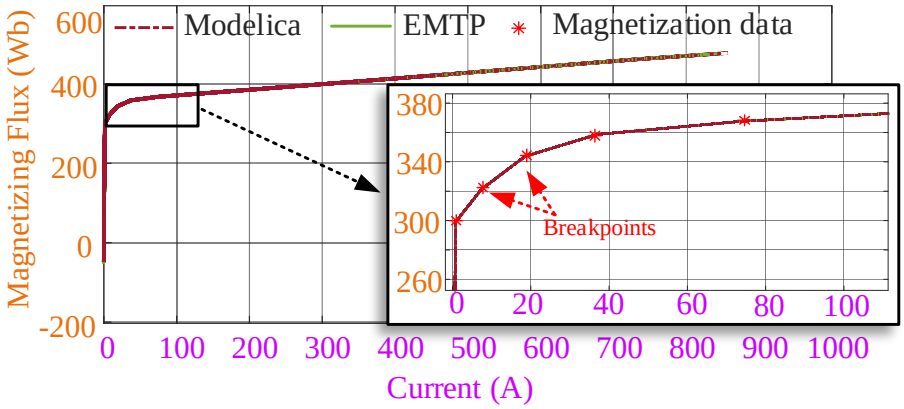


Figure 7.3 Current-Flux curve of magnetization branch in the LoadTransfo75 transformer; zoom-in on the knee-point solutions of Modelica and EMTP®

Table 7.1 shows the data and run-times of simulations carried out in Dymola and EMTP®. The CPU times are extracted from the average of 5-times "re-simulations." In Dymola, simulation is accomplished with 203 034 steps in 371.2 s, yielding 1.83 ms for each step. EMTP® outperforms Dymola with the ratio of 3.37:1 when the least error is favorite, i.e., $\Delta t = 1$ µs. Tolerance significantly impacts the CPU time and the number of time steps for the DASSL since the local error is tightly coupled with the logic for selecting the step size and order of integration. In this experiment, the simulation is repeated with the tolerance of 1e-2 as well. It causes a considerable

increase in the number of time steps, Jacobian, and function evaluations. Consequently, the CPU time increases with the ratio of 4:1, whereas the simulation accuracy does not change effectively (see Figure 7.2.(b)). The norm of error between these two simulations is reported 4.8e-3 for phase *b*. In both tolerances, the results are practically identical to EMTP® when $\Delta t = 1\ \mu s$. However, it should be noted that the solution methods in Modelica and EMTP® are fundamentally different, and a direct comparison of variable step solver with fixed-step one is not so fair. The time steps selected in Table 7.1 are for demonstration/comparison purposes; in reality, it is possible to choose even higher time steps without a significant loss of accuracy.

Table 7.1 Case 1: comparison of simulation performance

Characteristics	**Dymola**		**EMTP®**		
Solver	DASSL		Trapezoidal /Backward Euler		
Tolerance	1e-3	1e-2	Δt: 1 μs	Δt: 5 μs	Δt: 10 μs
Δt_{MIN}	0.115 fs	0.116 fs			
Δt_{MAX}	5.79 μs	0.16 μs			
No result points	203035	335261	601757	154367	81 661
No accepted steps	203034	335260	-		
f-evaluations	415437	760052	-		
J-evaluations	7393	337458	-		
CPU time (s)	371.2	1510.6	110.1	44.2	23.5
CPU-time for 1 step	1.83 ms	4.49 ms	0.18 ms	0.28 ms	0.28 ms
Performance ratio	1	0.24	3.37	8.39	15.79

7.3 Case 2: Analysis of Saturation in SM

This test case aims to verify the validity of the SM model in the saturation region; a large disturbance, including a sudden three-phase short-circuit, is applied at *t* = 50 ms near the terminals of SM "Portsmth_Cond" and lasts till *t* =150 ms. The SM protective relays detect the fault and trip the generator breaker at *t* = 200 ms. The parameters of both solvers are the same as in Case 1. Figure 7.4.(a) and (b) depict the graphs of phase-*a* terminal voltage and field current of the SM with and without saturation. As one can see, the results obtained from Modelica are superimposed on EMTP® ones. As time elapses, the difference between saturated and unsaturated curves is more distinguishable on the voltage curves (see Figure 7.4.(a)). It is observed from Figure 7.4.(b) that inclusion of saturation has an important impact on the excitation current needed for the generator operation.

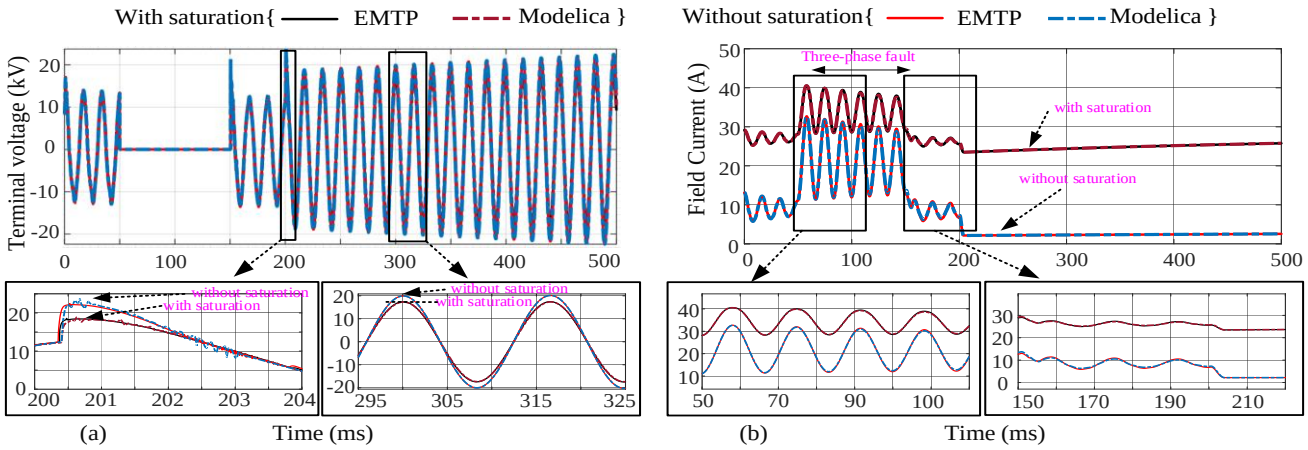


Figure 7.4 (a): Waveform of phase-*a* of stator voltage of SM with and without saturation model; the close-up views after load rejection. (b): Field current with and without saturation model; the zoomed views during and after the fault

Figure 7.5 illustrates the phase-*a* current of the stator. As one can see, the Modelica model yields the same results as EMTP® for both cases (with and without saturation). The current with saturation is lower than without saturation. It is seen that the effect of saturation on the current under the sub-transient state is more than the transient state and the difference decreases as time elapses.

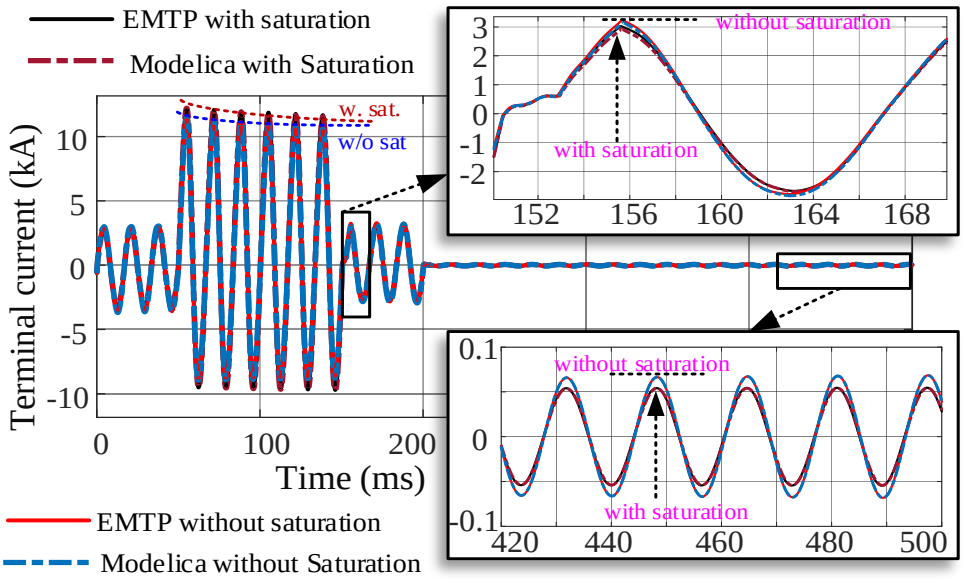


Figure 7.5 Phase-*a* stator current with and without saturation model; zoomed view after removing the fault and load rejection

7.4 Case 3: Lightning

In this case, it is assumed that lightning with the characteristics of 10kA, 8 /20 µs strikes phase-*a* of the line "TL_70_75" when the network is in steady-state at t = 95 ms. The surge arresters are located on the bus "SthPoint_138_075" to protect the high-voltage side of the transformer "LoadTransfo75" from transient overvoltages (see Figure 7.1.(b)). The simulation is run for 130 ms with the step sizes of 0.1 µs (depicted by black curve) and 10 µs (shown by red curve) in EMTP®. Other parameters of solvers in both tools are as the Case 1. Figure 7.6 shows the phase-*a* voltage waveform of the arrester ZnO1. As one can see, the results obtained from the Modelica arrester model are identical to EMTP® when $\Delta t = 0.1\ \mu s$. A high frequency transient (1300 Hz) is observed due to the strike of lightning.

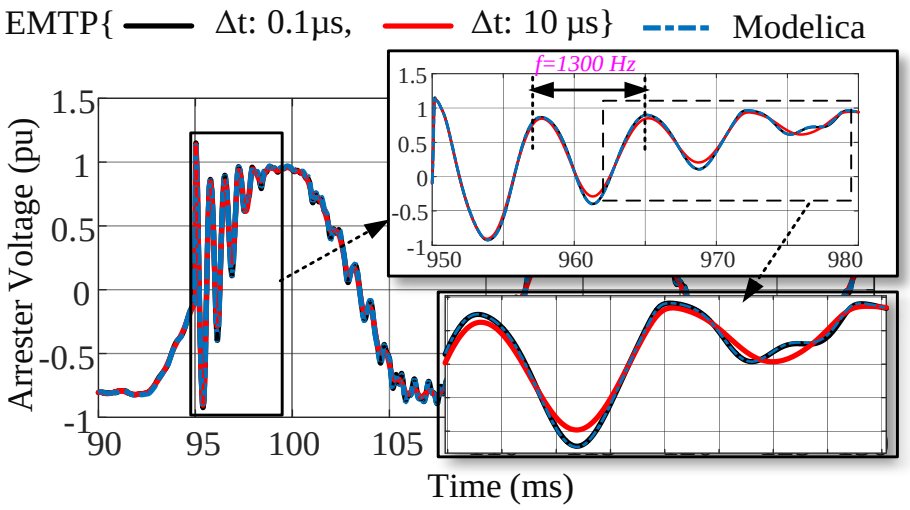


Figure 7.6 Voltage waveform of surge arrester ZnO1 on the bus SthPoint_138_075, DASLL solver: Tol=1e-3, EMTP® solver: Trapezoidal /backward Euler with Δt=0.1 µs and 10 µs

Table 7.2 compares the performances of simulations in both tools. In Dymola, simulation is accomplished with 51513 steps in 87.2 s, which yields 1.69 ms for each time step. In this case, Dymola outperforms EMTP®'s best result, that is when $\Delta t = 0.1$ µs, with the ratio of 5.56:1.

This test case is designed to show the potential advantages of variable time step solvers over fixed time step ones (like EMTP®). It is intended to illustrate that a very small time step used for the short duration of the very high transient has a penalizing effect on EMTP®, but not on Modelica

solver. Modelica integrator expectedly reverts to a very small time step only for a short duration. It would have been possible to apply lightning in EMTP® at simulation time t = 0 s, and in which case the performance results would have been much better; nevertheless, our demonstration remains valid. A more practical example is the breaker arc model that also forces the usage of very small time steps and may be triggered at any point of time. It will effectively give an advantage to Modelica since, in this case, it is required to capture more extended simulation periods.

Table 7.2 Case 3: comparison of simulation performance

Characteristics	**Dymola**	**EMTP®**	
Solver	DASSL	Trapezoidal /BE	
Tolerance	1e-3	Δt: 0.1 μs	Δt: 10 μs
Δt_{MIN}	0.623 ps		
Δt_{MAX}	5.56 μs		
No result points	51514	1335308	21637
No accepted steps	51513	-	
f-evaluations	105576	-	
J-evaluations	1503	-	
CPU time (s)	87.2	485.6	9.9
CPU-time for 1 step	1.69 ms	0.36 ms	0.45 ms
Performance ratio	1	0.179	8.8

7.5 Evaluation of Arc Models

In this section, the accuracy of arc models developed in MSEMT library will be investigated in different circuits. The arc model is a highly nonlinear component that plays an essential role in analyzing transient recovery voltage. In the following sections, the models of Cassie, Mayr, and combination of both models will be evaluated.

7.5.1 Comparison of Cassie and Mayr Arc Models

Arc model was explained in Section 3.6.2.4. In this section, verification of proposed Modelica models using the EMTP® is addressed. Figure 7.7 shows two identical circuits reproduced from [43] using the MSEMT library to test arc models' accuracy: one with Mayr and one with Cassie arc model. The circuit can be a simple representation of a circuit breaker interrupting a short-line fault. The circuit is energized with a sinusoidal voltage source with the magnitude of 41.8 kV RMS, 60 Hz. The circuit at the left side of the circuit breaker reproduces a 2-parameter IEC transient

recovery voltage, while the circuit at the right side represents a short transmission line that is short-circuited. The parameters of the Mayr arc model are given as $\tau_m = 0.3$ µs, $p_0 = 30900$ W, $g_{m0} = 1e4$ S and $T_{trip} = 20$ ms.

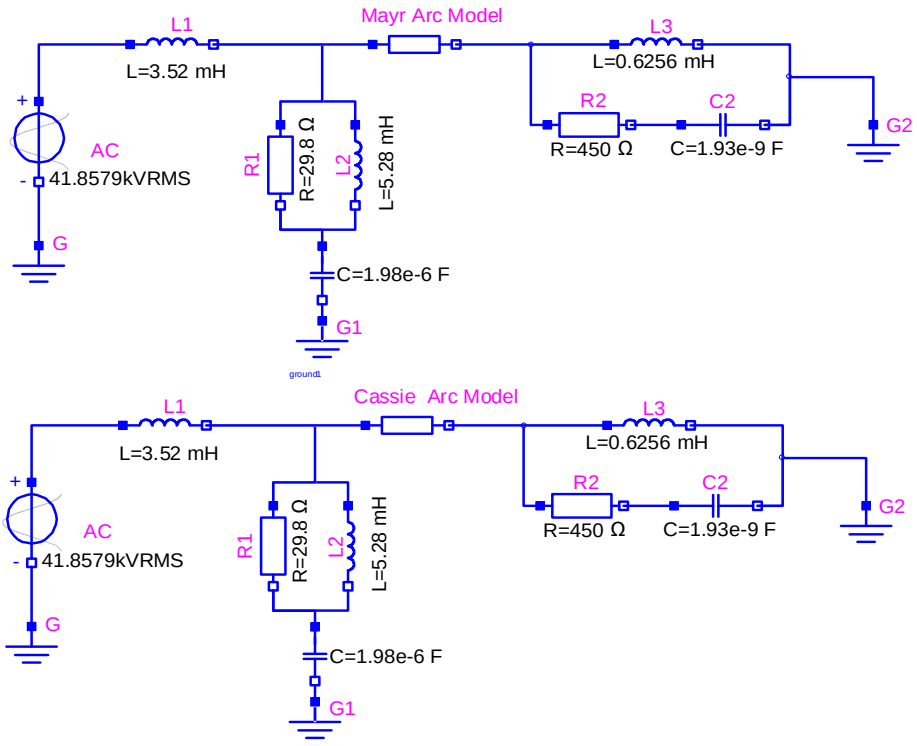


Figure 7.7 The proposed circuit for the verification and comparison of Mayr and Cassie arc models

Simulation is run using trapezoidal/backward Euler integrator with the time step of 0.1 µs and simulation time of 400 ms in EMTP®. The same circuit is simulated with IDA solver, Tol=1e-6 in Modelica. Figure 7.8 compares the curves of voltage and current of arc model obtained from Modelica and EMTP® solutions. As one can see, the results obtained from Modelica are identical with the EMTP® solutions.

Figure 7.9 shows the arc conductance/resistance of the Mayr arc model. It is observed that the arc resistance increases from zero to almost infinite quickly at the instant of interruption of current. After the interruption, the transient recovery voltage builds up across the circuit breaker contacts; hence a high-frequency transient recovery voltage (2-parameter) oscillation is observed over the breaker in the Mayr model in Figure 7.8.

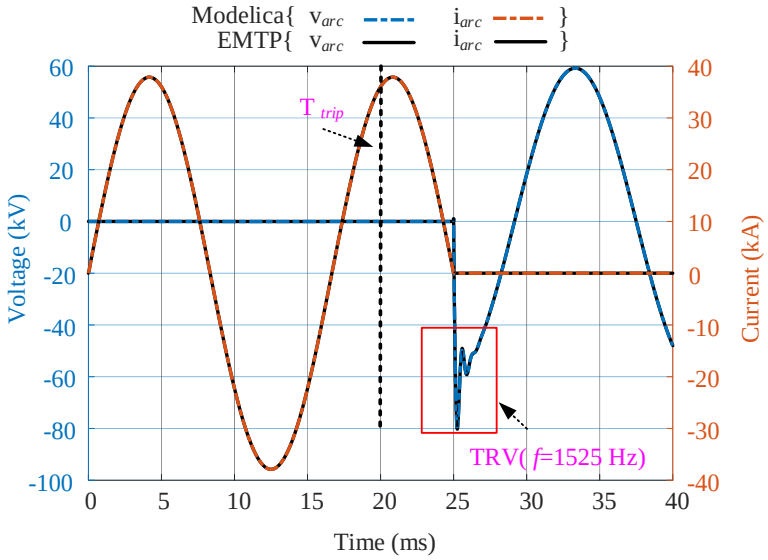


Figure 7.8 The curves of voltage and current obtained from the Mayr arc model in Modelica and EMTP®

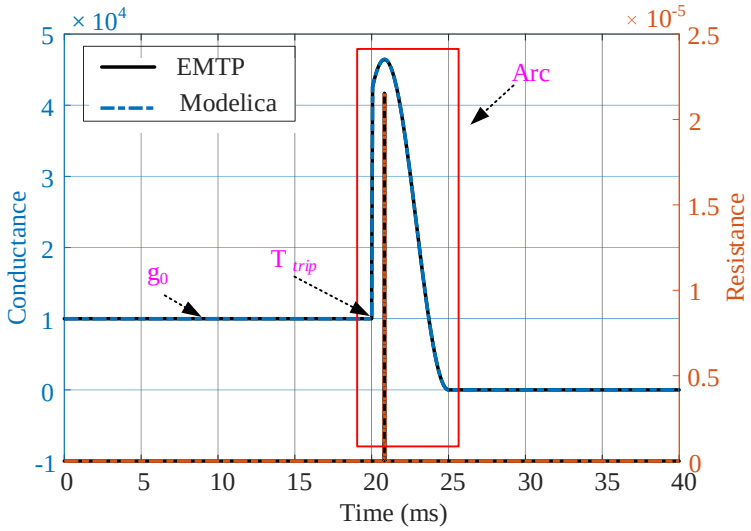


Figure 7.9 Arc conductance for Mayr model

Once again, the same circuit is simulated with the same solver parameters, except the Cassie arc model is used in this case. The parameters of the Cassie arc model are: $\tau_c = 1.2\ \mu s$, $v_0 = 3850\ \mathrm{V}$, $g_{c0} = 1e4\ \mathrm{S}$ and $T_{trip} = 20\ ms$. Figure 7.10 illustrates the results for arc voltage and current

obtained from both simulators. In this case, it is also observed that the solutions of both solvers are in excellent agreement.

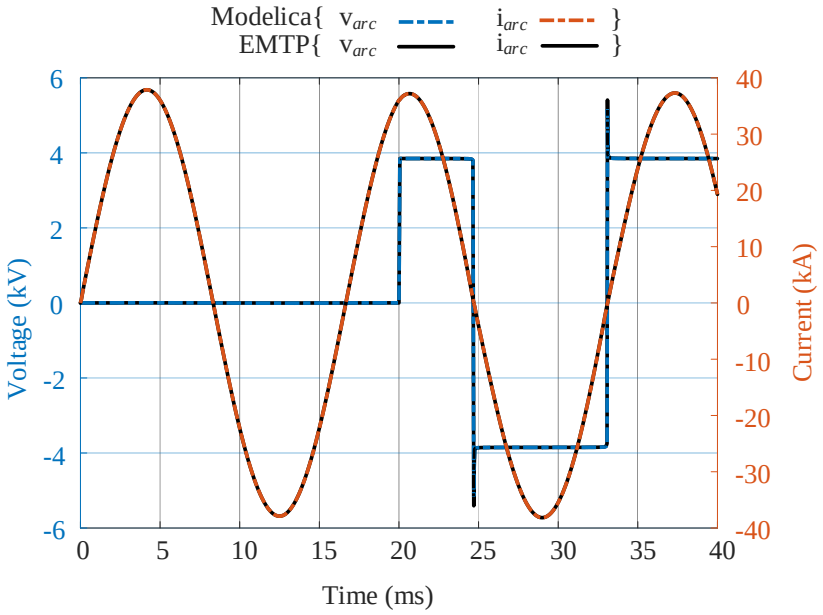


Figure 7.10 The curves of voltage and current obtained from Cassie arc model in Modelica and EMTP®

Comparing Figure 7.8 and Figure 7.10 shows that: (1) both arc voltage and current passe zero simultaneously (because arc behaves as a nonlinear resistor). (2): The Cassie model fails to interrupt the short circuit current, while the Mayr arc model has a successful interruption.

7.5.2 Cassie-Mayr Arc Model

This section deals with the Cassie-Mayr arc model in a kilometric fault test of a 420-kV CB. The test circuit is reproduced from the EMTP® example using the MSEMT library. Figure 7.11 shows the schematic of the circuit. The circuit is energized with a voltage source of 420 kV RMS, 60 Hz. The circuit at the right side represents a short (d=500 m) untransposed transmission line (CP-line model). The parameters of the arc model are given as $\tau_m = 0.5$ µs, $\tau_c = 1$ µs $p_0 = 100$ kW, $v_0 = 2000$ V, $g_0 = 5e7$ S and $T_{trip} = 28\,ms$. A single-phase earth fault on phase-*a* occurs at *t* = 1 ms at the end of TLM. The high voltage circuit breaker, D, trips the phase-*a* at *t* = 28 ms. The healthy phases remain closed.

7.5.2.1 Case 1: Source Inductance, L1=9 mH

In this case, the simulation is run using the trapezoidal/backward Euler solver with the time step of 0.1 µs and simulation time of 50 ms in EMTP®. The same circuit is simulated with IDA solver, Tol=1e-6 in Modelica. In this simulation, L1, which represents the source inductance, is 9 mH.

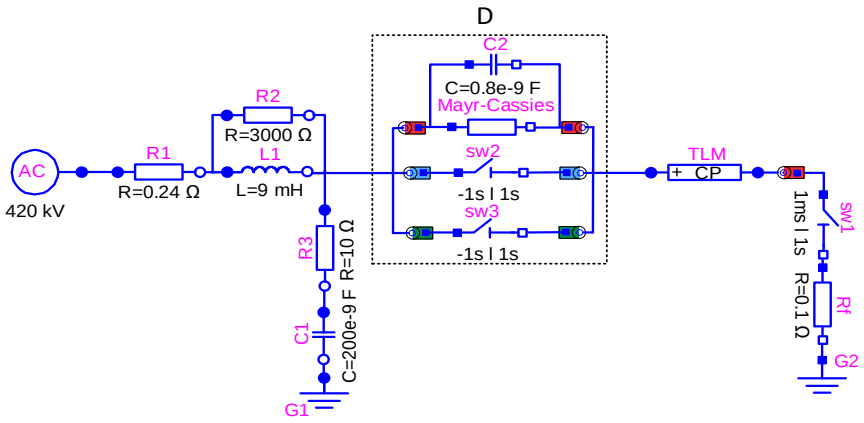


Figure 7.11 Kilometric fault test of a 420-kV CB

Figure 7.12 shows the waveforms of voltage and current of the arc model (phase-*a* of circuit breaker D). As one can see, the circuit breaker fails to interrupt the fault current in both simulators. The fault current, 90 kA, passes through the contacts of the circuit breaker by the end of simulations. This is a catastrophic status for the circuit breaker and may damage it severely.

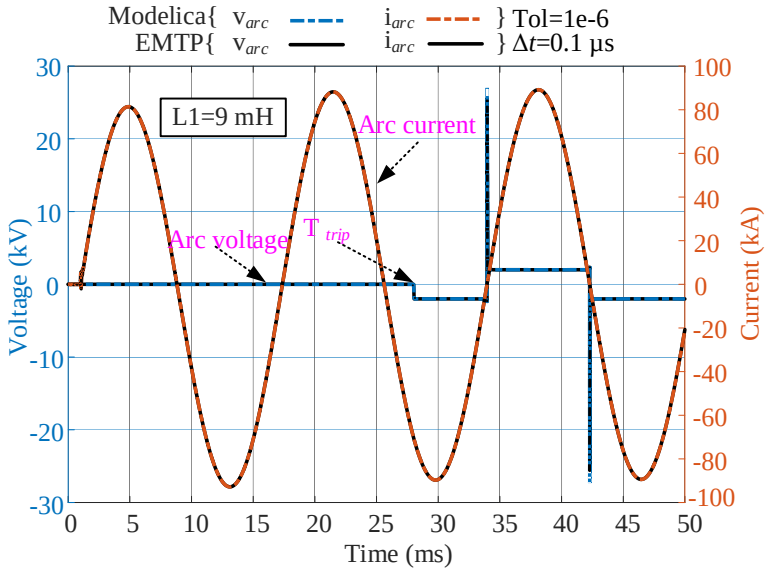


Figure 7.12 The voltage and current curves obtained from the Cassie-Mayr model, L1=9 mH

7.5.2.2 Case 2: Source Inductance, L1=10 mH, Δt= 0.1 µs in EMTP®

In the second attempt, we change the source inductance, L1=10 mH, and repeat the simulation with the same parameters as Case 1. As shown in Figure 7.13, the circuit breaker model in the Modelica simulation succeeded in breaking the fault current (79kA). A TRV occurs after the interruption of current on the terminals of the breaker. The simulation in EMTP® shows that the fault current continues by the end of the simulation. As a result, the selected time step for the simulation in EMTP® is not correct.

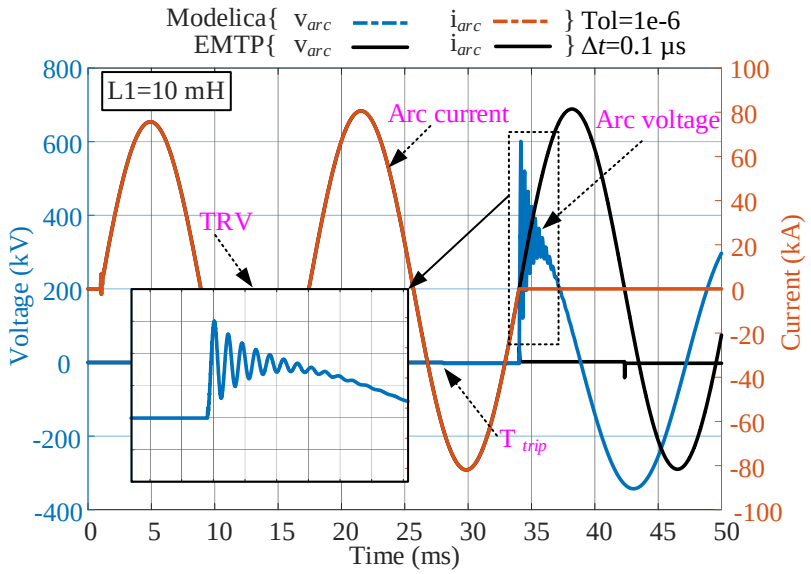


Figure 7.13 The curves of voltage and current obtained from the Cassie-Mayr arc model in Modelica and EMTP®, L1=10 mH, Δt= 0.1 µs in EMTP®

7.5.2.3 Case 3: Source Inductance, L1=10 mH, Δt= 0.01 µs in EMTP®

In the third attempt, we simulate the same circuit of Case 2 with the time step of 0.01 µs in EMTP®. The simulation parameters in Modelica remain the same as in previous cases. Figure 7.14 shows the voltage and current of the arc model. As one can see, the results obtained from both simulators are identical and illustrates a TRV on the terminal of circuit breaker D. In both simulators, the fault current is cleared when it passes zero-crossing after T_{trip}.

Table 7.3 shows the performance of simulations in cases 2 and 3. As one can see, the Modelica simulator outperforms EMTP® in case 3, where the results are identical, in the ratio of 4.33:1.

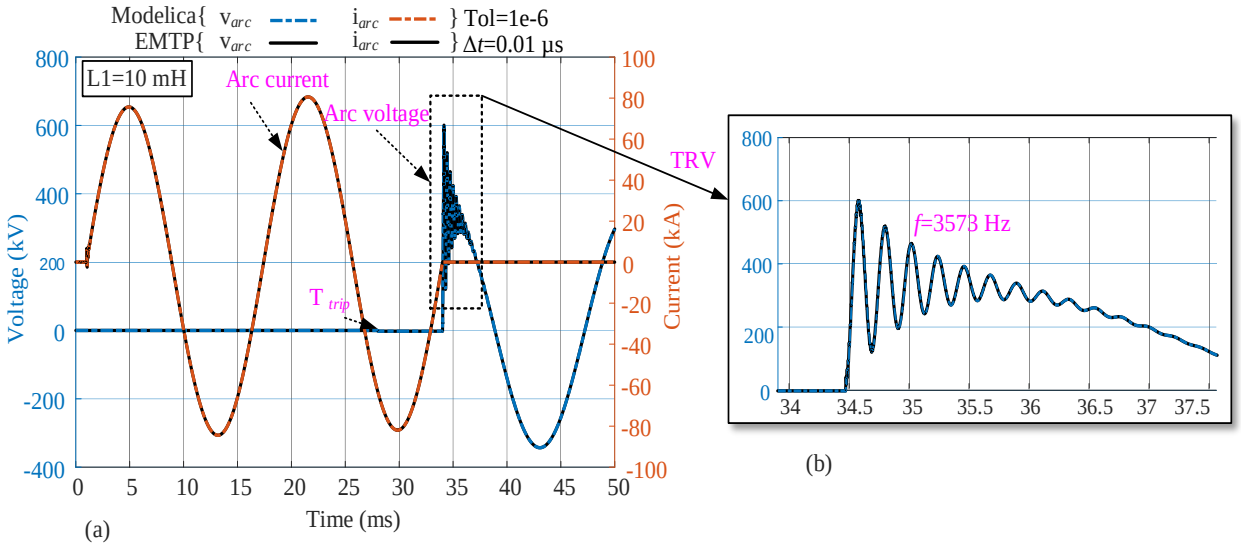


Figure 7.14 (a): The curves of voltage and current obtained from the Cassie-Mayr arc model in Modelica and EMTP®, L1=10 mH, Δt= 0.01 µs in EMTP®, (b); the zoom-in plot of TRV on the terminals of the circuit breaker

Table 7.3 Comparison of simulation performance in Cases 2 and 3

Characteristics	**Dymola**	**EMTP®**	
Solver	IDA	Trapezoidal /BE	
Tolerance	1e-6	-	-
Δt	-	0.1 μs	0.01 μs
Δt_{MIN}	2.72e-16 s	-	-
Δt_{MAX}	2e-05 s	-	-
No of steps	41 285	500 003	5 000 003
f-evaluations	103 768	-	-
J-evaluations	25 930	-	-
CPU time (s)	66.8	24.28	289.36
CPU-time for 1 step	0.134 ms	0.048 ms	0.057 ms
Performance ratio	1	0.36	4.33

7.6 Conclusion

Modelica language has been considered for EMT simulations due to its advantages for creating models at very high abstraction levels. This chapter emphasizes the software-to-software validation of nonlinear models, including synchronous machines with magnetic saturation and the surge arrester and arc models. The first two nonlinear models are validated by comparisons with EMTP® in a large grid (IEEE 118-bus benchmark). It is shown that high-level modeling in Modelica is very accurate as compared to EMTP®. However, the performance is unsatisfactory, except when variable time-step is used advantageously for high-frequency transients of short duration in a long simulation interval. Comparison of CPU time for the simulation of IEEE 39-bus and IEEE 118-bus benchmarks shows that simulation of cases including the WB and CP-line models can be very long and scales poorly.

Concerning the arc models, the comparison of results obtained from both tools demonstrates the accuracy of Modelica models. About the simulation efficiency, it was demonstrated that Modelica simulator offers better CPU time than EMTP® for high-resolution simulations required for simulation of arc models. The MSEMT library created in Modelica is user-friendly. It can be used for didactic purposes as well.

As future work, the existing library can be further developed to cover the models of HVDC system and renewable energy sources. The developed EMT models can be used for model exchange and co-simulation incorporating FMI. Therefore, co-simulation of Modelica and EMTP® via FMI can be considered for future. Parallelization of computations in the equation-based approach can be an interesting subject. This is a challenge for improvement of simulation speed in Modelica.

CHAPTER 8 CONCLUSION AND RECOMMANDATIONS

Since the thesis covers various research subjects, therefore the conclusion and recommendation of each work have been described in the last section of its dedicated chapter. Refer to Sections 4.3, 5.7, 6.5 and 7.6.

REFERENCES

[1] B. P. Zeigler, *Theory of Modeling and Simulation*, New York, USA: John Wiley,1976.

[2] D. C. Augustin, M. S. Fineberg, B. B. Johnson, R. N. Linebarger, F. J. Sansom and J. C. Strauss. "The SCi Continuous System Simulation Language (CSSL)." *Simulation* 9, 1967, pp. 281-303.

[3] E. Hairer, S. P. Nørsett, and G. Wanner. "*Solving Ordinary Differential Equations, I: Non stiff Problems.*" Springer-Verlag Berlin Heidelberg, 2nd rev. ed. 1993. corr. 3rd printing 2008 edition, 1993.

[4] MathWorks. The Mathworks - Simulink - Simulation and Model-Based Design. [Online]. Available: http://www.mathworks.com/products/simulink/

[5] R. Riaza. "DAEs in Circuit Modelling: A Survey." In *Surveys in Differential-Algebraic Equations I*. Achim Ilchmann; Timo Reis (eds.). Springer Science & Business Media, 2013.

[6] D. A. Calahan, Computer-*aided network design*. McGraw-Hill Education, 1972.

[7] U. M. Ascher, L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, US, 1998, p. 12.

[8] K. E. Brenan, S. L. Campbell, & L. R. Petzold, *Numerical solution of initial-value problems in differential-algebraic equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, US, 1995.

[9] R. W. Sebesta, *Concepts of programming languages*. Pearson Education Inc., 11th ed., 2015.

[10] D. Broman, "Meta-Languages and Semantics for Equation-Based Modeling and Simulation," Ph.D. dissertation, Dept. of Computer and Information Science, Linköping University, Linköping, Sweden, 2010.

[11] D. Zimmer, "Equation-based modeling of variable-structure systems," Ph.D. dissertation, Dept. of Computer Sciences, ETH Zurich, 2010.

[12] O. J. Dahl and K. Nygaard, "SIMULA: an ALGOL-based simulation language," *Communications of the ACM*, vol. 9, no. 9, 1966, pp. 671–678.

[13] D. Broman, P. Fritzson, S. Furic, “Types in the Modelica Language,” in *Proc. of the Fifth International Modelica Conference*, Vienna, Austria, vol. 1, 2006, pp. 303-315.

[14] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. 2 ed., John Wiley & Sons, 2014.

[15] F. E. Cellier & J. Greifeneder, *Continuous system modeling*, Springer Science & Business Media, 2013.

[16] F. E. Cellier, & E. Kofman, *Continuous system simulation*, Springer Science & Business Media, 2006.

[17] R. Tarjan, “Depth-first search and linear graph algorithms,” in *SIAM journal on computing*, vol. 1, no 2, 1972, pp.146-160.

[18] I. S. Duff, A. M. Erisman & J. K. Reid, *Direct methods for sparse matrices*, Oxford University Press, 2017.

[19] K. Kaya, J. Langguth, F. Manne, & B. Uçar, “Push-relabel based algorithms for the maximum transversal problem,” *Computers & Operations Research*, vol. *40*, no 5, 2013, pp. 1266-1275.

[20] J. Frenkel, G. Kunze, & P. Fritzson, “Survey of appropriate matching algorithms for large scale systems of differential-algebraic equations,” in *Proc. of the 9th International MODELICA Conference*, 2012, Munich, Germany, pp. 433-442.

[21] H. Elmqvist, M. Otter, “Methods for tearing systems of equations in object-oriented modeling.” In *Proc. of European Simulation Multiconference (ESM)*, 1994, vol. 94, pp. 1-3.

[22] E. B. Kathryn, S. L. Campbell, L. R. Petzold, *Numerical Solution of Initial–Value Problems in Differential-Algebraic Equations*, North-Holland, New York, 1989.

[23] C. C Pantelides, "The consistent initialization of differential-algebraic systems," in *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no 2, 1988, pp. 213-231.

[24] S. E. Mattsson, G. Söderlind, “Index reduction in differential-algebraic equations using dummy derivatives,” in *SIAM Journal on Scientific Computing*, vol. 14, no 3, 1993, pp. 677-692.

[25] S. E. Mattsson and G. Soderlind, "A new technique for solving high-index differential-algebraic equations using dummy derivatives," in *IEEE Symposium on Computer-Aided Control System Design*, 1992, pp. 218-224.

[26] I. S. Duff. “On Algorithms for Obtaining a Maximum Transversal.” In *ACM Transactions on Mathematical Software*, vol. 7, no 3, 1981, pp. 315-330.

[27] I. S. Duff, J. K. Reid. “An Implementation of Tarjan’s Algorithm for the Block Triangularization of a Matrix,” in *ACM Transactions on Mathematical Software*, vol. 4, no 2, 1978, pp.137–147.

[28] Modelica® - A Unified Object-Oriented Language for System Modeling and Simulation, Version 3.5, Feb 2020, Modelica Association, [Online]. Available: https://modelica.org/.

[29] F. Milano, *Power system modeling and scripting*, Springer Science & Business Media, 2010.

[30] P. Fritzson *et al.*, “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development,” *Modeling, Identification and Control*, vol. 41, no. 4, 2020, pp. 241–295.

[31] A. B. Dynasim, *Dynasim AB*, [Online]. Available: http://www.dynasim.se.

[32] K., Ryzhov Rozhdestvensky *et al.*, “Description of the Wolfram SystemModeler,” in *Computer Modeling and Simulation of Dynamic Systems Using Wolfram SystemModeler*, Springer, Singapore, 2020, pp. 23-87.

[33] S. Lynch, “An Introduction to Wolfram SystemModeler,” in *Dynamical Systems with Applications Using Mathematica®*, Birkhäuser, Cham, 2017, pp. 509-522.

[34] M. Jirstrand, J. Gunnarsson, & P. Fritzson, “MathModelica--a new modeling and simulation environment for Mathematica,” in *Proc. International Mathematica Symposium*, 1999.

[35] Maplesoft, *MapleSim—High-performance multi-domain modeling and simulation*. [Online]. Available: http://www.maplesoft.com/products/maplesim/index.aspx.

[36] J. Åkesson, M.Gäfvert & H. Tummescheit, “Jmodelica—an open-source platform for optimization of Modelica models,” in *Proc. of 6th Vienna International Conference on Mathematical Modelling*, 2009.

[37] J. Mahseredjian, V. Dinavahi and J. A. Martinez, "Simulation Tools for Electromagnetic Transients in Power Systems: Overview and Challenges," in *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1657-1669, July 2009.

[38] P.M. DeRusso, A.A. Desrochers, R.J. Roy, C.M. Close, *State variables for engineers*. John Wiley & Sons, Inc., 1997.

[39] F. Li, W. Peng-Yung. “A New Method for Establishing State Equations: The Branch Replacement and Augmented Node-Voltage Equation Approach,” in *Proc. of Circuits, systems, and signal*, vol. 21, pp. 149-161, 2002.

[40] C. Dufour, J. Mahseredjian and J. Bélanger, "A Combined State-Space Nodal Method for the Simulation of Power System Transients," in *IEEE Transactions on Power Delivery*, vol. 26, no. 2, pp. 928-935, April 2011.

[41] T. Hassell, W. Weaver, A. Oliveira, “Using MATLAB’s Simscape modeling environment as a simulation tool in power electronics and electrical machines courses," in *2013 IEEE Frontiers in Education Conference (FIE)*, Oklahoma City, OK, USA, 2013 pp. 477-483.

[42] Simscape Language Guide 2020, [available online]: www. Mathwork.com

[43] Simscape Electrical Specialized Power System user manual, 2020

[44] L. M. Wedepohl and L. Jackson, "Modified nodal analysis: an essential addition to electrical circuit theory and analysis," in *Engineering Science and Education Journal*, vol. 11, no. 3, pp. 84-92, June 2002.

[45] L. O. Chua, *Computer-aided analysis of electronic circuits: Algorithms and computational techniques*, Prentice-Hall, 1975.

[46] F. N. Najm, R. C. Dumas, *Circuit simulation*, vol. 9, 2010, Hoboken, NJ: Wiley.

[47] C.W. Ho, A. Ruehli and P. Brennan, "The modified nodal approach to network analysis," in *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504-509, June 1975.

[48] J. Mahseredjian and F. Alvarado, "Creating an Electromagnetic Transients Program in MATLAB: MatEMTP," in *IEEE Transactions on Power Delivery*, vol. 12, no. 1, pp. 380-388, Jan. 1997.

[49] J. Mahseredjian, S. Dennetière, L. Dubé, B. Khodabakhchian, L. Gérin-Lajoie, “On a new approach for the simulation of transients in power systems,” in *Electric Power Systems Research*, vol 77, No 11, 2007, pp 1514-1520.

[50] J. Mahseredjian: “Simulation des transitoires électromagnétiques dans les réseaux électriques”, Édition ‘Les Techniques de l'Ingénieur’, Février 10, 2008, Dossier D4130, 2008.

[51] A. Ametani, “Chapter 3: Simulation of electromagnetic transients with EMTP-RV” in *Numerical analysis of power system transients and dynamics*. The Institution of Engineering and Technology (IET), UK, 2015.

[52] E. Haginomori, T. Koshiduka, J. Arai & H. Ikeda, Power *system transient analysis: theory and practice using simulation programs (ATP-EMTP)*, John Wiley & Sons, 2016.

[53] H. W. Dommel, "Digital Computer Solution of Electromagnetic Transients in Single-and Multiphase Networks," in *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-88, no. 4, pp. 388-399, April 1969.

[54] Manitoba HVDC Research Centre (Canada). User’s Guide on the Use of PSCAD. Winnipeg, 2004.

[55] F. M. Gonzalez-Longatt & J. L. Rueda, *PowerFactory applications for power system analysis*. Springer, 2014.

[56] J. R. Marti and J. Lin, "Suppression of numerical oscillations in the EMTP® power systems," in *IEEE Transactions on Power Systems*, vol. 4, no. 2, pp. 739-747, May 1989.

[57] R. Alexander, "Diagonally implicit Runge–Kutta methods for stiff ODE’s." *SIAM Journal on Numerical Analysis* 14.6,1977, pp. 1006-1021.

[58] T. Noda, K. Takenaka and T. Inoue, "Numerical Integration by the 2-Stage Diagonally Implicit Runge-Kutta Method for Electromagnetic Transient Simulations," in *IEEE Transactions on Power Delivery*, vol. 24, no. 1, pp. 390-399, Jan. 2009.

[59] T. Noda, T. Kikuma & R. Yonezawa, “Supplementary techniques for 2S-DIRK-based EMT simulations,” in *Electric power systems research*, *115*, 2014, pp. 87-93.

[60] X. Fu, S. Mouhamadou Seye, J. Mahseredjian, M. Cai and C. Dufour, "A Comparison of Numerical Integration Methods and Discontinuity Treatment for EMT Simulations," in *Proc. 2018 Power Systems Computation Conference (PSCC)*, 2018, pp. 1-7.

[61] Dynaωo, open-source simulator for power systems, [Online]. Available: https://dynawo.github.io/.

[62] A. Guironnet, F. Rosière, G. Bureau and M. Saugier, "Speed-up of Large-Scale Voltage Stability Simulations within a Fully Separated Modeler/Solver Framework," in *Proc. 2021 International Conference on Smart Energy Systems and Technologies (SEST)*, 2021, pp. 1-6.

[63] D. Fabozzi and T. Van Cutsem, "Simplified time-domain simulation of detailed long-term dynamic models," in *Proc. 2009 IEEE Power & Energy Society General Meeting*, Calgary, AB, 2009, pp. 1-8.

[64] A. G. Taylor, A. C. Hindmarsh, "User documentation for KINSOL, a nonlinear solver for sequential and parallel computers." Lawrence Livermore National Lab., CA, US, 1998.

[65] Modelica®-A Unified Object-Oriented Language for Physical Systems Modeling - Version 1, September 1997. [Online Available] http: //www.modelica.org.

[66] Modelica Association. [Online Available] http://www.modelica.org

[67] Julia programming language. [Online Available]: https://julialang.org/

[68] B. Lie, A. Palanisamy, A. Mengist, L. Buffoni, M. Sjölund, A. Asghar, & P. Fritzson, "OMJulia: An OpenModelica API for Julia-Modelica Interaction," in *Proc. 2019 13th International Modelica Conf*. Linköping, Sweden,

[69] H. Elmqvist, M. Otter, A. Neumayr, G. Hippmann, "Modia - Equation Based Modeling and Domain Specific Algorithms," in *Proc. 2021 14th International Modelica Conf*., Linköping, Sweden, pp. 73-86.

[70] The Functional Mock-up Interface. [Online Available] https://fmistandard.org

[71] T. Blochwitz, *et al*. "The functional mockup interface for tool independent exchange of simulation models," in *Proc. of the 8th International Modelica Conf*., Linköping, Sweden, 2011.

[72] L. Vanfretti, W. Li, T. Bogodorova and P. Panciatici, "Unambiguous power system dynamic modeling and simulation using Modelica tools," in *Proc. 2013 IEEE Power & Energy Society General Meeting*, 2013, pp. 1-5.

[73] F. J. Gómez, L. Vanfretti and S. H. Olsen, "CIM-Compliant Power System Dynamic Model-to-Model Transformation and Modelica Simulation," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 3989-3996, Sept. 2018.

[74] F. J. Gómez, L. Vanfretti and S. H. Olsen, "Binding CIM and Modelica for consistent power system dynamic model exchange and simulation," in *Proc. 2015 IEEE Power & Energy Society General Meeting*, 2015, pp. 1-5.

[75] A. Masoom, J. Mahseredjian, T. Ould-Bachir, A. Guironnet, "MSEMT: An Advanced Modelica Library for Power System Electromagnetic Transient Studies." in *IEEE Transactions on Power Delivery*, 2021.

[76] J. Joss. "Algorithmisches Differenzieren." Ph.D. dissertation, Swiss Federal Institute of Technology, Zurich, Switzerland, 1976.

[77] L. Petzold. Description of DASSL: a differential/algebraic system solver. Tech. report Sandia National Labs., Livermore, CA (USA), Sept. 1982.

[78] A. C. Hindmarsh et al. "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, " in *ACM Trans. Math. Software.* 31.3, Sept. 2005, pp. 363–396.

[79] S. G. Krantz & H. R. Parks, *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012.

[80] A. C. Hindmarsh, R. Serban., A. Collier, User Documentation for IDA v2.6.0, *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, May 2009.

[81] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd. Ed., Society for Industrial and Applied Mathematics, USA, 2003.

[82] R. S. Dembo, S. C. Eisenstat, T. Steihaug. "Inexact newton methods," in *SIAM Journal on Numerical analysis*, vol. 19.2,1982, pp. 400-408.

[83] P. N. Brown and A. C. Hindmarsh. "Reduced Storage Matrix Methods in Stiff ODE Systems," in *Journal of Applied Math. & Comp.*, vol. 31, 1989, pp. 49-91.

[84] W. Braun, *et al.* "Fast simulation of fluid models with colored jacobians," in *Proc. of the 9th International Modelica Conf.*, 2012, Munich; Germany.

[85] A. H. Gebremedhin, F. Manne, and A. Pothen. "What color is your Jacobian? Graph coloring for computing derivatives." in *Society for Industrial and Applied Mathematics (SIAM)* review, vol. 47.4, 2005, pp. 629-705.

[86] J. Åkesson, *et al.* "Generation of sparse jacobians for the function mock-up interface 2.0." in *Proc. of the 9th International Modelica Conf.*, 2012.

[87] K. Soetaert, J. Cash, & F. Mazzia, *Solving differential equations in R*. Springer Science & Business Media, 2012.

[88] W. Braun, F. Casella & B. Bachmann, "Solving Large-scale Modelica models: New Approaches and Experimental Results using OpenModelica," in *Proc. 2017 International Modelica Conf.*, Prague, Czech Republic, 2017.

[89] V. Brandwajn, "Damping of numerical noise in the EMTP solution", EMTP Newsletter, vol. 2, no. 3, pp 10-19, 1982.

[90] J. Lin and J. R. Marti, "Implementation of the CDA procedure in the EMTP," in *IEEE Transactions on Power Systems*, vol. 5, no. 2, pp. 394-402, May 1990.

[91] H. Lundvall, P. Fritzson, and B. Bachmann, 'Event Handling in the OpenModelica Compiler and Runtime System', Linköping University Electronic Press, Linköping, 2008.

[92] N. Mohan, T.M. Undeland, and W.P. Robbins, *Power Electronics: Converters, Applications, and Design*, John Wiley & Sons, Inc., New York, 1995.

[93] J. Mahseredjian, L. Dube, Ming Zou, S. Dennetiere and G. Joos, "Simultaneous solution of control system equations in EMTP," in *IEEE Transactions on Power Systems*, vol. 21, no. 1, pp. 117-124, Feb. 2006.

[94] OpenModelica documentations, [Online Available]: https://www.openmodelica.org/

[95] "Excitation System Models for Power System Stability Studies," IEEE Committee Report. IEEE Transactions on Power Apparatus and Systems, vol. PAS-100, No. 2 February 1981.

[96] "IEEE Recommended Practice for Excitation System Models for Power System Models for Power System Stability Studies," IEEE Standard 421.5-2005.

[97] P. Pourbeik *et al*., “Dynamic Models for Turbine-Governors in Power System Studies,” Tech. report PES-TR1. IEEE Power & Energy Society Jan 2013.

[98] J. R. Marti, "Accurate Modelling of Frequency-Dependent Transmission Lines in Electromagnetic Transient Simulations," in *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-101, no. 1, pp. 147-157, Jan. 1982.

[99] A. Morched, B. Gustavsen and M. Tartibi, "A universal model for accurate calculation of electromagnetic transients on overhead lines and underground cables," *in IEEE Transactions on Power Delivery*, vol. 14, no. 3, pp. 1032-1038, July 1999.

[100] I. Kocar and J. Mahseredjian, "Accurate frequency dependent cable model for electromagnetic transients," 2016 IEEE Power and Energy Society General Meeting (PESGM), 2016, pp. 1-1.

[101] P. C. Krause, O. Wasynczuk, S. D. Sudhoff, and S. Pekarek, “Synchronous machines” in *Analysis of electric machinery and drive systems*, 3rd d. NY, USA, IEEE Press, 2013.

[102] U. Karaagac, J. Mahseredjian and O. Saad, "An efficient synchronous machine model for electromagnetic transients," *IEEE Trans. Power Delivery*, vol. 26, no. 4, 2011, pp. 2456-2465.

[103] P. Kundur, “Synchronous machine parameters," in *Power system stability and control*, New York, USA: McGraw-Hill, 1994.

[104] J. A. Martinez, J. Mahseredjian and B. Khodabakhchian, "Parameter determination for modeling system transients-Part VI: Circuit breakers," in *IEEE Transactions on Power Delivery*, vol. 20, no. 3, pp. 2079-2085, July 2005.

[105] M. Lambert, "Transformer modeling for low-and mid-frequency electromagnetic transients’ simulation." Ph.D. dissertation, Dept. Electrical Engineering, École Polytechnique de Montréal, Montreal, 2014.

[106] W. H. Kersting, “Radial distribution test feeders,” in *IEEE Transactions on Power Systems*, vol. 6 (3), Aug 1991, pp. 975–985.

[107] A. Masoom, T. Ould-Bachir, J. Mahseredjian, A. Guironnet, N. Ding, “Simulation of electromagnetic transients with Modelica, accuracy and performance assessment for transmission line models,” In *Electric Power Systems Research*, vol. 189, 2020, 106799.

[108] A. Haddadi, J. Mahseredjian, “Power system test cases for EMT-type simulation studies,” CIGRE, Paris, France, Tech. Rep. CIGRE WG C, 4, 2018, pp. 1-142.

[109] M.E. Hosea, L.F. Shampine, “Analysis and implementation of TRBDF2,” in *Applied Numerical Mathematics*, vol. 20, no. 1, pp. 21-37, 1996.

[110] PEGASE: Pan European Grid Advanced Simulation and state Estimation, [Online]. Available: https://cordis.europa.eu/project/id/211407

[111] iTesla: Innovative Tools for Electrical System Security within Large Area, [Online]. Available: https://cordis.europa.eu/project/id/283012

[112] L. Vanfretti T. Rabuzin, M. Baudette, and M. Murad, “iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations, ” in *SoftwareX*, 18 May 2016.

[113] M. Baudette, M. Castro, T. Rabuzin, J. Lavenius, T. Bogodorova, L. Vanfretti, OpenIPSL: Open-Instance Power System Library - Update 1.5 to “iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations, ” in *SoftwareX*, vol. 7, January-June 2018, pp. 34-36.

[114] A. Bartolini, F. Casella, A. Guironnet. “Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library”. In *Proc. 2019 International Modelica Conf.*, Regensburg, Germany,2019, Feb 2019.

[115] H. Lundkvist, and A. Yngve. "Accelerated Simulation of Modelica Models Using an FPGA-Based Approach." Master thesis, Dept. of Electrical Engineering, Linköping University, 2018.

[116] P. Gibert, P. Panciatici, R. Losseau, A. Guironnet, D. Tromeur-Dervout and J. Erhel, "Speedup of EMT simulations by using an integration scheme enriched with a predictive Fourier coefficients estimator," In *Proc.* 2018 *IEEE PES Innovative Smart Grid Technologies Conf. Europe (ISGT-Europe)*, Sarajevo, 2018, pp. 1-6.

[117] E. Kofman, J. Fernández, D. Marzorati, “Compact sparse symbolic Jacobian computation in large systems of ODEs,” in *Applied Mathematics and Computation*, vol. 403, p. 126181, Aug. 2021.

[118] A. Guironnet, M. Saugier, S. Petitrenaud, F. Xavier, and P. Panciatici, “Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems,” in *Proc. 2018 IEEE PES Innovative Smart Grid Technologies Conf. Europe (ISGT-Europe)*, Sarajevo, 2018.

[119] T. A. Davis and E. Palamadai Natarajan, “Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems,” in *ACM Trans. on Mathematical Software*, vol. 37, no. 3, pp. 36:1–36:17, Sep. 2010.

[120] X. Chen, Y. Wang & H. Yang “NICSLU: An adaptive sparse matrix solver for parallel circuit simulation,” in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 32(2), 261-274:2013.

[121] L. Razik, L. Schumacher, A. Monti, A. Guironnet and G. Bureau, "A comparative analysis of LU decomposition methods for power system simulations," in *2019 IEEE Milan PowerTech*, Milan, Italy, pp. 1-6.

[122] A. Masoom, A. Guironnet, A.A. Zeghaida, T. Ould-Bachir, J. Mahseredjian, “Modelica-based simulation of electromagnetic transients using Dynaωo: Current status and perspectives,” in *Electric Power Systems Research*, vol. 197, 2021, 107340.

[123] IEEE Working Group 3.4.11, "Modeling of metal oxide surge arresters," *IEEE Trans. Power Delivery*, vol. 7, no. 1, pp. 302-309, Jan. 1992.

[124] A. Masoom, J. Mahseredjian, T. Ould-Bachir, A. Guironnet “Electromagnetic Transient Modeling of Large Power Networks with Modelica,” in *Proc. of 14th International Modelica Conf.*, Linköping, Sweden, Sep. 2021.