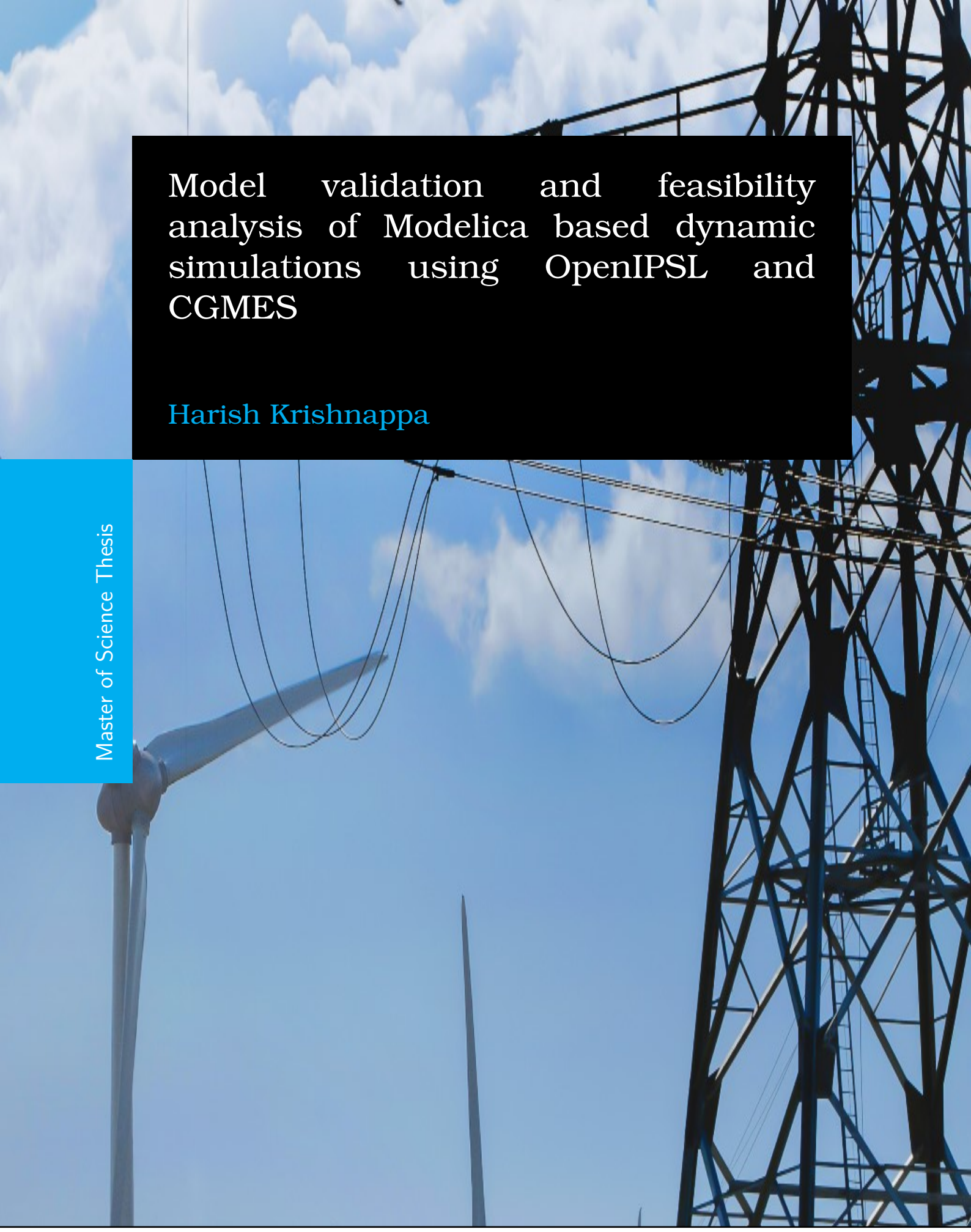


Model validation and feasibility analysis of Modelica based dynamic simulations using OpenIPSL and CGMES

Harish Krishnappa

Master of Science Thesis

TU Delft Delft University of Technology

Electrical Sustainable Energy

Model validation and feasibility analysis of Modelica based dynamic simulations using OpenIPSL and CGMES

by

Harish Krishnappa

in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering at Delft University of Technology

To be defended publicly
on Friday August 25, 2017 at 15:00 in TU Delft.

Student number:	4477847	
Project duration:	December 1, 2016 – August 25, 2017	
Supervisors:	Dr. ir. Jose L. Rueda Torres,	TU Delft
	Ir. Sander Franke	TenneT TSO B.V.
Thesis committee:	Prof. dr. Peter Palensky,	TU Delft
	Dr. ir. Jose L. Rueda Torres,	TU Delft
	Dr. ir. Luigi Vanfretti,	RPI, USA
	Ir. Sander Franke	TenneT TSO B.V.

TU Delft Delft University of Technology

Acknowledgement

First and foremost, I would like to thank TenneT TSO for giving me the opportunity to conduct my master thesis and, TU Delft for its guidance through my Master's course. This thesis would not have been possible without the kind support and help of many individuals from these organizations. This thesis was supported by the System Operations department of TenneT TSO.

I am thankful to my supervisors Sander Franke (TenneT) and Prof. Jose Rueda Torres (TU Delft) for their insights and guidance throughout the thesis and steering me in the right direction whenever they thought I needed it. I would like to thank Prof.Luigi Vanfretti (RPI, USA) for his guidance, especially during the initial stages of this thesis. I would like to thank Prof. Peter Palensky (Chair, TU Delft) for his insightful comments and encouragement to go ahead with this open-source crusade.

Besides, I would like to thank Francisco Gomez (Phd, KTH) for all the help with Modelica scripting and for specific insights into the modelling process. Also, am grateful to Svein Olsen (Statnett SF) for his help in formulating the thesis objectives.

I am grateful to all my TenneT collegues, Susana de Graff, Frank Spaan, Tofan Fadriansyah, Jasper van Casteren, Frank Nobel, Nikoleta Kandalepa, Pim Jacobs, Vinay Sewdien and Marco Pavesi for their support. I would also like to thank my fellow interns, Camiel van Altenborg and Michelle Porte for their constant support and for all the fun, yet resourceful conversations.

I am also grateful to my Master co-ordinator, Laura Ramirez Elizondo and Academic counsellor Agaby Masih for their support and guidance in the beginning of this thesis. I would also like to thank my friends, Avinash Prakash, Aravinth Thamizh, Aniket Lewarkar, Prateek Gupta, Siddharth Kumar, Digvijay Gusain, Chetan Kumar, Deesh Dileep and all the colleagues in IEPG group at TU Delft for all the technical and moral support. Also, the Willemse family van Arnhem deserve a special thanks because they have been like a second family to me.

I would like to offer my special thanks to my childhood friend Santhosh Kumar , who although no longer with us, continues to inspire by his example.

Finally, I must express my very profound gratitude to my family and all my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

"There are only two options:
Make progress or make excuses."

- Tony Robbins

Abstract

The European Union's energy and climate policy objectives for 2030 targets to achieve at least 27% of the generated electricity from renewables. Such large-scale integration of renewable Energy Sources (RES) into the grid, will have drastic effects on the electricity grid structure, system operations and the functioning of the electricity market itself. In order to tackle the intermittency challenges posed by RES, smarter operating processes are essential. This requires accurate simulation tools and an efficient exchange of information between the energy players in Europe (TSOs, DSOs and other private generators). The Common Grid Model Exchange Standard (CGMES) was developed to support data exchanges between these energy players. The CGMES based Common Information Model (CGMES-CIM) files contain all the information about the grid under study including the powerflow values. The current version of CGMES-CIM supports full interoperability with respect to steady state simulations but is quite challenging to use for dynamic simulations. Open-Instance Power System Library (OpenIPSL), which is a Modelica based power system library, seems to be one of the possible solutions that can overcome the interoperability issue with respect to dynamic simulations.

This thesis focuses on extending the OpenIPSL with PowerFactory based models and provides a proof of concept to automatically initialize the Modelica grid model using CGMES-CIM. OpenModelica, an open-source Modelica simulation environment is used in this project to carry out dynamic studies on a modified Bonneville Power Administration (BPA) grid model. In order to perform dynamic studies, the Modelica grid model first needs to be populated with powerflow values. OpenModelica does not include powerflow option and therefore, powerflow solution needs to be obtained from another simulation tool and the values need to be subsequently loaded into OpenModelica to perform dynamic simulations; this process is called Initialization. Thus, to overcome the challenges with initialization of Modelica models, this thesis presents a proof of concept that directly utilizes the CGMES-CIM files for initialization of Modelica based grid models. The interfacing process between CIM and Modelica based grid model is done using Python.

To test the concept, Modelica based dynamic simulations were carried out and compared with reference results (signal records obtained through time domain simulations) from PowerFactory. For this purpose, several PowerFactory based Modelica models were developed and validated, extending the OpenIPSL. These models were subsequently used to create and validate a modified BPA grid model, which serves as a test case. This thesis also proposes a method to directly convert CGMES-CIM files to Modelica files. This conversion helps achieve complete automation of dynamic simulations in a Modelica environment. Therefore, this thesis contributes to the OpenIPSL by developing PowerFactory based models and, also proposes a new CGMES-CIM to Modelica converter.

Table of Contents

Acknowledgement iii

Abstract v

1 Introduction 1

1-1 Background . . . 1

1-1-1 CGMES-CIM . . . 3

1-1-2 Dynamic profile in CGMES-CIM . . . 4

1-2 Problem definition . . . 5

1-3 Motivation . . . 6

1-4 Goals and research questions . . . 6

1-5 Contributions . . . 9

1-6 Approach and thesis outline . . . 9

2 Modelica power system library 11

2-1 Introduction to Modelica . . . 11

2-1-1 Main characteristics . . . 12

2-1-2 Softwares based on Modelica . . . 12

2-2 Modelling environment . . . 13

2-2-1 Attributions and equations . . . 13

2-2-2 Simulation parameters . . . 14

2-3 Application example . . . 15

2-4 Modelica libraries . . . 17

2-4-1 Modelica standard library (MSL) . . . 17

2-4-2 Open-Instance Power System Library . . . 17

3 PowerFactory based Modelling **19**
3-1 DIgSILENT PowerFactory . . . 19
3-2 PowerFactory Models . . . 21
3-2-1 Synchronous generator . . . 22
3-2-2 Loads . . . 25
3-2-3 Transformer . . . 27
3-2-4 Transmission line . . . 28
3-2-5 Exciter system . . . 29
3-2-6 Overexcitation Limiter . . . 30
3-2-7 Speed-Governing System . . . 31
3-2-8 Steam Turbine system . . . 32
3-2-9 OLTC . . . 33
3-2-10 Asynchronous machine . . . 34

4 CIM based initialization for dynamic simulation **37**
4-1 Standards for Model Information Exchange . . . 37
4-2 Common information model standards . . . 38
4-3 CIM UML . . . 38
4-4 CGMES files . . . 41
4-5 CGMES-CIM based initialization of Modelica grid models . . . 44
4-5-1 Method 1: Initialization of Modelica based grid models . . . 45
4-5-2 Method 2: Model to model transformation . . . 47

5 Result discussion **49**
5-1 Considered test system . . . 49
5-2 Model Validation . . . 50
5-2-1 Simulation set-up . . . 51
5-2-2 Synchronous Generator . . . 51
5-2-3 Excitation system . . . 52
5-2-4 Governor system . . . 54
5-2-5 Load . . . 55
5-2-6 Transformer . . . 56
5-2-7 Transformer tap changers . . . 57
5-2-8 Complete model . . . 58
5-3 CGMES-CIM based Initialization in Modelica . . . 59
5-4 CGMES-CIM to Modelica converter . . . 63

6 Conclusions and future scope **69**
6-1 Conclusion . . . 69
6-2 Future scope and recommendations . . . 70

A **77**
A-1 Synchronous generator . . . 77
A-1-1 Parameters and equations . . . 78
A-1-2 Generator code . . . 78
A-2 Load . . . 84
A-3 Transmission Line . . . 87
A-4 Governor . . . 88
A-5 Exciter . . . 88
A-6 OLTC . . . 89

B **91**
B-1 Grid parameters . . . 91
B-2 Validation results of uncontrolled generator . . . 92
B-3 Generator validation with excitation system . . . 94
B-4 Generator validation with exciter and governor system . . . 96
B-5 Complete Model . . . 98

C **101**
C-1 CGMES based Initialization . . . 101
C-2 Model to model transformation . . . 102

List of Figures

1-1 Types of simulation studies and respective tools used . . . 2
1-2 CIM model part types . . . 4
1-3 Issue with the dynamic model exchange . . . 5
1-4 Overall Idea . . . 7
1-5 Work division . . . 7
1-6 Initialization process for dynamic simulations in OpenModelica . . . 8
1-7 'All in one' test system or modified Bonneville Power Administration (BPA) test system . . . 8
1-8 Workflow . . . 9
1-9 Thesis outline . . . 10

2-1 Modelica Process [1] . . . 12
2-2 OMEdit - OpenModelica Connection Editor . . . 13
2-3 Simulation options in OpenModelica . . . 14
2-4 Text view of electrical bus . . . 15
2-5 Model connections using PwPin . . . 15
2-6 Modelica based generator model displayed in text, icon and diagram view (shown using OpenModelica connection editor) . . . 16
2-7 Modelica library . . . 17
2-8 OpenIPSL . . . 18

3-1 Model definition of simple exciter system consisting DSL blocks . . . 20
3-2 Composite frame of the generator . . . 20
3-3 Common model of user defined AVR system . . . 21
3-4 Composite model of a generator system . . . 21
3-5 Equivalent d-axis circuit . . . 22
3-6 Equivalent q-axis circuit for round-rotor machine . . . 22

3-7 Coordinate system of Synchronous and Rotor frames* 23
3-8 Relationship between coordinate systems 23
3-9 Modelica model of the synchronous machine based on PowerFactory round rotor model (Machine model 2.2) 25
3-10 Representation of mixture of static and dynamic loads in PowerFactory* 26
3-11 Model that approximates the behaviour of the non-linear dynamic load* 26
3-12 Non-linear dynamic load modelled based on PowerFactory in Modelica 27
3-13 Positive sequence (per unit) equivalent circuit of a transformer* 27
3-14 Transformer model in Modelica and the dialog box for parameter input 28
3-15 Equivalent circuit of the three phase line* 28
3-16 Transmission model in Modelica and dialog box for parameter input 29
3-17 Simplified IEEE ST1A excitation model* 29
3-18 Over excitation limiter* 30
3-19 Exciter and limiter modelled in Modelica 30
3-20 Typical speed-governing system* 31
3-21 Speed-governing steam turbine system* 31
3-22 Linear model of turbine system 32
3-23 Governor and turbine model in Modelica based on PowerFactory 32
3-24 Tap changer is modelled at the HV side* 33
3-25 Equivalent circuit of the three-phase line* 33
3-26 off-load tap changer block in Modelica 34
3-27 On-load tap changer block in Modelica 34
3-28 General equivalent circuit of asynchronous machine in PowerFactory 35
3-29 Simple cage rotor model 35
3-30 Modelica model 35

4-1 Power transformer represented in CIM-UML* 38
4-2 UML class diagram for the classes of a transformer* 39
4-3 CIM-UML representation of Load based on CGMES v2.5 40
4-4 Different CIM profiles according to CGMES v2.5 41
4-5 The powerflow values inside the SV profile 42
4-6 Graph structure of the attributes of the CIM classes, SvPowerFlow and SvVoltage inside SV profile 43
4-7 Graph structure of the attributes of the CIM class, TopologicalNode inside TP profile 43
4-8 Flowchart to obtain the name of the bus and its respective bus voltages and angles 44
4-9 Dialog box for powerflow input in a generator model 45
4-10 Initialization process in Modelica using CGMES-CIM files 45
4-11 Model to Model transformation; CGMES-CIM to Modelica script conversion . . . 48

5-1 Test system modelled in PowerFactory 49

5-2 Grid model in PowerFactory 50
5-3 Grid model in Modelica 50
5-4 Terminal voltage and speed variation in an Uncontrolled generator 52
5-5 Comparison of terminal voltage with and without excitation system 52
5-6 Test system used to validate excitation system 53
5-7 Terminal voltage and electrical torque obtained from exciter validation 53
5-8 Test system for governor validation 54
5-9 Comparison of speed responses with and without the Governor system 54
5-10 Comparison of Modelica and PowerFactory responses with the governor system . 55
5-11 Difference in static and dynamic load responses 55
5-12 Test system for transformer validation in PowerFactory 56
5-13 Test system for transformer validation in OpenModelica 56
5-14 Comparison of Modelica and PowerFactory responses for transformer validation . 56
5-15 Voltages when the tap changer position is at 93 57
5-16 Complete model of the modified BPA system in Modelica 58
5-17 Voltages at the buses when load is increased by 100% between 5-7sec 59
5-18 Process used in the thesis for CGMES-CIM based initialization of Modelica models 60
5-19 Dialog box of a Modelica based generator showing dummy initial values 60
5-20 Part of modelica script that undergoes changes 60
5-21 Generator output obtained from the Python based CGMES reader 61
5-22 SV profile of the the test system for scenario 1 62
5-23 Voltage magnitude at Bus3 for different scenarios 63
5-24 The test system modelled in PowerFactory 64
5-25 Process followed in this thesis to obtain the Modelica model directly from CGMES-CIM 65
5-26 Dynamic profile showing the generator parameters required for dynamic simulations 66
5-27 Automatically generated test system in OpenModelica using the proposed converter 66
5-28 Volatge at Bus1 for a load increase of 100% at LoadC between 5-7 sec. 67
5-29 Voltages at all the buses when a perturbation is applied 67

A-1 The parameters of round rotor 2.2 generator in PowerFactory 78
A-2 Governor parameter description 88
A-3 Exciter parameter description 88
A-4 off-load tap changer in Powerfactory - 1 89
A-5 off-load tap changer block in Powerfactory - 2 89
A-6 on-load tap changer block in Powerfactory 90
A-7 Voltage at the Load when OLTC is used in OpenModelica 90

B-1 Comparison graphs for uncontrolled generator - 1 92
B-2 Comparison graphs for uncontrolled generator - 2 93

B-3 Comparison graphs for a generator with excitation system-1 94
B-4 Comparison graphs for a generator with excitation system-2 95
B-5 Comparison graphs for a generator with excitation system and governor system-1 96
B-6 Comparison graphs for a generator with excitation system and governor system-2 97

C-1 Generator output obtained from the python based CGMES reader for IEEE 14 bus system 101

List of Tables

2-1 Variables and class overview 14

5-1 Simulation set-up in OpenModelica 51
5-2 Results when tap changer is added at the HV side of the transformer (steady-state values) 57
5-3 My caption 58
5-4 Comparison of steady state voltage values 59
5-5 Powerflow values of the 9-bus test system 63

A-1 Input Definition of the RMS-Model 77
A-2 State Variables 77
A-3 Output Definition 78

B-1 Synchronous machine parameters used in the validation process 91
B-2 Line parameters used in uncontrolled generator model 91
B-3 Exciter parameter values used for validation 95
B-4 Governor system parameter values used for validation 98
B-5 Line parameters used in modified BPA system for Transmission line near the load 98
B-6 Line parameters used in both of the parallel transmission lines 98
B-7 Exciter parameter values used for validation of modified BPA system 99
B-8 Governor system parameter values used for validation for BPA test system 99
B-9 Tranformer near G1 99

Glossary

- **CGMES-CIM** Common Grid Model Exchange Standard based Common Information Model
- **RES** Renewable Energy Sources
- **ENTSO-E** European Network of
- **CGMES** Common Grid Model Exchange Standard
- **TSO**Transmission System Operators for Electricity
- **UML** Unified Modeling Language
- **CIM** Common Information Model
- **CPS** cyber physical systems
- **OpenIPSL** Open-Instance Power System Library
- **ICT** Information and Communication Technology
- **IDE** Integrated Development Environment
- **MSL** Modelica standard library
- **iTesla** Innovative Tools for Electrical System Security within Large Areas
- **DSL** DIgSILENT Simulation Language
- **PU** Per-unit
- **EMT simulations** Electromagnetic simulations
- **OEL** over-excitation limiter
- **EC** European Commission
- **IOP** Interoperability tests

Chapter 1

Introduction

In this chapter, a background to the field of research is presented. Introduction to the concepts like information model exchange, CIM, CGMES, OpenIPSL and OpenCPS will also be provided in this chapter. Finally, the problem definition, goals, research questions and outline of this thesis project are described.

1-1 Background

The European Union's energy and climate policy objectives for 2030 has led to the large integration of Renewable Energy Sources (RES) into the grid. The contribution of renewable energy in the electricity grid had increased to 13% in 2012 as a proportion of total energy consumed and is expected to rise to 21% in 2020 and 24% in 2030 [2]. This has impacts on the electricity grid infrastructure, operations and the functioning of the electricity market itself. With penetration of new technologies and the growth of large scale power systems, the need for complex power system simulation has increased. With this in mind, achieving valuable simulation results has become one of the important research questions in the area of electrical power engineering [3].

Power system simulation can be classified into steady state and dynamic simulations. Steady state simulations involve computations to find new system equilibrium. The dynamic simulations on the other hand, computes system evolution through time and are slower than steady state simulations [4]. Different kinds of models are designed to meet different simulation requirements. The emphasis of modelling complexity mainly deals with the type of study, described below and shown in 1-1.

- **Electromagnetic transient study**: This study provides information about interactions between the magnetic field of inductances and electrical field of capacitance in power systems [5]. Consists the most detailed type of power system models. Studies like the analysis of transients after asymmetrical faults, or operations

of power electronics converters are covered. EMTP-RV, PSCAD/EMTDC, etc. are mainly used to study the electro-magnetic transient of the system after large or small disturbances, in the time scale between microseconds and seconds. Usually, it is difficult to simulate very large networks and needs to be dealt with by using discrete solvers. This kind of study in Powerfactory is referred to as EMT simulation study.

- **Electromechanical transient study**: Deals with the interaction between the mechanical energy stored in the inertia of rotating machines and the electrical energy stored in the system [5]. The most common softwares are PSS/E, Eurostag, Simpower, PSAT, etc. These kind of software tools mainly simulates and investigates the response of system after large or small disturbances, such as short-circuit fault, opening of transmission lines, loads and generators. These tools are also used to study the system ability to maintain stable operation. In case of large networks,these models are usually simplified (equivalent models are used for faster computations). This kind of study in PowerFactory is referred to as RMS simulation study.
- **Quasi steady state dynamics study**: This kind of study uses a simplified representation of power system components, which can be used in long-term dynamic simulation. The models used are simplified further by neglecting most of the dynamics (by replacing the differential equations by algebraic equations).

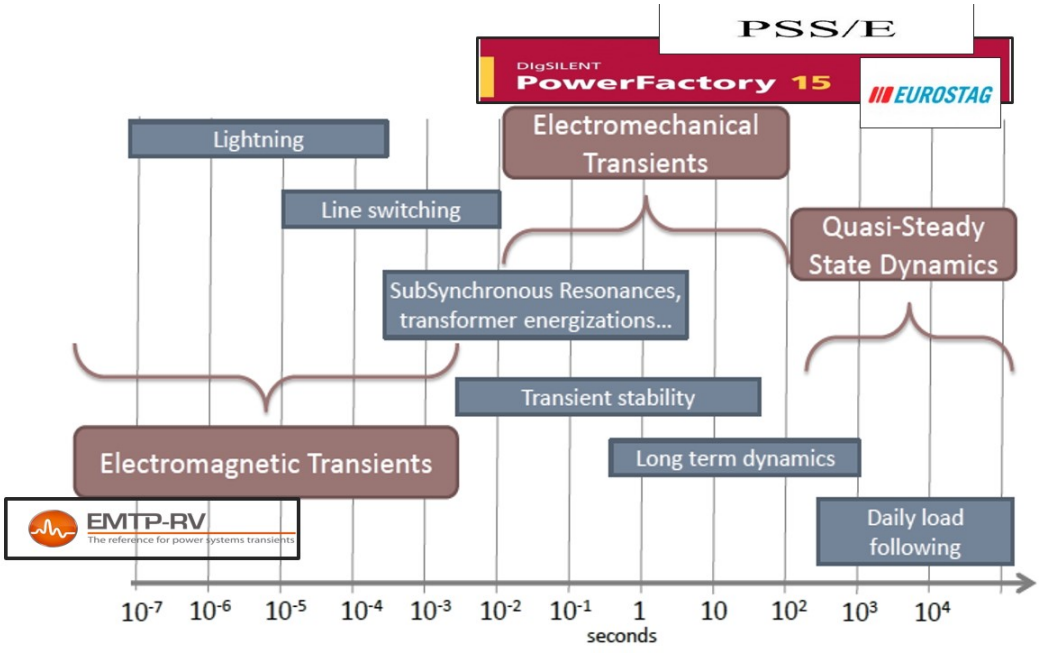


Figure 1-1: Types of simulation studies and respective tools used

Simulation tools like DIgSILENT PowerFactory, SimPowerSystems, etc. help the user with varied types of power system studies. These tools consider both the electromagnetic transient and electromechanical transient models and can provide simulation results in a broader time range [6]. They also help obtain more detailed and accurate simulation solutions. However, the libraries integrated in the tools are commonly closed for modifications (black-box models), which limit the flexibility of simulation to some extent [6].

1-1-1 CGMES-CIM

A powerful modelling language which can realize, not only an accurate model representation, but also increase computational efficiency of model simulation could be an effective solution [7]. Usually for a energy player (TSOs, DSOs and other private generators), the behaviour of the neighbouring control areas becomes significant during their grid development and maintenance. For this, they would require accurate simulation tools and data from their neighbouring energy players. In this regard, an efficient exchange of information between the energy players in Europe (TSOs, DSOs and other private generators) becomes very crucial [8].

To enable better coordination and data exchange between TSOs, a commonly agreed exchange format standard called Common Grid Model Exchange Standard (CGMES) was introduced by the European Network of Transmission System Operators for Electricity (ENTSO-E) [8]. ENTSO-E represents around 42 electricity transmission system operators from around 35 countries in Europe. CGMES is based on Common Information Model (CIM).

CIM is based on the Unified Modeling Language (UML) and, uses classes and relations to represent the semantic information. The CIM follows Object-Oriented Programming (OOP) principles defining network analysis data in terms of building blocks, which contain the basic components and topology of the power network as shown in Figure 1-2 [7]. A CIM represented power system objects can be quickly converted to classes in an appropriate object-oriented programming language application [7].

Data in CGMES-CIM is partitioned into types of profile model parts as shown in Figure 1-2.

1. Invariant profile model types include the data that describe the qualities of the grid that are inherent in its construction and will not typically change except as a result of new construction activity. (the term “invariant” here emphasizes the goal that this information about a given element should be the same in every study representing that element) [7].
2. Variant profile model types are those that the engineer will set up differently for different kinds of studies. For example, a capacity planning study might look at a heavy-load scenario, while a real-time state estimator studies current load conditions [7].

The CGMES-CIM profiles are:

- Equipment profile (EQ) profile provide the details about the basic steady-state characteristics of all the equipments connected to the grid.
- Diagram layout (DL) profile defines how grid parts may be organized into schematic diagrams [7].
- Short-circuit (SC) profile defines additional data required for short-circuit analysis [7].
- Dynamic (DY) profile defines additional data required for dynamic analysis.
- Geographical layout (GL) profile defines geographical data [7].

- Steady-state hypothesis (SSH) profile defines the input choices for power flow, consisting mainly of component status, generation and load values, regulation targets, limits [7].
- Topology (TP) defines the bus-branch topology that results from eliminating switches from the model [7].
- State variables (SV) defines the steady-state solution (the solution after powerflow).

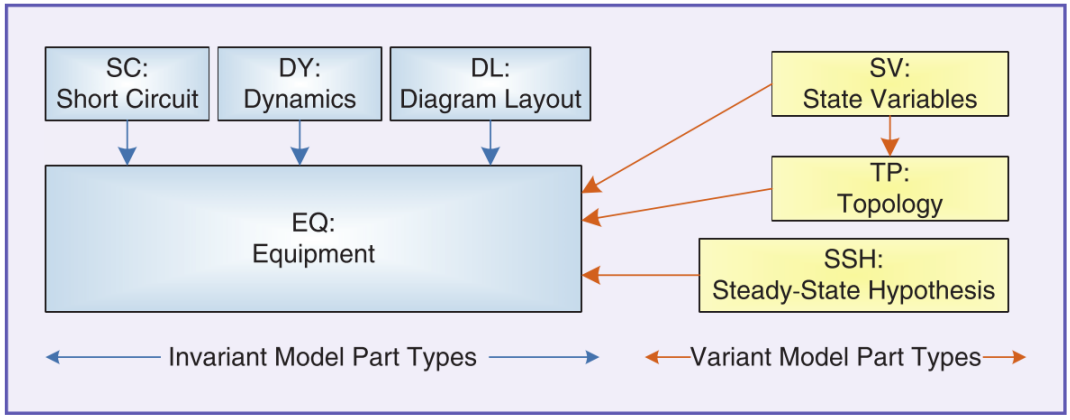


Figure 1-2: CIM model part types *

1-1-2 Dynamic profile in CGMES-CIM

The DY (dynamics) profile supports the exchange of dynamic behaviour models that are defined by IEEE / CIGRE standards for Power System Stability Analysis [9]. There are three ways in which the current version of the dynamics profile is designed to support under CGMES [10]:

- **Standard model exchanges**: A simplified approach for exchanging dynamic models. The behaviours of the models are defined in a standard manner. The current profile supports a set of standard models only.
- **User-defined model exchanges**: It is a way to exchange full information on user defined models. Currently, the DY profile is not fully equipped for this type of exchange, to be precise, it does not support the mechanism to model the individual elements from the control blocks and describe how they are linked to each other [10].
- **Proprietary models exchange**: This process provides users with the ability to exchange the parameters of a model representing a vendor-proprietary tool, where an explicit public description of the model is not desired. It is a process to exchange proprietary models (black box models like .dll etc). All parties participating in the exchange should have the model (.dll, etc.). Only parameters of models are exchanged. The current version allows for the exchange of model information like the name and description with unlimited number of parameters per model [10].

*Figure taken from reference [7]

1-2 Problem definition

One of the developments in the field of power systems is the effort to achieve a common language that supports interoperability between power system tools with respect to dynamic simulations [2] - [11]. CGMES based CIM helps in exchange of dynamic data by sharing the parameters(values) of associated blocks needed for dynamic studies [9].

Within the current CGMES (v2.4), dynamic standard models are described in both graphical (black box/control block diagrams) and textual format [9], [12], which could lead to different interpretations. Therefore, a reliable, organised approach is required to capture, maintain and exchange information of each standard model (IEEE models) [10]. Also, the utilities often introduce new equipments in their grid, develop new or improved standard models and maybe even add new modelling functionalities for business processes. Therefore, the classes in the standards need to be extended every single time, which becomes a complex task for all the energy players involved. Thus, unambiguous model exchange between different applications/software platforms might become challenging [10].

Another drawback could be that different non-standard components (like exciters and other control systems) are not consistent in all platforms due to different modelling philosophy, assumptions and simplifications. Also, the conventional block diagram modelling forces the user to share only parameters, which sometimes leads to different interpretations in power system tools [12]. The mismatch could be as shown in Figure 1-3.

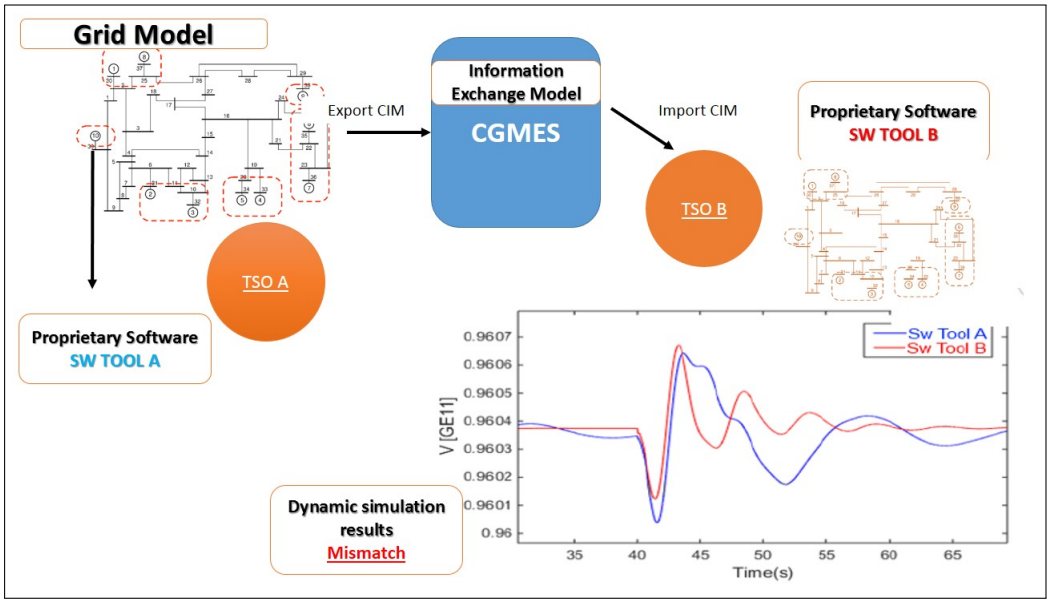


Figure 1-3: Issue with the dynamic model exchange

Additionally, within an utility, there is need for a common dynamic grid model that could be used during all the stages of the grid development. Usually, the utilities use different tools (PSSe, PowerFactory and EMS) at different stages of grid development and analysis. Here, the interoperability might be a huge factor, as the grid models need to undergo specific changes before it can be simulated in another platform. Often, this requires expert intervention and is time consuming.

1-3 Motivation

As discussed, there are limitations in the exchange of dynamic models by making use of CGMES for unambiguous dynamic model sharing. Though most of the power system simulation tools are reasonably user-friendly and computationally efficient, they have a closed architecture and different modelling philosophies [13] & [14]. Thus, there is motivation to use an open-source modelling language such as Modelica, to describe electric networks [13]. Also, the latest version of CGMES v2.5 proposes Modelica for user defined dynamic model exchanges. Therefore, Modelica based dynamic model exchanges could become very crucial in the near future among the energy companies for model exchange process and co-ordination.

In general, an equation-based modelling language helps to know the exact model (in terms of equtions and parameters) of the system independently of the software in which it is modelled. An explicit mathematical representation can be made using Modelica that helps to wave out any ambiguity about the model, while enabling simulations with diverse tools. Also, Modelica separates the solver from the model, which is an advantage while working with a large system and the models can be easily exchanged between simulation environments[12].

The development of a Modelica library for phasor time-domain simulation of power systems was first initiated during the iTesla project and later moved into the Open Cyber Physical Systems (OpenCPS) project. The Open-Instance Power System Library (OpenIPSL) contains a set of power system component models necessary for the execution of prototype demonstrations on the iTesla platform. Currently, OpenIPSL already consists PSAT and PSSE compliant dynamic models in it's library, but it lacks PowerFactory models. Several TSOs might revert back to using PowerFactory for dynamic simulations in the near future because of its features and improved efficiency. Addition of PowerFactory based models into OpenIPSL would find significance with TSOs using other proprietary tools. Also, within a TSO, there is a great need for one grid model for dynamic simulations that utilizes initialization data from different sections of grid development like the grid planning stage, day ahead, hourly planning stage and from the Energy management systems (EMS).

1-4 Goals and research questions

The overall objective of this thesis project is to make use of CGMES-CIM files directly to simulate the dynamic responses of a grid with a fixed topology. The CGMES-CIM files contain all the information regarding the grid under study. This information is converted to specific Modelica class files utilizing the OpenIPSL library files (Mapping algorithm is utilized for this purpose) and Python. Therefore, the user would be able to perform dynamic studies with just a click of a button, and the dependency on proprietary tools is avoided. The overall idea of the project is described in Figure 1-4. The idea described in Figure 1-4 is further simplified to obtain the objective of the project:

"Investigate the feasibility of using open source software to overcome interoperability issues with dynamic model exchange, using CGMES and OpenIPSL"

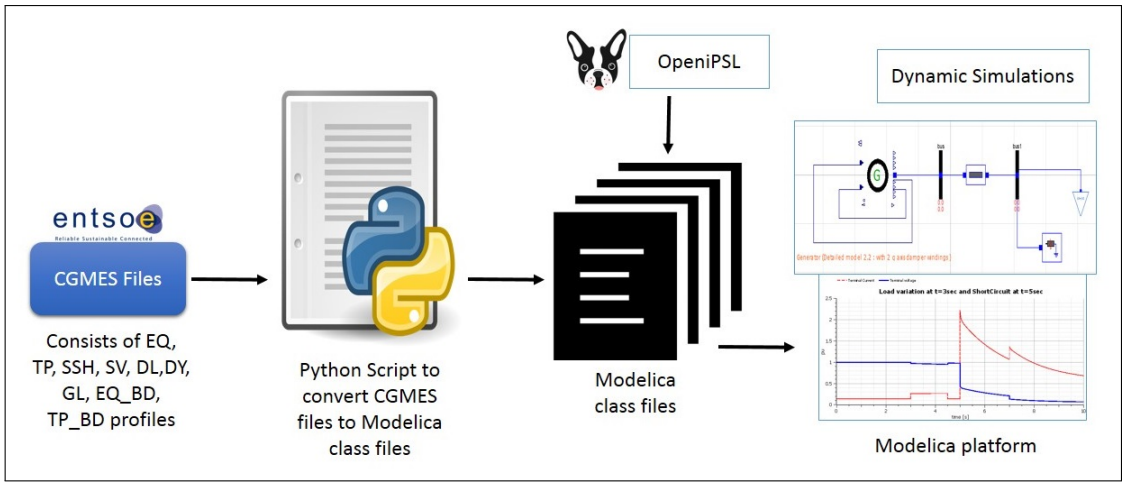


Figure 1-4: Overall Idea

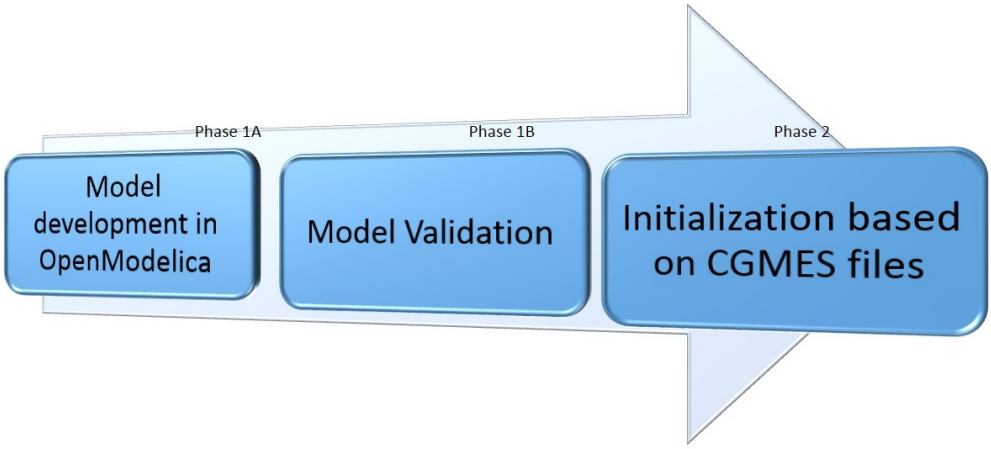


Figure 1-5: Work division

The open source software considered is OpenModelica and Modelica language would be used to model the grid elements with PowerFactory as a reference tool. Phase 1 of the thesis is to develop PowerFactory models in OpenModelica and validate them. And in this way extending the OpenIPSL library with the non-existing PowerFactory (dynamic) models. Phase 2 of this Master Thesis deals with making use of CGMES model for the initialization of the Modelica based grid model. The work division is shown in Figure 1-5. OpenModelica doesn't contain its own powerflow toolbox, powerflow needs to be carried in other simulation tools and later loaded into OpenModelica and this process is called initialization.

Note: All models in OpenIPSL are programmed in such way that by introducing a power flow solution consisting of Voltage magnitude in pu, Voltage angle in degrees, Active power in MW and reactive power in MVAr (usually from another tool), the initial guess is computed as a parameter within each model (eg: generator model) and are provided into the initial equations that are used to solve the overall initialization problem [15].

The most important research questions that are investigated:

1. Is it possible to model and validate a PowerFactory based grid models in OpenModelica in terms of dynamic simulations and thereby extending the openIPSL library?
2. Can the initialization process (shown in Figure 1-6) in Modelica be implemented by directly utilizing the CGMES files (SV,SSH profiles)?

The test system chosen (shown in Figure 1-7 [16]) is a modified Bonneville Power Administration (BPA) test system described in [17]. The test system can capture transient (angle), frequency and voltage instability phenomena (resulting in system collapse), all within one single system.

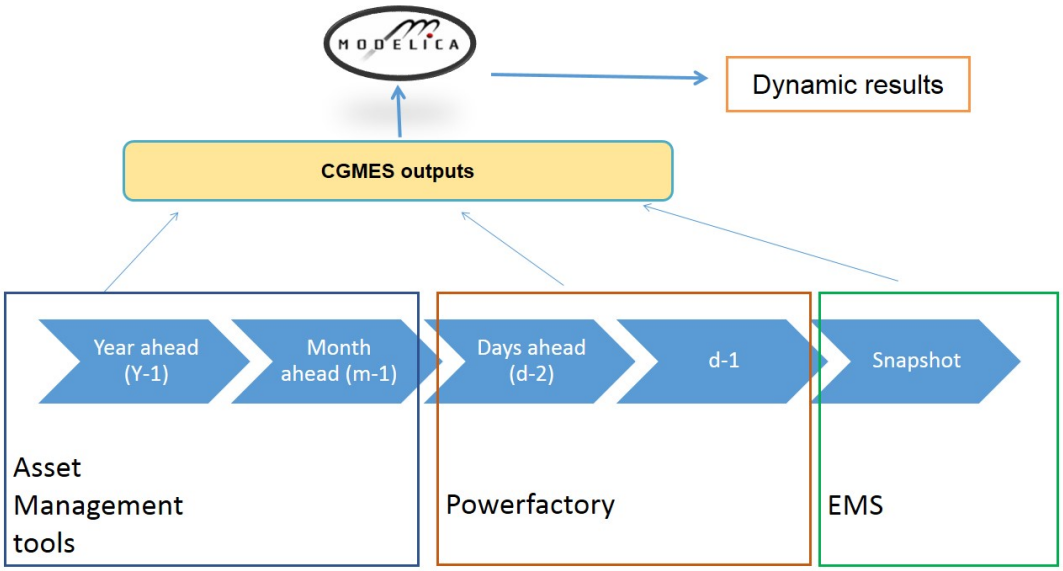


Figure 1-6: Initialization process for dynamic simulations in OpenModelica

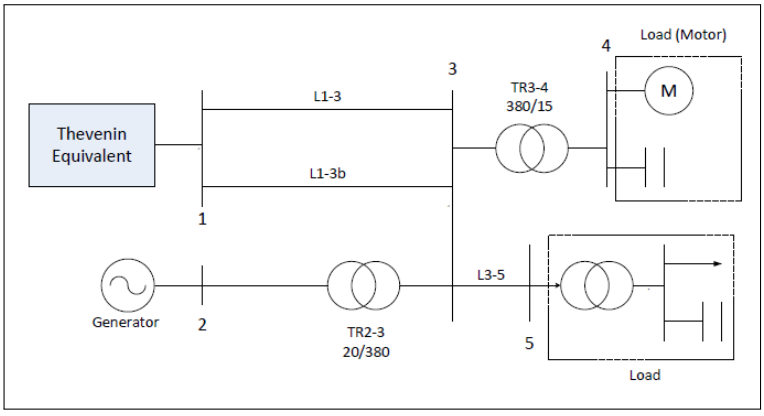


Figure 1-7: 'All in one' test system or modified Bonneville Power Administration (BPA) test system

Since, the proof of concept is carried out using PowerFactory models, research question (1) needs to be implemented first. Later on, using (1), the concept of importing initialization data directly from CIM files would be tested and verified, as described in (2). Figure 1-8 describes the workflow implemented in this thesis. The research questions were framed focusing on aspects that TenneT and/or other similar TSOs are concerned about.

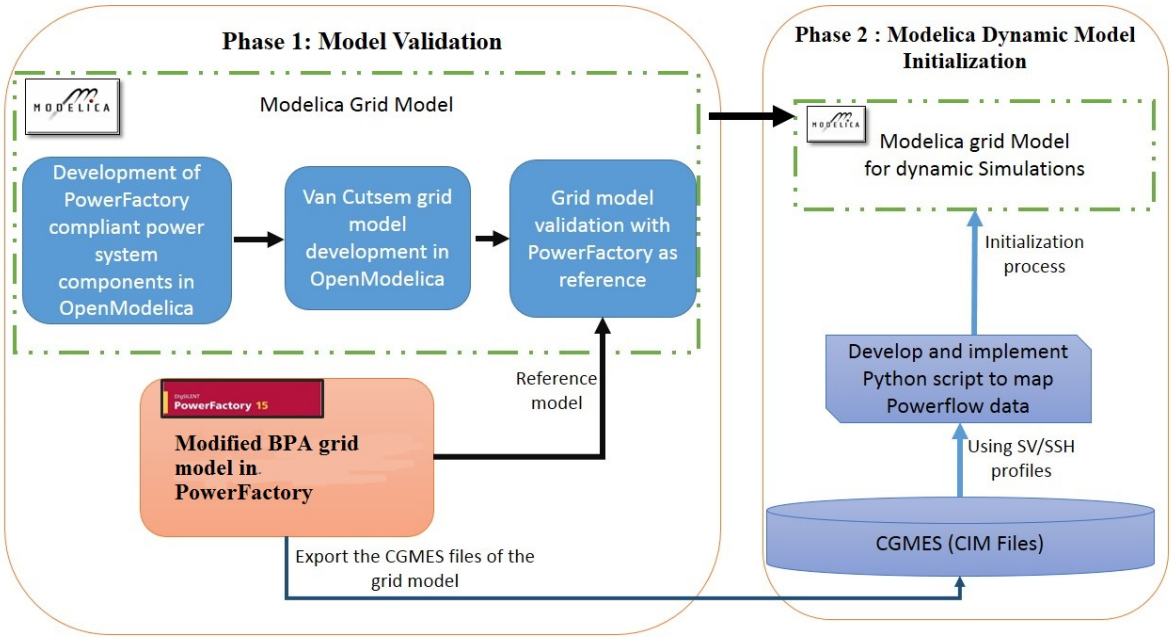


Figure 1-8: Workflow

The objectives will be achieved in order to evaluate the feasibility of utilizing Modelica with OpenIPSL and CGMES for internal dynamic model exchanges within a TSO.

1-5 Contributions

- Contribute models to the Open-Source Modelica Power system library - Update OpenIPSL with PowerFactory based models.
- Develop tutorials and documentation for an Intelligent Electrical Power Grids (IEPG) course - Make video tutorials and develop instructions for Modelica based course work.
- One conference and one journal paper.

1-6 Approach and thesis outline

This chapter starts with a brief introduction to the thesis topic and why this project is of great importance to the field of electric power system simulations. The outline of the

project, its main goal and objectives are all explained in this chapter. Literature survey was carried out to understand the current practices with respect to dynamic model exchanges, CGMES, CIM and Modelica. The second chapter discusses Modelica language in detail and also includes details regarding using Modelica for power system dynamic studies. This is followed by, third chapter which discusses the implementation of PowerFactory based models in OpenModelica. Modelling philosophy in PowerFactory is discussed. Chapter 4 describes CIM, CGMES and python scripting is used for mapping the initial values directly from the CIM files into Modelica grid model. Chapter 5 shows the completed test system model in Modelica. Validation of component models, the results from the CGMES-CIM to Modelica converter and the initialization process is discussed. Last chapter of this report concludes the thesis and gives details on the probable future developments that can follow this thesis project. Figure 1-9 can be referred for an overview of the chapters in this thesis report.

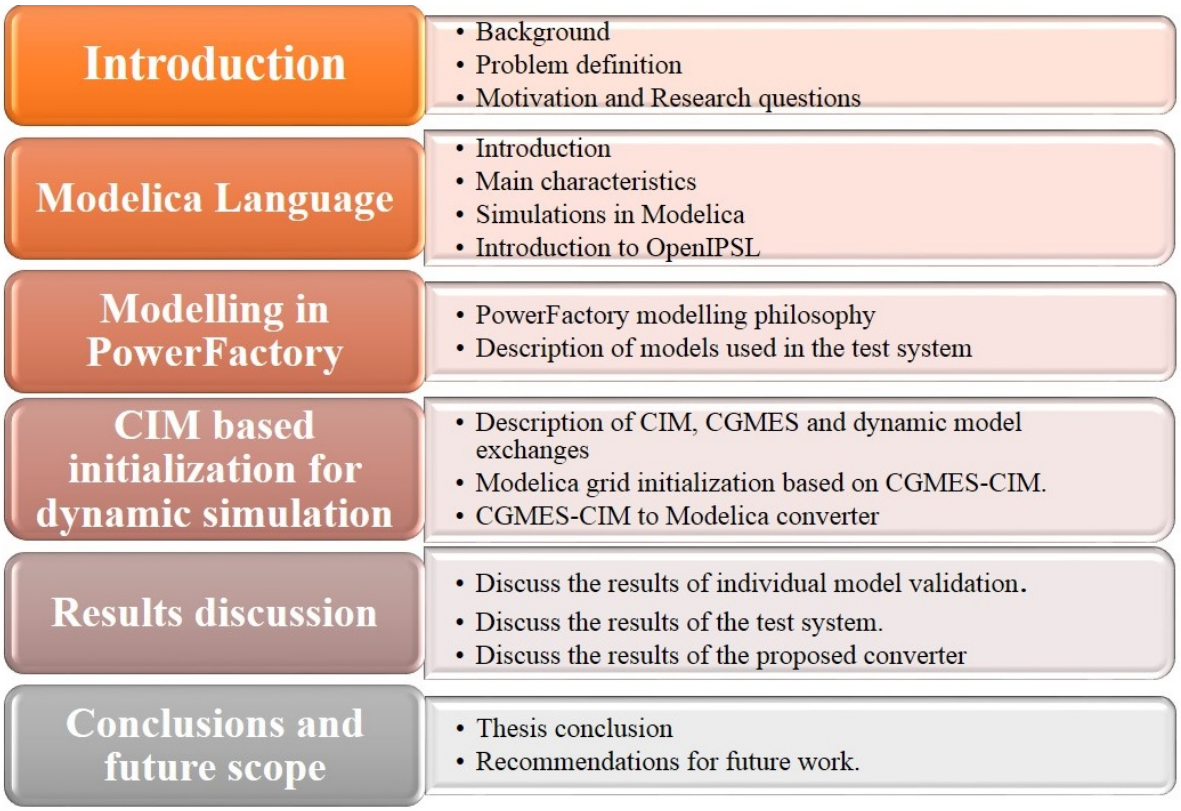


Figure 1-9: Thesis outline

Chapter 2

Modelica power system library

This chapter deals with the introduction of Modelica programming language. Modelica's main constructs encountered in power system library is discussed using an application example. Finally, the power system library used in the thesis - the OpenIPSL is described.

2-1 Introduction to Modelica

Modelica is an open-source object-oriented programming language for modelling of physical systems containing electrical, mechanical, hydraulic, thermal or any other process-oriented sub-components [6]. Therefore, Modelica based modelling could play a vital role in CPS, which is a combination of physical processes and Information and Communication Technology (ICT) [18], [19].

The Modelica language is used for modelling large, complex and multi-domain systems. It is designed in a way to support library developments and model exchanges. It's a modern language based on acausal modelling with mathematical equations and object-oriented constructs to facilitate re-usability of models [20].

After compilation of Modelica model into machine code by the Modelica compiler, execution of the code takes place with a numerical solver that is capable of solving combinations of Differential and Algebraic Equations (DAEs). A Modelica model is not coupled to any specific solver and therefore can be easily exchanged between simulation environments [1]. There are different Integrated Development Environment (IDE) based on a standardized Modelica language. IDEs provide the development environment and simulation facility. Therefore, several steps are involved in order to simulate a model, specifically the model is compiled into C-code and binded to a desired solver as shown in Figure 2-1.

Modelica helps the user to focus on what he/she wants to model rather than the

computational realizations. This way, the user can achieve the desired goals without being fully aware of the processing scheme [21].

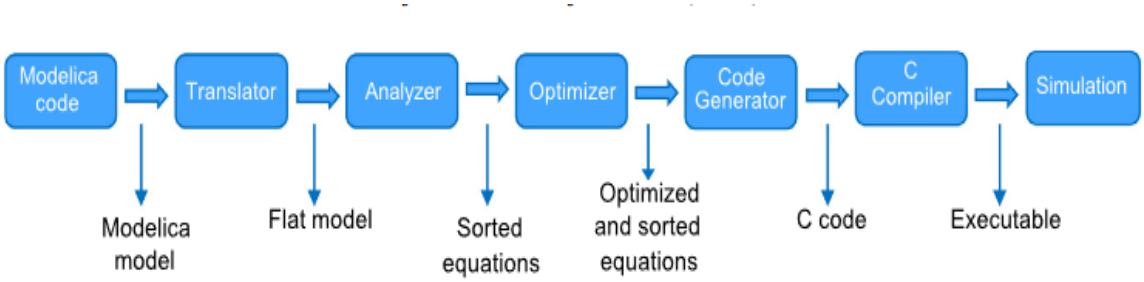


Figure 2-1: Modelica Process [1]

The developers of Modelica, the Modelica Association have developed Modelica standard library that includes about 1600 model components and 1350 functions from many domains. The latest version, *Modelica 3.4* was released in April 2017 [20].

2-1-1 Main characteristics

Modelica as a modelling language offers its own unique capabilities which play an important role in power system studies. Few of the characteristics are mentioned [6],[22]:

1. **Equation-based**. Modelica is based on equations instead of assignment statements. The equation level modelling allows for greater flexibility as they do not prescribe a certain data flow. Such modelling also helps decompose complex systems into simple sub-models making it easier to understand, share and reuse.

2. **Object-oriented**. Modelica uses a general class concept, which unifies classes and general sub-typing into a single language construct. Re-usability and evolution of models is made easier.

3. **Multi-domain modelling**. Model components from different physical domains can be connected and simulated without interference of each other.

4. **Model exchange**. In Modelica, dynamic models can be exported as Functional Mock-up Units (FMU) and used in other simulation environments. (eg: Simulink)

5. **Re-usability**. Within Modelica, the definition of each component consists of equations using only local variables and connectors. Thus, there is no connection between a component and the rest of the system, except from the connector.

2-1-2 Softwares based on Modelica

Modelica is a non-proprietary (open-source) language so it can be used for free by anyone. This enabled a wide range of softwares based on Modelica. A lot of commercial (Dymola, CyModelica, MapleSim etc.) and free software tools (JModelica, Scicos, SimForge etc.) are

now available for industrial and academic use. The biggest difference among them is the interface and solvers they use. An open-source platform called OpenModelica (see Figure 2-2) is utilized in this thesis project.

OpenModelica Connection Editor is an advanced open-source Modelica based software tool. It provides friendly graphical user interface with easy-to-use model creation, simulation of models, and plotting of results. The interface is extensible enough to support user-defined extensions. Models can be in both code form and graphical form [23].

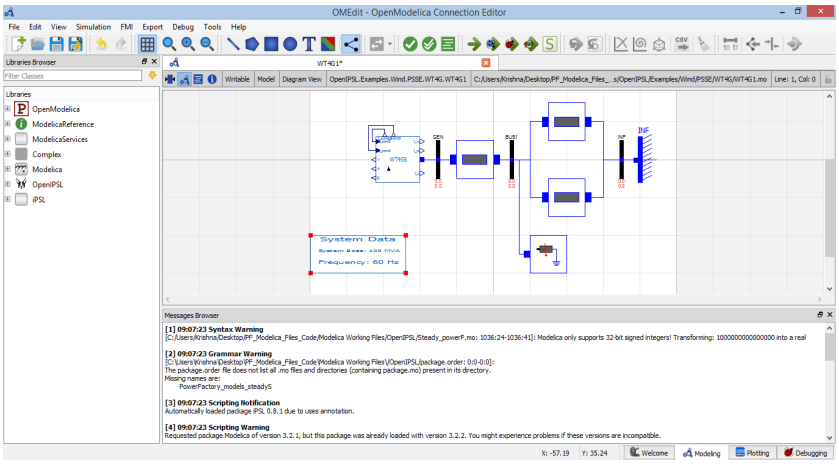


Figure 2-2: OMEdit - OpenModelica Connection Editor

2-2 Modelling environment

The basic structure of modelling in Modelica is the class. A class consists of name, list of declaration of its attributes and equations. Every object is an instant of corresponding class which is defined by its data and behaviour [24].

2-2-1 Attributions and equations

The attributions of a Modelica class contains constants, parameters, and variables. By default, all of the variables are continuous time variables which change their values continuously during simulation [24]. Variables can be declared by prefixes constant, parameter, discrete. Constants will never change their value after their definition, eg: constant Real pi=3.1415. If the variables have prefixed parameter, then they can be assigned by users before simulation or after the stage of initialization, but will remain constant during time-dependent simulation. The discrete variables can change their values only at event instants during simulation. Furthermore, time is a global built-in variable, which can be used without declaration [20]. For the sake of readability, different types of variables and classes can be defined in Modelica as shown in Table 2-1. Equations in the model are followed by the keyword 'equation'. The differential equations are defined using the 'der' operator, eg: der(x)=x+1, where, x is the state variable. Each state requires an initial condition, eg: der(x)= 0 which can also be defined during its parameter definition, eg. some_parameter_name(start=0).

Table 2-1: Variables and class overview

Keywords	Description
Real	Default variable type ;floating point eg. 50.022
Integer	Default variable type; integer eg. 1, 579
Boolean	Default variable type; eg. true, false
String	Default variable type; eg. "HelloWorld"
type	Class to define variable types
connector	Class to define interfaces
model	Class to define model components
package	Class to define library, no equations here.
block	Class just like model-class, consists only public input, output an parameter

2-2-2 Simulation parameters

As discussed in the previous section, the simulation solvers in Modelica programming environments are decoupled from the model. This makes it easier to exchange models between different simulation platforms. The basic characteristics that will change the outputs of the simulations are the solvers used, the integration time step and the tolerance. The user can choose between several kinds of solvers. In the example shown in Figure. 2-3, the simulation time is set for 10 sec and the tolerance of the solver is set to 1e-6. The solver is radau5 - Radau IIA with three points. the size of the solver is set by the number of intervals.

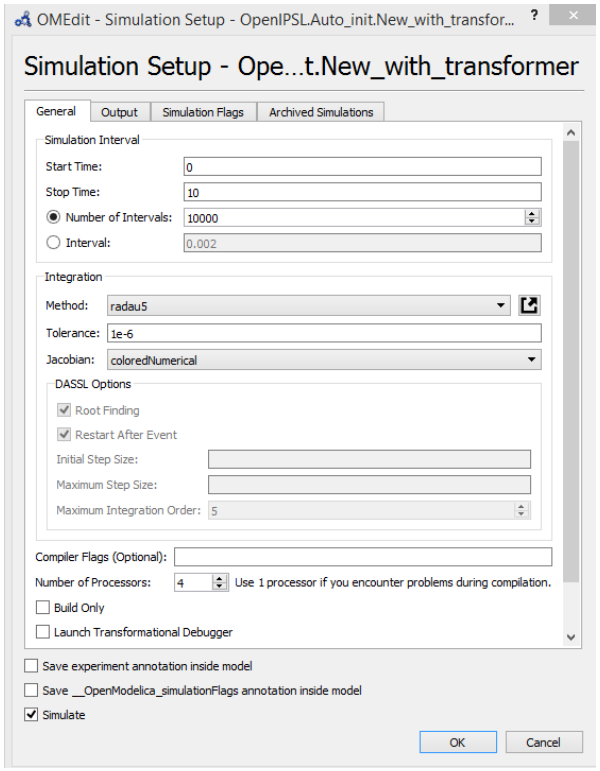


Figure 2-3: Simulation options in OpenModelica

2-3 Application example

Complex models can be created by introducing equations that are accurate using Modelica. Code of an electrical bus is shown in Figure 2-4. In Modelica, defining a parameter or a variable is pretty straightforward, as shown in Figure 2-4. Modelica works on the concept of class to represent models. The collection of basic models forms a library.

Models are connected by connectors, which are special Modelica classes that define the connection of two or more components[6] . In the example shown in Figure 2-4, *PwPin* connectors are used for electrical components and similar class called *Impin* connectors exist for non-electrical components.

```
1   within OpenIPSL.Electrical.Buses;
2   model Bus "Bus model 2014/03/10"
3
4   OpenIPSL.Connectors.PwPin p(vr(start=V_0*cos(angle_0*Mc
    vi(start=V_0*sin(angle_0*Modelica.Constants.pi/180))) a
14
15    Real V(start=V_0) "Bus voltage magnitude (pu)";
16    Real angle(start=angle_0) "Bus voltage angle (deg)";
17    parameter Real V_0=1 "Voltage magnitude (pu)"
18    annotation (Dialog(group="Power flow data"));
19    parameter Real angle_0=0 "Voltage angle (deg)"
20    annotation (Dialog(group="Power flow data"));
21
22  equation
23    V = sqrt(p.vr^2 + p.vi^2);
24    angle = atan2(p.vi, p.vr)*180/Modelica.Constants.pi;
25    p.ir = 0; p.ii = 0;
26    annotation ( ...);
104 end Bus;
```

Model of connectors

Variables

Parameters

Equations

Figure 2-4: Text view of electrical bus

Variables vr, vi, ir and ii are defined to present the real and imaginary parts of voltages and currents respectively. Two connection rules apply: the voltages must be equal and sum of currents should be zero as shown in Figure 2-5 [6] and Equation 2-1.

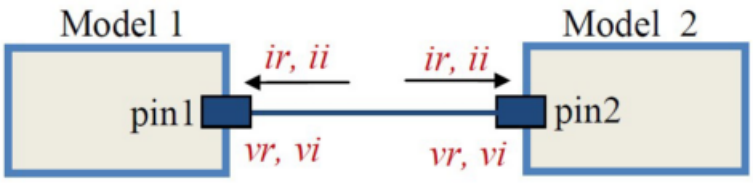


Figure 2-5: Model connections using PwPin

$$\begin{aligned} pin1.ir + pin2.ir &= 0; \\ pin1.ii + pin2.ii &= 0; \\ pin1.vr &= pin2.vr; \\ pin1.vi &= pin2.vi; \end{aligned} \tag{2-1}$$

Three types of model views are possible while working on models in OpenModelica and is as shown in Figure 2-6. The contents of these views are inter-dependent, that is the change in one particular view induces similar change in other two views.

1. **The text view**: the code is written. All the graphical modifications will have an impact on it.

2. **The icon view**: the graphical representation is designed. Allows to define the model's graphical appearance.

3. **The diagram view**: Is where the user can drag and drop previously made models, the models will appear as an instance with the appearance created in the icon view [6].

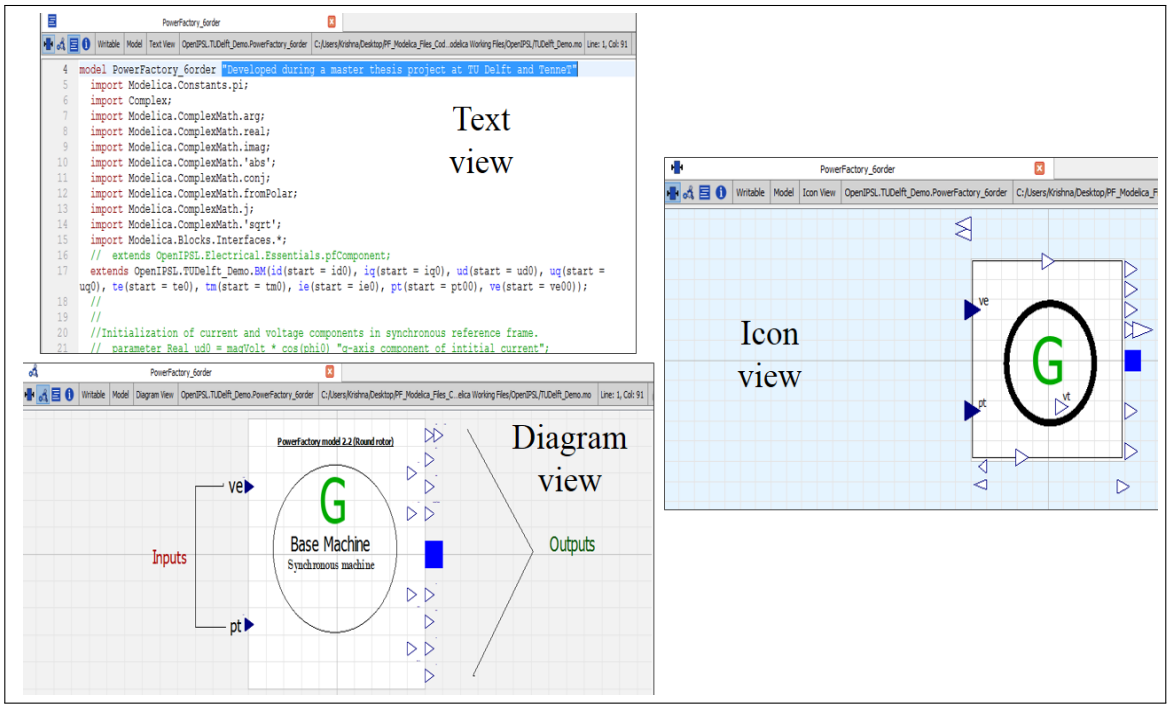


Figure 2-6: Modelica based generator model displayed in text, icon and diagram view (shown using OpenModelica connection editor)

2-4 Modelica libraries

2-4-1 Modelica standard library (MSL)

Modelica Standard Library (MSL) is a free (standard conform) library developed by Modelica Association. The brief view of the library is shown Figure.2-7. It is freely available in the source code and can be modified to be used in commercial software. The elements in the library can be used to model multi-domain system which can include: 1D or 3D mechanical, electrical (analog, digital, machines), control systems thermal, fluid and hierarchical state machines [24].

To build a system, one can drag-and-drop the components from the library to the graphical edit screen. Additionally, numerical functions, functions for strings, files and streams are also included in the library. This thesis utilize several basic blocks such as transfer functions and some numerical functions to build the components of control blocks of the generator system and other grid elements.

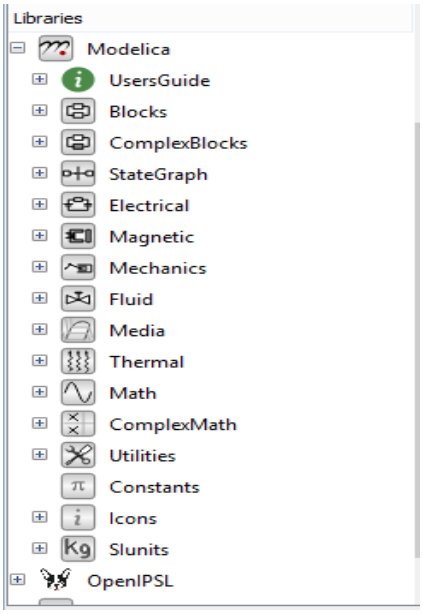


Figure 2-7: Modelica library

2-4-2 Open-Instance Power System Library

To overcome the growing complexity in pan European Transmission system, FP7 iTesla (Innovative Tools for Electrical System Security within Large Areas) was initiated to create a toolbox for operating the European transmission network [1].

OpenIPSL started as a fork of iPSL and is actively developed by SMartTS Lab members and other researchers from all over the world, as a research and education oriented library for power systems. These models are now part of OpenCPS project. OpenIPSL is an open-source Modelica library for power systems. It contains a set of power system

components for phasor time domain modelling and simulation. The models within this library have been validated against a number of reference tools. The main package OpenIPSL consists of several packages like Connectors, Electrical, Interfaces and Non-Electrical.

1. Connectors : This package contains three connectors; PwPin, is used for treating voltage and current as complex variables, ImPin is a simple connector for real variables and PwcobPin is used for changing from machine power base to system power base.

2. Electrical : This is the main package of the library with all the power system component models for phasor time domain representation. It contains several sub-packages like Bus,Branch, Machine, Loads, Controls etc. Several of these models are based on different proprietary software tools like PSAT, PSSE and Eurostag.

3. Examples : Contains set of examples that show the usage of the power system component models.

4. Interfaces : contains models used for data conversion. These models help to exchange data between the library and other Modelica libraries.

5. Non-Electrical : This package comprises of functions, blocks, and specialized models that is used to build the power system component models. They can be transfer functions, logical operators and so on., These models perform specific operations and are not available in the MSL.

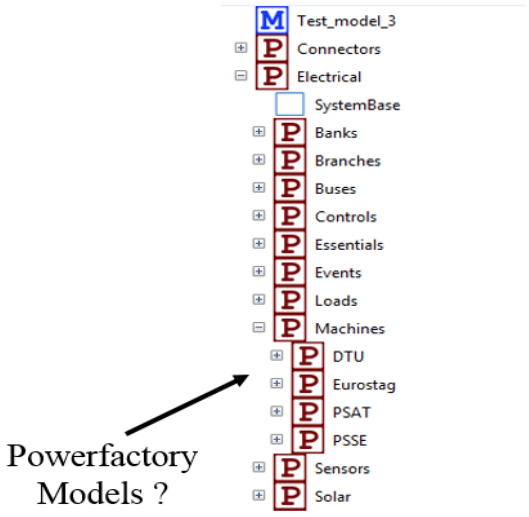


Figure 2-8: OpenIPSL

The current OpenIPSL consists of models based on simulink, Eurostag, PSAT and PSSe. Therefore, OpenIPSL is updated with the PowerFactory based models, which are created during the course of this thesis project. These models are developed based on the requirements of the test system considered in this thesis. The following chapter would provide the information regarding the modelling philosophy used in PowerFactory.

Chapter 3

PowerFactory based Modelling

In this thesis, PowerFactory is used as a reference to develop models in Modelica, and hence the modelling philosophy used in PowerFactory is discussed in this chapter. This is followed by detailed explanation of the mathematical representation of each electrical model that is used in this thesis. It starts from electrical element such as a load, transformer, generator and subsequently describes the governor and exciter systems used in this thesis.

3-1 DIgSILENT PowerFactory

Different events in the power system would cause varied disturbances in the system, which sometimes results in major black-outs. Possibility to predict and simulate the power system stability for different types of disturbances is quite crucial in grid operations [3]. Sophisticated simulation tools like DIgSILENT PowerFactory have emerged to integrate scientific approaches to assist power engineers in both academic and industry-oriented research studies [25]. PowerFactory includes load flow calculation, optimal power flow calculation, contingency analysis, protection, RMS/EMT simulation, reliability assessment and many other functionalities. It also supports interfacing with other packages (e.g. MATLAB, Phyton) for data exchange.

The PowerFactory modelling approach combines both graphical and script based modelling methods. It is also based on basic hierarchical levels of time-domain modelling [3]:

1. **DSL Block Definitions** are based on the "DIgSILENT Simulation Language" (DSL). They form the basic building blocks to represent transfer functions and differential equations for the more complex models [3].

2. **Common models**, are based on the DSL Block Definitions and are the front-end of the user-defined transient models as shown in Figure 3-3.

3. **Composite models** are based on composite frames. They help to combine and interconnect several elements (built-in models) and/or common models as shown in Figure 3-4. The composite frames (shown in Figure 3-2 [26]) enable the reuse of the basic structure of the composite model [3].

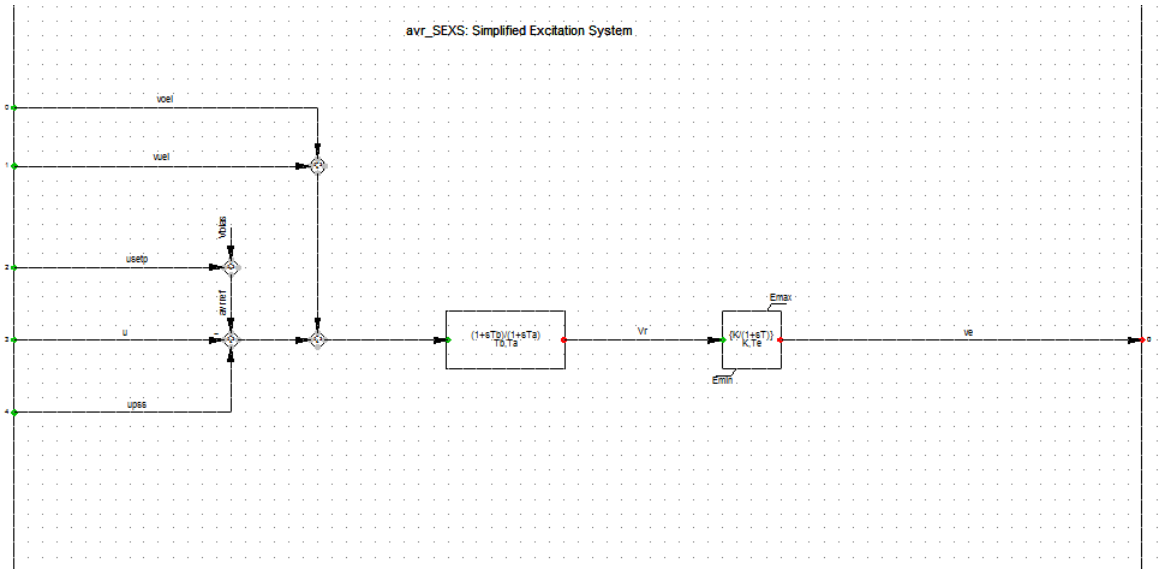


Figure 3-1: Model definition of simple exciter system consisting DSL blocks

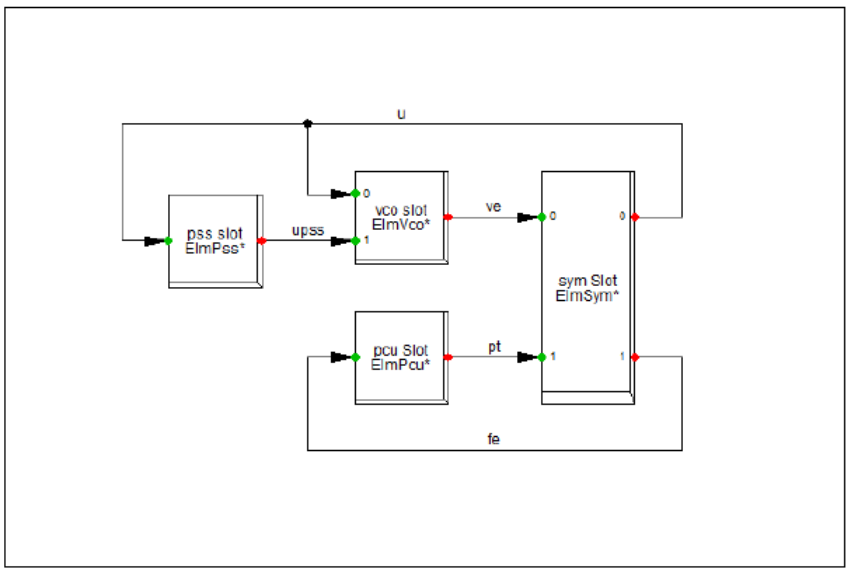


Figure 3-2: Composite frame of the generator

The Common Model combines general time-domain models or Block Definitions with a set of parameter values and creates an integrated time-domain model. The Composite Model on the other hand, connects a set of time-domain models inside a diagram (which is a composite frame).

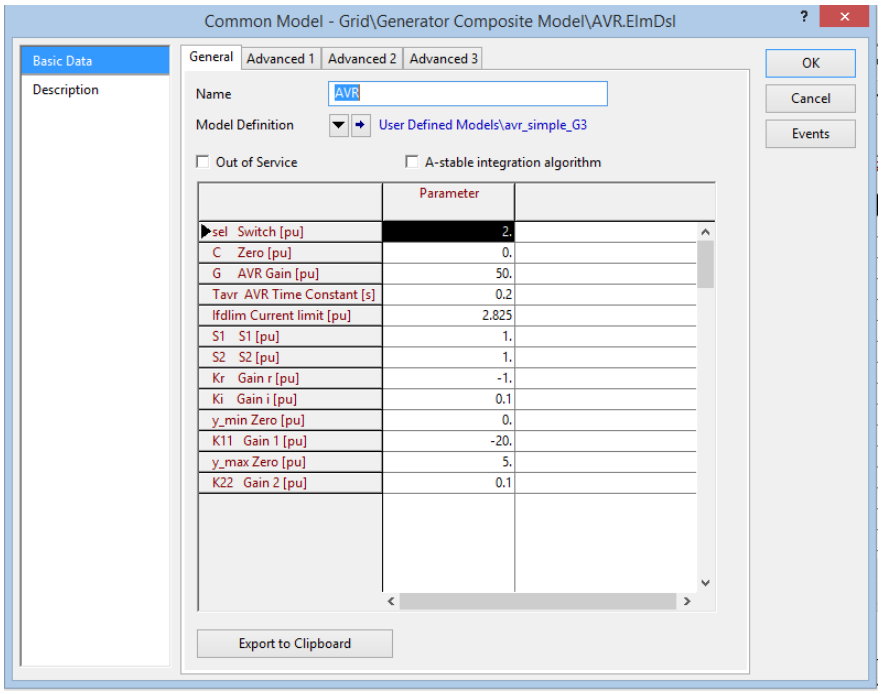


Figure 3-3: Common model of user defined AVR system

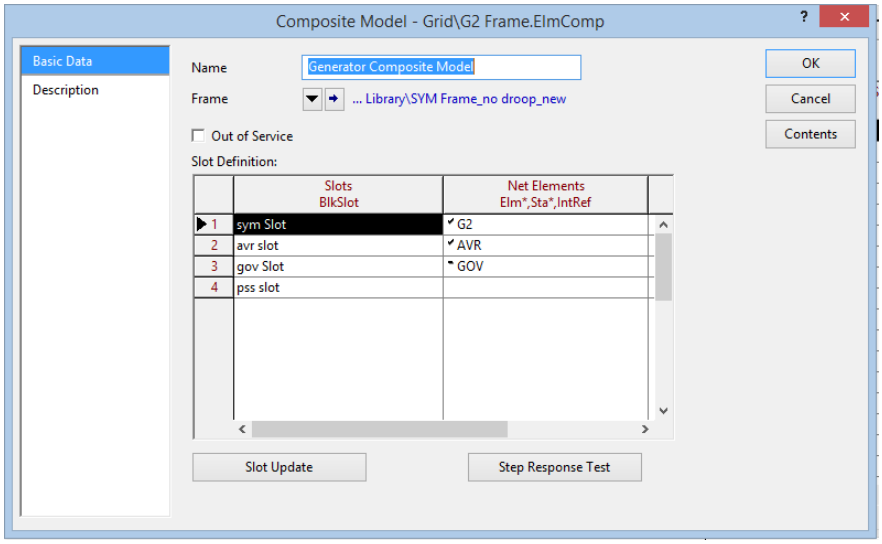


Figure 3-4: Composite model of a generator system

3-2 PowerFactory Models

This thesis considers 'All-in-one' or a modification of the BPA test system [27]. The test system captures transient (angle), frequency and voltage instability phenomena within one single system. In this thesis, models are developed specifically for this test system.

3-2-1 Synchronous generator

PowerFactory offers three types of synchronous machine models, Model 2.1 (salient pole machine), Model 2.2 (round rotor machine) and Model 3.3 (detailed generator model). They are all represented in a rotor reference system (dq-reference frame) [28]. This thesis makes use of round rotor system. The Model 2.2 is defined by six state variables and therefore is a sixth order generator model.

The rotor d-axis is always modelled by two rotor loops representing the excitation or field winding and the 1d-damper winding. The equivalent d-axis circuit is shown in Figure 3-5 [28]. For the q-axis, PowerFactory supports two models, a salient-pole rotor machine model having only the 1q damper winding and a round-rotor machine model with the 1q and 2q damper windings. The equivalent q-axis circuit for a round rotor system is shown in Figure 3-6 [28].

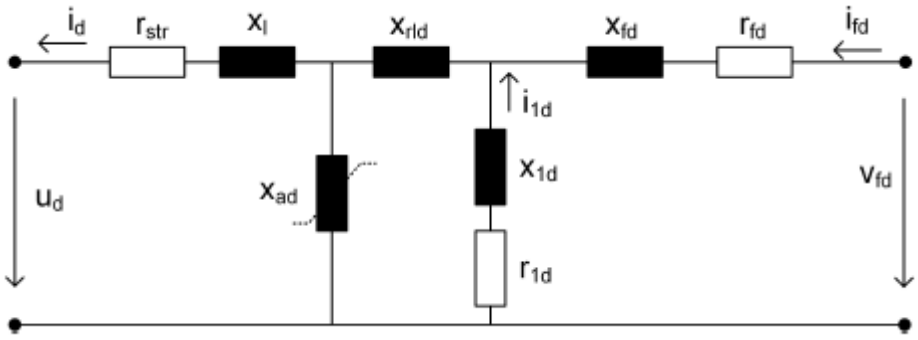


Figure 3-5: Equivalent d-axis circuit

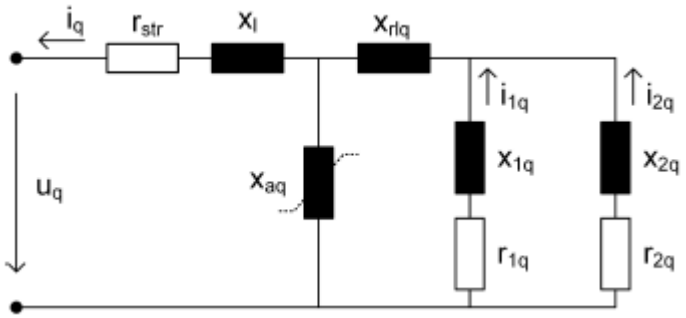


Figure 3-6: Equivalent q-axis circuit for round-rotor machine

Individual synchronous machines variables are transferred to rotor reference frame, however the network quantities are expressed in synchronous reference frame. The coordinate system is shown in Figure. 3-7 [24].

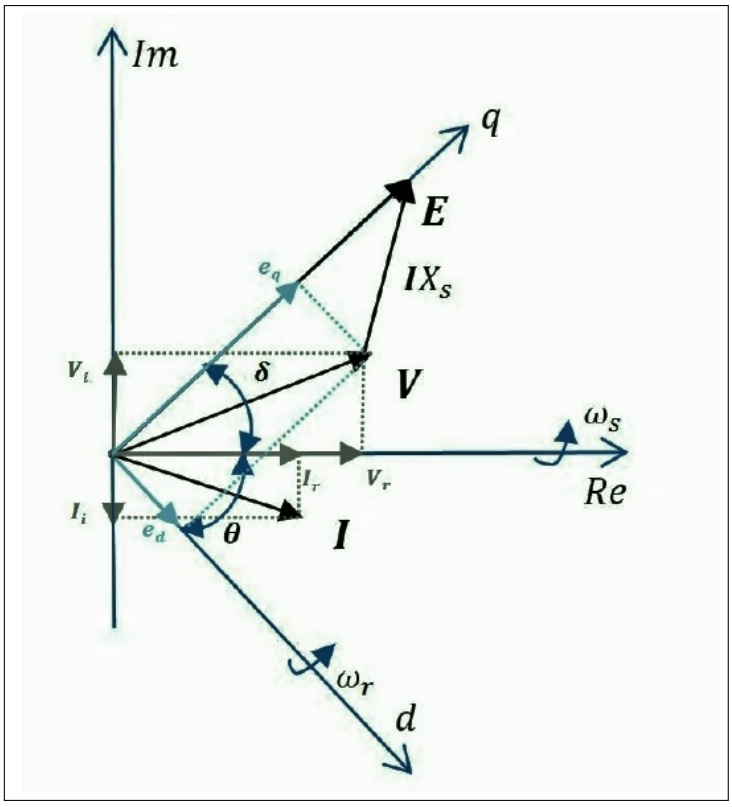


Figure 3-7: Coordinate system of Synchronous and Rotor frames*

Where, the generator is represented by a voltage source E behind a dynamic impedance X_s. The rotor angle here is angular spatial positions of the generator rotor shaft. The synchronously rotating reference axes leading the dq-axes by angle $\theta = \frac{\pi}{2}$ - δ. Thus, relationship between the quantities expressed in $Re - Im$ axes and dq-axes are as following:

$$\begin{bmatrix} V_r \\ V_i \end{bmatrix} = \begin{bmatrix} sin(\delta) & cos(\delta) \\ -cos(\delta) & sin(\delta) \end{bmatrix} \times \begin{bmatrix} e_d \\ e_q \end{bmatrix}$$

Figure 3-8: Relationship between coordinate systems

Normally, for large-scale stability studies the transformer voltage terms and the effect of speed variations are neglected in the stator voltage equations. The stator dynamics are relatively fast for stability studies. Therefore, for RMS-simulations, the derivatives of the stator quantities (transformer voltage terms) are not considered in the equations. This allows also using bigger time steps compared to the EMT model [26]. Taking into account this simplification, the stator voltages are given by:

$$u_d = u_d^{''} - r_{str} * i_d + n * x_q^{''} * i_q \tag{3-1}$$

$$u_q = u_q^{''} - r_{str} * i_q + n * x_d^{''} * i_d \tag{3-2}$$

*Figure taken from reference [26]

where the sub-transient voltages are given by,

$$\begin{aligned} u_d^{''} &= -n * \psi_q^{''} \\ u_q^{''} &= n * \psi_d^{''} \end{aligned} \tag{3-3}$$

The state equations of the round rotor system consists of 4 rotor rotor voltage equations and 2 equations of motion. The variables and parameters of the sixth order synchronous machine model are detailed in Figure A-1-2.

$$v_{fd} = r_{fd} * i_{fd} + \frac{1}{w_n}\frac{d\psi_{fd}}{dt} \tag{3-4}$$

$$0 = r_1 d * i_{1d} + \frac{1}{w_n}\frac{d\psi_{1d}}{dt} \tag{3-5}$$

$$0 = r_1 q * i_{1q} + \frac{1}{w_n}\frac{d\psi_{1q}}{dt} \tag{3-6}$$

$$0 = r_2 q * i_{2q} + \frac{1}{w_n}\frac{d\psi_{2q}}{dt} \tag{3-7}$$

where, $w_n = 2 * \pi * f_{nom}$ is the nominal angular frequency. Equations of motion defined by speed (n) and angle (ϕ):

$$\frac{dn}{dt} = \frac{t_m - t_e - t_{dkd} - t_{dpe}}{t_{ag}} \tag{3-8}$$

$$\frac{d\phi}{dt} = w_n * (n - fref) \tag{3-9}$$

where,

$$t_m = \frac{pt}{n} - dpu * n + addmt \tag{3-10}$$

$$t_e = \frac{i_q * \psi_d - i_d * \psi_q}{cosn} \tag{3-11}$$

All models in a Modelica library require initial guess values that should come from a solution of the steady state of the overall model. From these values, a Modelica tool solves the initialization problem for all algebraic and differential - state variables. Few initial equations for the mentioned state equations are defined in [28] and are as folows:

$$\phi_0 = arctan(u_t + (r_s tr + j * x_q) i_t) - \frac{\pi}{2} \tag{3-12}$$

$$t_{m0} = \frac{pt}{n} - (xmdm + dpu * n) \tag{3-13}$$

The remaining initial equations are discussed in the Appendix code A-1-2. The model in Modelica contains two inputs ve and pt (for inputs from exciter and governor control blocks).The initial equations are discussed in Listing 3.1. It also consists of several output including the torques as shown in Figure. 3-9. All the details of output/input parameters are discussed in Appendix A-1. A partial model called Base Machine (BM) is created so as to help further extension of PowerFactory synchronous machines like salient pole (Model 2.1) or detailed model (model 3.3) and the Modelica code is given in A-1-2.

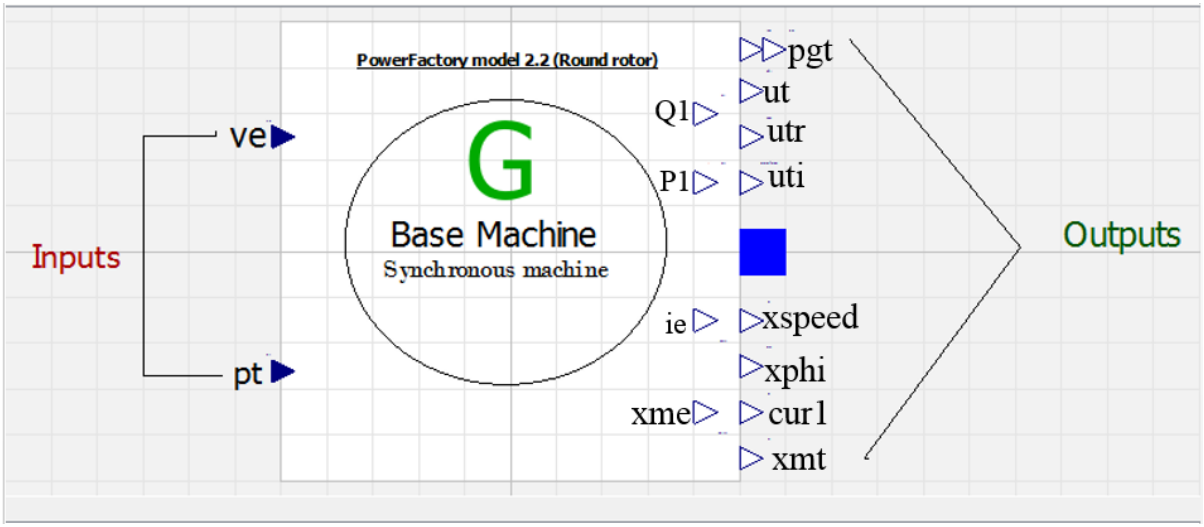


Figure 3-9: Modelica model of the synchronous machine based on PowerFactory round rotor model (Machine model 2.2)

Listing 3.1: Synchronous generator initialization equations

```
parameter Real te0 = (iq0 * psid0 - id0 * psiq0) / cosn"electrical torque";
parameter Real tm0 = pt00 / n0"initial mechanical torque";
parameter Real ie0 = xadu * ifd0"initial excitation current";
parameter Real ve00 = ie0"initial excitation voltage";
parameter Real ufd0 = rfd / xadu * ve00"initial field voltage";
parameter Real pt00 = te0 * n0"initial turbine power";
 //Initial conditions for rotor flux linkages
parameter Real ifd0 = ((xad + xl) * id0 + uq0 + rstr * iq0) / xad;
parameter Real psid110 = kfd * psifd0 + k1d * psi1d0;
parameter Real psiq110 = k1q * psi1q0 + k2q * psi2q0;
parameter Real psid0 = (-xd11 * id0) + psid110;
parameter Real psiq0 = (-xq11 * iq0) + psiq110;
parameter Real psifd0 = (-xad * id0) + (xad + xrl + xfd) * ifd0;
parameter Real psi1d0 = (-xad * id0) + (xad + xrl) * ifd0;
parameter Real psi1q0 = -xaq * iq0;
parameter Real psi2q0 = -xaq * iq0;
```

3-2-2 Loads

With RMS simulations in PowerFactory, three-phase loads are modelled as a combination of the static and the dynamic load as shown in Figure 3-10. The static part is modelled as a constant impedance and the dynamic portion of the load can be modelled as a linear load or as a non-linear load [29]. In this thesis both the static load and non linear dynamic loads are developed.

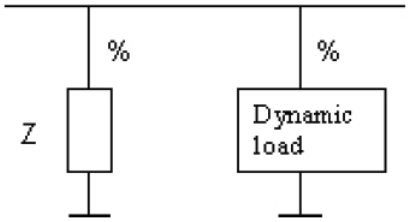


Figure 3-10: Representation of mixture of static and dynamic loads in PowerFactory*

The dynamic, voltage and frequency-dependent load model represented by the block diagrams shown in Figure 3-11 . It uses the three polynomial terms when modelling the voltage dependency of loads in PowerFactory .

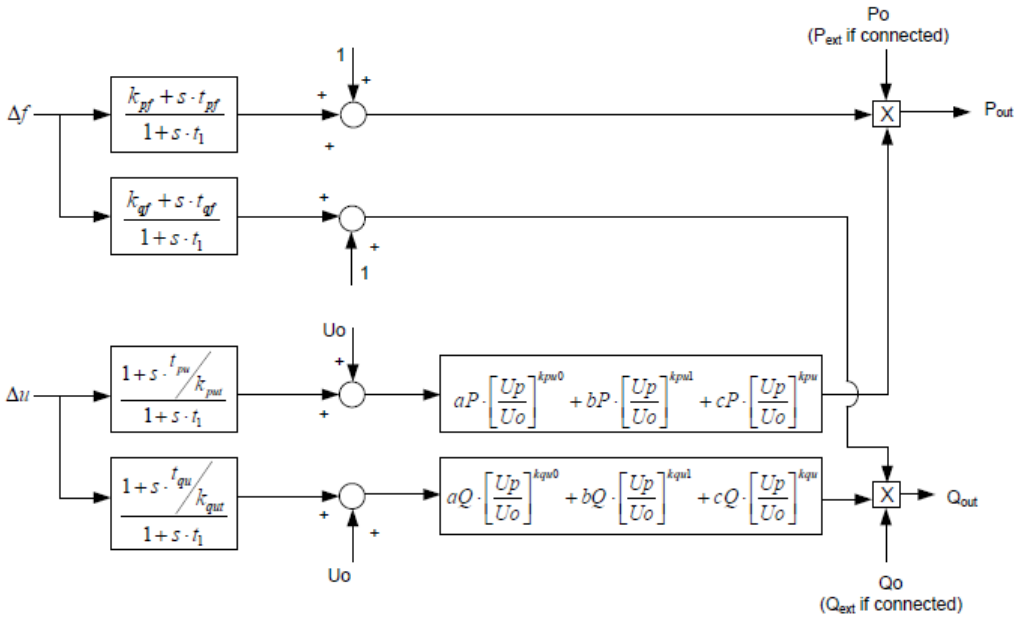


Figure 3-11: Model that approximates the behaviour of the non-linear dynamic load*

where,

$$\begin{aligned} k_{put} &= aP * k_{pu0} + bP * k_{pu1} + cP * k_{pu} \\ k_{qut} &= aQ * k_{qu0} + bQ * k_{qu1} + cQ * k_{qu} \end{aligned} \tag{3-14}$$

Figure. 3-12 shows the parameter dialog box, which is by default modelled for non-linear dynamic model. But, the model can work as a static model with right choice of parameter values and making certain values equal to zero. The Modelica code for the non-linear dynamic load is given in A-2.

*Figure taken from reference [29]

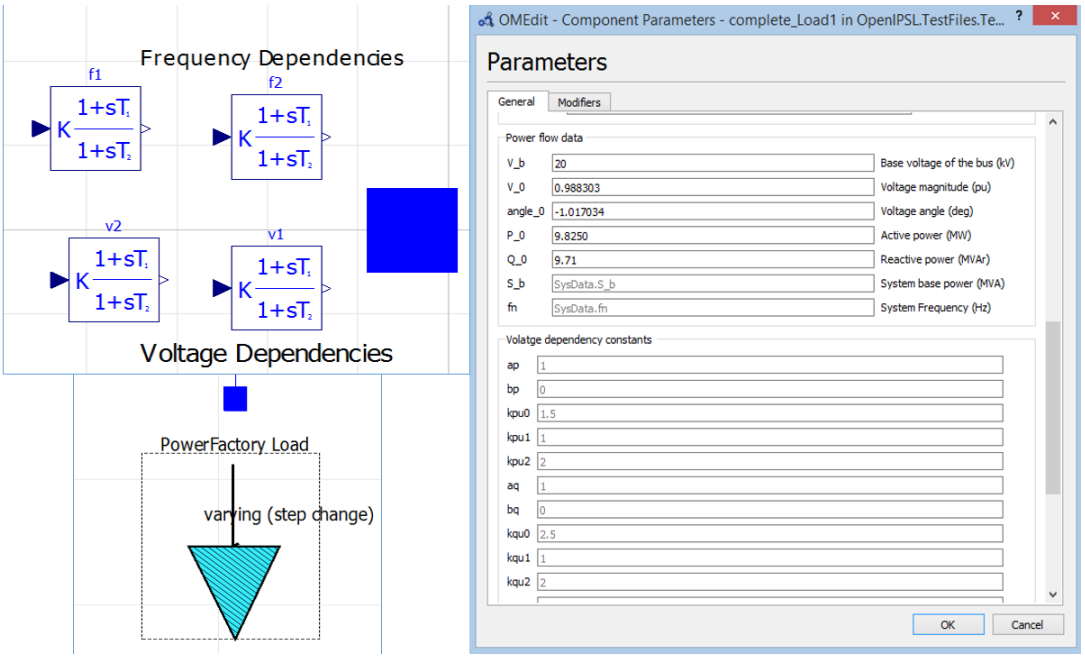


Figure 3-12: Non-linear dynamic load modelled based on PowerFactory in Modelica

3-2-3 Transformer

Two-winding transformers are used in the considered test system. The two-winding transformers in PowerFactory can be used to represent network transformers, phase-shifters, auto transformers or MV-voltage regulators [30]. The per-unit positive sequence equivalent circuit of the transformer is shown in Figure 3-13 . The leakage reactance and winding resistances are included on the HV and LV sides, and the magnetising branch accounts for core losses. These losses are represented by the magnetising reactance and a parallel resistance. Figure 3-14 shows the dialog box for parameter input.

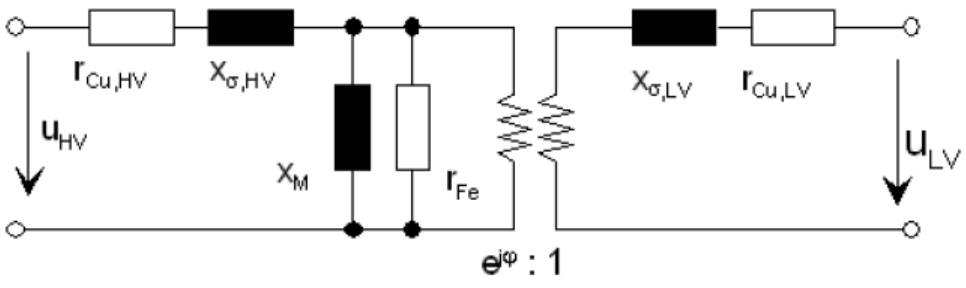


Figure 3-13: Positive sequence (per unit) equivalent circuit of a transformer*

*Figure taken from reference [30]

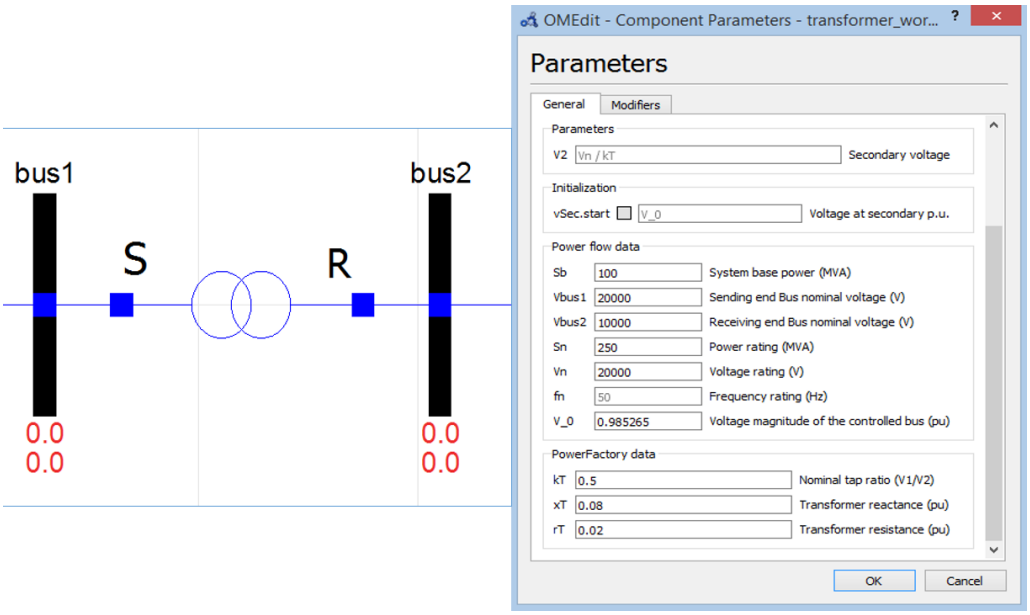


Figure 3-14: Transformer model in Modelica and the dialog box for parameter input

3-2-4 Transmission line

PowerFactory gives the option to choose between two transmission line models. The "Lumped parameters model (π -nominal)" or the "Distributed parameters model (π -equivalent)" [26]. The π -nominal model is a simplification of the π -equivalent model. Lumped parameter model is used in this thesis. The equivalent circuit for a three-phase Lumped parameters model is shown in Figure 3-15. The sum of all admittances connected to the corresponding phase is given by Y_s. Y_m gives the negative value of the admittances between two phases. The input parameters in the line type transmission line are defined in terms of positive and zero sequence impedances and admittances Z_1, Y_1, Z_0 and Y_0. The negative sequence values are assumed to be equal to the positive sequence.

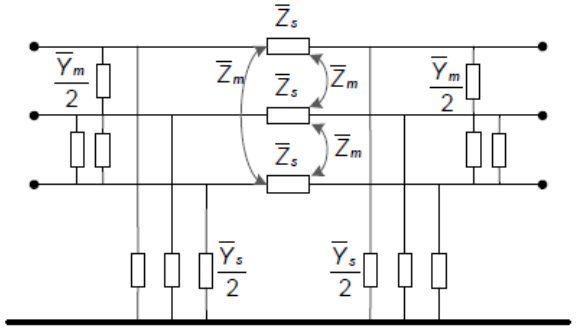


Figure 3-15: Equivalent circuit of the three phase line*

*Figure taken from reference [26]

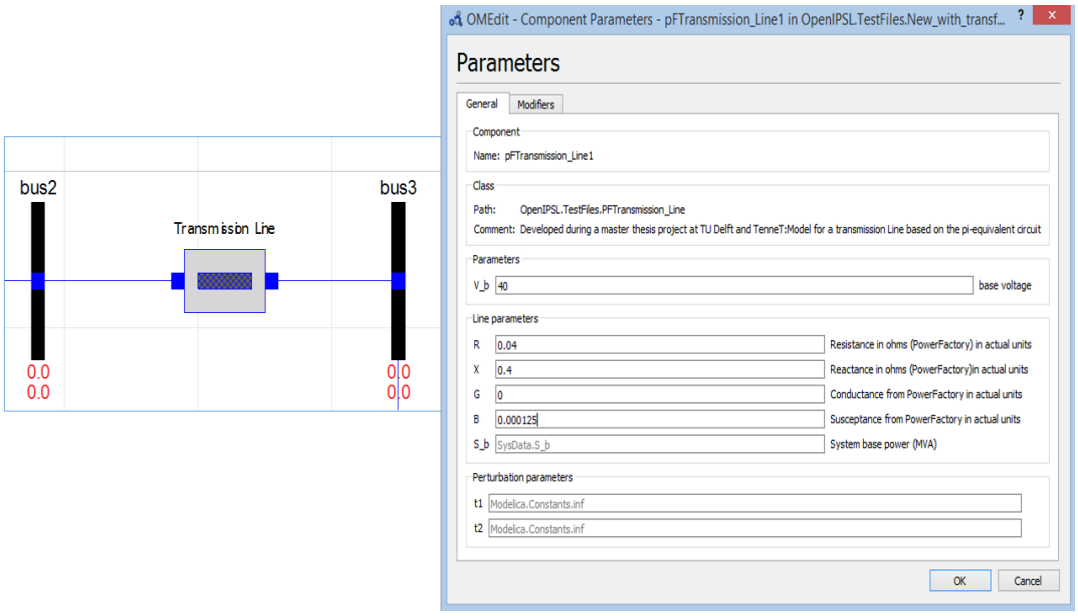


Figure 3-16: Transmission model in Modelica and dialog box for parameter input

3-2-5 Exciter system

Excitation systems should be capable of responding rapidly to a disturbance; should be designed to have fast acting response to enhance transient stability. In this thesis, a simplified ST1A model shown in Figure 3-17 is implemented. The input signal of the excitation system is the output of the voltage transducer, V_{TR}. This voltage is compared with the voltage regulator reference, V_{REF} . Thus, the difference between these two voltages gives the error signal that drives the excitation system. An additional signal from over-excitation limiter (OEL) output, V_{OEL} is also provided, which becomes non-zero only in the case of unusual conditions (disturbances).

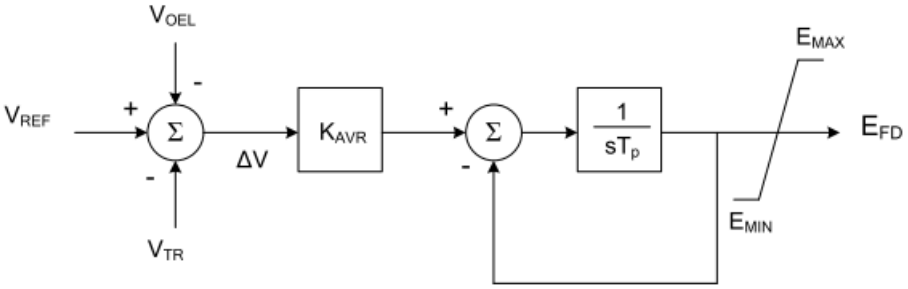


Figure 3-17: Simplified IEEE ST1A excitation model*

*Figure taken from reference [16]

3-2-6 Overexcitation Limiter

Certain slow acting phenomena, such as voltage collapse,might force machines to operate at high excitation levels for a sustained duration, this phenomenon can be captured using Over-Excitation Limiters (OEL). According to the IEEE recommended practice 421.5, OELs are required in excitation systems to capture slow changing dynamics associated with long-term phenomena. OEL protects the generator from overheating due to prolonged field over-currents. The overheating is seen either due to failure of a component inside the voltage regulator, or due to an abnormal system condition. In other words, OEL allows machines to operate for a defined time-overload period, and then reduces an excitation to a safe level. The OEL detects high field currents (IFD) and outputs a voltage signal (VOEL) which is sent to the excitation system summing junction as shown in Figure 3-18. This signal is equal to zero in normal operation condition.

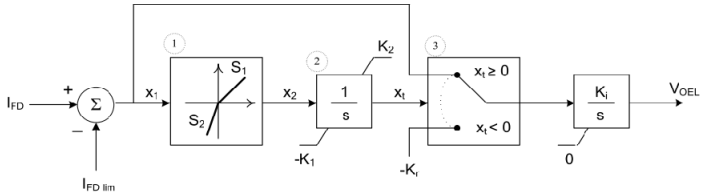


Figure 3-18: Over excitation limiter*

Both the exciter and OEL are modelled together in PowerFactory and a similar system is modelled in Modelica as shown in Figure 3-19. The inputs are terminal voltage (ut) and excitation current (ie). In addition to this, it contains inputs for initial parameters from the synchronous machine; initial excitation voltage (VE0) and initial terminal voltage (UT0) as shown in Listing 3.2.

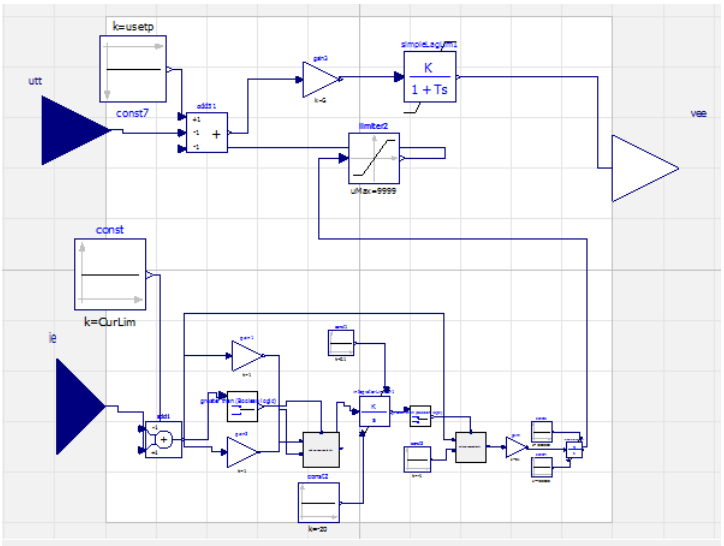


Figure 3-19: Exciter and limiter modelled in Modelica

*Figure taken from reference [16]

Listing 3.2: Exciter initialization equations

```
//Inputs from the synchronous machine
Modelica.Blocks.Interfaces.RealInput VEE "Initial Excitation voltage from
    machine";
Modelica.Blocks.Interfaces.RealInput UTT "Initial Terminal voltage from
    machine";
 //Parameter initializations
parameter Real VE0(fixed = false)"Initialization";
parameter Real UT0(fixed = false) "Initialization";
//
initial equation
VE0 = VEE;
UT0= UTT;
```

3-2-7 Speed-Governing System

A typical mechanical-hydraulic speed-governing system consists of a speed governor, a speed relay, hydraulic servomotors, and controlled valves, which are represented in the functional block diagram [16]:

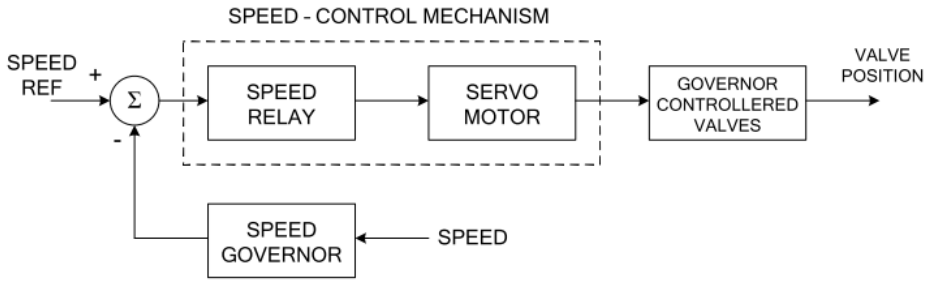


Figure 3-20: Typical speed-governing system*

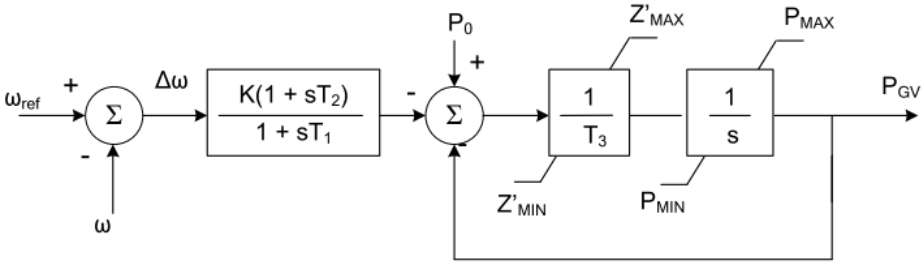


Figure 3-21: Speed-governing steam turbine system*

*Figure taken from reference [16]

3-2-8 Steam Turbine system

High pressure and high temperature steam is converted into rotating energy by the steam turbine, which in turn is converted into electrical energy by a generator. A steam system, tandem compound single reheat turbine, is used in this thesis. The turbine is represented by a simplified linear model as shown in Figure 3-22. The governor block consists inputs for initial parameters from the synchronous machine; initial turbine power (pt0) and initial power factor (cosn0) as shown in Listing 3.3.

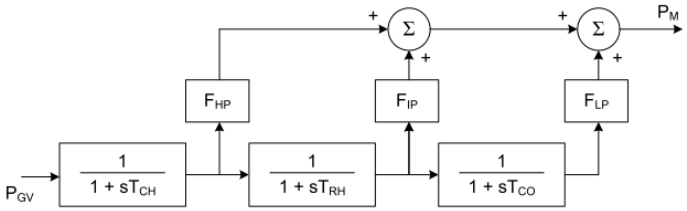


Figure 3-22: Linear model of turbine system

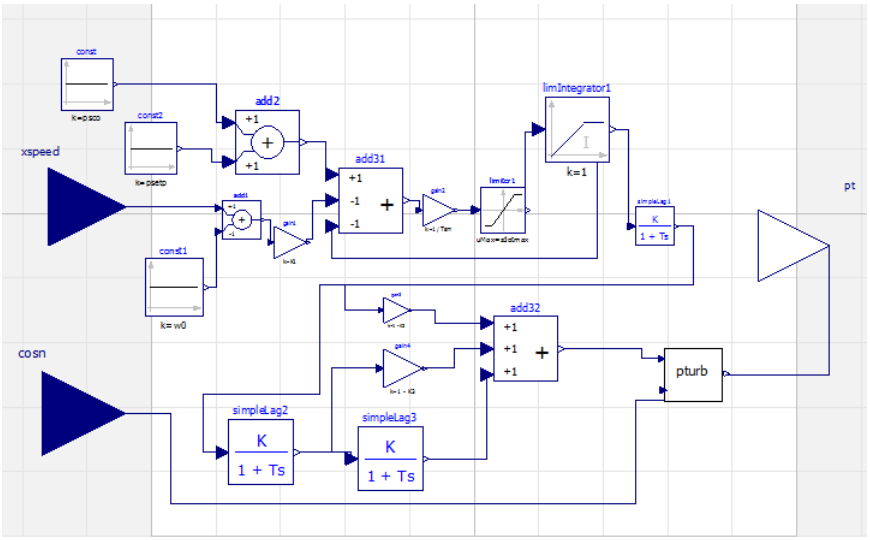


Figure 3-23: Governor and turbine model in Modelica based on PowerFactory

Listing 3.3: Governor initialization equations

```
//Inputs from the synchronous machine
Modelica.Blocks.Interfaces.RealInput xspeed "speed (pu) from the machine";
Modelica.Blocks.Interfaces.RealInput cosn "power factor from the machine";
Modelica.Blocks.Interfaces.RealInput PTT "turbine power from machine";
parameter Real cosn0(fixed=false);
parameter Real pt0(fixed=false);
//
initial equation
pt0=PTT;
cosn0=cosn;
```

3-2-9 OLTC

The tap changer is represented by an additional, ideal transformer connected to either the HV or LV side of the transformer as shown in Figure 3-24. For most applications, the winding ratio of this transformer is real and is defined by the actual tap position (in number of steps) multiplied by the additional voltage per step [30]. In this thesis, discrete tap changers are used where, only integer tap positions are considered. The tap changer is placed at HV side and the control node is at LV side, which means that the Tap controls the LV side. Asymmetrical tap changer model works as shown in Figure 3-25. where, nntap0 is the neutral position, tapmx and tapmn are the respective higher and lower tap positions.

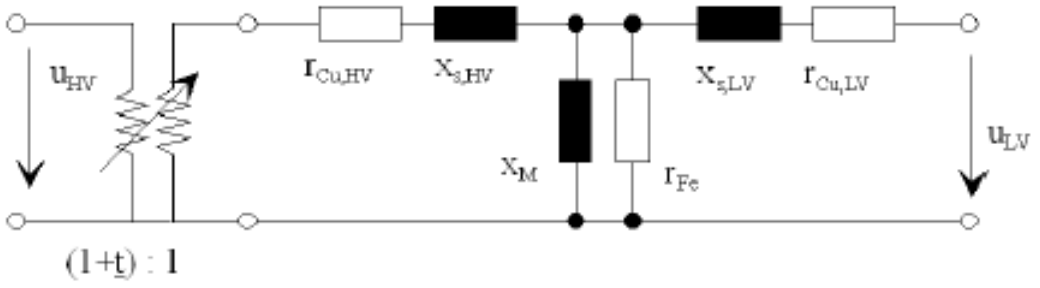


Figure 3-24: Tap changer is modelled at the HV side*

This kind of asymmtrical tap changer model uses a complex value du, which is expressed as:

$$du = du_{tap}(\cos(phitr) + j * \sin(phitr)) \tag{3-15}$$

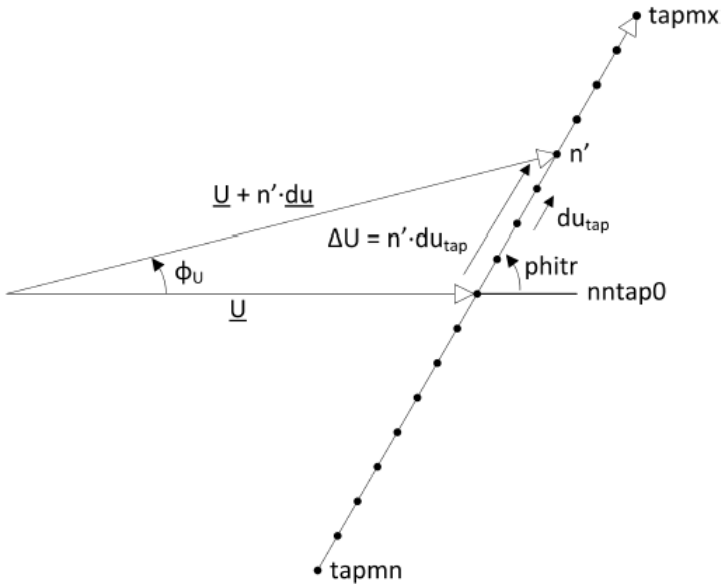


Figure 3-25: Equivalent circuit of the three-phase line*

*Figure taken from reference [30]

A separate block as shown in Figure 3-26 functions as a off-load tap changer and follows the equation 3-15.

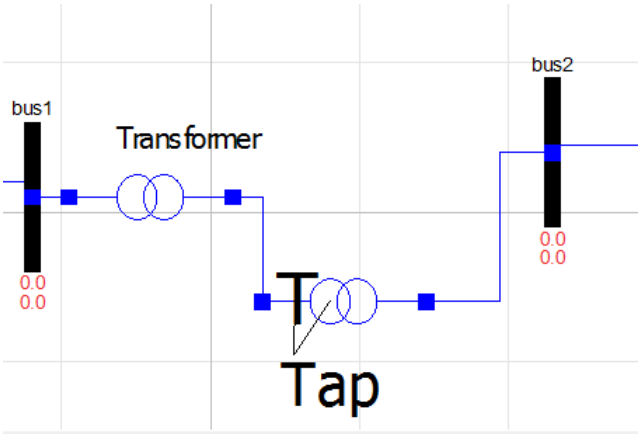


Figure 3-26: off-load tap changer block in Modelica

The 2-winding transformer model in PowerFactory is comprised of the 2-winding transformer element (ElmTr2), and the 2-winding transformer type (TypTr2). The transformer element allows input of data relating to the control of the transformer under steady-state conditions, and the transformer type allows input of the physical properties of the transformer [30]. In order to use the OLTC option, the user needs to input data in both the transformer element and type as shown in Appendix A-6.

Since, the exact logic for OLTC is not well documented in [30], an approximate PowerFactory based OLTC model is developed during this project.

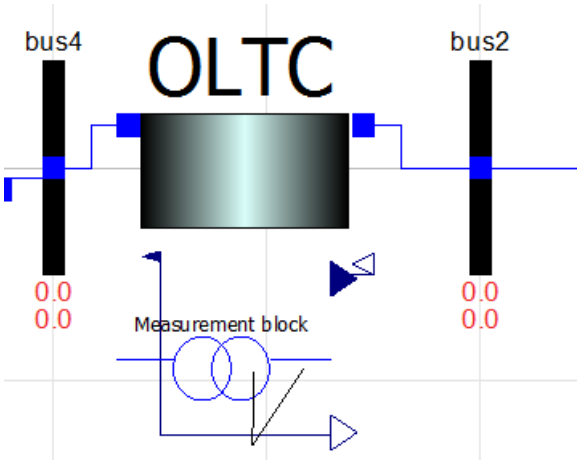


Figure 3-27: On-load tap changer block in Modelica

3-2-10 Asynchronous machine

In this thesis, single cage model is used. It is the simplest of the rotor impedance models. The model is characterised by a single R-L branch with a slip dependent rotor resistance.

This model is adequate for describing wound rotor motors, however it is unsuitable for motor starting studies with squirrel cage motors [31].

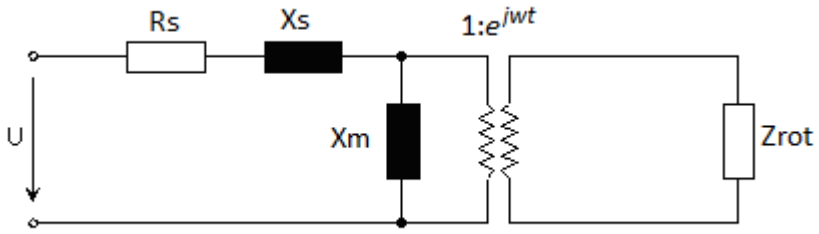


Figure 3-28: General equivalent circuit of asynchronous machine in PowerFactory

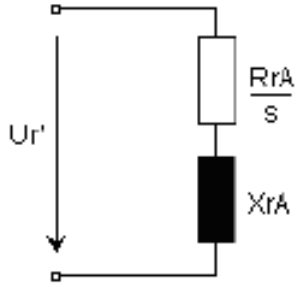


Figure 3-29: Simple cage rotor model

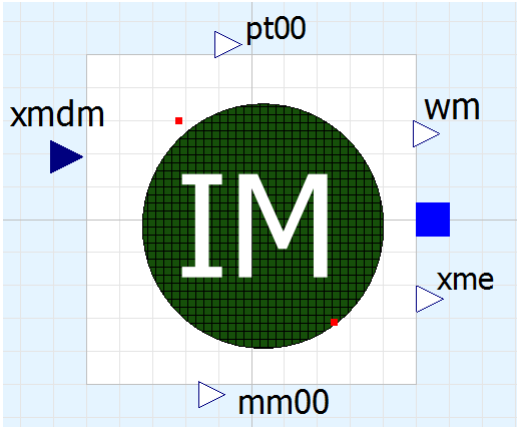


Figure 3-30: Modelica model

The voltage equations of an asynchronous machine model - single cage model:

$$u_s = r_s.i_s + j\frac{w_{ref}}{w_n} * x^{''}.i_s + \frac{x^{''}}{w_n}\frac{di_s}{dt} + j.\frac{w_{ref}}{w_n}.\psi^{''} + \frac{1}{w_n} * \frac{d\psi^{''}}{dt} \tag{3-16}$$

where the subtransient flux is defined as,

$$\psi'' = x_{sr}^{T}.x_{RR}^{-1}.\psi_R \tag{3-17}$$

where,
ψ_R is the rotor flux vector,
i_R is a rotor-current vector;
u_s, i_s and ψ_s are stator voltage, current and flux;
$w_n = \frac{dphi}{dt} = 2.\pi.f_{nom}$ is nominal angular frequency and
w_R is the rotor speed.

The single cage model constitutes of 3 state equations: One of the state equation is the speed equation and the remaining state equations are the rotor flux equations derived from 3-16.

$$\frac{dw_m}{dt} = \frac{m_e - m_m}{T_{ag}} \tag{3-18}$$

where,
T_{ag} is the total acceleration time constant in sec;
m_e is the electrical torque in p.u.;
m_m is the mechanical torque in p.u.;
w_m is the speed in p.u.

and the electric torque is given by,

$$m_e = \frac{i_s.r * \psi_s.i - i_s.i * \psi_s.r}{cos_n.eff/w_n} \tag{3-19}$$

where,
cos_n is the nominal power factor and
eff is the efficiency at nominal operation in p.u.

The following chapter discusses the use of model to model transformation approach to directly use the initial values (powerflow values) from the CGMES data and to generate the Modelica file that is 'dynamic simulation ready'.

Chapter 4

CIM based initialization for dynamic simulation

This chapter describes the model information exchange process with CGMES, which is closely related to the IEC- 61970 CIM standards. The structure of CGMES-CIM and its profiles are introduced. Finally, algorithm based on Python is described that obtains specific data from CGMES and helps automatically initialize the Modelica grid model. Also, the extension of this thesis work; Model to Model transformation (direct conversion of CGMES-CIM to a Modelica script file) is explained.

4-1 Standards for Model Information Exchange

The Common Grid Model Exchange Standard (CGMES) is developed by European Network of Transmission System Operators for Electricity (ENTSO-E) for interfacing between members' software in order to exchange power system modelling information as required by the ENTSO-E and TSO business processes [8]. The CGMES is a superset of the IEC Common Information Model (CIM) standard. It was developed to help transmission system operators (TSOs) with data exchanges in the areas of network planning, system operations and integrated electricity markets [9].

The major contribution of FP7 iTesla or the OpenCPS is the use of Modelica language for exchanging dynamic models. The latest draft version of CGMES v2.5 standard, proposes the use of Modelica for model exchanges and would require the proprietary tool owners to provide grid models to the users , both in CGMES-CIM and in Modelica [10]. The CGMES-CIM (v2.5) would be implemented by the European TSOs for operational and system development exchanges by the end of 2017 [32].

4-2 Common information model standards

A common information model is therefore required, that works with all the proprietary power system tools. Therefore the primary goal of the CIM standard is to improve model quality and to assure that information from different sources fit together in a structured and meaningful model. The CIM is based on Unified Modelling Language (UML) classes and relations to represent semantic information of the real power grid. Within CIM the power systems components are treated as objects that can be quickly converted to classes in an appropriate object-oriented programming language application. The CGMES-CIM works perfectly for the steady-state analysis. However, for dynamic simulations, it could be challenging to attain interoperability between proprietary tools as each proprietary tool follows its own modelling philosophy and simplifications.

4-3 CIM UML

The scope of this thesis limits the discussion of UML. However, the genral structure of UML is discussed with an example of a transformer. UML class (shown in Figure 4-2) diagrams provides visual representation of object hierarchies. Figure 4-1 shows the general CIM-UML layout of a power transformer.

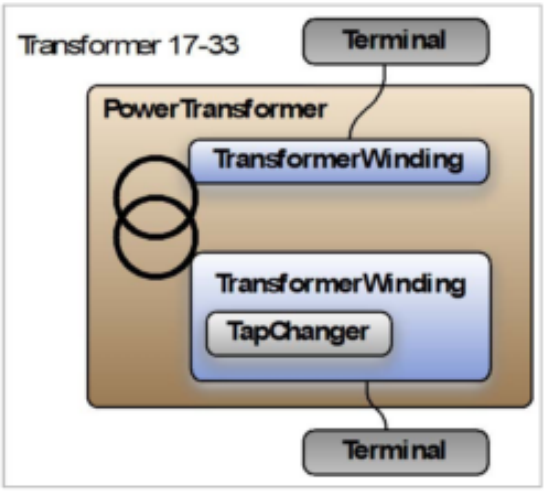


Figure 4-1: Power transformer represented in CIM-UML*

*Figure taken from reference [32]

Within CIM-UML, a two terminal power transformer becomes two Transformer Winding objects within a PowerTransformer container. If a tap changer is present to control one of the terminals then an instance of the TapChanger class is associated with the TransformerWinding object of that particular Terminal while still being contained within the PowerTransformer instance.

Thus, Figure 4-1 shows **4 CIM objects**: two TransformerWindings, one TapChanger and one PowerTransformer along with the Terminals associated with each TransformerWinding.

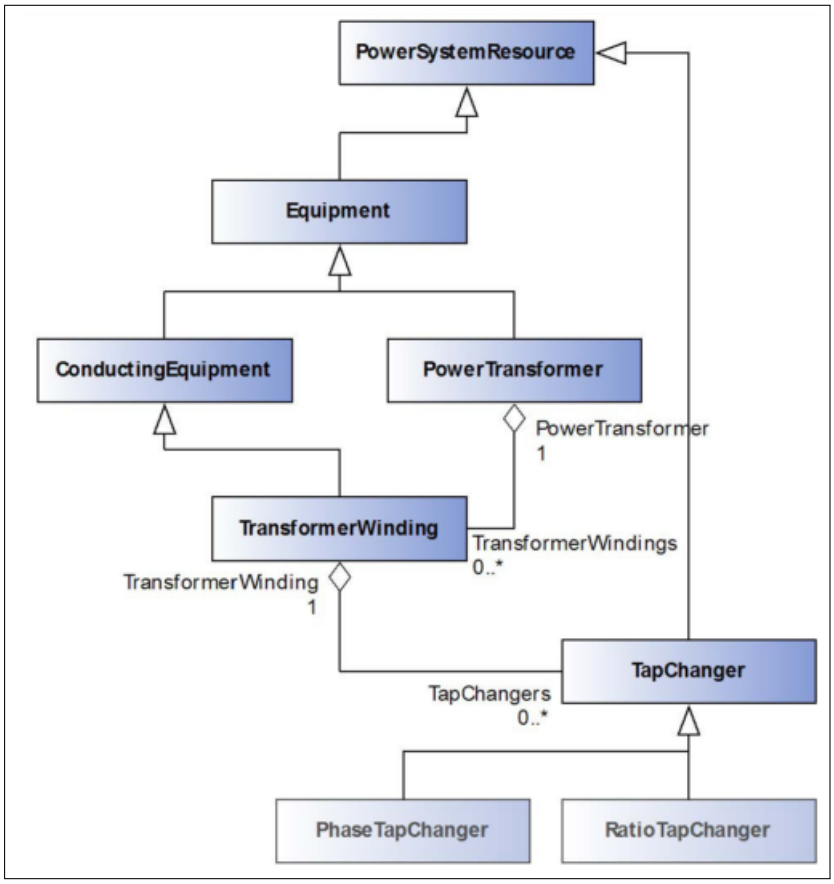


Figure 4-2: UML class diagram for the classes of a transformer*

*Figure taken from reference [32]

Classes are of different kinds and is explained using the class diagram of a transformer shown in Figure 4-2 [33]:

1. **Inheritance class**: Inheritance is often referred to as Generalisation. It defines a class as being a sub-class of another class. A sub-class inherits all the attributes of its parent-class, but can also contain its own attributes. In Figure. 4-1, Equipment inherits PowerSystemResource. Arrows are used to show the inheritance.

2. **Association class**: Here, the classes share different parent super-classes. In the Figure 4-2, TapChanger class is associated with the TransformerWinding object of that particular terminal while still being contained within the PowerTransformer instance.

3. **Aggregation class**: The Aggregation relationship defines a special kind of association between classes, indicating that one is a container class for the other. In the Figure

4-2, PowerTransformer is a container class of TransformerWinding. It is represented by a white diamond [33]. In PowerTransformer association with TransformerWinding, 1 PowerTransformer might have TransformerWinding from 0 to many (0..*).

4. **Composition class**: Composition is a specialised form of Aggregation where the "contained" object is a fundamental part of the "container" object, and that if the "container" is destroyed, all the objects that are related to it with a composition are similarly destroyed. It is represented by a shaded diamond shape.

Figure. 4-3 shows the load model according to the latest CGMES.

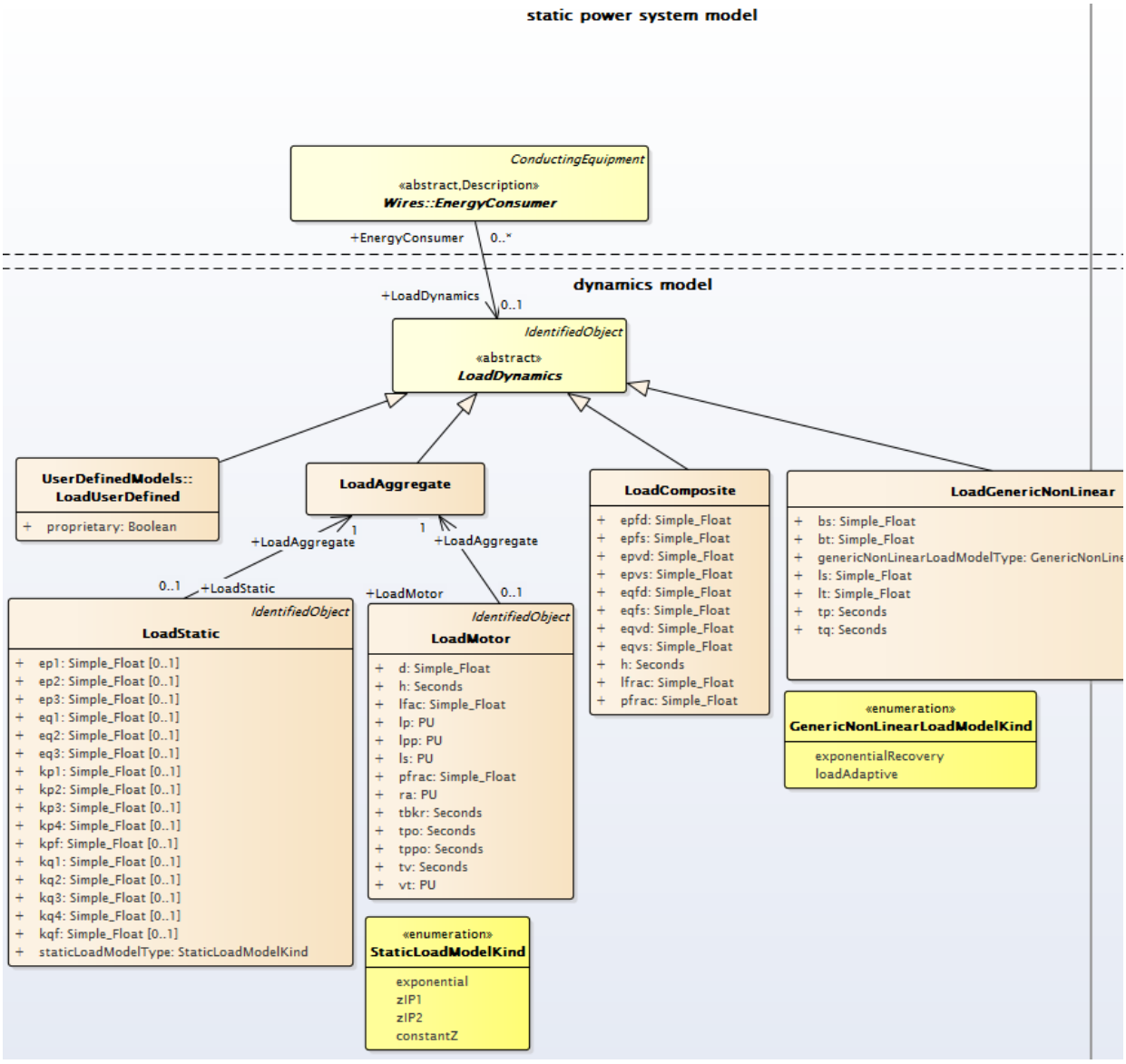


Figure 4-3: CIM-UML representation of Load based on CGMES v2.5

4-4 CGMES files

The Common Grid Model Exchange Standard (CGMES) is an ENTSO-E standard based on the IEC CIM, which stablishes a framework to assess the conformity of the suppliers' applications to the CGMES. The CGMES is used by European TSOs for operational and system development exchanges [32]. It contains several profiles that are interlinked as shown in Figure.

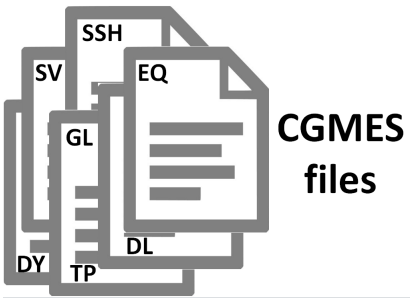


Figure 4-4: Different CIM profiles according to CGMES v2.5

Few of the profiles (they are the subset of classes, associations and attributes needed to accomplish a specific type of interface and based upon a canonical model) within CGMES are discussed below:

1. **Equipment profile**:The equipment profile is separated by three functional parts: EQ core, EQ operation and EQ short circuit. An equipment instance file describes the equipments in the grid model.
2. **Topology profile** :A topology instance file contains all topology objects for a grid model. These topology objects reference the corresponding equipment describing how equipment is electrically connected.
3. **Steady state hypothesis profile** :A steady state hypothesis instance file contains all objects required to exchange input parameters to be able to perform load flow simulations.
4. **State variables profile** : A state variable instance file contains all objects required to complete the specification of a steady-state solution.
5. **Dynamics profile** : A dynamics instance file represents the parameters necessary to model dynamic behaviour of the power system, e.g. transient and subtransient reactances of synchronous machines, parameters of the control block diagrams of excitation systems, turbine, governors, power system stabilisers, etc.
6. **Diagram layout profile** : Diagram layout profile standard and contains data necessary for the model diagram.
7. **Geographical location profile** : A geographical data instance file contains GIS data and is constructed based on IEC 61968-4, although it is limited to the classes which cover ENTSO-E needs.

The CGMES-CIM can also be used in the eXtensible Markup Language (XML), which is a standard format that stores machine-readable data in a structured, extensible format. This thesis uses XML format to read the powerflow values from the CGMES-CIM files. All the CGMES-CIM profiles are associated with each other through Resource Description Framework identification (RDFid).

RDF identifications help create relationships between XML nodes. For example, the initial voltage and angle of a bus is obtained from SV profile, the corresponding name of the bus is obtained from TP profile. The class object in SV profile and that of TP profile are associated by a common RDFid. The SV profile for a simple two bus system with a generator and a load is described in Figure 4-5.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with PowerFactory 16.0.4 (digcimdb.dll ServicePack 0) -->
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2013/CIM-schema-cim16#" xmlns:entsoe="http://ents
  <md:FullModel rdf:about="urn:uuid:_54bc4bc6-95b3-49a1-be31-f114cdc418c1">
    <md:Model.DependentOn rdf:resource="urn:uuid:_732970a1-9c8f-4ae7-9bcf-ae7f83d6b7fd" ,
    <md:Model.DependentOn rdf:resource="urn:uuid:_561362d2-828c-4af4-823d-dd72cfb0004a" ,
    <md:Model.created>2017-05-22T10:06:41</md:Model.created>
    <md:Model.modelingAuthoritySet>tennet</md:Model.modelingAuthoritySet>
    <md:Model.profile>http://entsoe.eu/CIM/StateVariables/4/1</md:Model.profile>
    <md:Model.scenarioTime>2011-03-07T14:18:25</md:Model.scenarioTime>
  </md:FullModel>
  <cim:SvVoltage rdf:ID="_7a7a3374-4bd9-4186-b3bc-dc8d8a137b82">
    <cim:SvVoltage.TopologicalNode rdf:resource="#_d6e8d2ad-bab6-4f76-bb23-c4256fbb0ae4"
    <cim:SvVoltage.angle>-0.52162</cim:SvVoltage.angle>
    <cim:SvVoltage.v>19.7772</cim:SvVoltage.v>
  </cim:SvVoltage>
  <cim:SvVoltage rdf:ID="_c1dd63b3-44cc-4bfd-bdc1-7125c59df764">
    <cim:SvVoltage.TopologicalNode rdf:resource="#_1af989dd-6ab0-4a63-aa74-f5115ec0e29e"
    <cim:SvVoltage.angle>0.</cim:SvVoltage.angle>
    <cim:SvVoltage.v>20.</cim:SvVoltage.v>
  </cim:SvVoltage>
  <cim:SvPowerFlow rdf:ID="_7274eb37-852c-4fb2-a27d-4db0228c3145">
    <cim:SvPowerFlow.Terminal rdf:resource="#_a1487b26-7f8b-4bba-a23b-b05f139de76e" />
    <cim:SvPowerFlow.p>10.</cim:SvPowerFlow.p>
    <cim:SvPowerFlow.q>10.</cim:SvPowerFlow.q>
  </cim:SvPowerFlow>
  <cim:SvPowerFlow rdf:ID="_4411db51-4956-4b34-baae-3d15fe04e107">
    <cim:SvPowerFlow.Terminal rdf:resource="#_1d9cdf39-61f9-4ee6-8d8d-5c32f36ec3e3" />
    <cim:SvPowerFlow.p>-10.0204</cim:SvPowerFlow.p>
    <cim:SvPowerFlow.q>-10.1545</cim:SvPowerFlow.q>
  </cim:SvPowerFlow>
```

Figure 4-5: The powerflow values inside the SV profile

To better interpret the xml classes and attributes, online tool known as Xmlgrid was utilized, which gives the graphical structure of attributes within CIM classes as shown in Figure. 4-6 and 4-7. This kind of representation helps to understand the attributes and association that CIM classes have between them. For example, CIM class SvVoltage has attributes *angle* and *v*. Also, this class is associated with the TP profile - CIM class, TopologicalNode, from which the name of the respective bus in the grid model is obtained.

This process is shown using a flowchart in Figure 4-8. Similar logic can be developed to harvest the other load flow values (active power (p) and reactive power (q)) from the CGMES-CIM profiles.

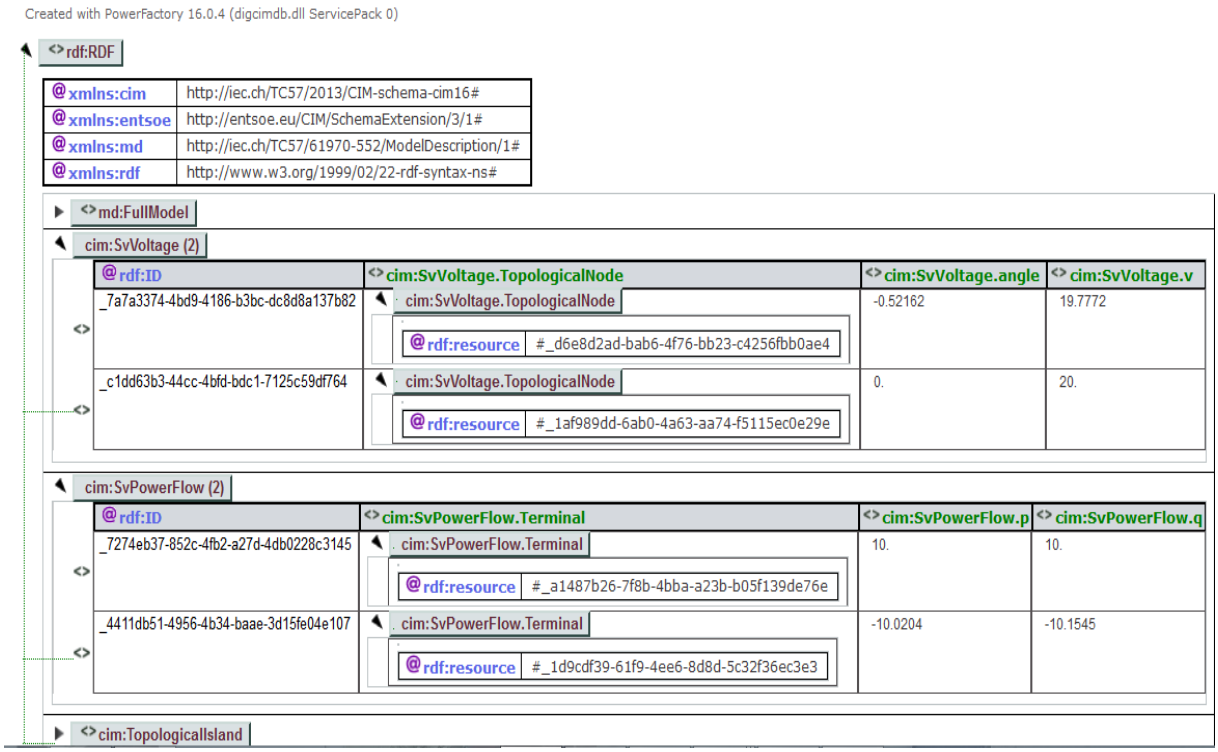


Figure 4-6: Graph structure of the attributes of the CIM classes, SvPowerFlow and SvVoltage inside SV profile

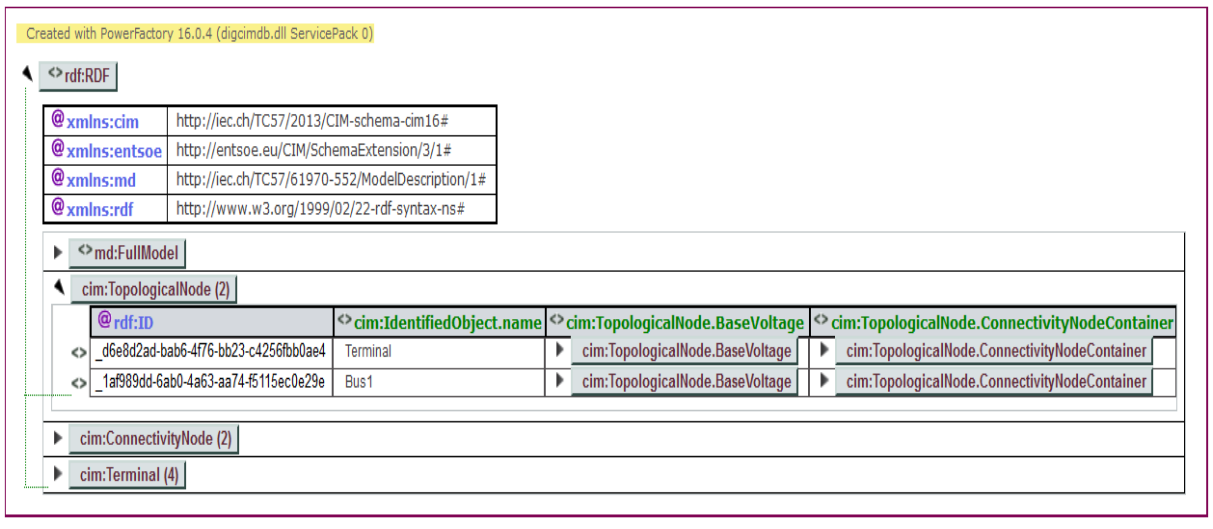


Figure 4-7: Graph structure of the attributes of the CIM class, TopologicalNode inside TP profile

The flowchart to obtain all the bus values and the names of the elements in the grid is shown in Figure 4-8. If there is a bus named Bus1, then all its values are obtained from SV profile and using its respective RDFid, its name is obtained from TP profile to obtain a list: (BUS_NAME, Voltage, Angle) = [Bus1, 1.04, 0.5213]

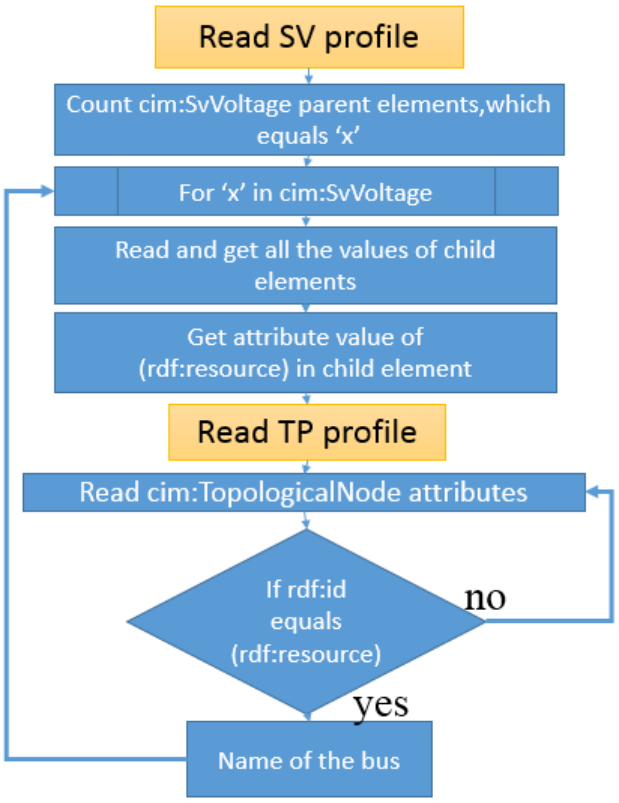


Figure 4-8: Flowchart to obtain the name of the bus and its respective bus voltages and angles

4-5 CGMES-CIM based initialization of Modelica grid models

All the models in OpenIPSL are programmed in such way that by introducing a power flow solution from another tool [22]. Figure 4-9 shows the dialog box of a generator in the grid, where the powerflow values (Voltage at the generator bus (V_0), angle ($angle\_0$), active power (P_0), reactive power (Q_0)) is obtained from other tool and manually input into Modelica models. This process could be very stressful for a large grid model and there is also a high chance for human-errors while entering the powerflow values.

Using the powerflow values, the remaining initial guess values are computed as a parameter within each model (ex: initial excitation voltage $ve0$ is computed within the synchronous machine model). These initial values are subsequently provided into the initial equations that are used to solve the overall initialization problem

Figure 4-9: Dialog box for powerflow input in a generator model

In order to overcome this tedious nature of working with Modelica models, a CGMES-CIM reader is developed that harvests the powerflow data and dumps it into respective component model (Generator or Load) in Modelica based grid model. The CGMES reader is interfaced using Python. Therefore, this thesis proposes a automatic process to harvest data directly from CGMES-CIM and initialize the Modelica based grid model.

4-5-1 Method 1: Initialization of Modelica based grid models

Here, the powerflow values are directly harvested from the CGMES files. The existing Modelica script for a specific grid model is also used for manipulation. The names of the equipment (eg: Gen_Delft or Load_Arnhem) are looked up and their respective powerflow values are updated accordingly. The algorithm developed tries to locate the names and the respective equipments through the SV profile. **NOTE: Different algorithms can be developed with this idea.**

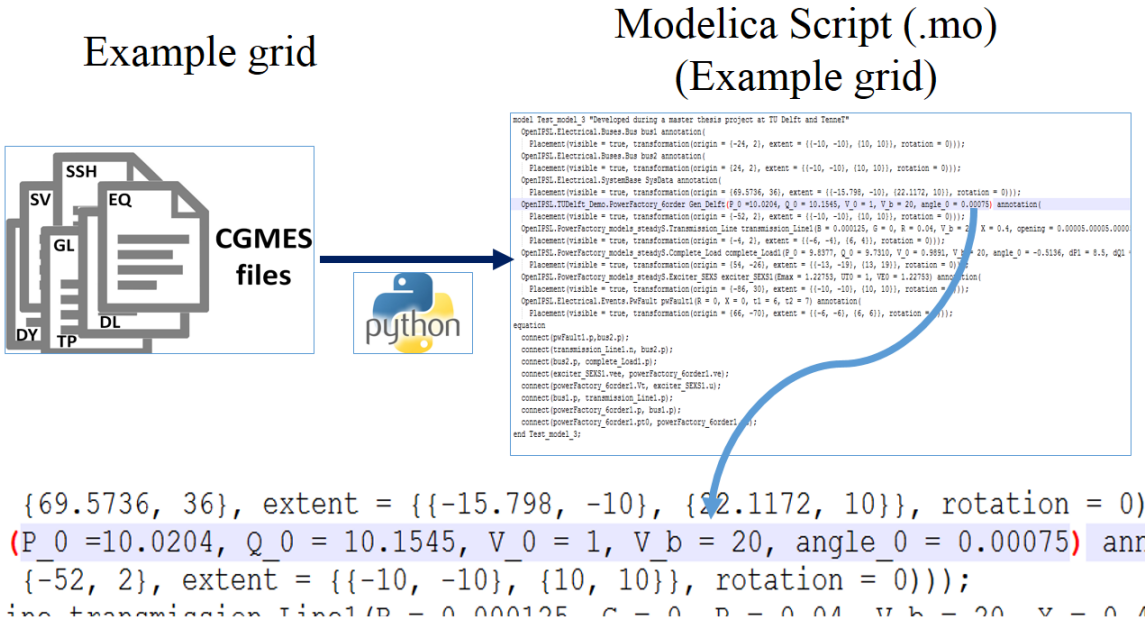


Figure 4-10: Initialization process in Modelica using CGMES-CIM files

The process is carried out through following steps:

1. **Access CGMES files**: Find all the buses, generators and loads
2. After deducing the number of elements, find their corresponding names.
3. **Access the corresponding Modelica script file**: Look up for the name of the element (Generators and Loads), replace the dummy powerflow values in the models by the actual powerflow values from the CGMES-CIM.

Listing 4.1: Python code snippet for Method 1

```
import fileinput
import sys

#Function to replace the specific line
def replaceAll(file,searchExp,replaceExp):
for line in fileinput.input(file, inplace=1):
if searchExp in line:
line = line.replace(searchExp,replaceExp)
sys.stdout.write(line)

# This is an example code snippet for Method 1
text = "GEN_delft"   # if any line contains this name, I want to modify the
    whole line (Only in the parameters section).
#In actual code, text will be from a CGMES reader.
x = fileinput.input(files="new.mo")
for line in x:
print line
if text in line:
Sentence=line
x.close()
#Copy the entire line into a new file
x1 = open(r'newfile.mo', 'w')
x1.write(line + '\n')
x1.close()
break

#Obtain the powerflow values for GEN_delft, text1 is just an example here,it
     will be replaced by a list of lists in the actual script
text1='P_0 = 9.8572, Q_0 = 9.8767, V_0 = 1, _b = 20, angle_0 = 0.00075)
    annotation('
#Read from the temporary file
datafile = open('newfile.mo', 'r')
#Create my final file
smallerdataset = open('FinalScript.mo', 'w')
for counter, line in enumerate(datafile):
print counter
smallerdataset.write(line.split('(', 2 )[0]+'('+text1)
New_Sentence = line.split('(', 2 )[0]+'('+text1
break
smallerdataset.close()
replaceAll('new.mo',Sentence,New_Sentence)

#This is an example script
```

Code description: The Python code developed in this thesis are based on standard packages and are developed from scratch. Although, there are existing Python packages that enable direct interaction of Python with OpenModelica, they were not utilized in this thesis, keeping in mind the IT-security constraints within an utility.

In Method 1, specific code-line manipulation is carried out. It uses 3 .mo files are used for manipulation, original Modelica script file (new.mo), where dummy powerflow values exist, temporary file (newfile.mo) and a final script (FinalScript.mo) that contains the powerflow values from CGMES-CIM as shown in the Listing 4.1.

Advantages of Method 1:

1. Easy to initialize the Modelica grid models, where manual input of values is avoided. (Everytime, the generator dispatch values are changed or the load consumption is changed, it becomes easier to load the powerflow values using this method.)
2. Mapping is based on "names" of the elements (eg: *Gen*_001), which increases readability of the code.

Rules for using Method 1:

1. It requires 2 similar grid models in both the software environments (even the names of the elements should be same), which means a Modelica based grid model has to be created already.
   If a transmission line is removed in the another tool, the same line should be removed in Modelica model as well. This method works only if the physical grid models in both the tools are the same.
2. The Python script handles both the .mo file and the CGMES files at once (Could be time consuming for large system and may consume more system memory).
3. Only converged power flow values are to be used. In this thesis, the CGMES-CIM files are obtained through PowerFactory and for all the methods provided in this thesis, the powerflow is assumed to be converged.

With this method, the objective of this thesis project is accomplished. However, this thesis goes a step ahead to achieve Model to model transformation, which provides complete automation and overcomes the short-comings of Method 1.

4-5-2 Method 2: Model to model transformation

The Model-to-Model transformation (M2M) [32] is a process of converting a target model from a source model. There have been several attempts at converting CIM files to Modelica script [18]. The work in [18] also describes Model to Model transformation for a IEEE 9-bus system. However, the output Modelica model is in the form of equations (only text) as it doesn't take into account the Diagram Profile (DL profile). Similarly, there exists a Power Systems on Modelica (PSM) tool [34] that automatically generates and simulates power systems in

Modelica. However, it is implemented for ENTSO-E CIM Profile 1 (the older version of CGMES-CIM), which again outputs the Modelica model in the form of equations.

In this thesis, CGMES-CIM files are used for the conversion. In addition, the diagram profile (DL profile) is also utilized to obtain the graphical representation, which was missing in the previous other tools. Figure 4-11, depicts the process of obtaining Modelica script directly from CGMES-CIM.

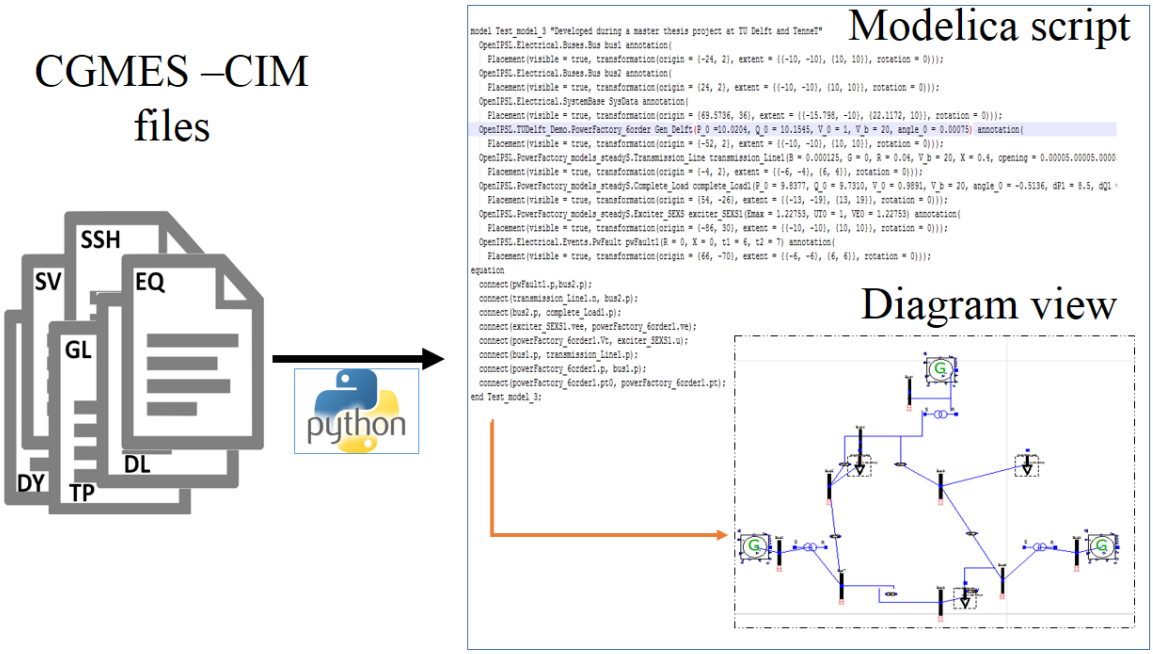


Figure 4-11: Model to Model transformation; CGMES-CIM to Modelica script conversion

In order to test the M2M method, a 9-bus system is considered, which uses all the PowerFactory models developed in this thesis. The proposed CGMES-CIM reader can work with any grid consisting of synchronous generators, loads, transmission lines, transformers and buses. The results of this method and the validation results of the PowerFactory based Modelica models are described in the following chapter.

Chapter 5

Result discussion

This chapter deals with the results obtained from model validation in Modelica with PowerFactory as a reference tool. The test system considered is described first and the components developed specifically for the test system is described one by one. Also, the results obtained from the CGMES to Modelica converter is described in the end.

5-1 Considered test system

As described in the previous chapter, this thesis develops PowerFactory Based Modelica models specifically for the considered test system as shown in Figure 5-1. The 'All-in-one' test system consists of a local area connected to a strong grid (Thevenin Equivalent) by two 380 kV transmission lines. A motor load (rated 750 MVA, 15 kV) is connected at Bus 4 and supplied via a 380/15 ratio transformer. A local generator (rated 450 MVa, 20 kV) is connected at Bus 2 to supply the loads through a 20/380 ratio transformer. Also, a Load Tap Changer (LTC) is present at the Load.

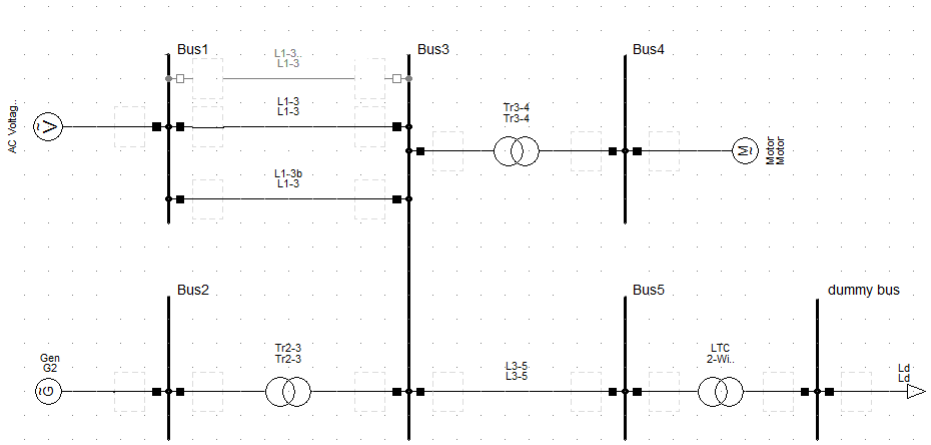


Figure 5-1: Test system modelled in PowerFactory

Therefore, individual grid component models starting from the Synchronous machine is first Modelled in Modelica and then validated against PowerFactory.

5-2 Model Validation

For the validation process, a simple 3 bus system with a load is considered. Since, the models had to be testes from scratch, few existing components from the OpeniPSL like the PSAT static load and PSAT based transmission lines (Lumped parameter type) were initially used. The grid model followed for testing is as shown in Figure 5-2.

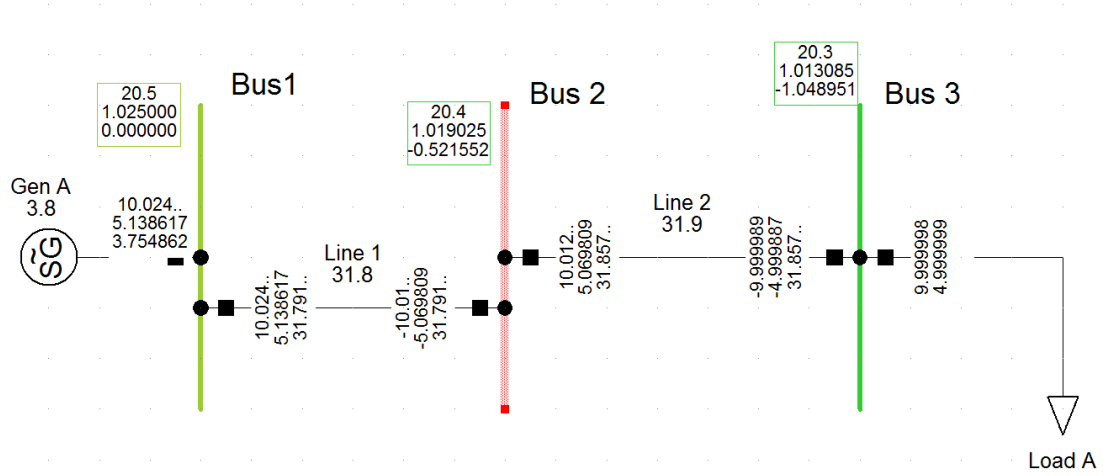


Figure 5-2: Grid model in PowerFactory

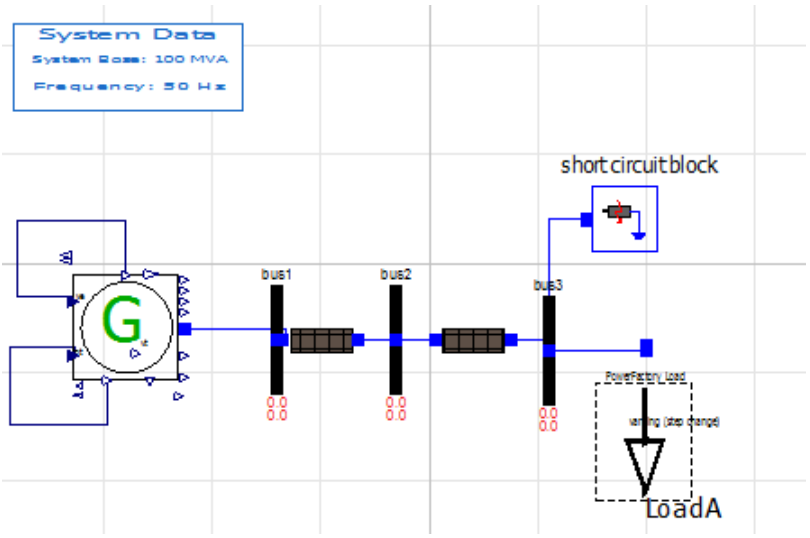


Figure 5-3: Grid model in Modelica

Validation tests are carried out using this simple system. Thus a single generator model was

validated first and then each control system was validated using the validated generator. The perturbations introduced were events such as three phase to ground faults, load variations (both increase and decrease in load) to analyse the transient behaviour. All of the perturbations were modelled in the same way in Modelica as they are in PowerFactory.

During the validation process, the results are obtained both qualitatively and quantitatively. The graphical observation only provides an insight of the validity of a model. A quantitative assessment on the other hand, allows to measure the validity of a model response against its reference. It gives a numerical value for better assessment.Root Mean Square Error (RMSE) is used for quantitative assessment.

RMSE is given by equation 5-1. Where, n is the total number of discrete measurement points.
x_1, x_2,. . . , x_n are the discrete measurement points at time t_1, t_2,. . . ,t_n for Modelica and
y_1, y_2,. . . ,y_n are the discrete measurement points at time t_1, t_2,. . . ,t_n for PowerFactory.

$$RMSE = \sqrt{\frac{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + ... + (x_n - y_n)^2}{n^2}} \tag{5-1}$$

All the simulation graphs contain the RMSE on the bottom right corner.

5-2-1 Simulation set-up

Power flow computations were performed in PowerFactory and the same power flow solution was used in OpenModelica to initialize the Modelica grid model. The simulation set up is given in Table 5-1. However, the actual solvers used in PowerFactory were not known and a solver that gives the closest response was chosen in OpenModelica to carry out the simulations.

Table 5-1: Simulation set-up in OpenModelica

	Value
Start time	0
Stop time	10
Interval step	0.001
Integration method	Radau5
Tolerance	1e-6
Non-linear solver	Hybrid

5-2-2 Synchronous Generator

The equations and parameters used for the generator are given in the Appendix A-1. Few results are shown here and the remaining can be found in the Appendix B-2. The perturbations are:
Load increase by 100% between 2-3sec
3phase short circuit at bus 2 between 4-5sec
Load decrease by 200% between 7-9sec.

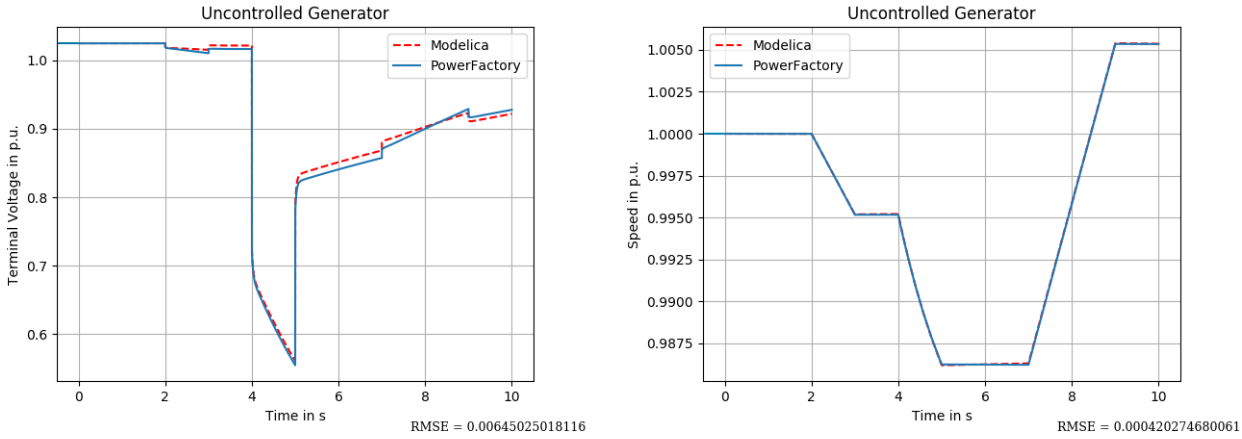


Figure 5-4: Terminal voltage and speed variation in an Uncontrolled generator

5-2-3 Excitation system

Simplified IEEE ST1A excitation model is used in this thesis. The parameter values and other results are discussed in the Appendix. Figure 5-5 shows that the voltage is brought back to the nominal value with the exciter. Figure 3-18 shows the graphs from Modelica and PowerFactory.

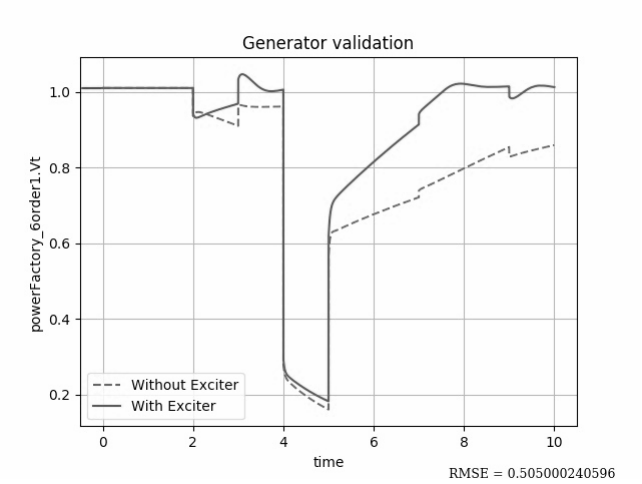


Figure 5-5: Comparison of terminal voltage with and without excitation system

A test system was created in PowerFactory as shown in Figure 5-6 to test for different perturbations:

1. Active and Reactive load increased by 100% at Bus2, starting from 2s to 3s.
2. Line 1 is opened at 4s and closed at 5sec.
3. Active and Reactive load decreased by 200% at Bus4, starting from 6s to 7s.

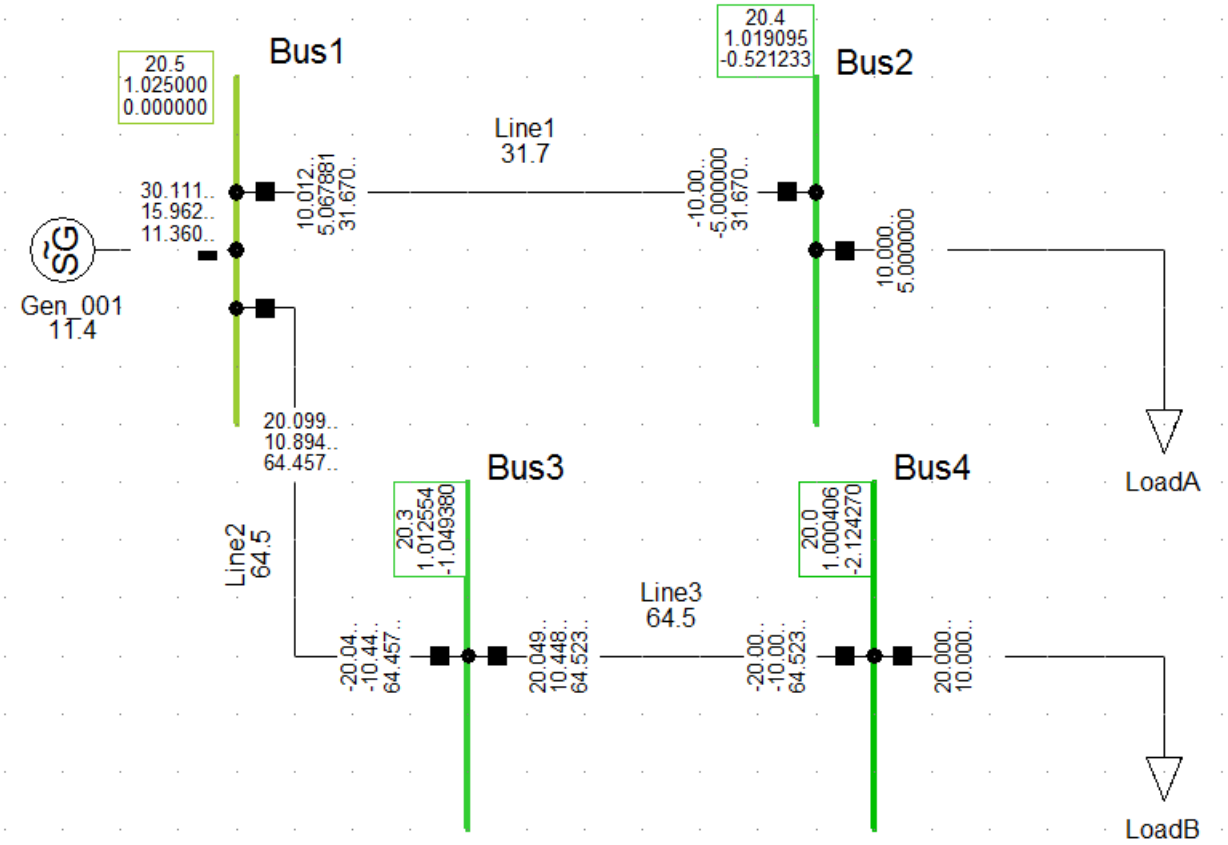


Figure 5-6: Test system used to validate excitation system

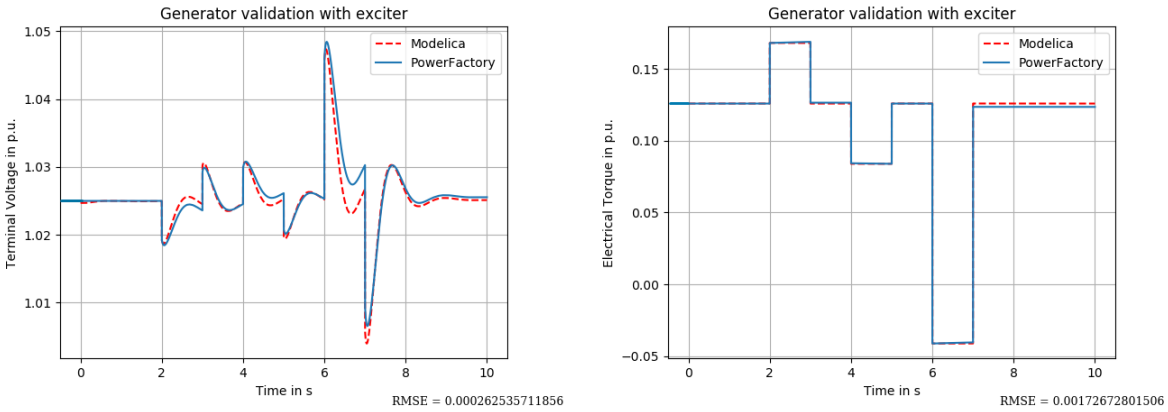


Figure 5-7: Terminal voltage and electrical torque obtained from exciter validation

The parameter values of the exciter as same as shown in Appendix A-5. The line parameter values are shown in Table B-2. The generator parameter values are shown in Table B-1. The remaining results of exciter validation are shown in Appendix B-3.

5-2-4 Governor system

Governor system was also modelled for the test system. Similar grid system as that of exciter was taken for analysis (see Figure. 5-8). Figure. 5-9 shows the effect of the governor system. (Note: The governor in this example is not tuned as it is beyond the scope of this thesis.)

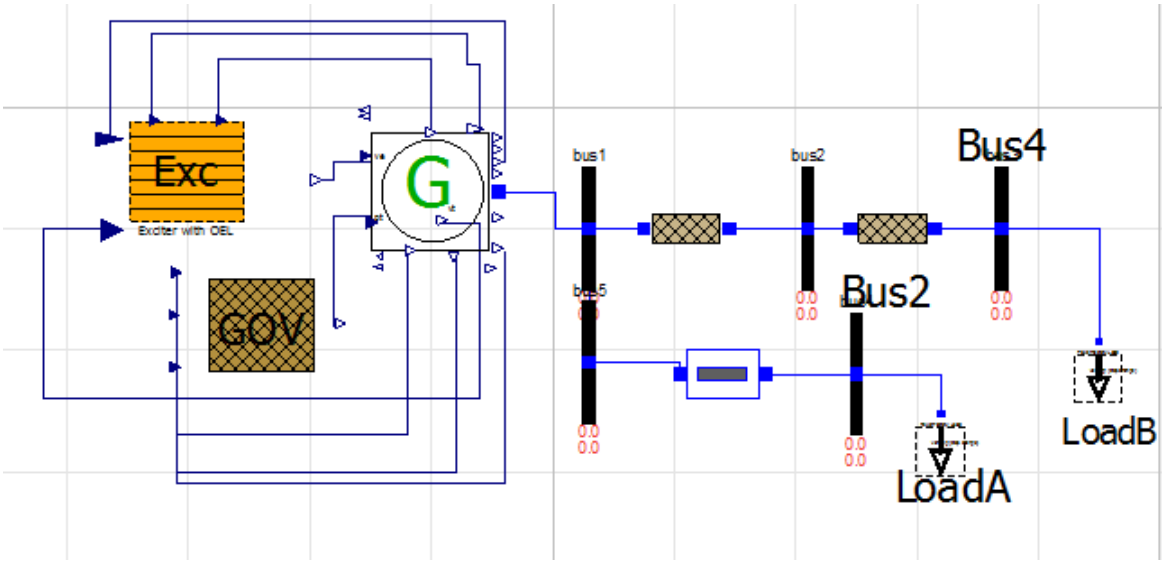


Figure 5-8: Test system for governor validation

The impact of the governor system described in the Chapter 3 is initially checked. Figure.5-10 shows the responses on speed and electrical torque after a load variation of 100% between 5-7sec. The values of the governor used are tabulated in the Appendix A-5.

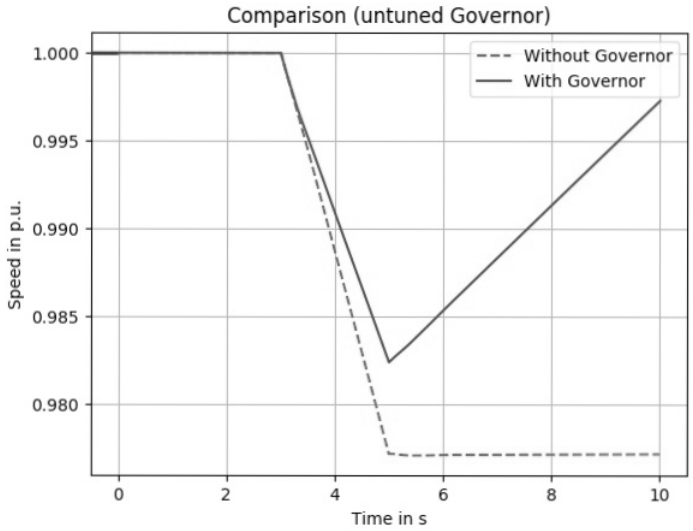
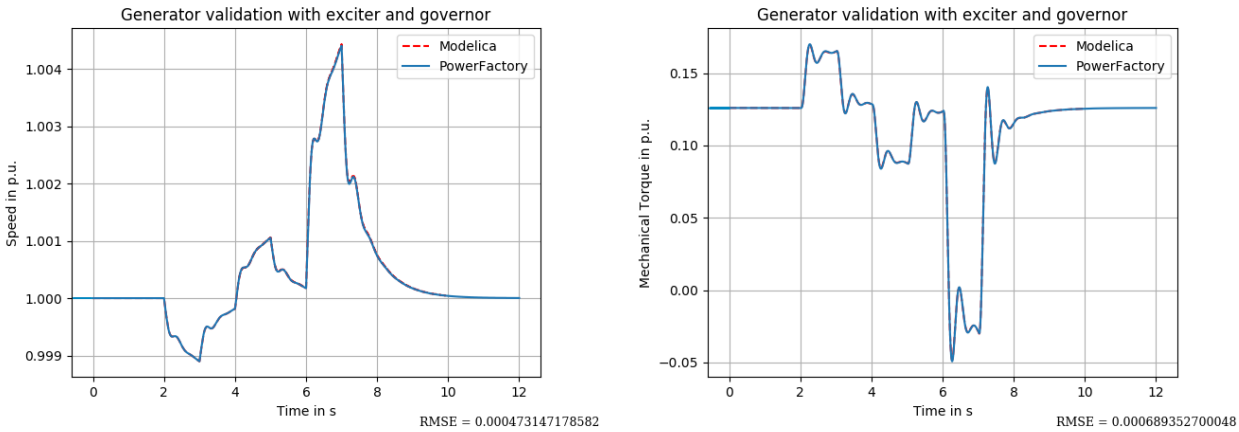


Figure 5-9: Comparison of speed responses with and without the Governor system

Figure.5-10 shows the responses on speed and voltage at Load B after the application of the described perturbations. The remaining results are shown in Appendix A-4.



(a) Speed (b) Mechanical torque

Figure 5-10: Comparison of Modelica and PowerFactory responses with the governor system

5-2-5 Load

100% non-linear load is used in this thesis. A non-linear PowerFactory based load model is modelled in Modelica. However, for the initial model validation in this thesis, a static load was used from the PSAT sub-library. Figure. 5-11 shows the difference in response of the PowerFactory based dynamic load model with the PSAT load model. The grid model shown in Figure 5-2 is used and same perturbations are applied. Care must be taken to choose an appropriate load model to avoid error in the simulations.

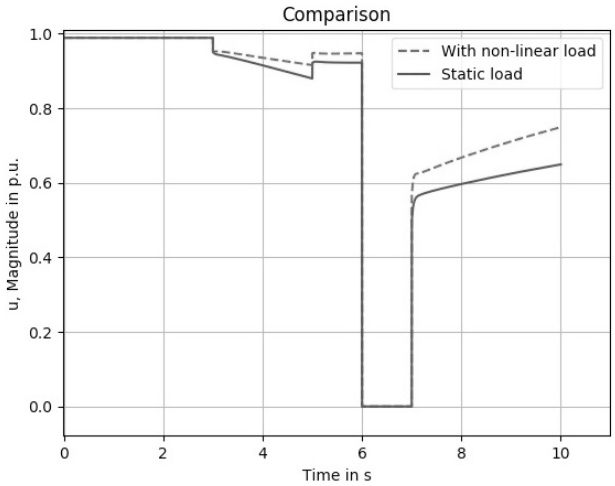


Figure 5-11: Difference in static and dynamic load responses

5-2-6 Transformer

Transformer was modelled by using the PSAT transformer model from the OpeniPSL, to obtain the same response as PowerFactory. In the test system shown in Figure 5-12 is replicated in OpenModelica as shown in Figure 5-13.

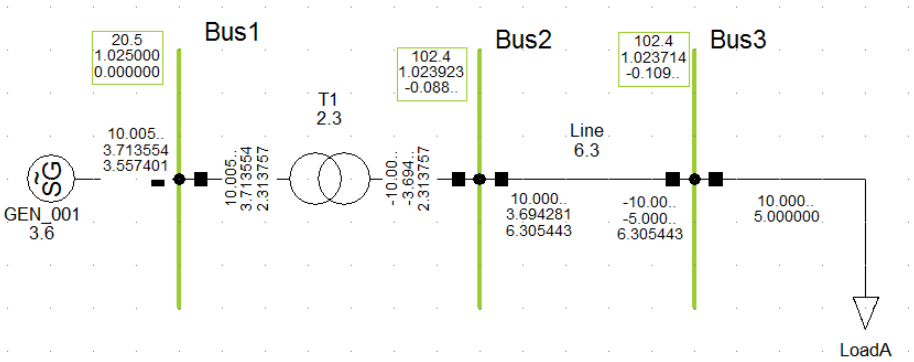


Figure 5-12: Test system for transformer validation in PowerFactory

The load is increased by 100% between 3-5 sec and the responses of the primary and secondary voltages of the transformer is shown in Figure. 5-14.

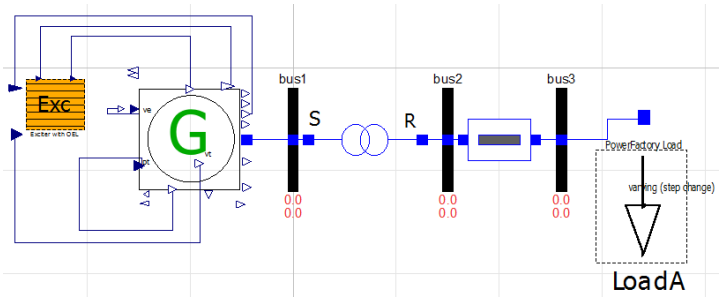
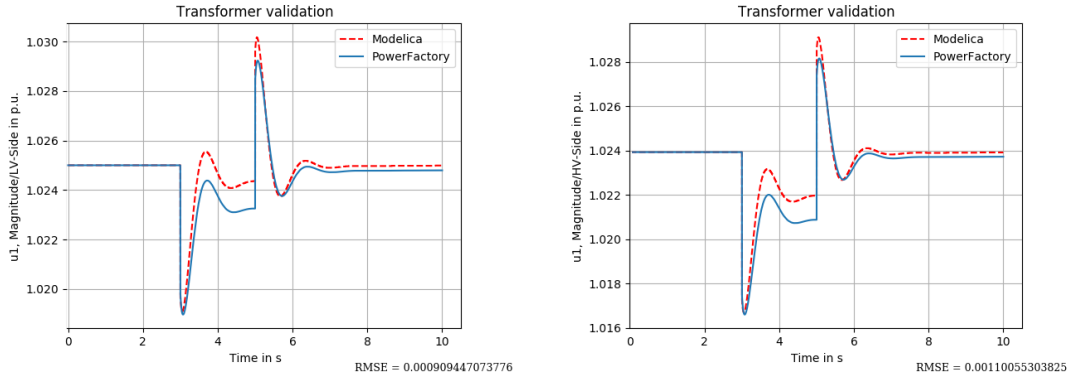


Figure 5-13: Test system for transformer validation in OpenModelica



(a) Primary voltage (b) Secondary voltage

Figure 5-14: Comparison of Modelica and PowerFactory responses for transformer validation

Note: The transformer model used in this thesis is a modified form of TwoWindingTransformer from the PSAT library. Significant error is seen when this model is used in the simulation. Figure 5-14 shows the error during and after the disturbance.

5-2-7 Transformer tap changers

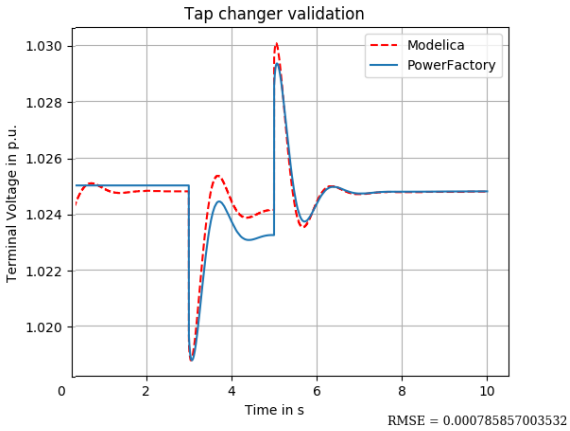
PowerFactory based tap changers based on asymmetrical type described in the section 3-2-9 is modelled and validated for the transformer test system shown in Figure 5-13. Table 5-2 shows the the values of tap positions for which the model was validated.

The fixed ratio tap changer works based on the equation in 3-25. where, $phitr = 0$, du_{tap}=$0.01 * n$ and n is the number taps moved from the neutral position. For example, if the tap position is set to 93 and the neutral 95, then the secondary voltage decreases by $2 * 0.01V$, to give $1.0047V$ at the secondary side.

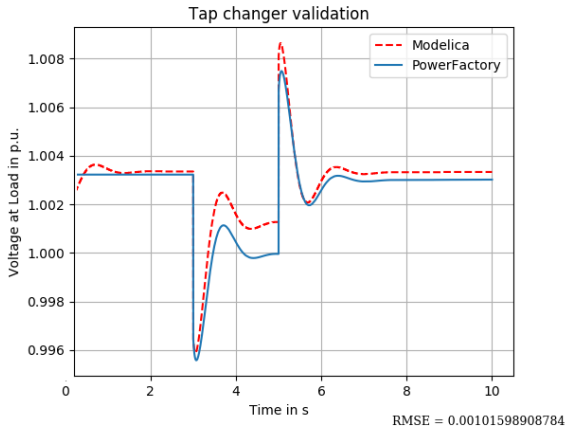
The results of transformer tap position at 93 is shown in Figure 5-15, when a load is increased by 100% between 3-5 sec. The tap changer is added to the existing modified PSAT transformer used in Figure 5-13. Therefore, significant eror can be seen in the simulation responses of the models that include tap changers.

Table 5-2: Results when tap changer is added at the HV side of the transformer (steady-state values)

Tap changer at HV side	Value	Transformer primary	Transformer Secondary
Minimum tap position	90	-	-
Maximum tap position	100	-	-
Neutral tap position	95	-	-
Actual tap position 1	90	1.0247	0.972507
Actual tap position 2	95	1.0247	1.0237
Actual tap position 3	93	1.0247	1.0032
Actual tap position 4	100	1.0247	1.0751



(a) Generator terminal voltage



(b) Voltage at the load

Figure 5-15: Voltages when the tap changer position is at 93

5-2-8 Complete model

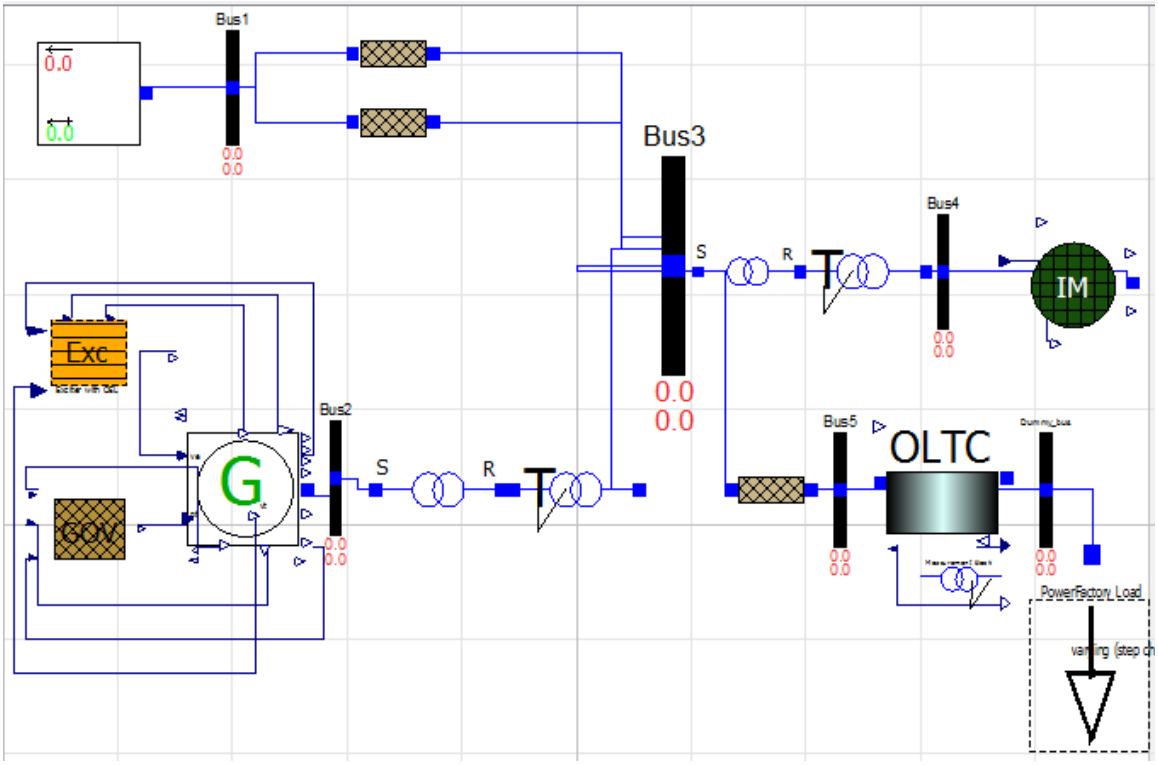


Figure 5-16: Complete model of the modified BPA system in Modelica

Due to various reasons like the time constraint for this thesis, unclear documentations, different algorithms and solvers, few component models are incomplete and not utilized in the test system. However, the test system works without these component models. The list of working models are given in Table 5-3. Therefore, the OLTC and the induction machine are inactive in the test system.

Table 5-3: My caption

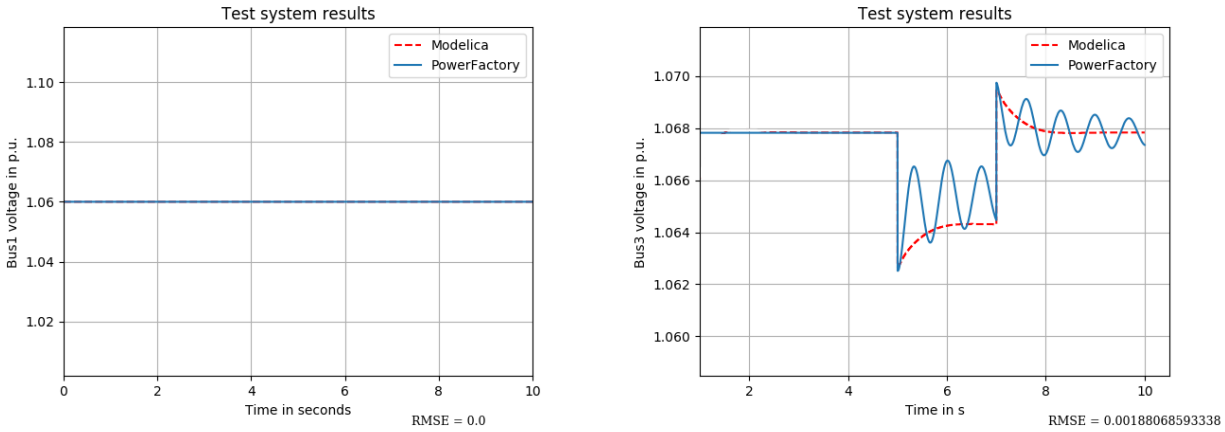
Componenet	Status
Synchronous generator (round rotor) model-2.2	Working
Non-linear dynamic load	Working
Transformer(normal)	Working
Transformer(with taps)	Working
Off-load tap changer	Working
Exciter: mod ST1A	Working
Governor	Working
OLTC	Incomplete
Induction Machine (3rd order)	Incomplete
Infinite Bus	Incomplete

Table 5-4: Comparison of steady state voltage values

Steady state voltage values at the buses		
BUS	**PowerFactory**	**Modelica**
Bus1	1.06	1.06
Bus2	1.04	1.03629
Bus3	1.0678	1.06783
Bus4	1.006987	1.007
Bus5	1.067063	1.06708

The parameter values of the test system is given in Appendix B-5. The steady comparison of Modelica based model and that of PowerFactory is given in Table 5-4.

The infinite bus used in the validation process belongs to the PSAT library of OpeniPSL. The exact dynamics of the infinite bus are missing in the test carried out in this thesis. Figure 5-17 shows that missing dynamics in the Modelica based grid model.



(a) Voltage at Bus1 (b) Voltage at Bus3

Figure 5-17: Voltages at the buses when load is increased by 100% between 5-7sec

5-3 CGMES-CIM based Initialization in Modelica

In order to automatize the process of initialization in Modelica based models, the CGMES-CIM based initialization is carried out. Due to practical constraints, CGMES-CIM files are obtained from PowerFactory. The workflow is described in the Figure 5-18.

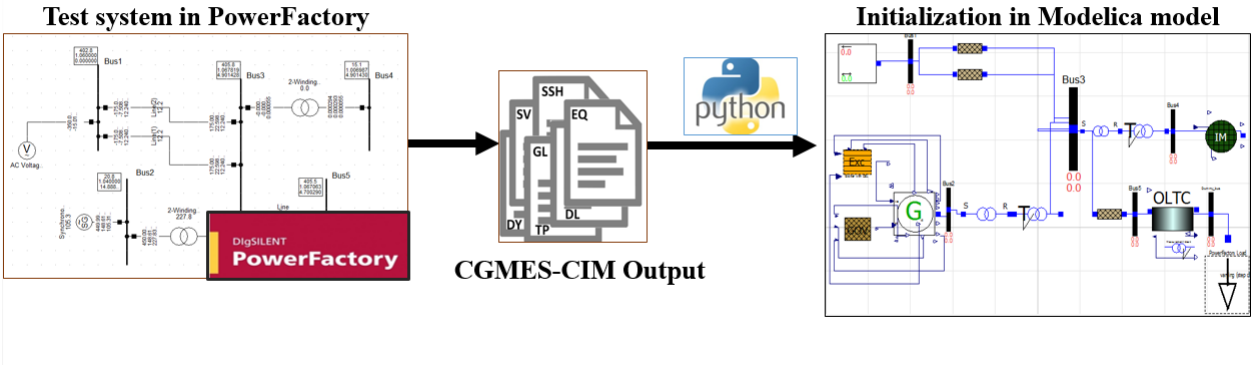


Figure 5-18: Process used in the thesis for CGMES-CIM based initialization of Modelica models

Different scenarios were created in PowerFactory for the considered BPA test system. The developed Python based CGMES-CIM reader gets all the powerflow values from CGMES-CIM to initialize the Modelica based grid models. This Python based interface, matches the names of the generators and loads and replaces the dummy initial values (Modelica models by default will have some random initial values as shown in Figure 5-19.) with the ones from the CGMES-CIM. The change is similar to the one shown in Figure 5-20.

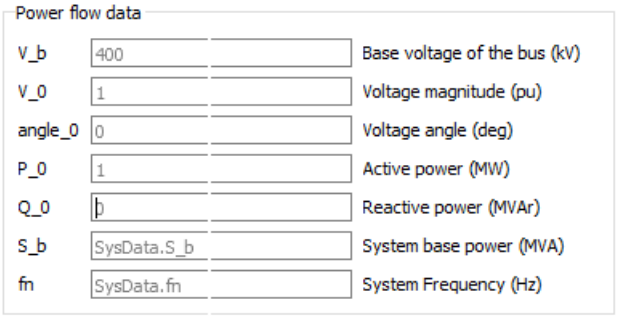


Figure 5-19: Dialog box of a Modelica based generator showing dummy initial values

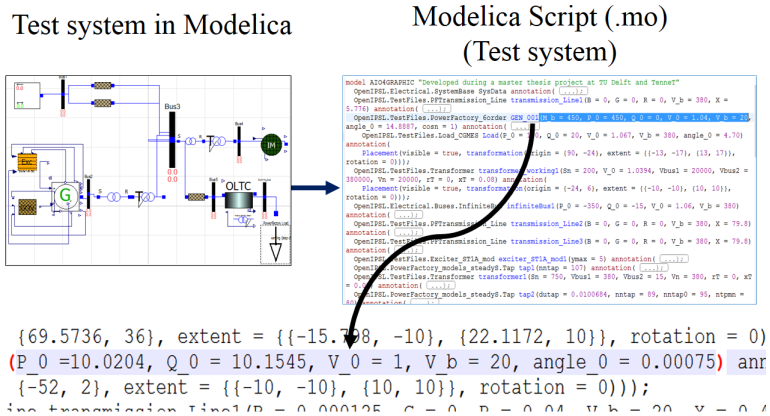


Figure 5-20: Part of modelica script that undergoes changes

Therefore, different scenarios are created and the respective CGMES-CIM files are used to obtain different responses based on different initialization values in OpenModelica automatically.

1. Scenario 1: The test system (Figure 5-16) is used with all the original values.
2. Scenario 1: The load is decreased by 50%.
3. Scenario 1: The active power dispatch of the generator is reduced by 50%.

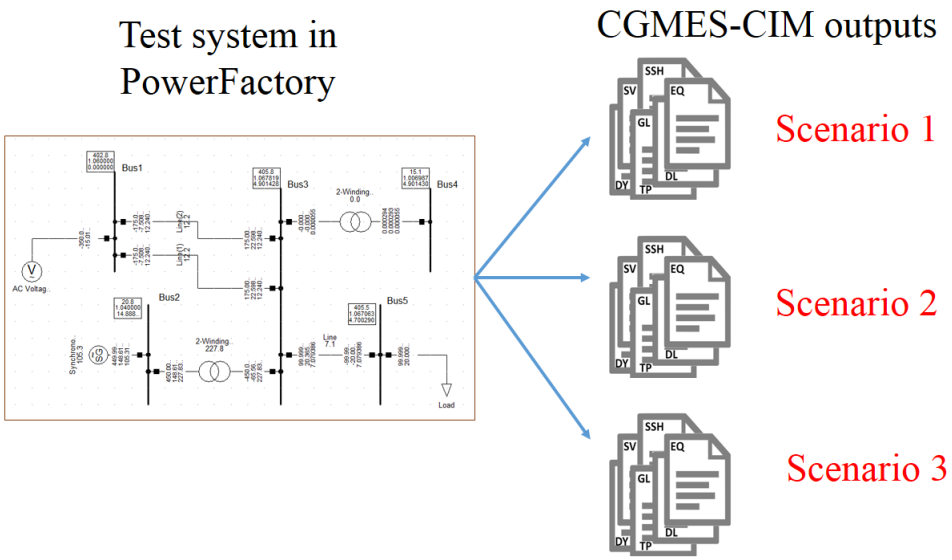


Figure 5-21: Generator output obtained from the Python based CGMES reader

Listing 5.1: Results obtained from the CGMES-CIM reader for Scenario 1

```
*** The new powerflow values are: ***
#Total_Gen_PF list reads [Generator name, Active power, Reactive power,
    Voltage, Angle]:
[('Gen_001', '449.999', '148.619', '1.04', '14.8888')]
#SlackGenData list gives the Corresponding base voltage [Generator name,
    Base voltage, nominal voltage, powerfactor]:
('Gen_001', '450.', '20.', '1.')
#Total_Infinitebus_details gives [Infinite_bus_name, Active power, Reactive
    power, voltage, angle, base voltage in kV]:
[('Inf_bus', '350', '15', '1.06', '0', '380.')]
#Total_Load_details gives [Load name, Active power, Reactive power, voltage,
     angle, base voltage, xaxis, yaxis]:
[('Load', '100.0', '20.0', '1.0670631578947367', '4.70029', '380.', '161.875
    ', '118.125')]
```

As described in the section 4-4, the powerflow values are defined inside the SV profile. The SV profile for scenario-1 is shown in Figure 5-22. The CGMES-CIM reader maps the powerflow values directly to the modelica script with the help of element names. The CGMES-CIM reader output for scenario-1 is shown in Listing 5.1. The response of the voltages at Bus3 for different initilization values is shown in Figure 5-23.

```
<cim:SvPowerFlow rdf:ID="_42087de5-4fe5-44df-8089-6521cfd43268">
  <cim:SvPowerFlow.Terminal rdf:resource="#_4a5f0703-cfa2-4c88-a0d2-f80333a3623d" />
  <cim:SvPowerFlow.p>100.</cim:SvPowerFlow.p>
  <cim:SvPowerFlow.q>20.</cim:SvPowerFlow.q>
</cim:SvPowerFlow>
<cim:SvPowerFlow rdf:ID="_8cc92cc3-cbd1-47b0-8f71-483b6e157d8e">
  <cim:SvPowerFlow.Terminal rdf:resource="#_d7d50ad4-64a3-4de0-a980-179bd575e93a" />
  <cim:SvPowerFlow.p>-449.999</cim:SvPowerFlow.p>
  <cim:SvPowerFlow.q>-148.619</cim:SvPowerFlow.q>
</cim:SvPowerFlow>
<cim:SvPowerFlow rdf:ID="_e17c0454-fc84-4ac3-8531-b75463ac1ae9">
  <cim:SvPowerFlow.Terminal rdf:resource="#_f322a3fb-555b-4553-9cb8-19219d2e4baa" />
  <cim:SvPowerFlow.p>350.</cim:SvPowerFlow.p>
  <cim:SvPowerFlow.q>15.0162</cim:SvPowerFlow.q>
</cim:SvPowerFlow>
<cim:SvVoltage rdf:ID="_d662e501-1406-43c1-add4-d6a8db02d1ac">
  <cim:SvVoltage.TopologicalNode rdf:resource="#_ee230f19-f08c-49c2-ac2b-15f5e84324ba" />
  <cim:SvVoltage.angle>4.70029</cim:SvVoltage.angle>
  <cim:SvVoltage.v>405.484</cim:SvVoltage.v>
</cim:SvVoltage>
<cim:SvVoltage rdf:ID="_13f16505-f6b0-4ffa-9e4b-075ba1c38ac5">
  <cim:SvVoltage.TopologicalNode rdf:resource="#_1d4d7e8d-1f67-4474-8551-9d9b64bbf66c" />
  <cim:SvVoltage.angle>4.90143</cim:SvVoltage.angle>
  <cim:SvVoltage.v>405.771</cim:SvVoltage.v>
</cim:SvVoltage>
<cim:SvVoltage rdf:ID="_9c7a8b47-5a7e-4858-b31d-7f9626174524">
  <cim:SvVoltage.TopologicalNode rdf:resource="#_f2db42b1-9168-4166-b88c-29e8050fd9d5" />
  <cim:SvVoltage.angle>14.8888</cim:SvVoltage.angle>
  <cim:SvVoltage.v>20.8</cim:SvVoltage.v>
</cim:SvVoltage>
<cim:SvVoltage rdf:ID="_1082b46c-12aa-48b5-b4da-d71badd61cff">
  <cim:SvVoltage.TopologicalNode rdf:resource="#_37836c8c-39b7-403a-890a-7a0f8eb98adb" />
  <cim:SvVoltage.angle>0.</cim:SvVoltage.angle>
  <cim:SvVoltage.v>402.8</cim:SvVoltage.v>
</cim:SvVoltage>
<cim:SvVoltage rdf:ID="_79db9c12-fbed-4ba5-90d9-115d53e485aa">
  <cim:SvVoltage.TopologicalNode rdf:resource="#_01f5932e-dabe-4b3e-aac5-f1d30b7e143b" />
  <cim:SvVoltage.angle>4.90143</cim:SvVoltage.angle>
  <cim:SvVoltage.v>15.1048</cim:SvVoltage.v>
```

Figure 5-22: SV profile of the the test system for scenario 1

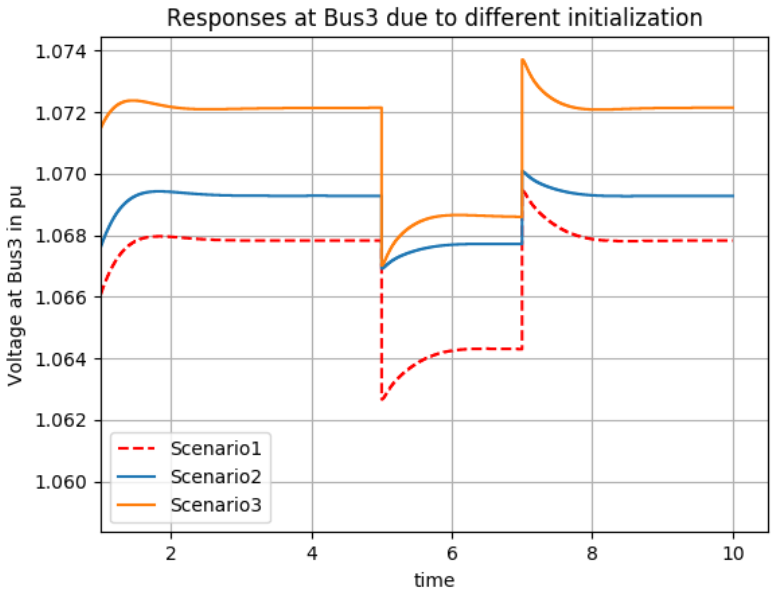


Figure 5-23: Voltage magnitude at Bus3 for different scenarios

5-4 CGMES-CIM to Modelica converter

The test system considered is inspired by the IEEE 9-bus system. The 9-bus system used in this thesis consists of three generators, six transmission lines, three loads and three transformers. Generator Gen1 has a modified ST1A exciter. The complete data of the test system is provided in the Appendix C-2.

Powerflow values of the test system are shown in Table 5-5:

Table 5-5: Powerflow values of the 9-bus test system

Powerflow values of the test system				
Element	Active power	Reactive power	Voltage magnitude	Voltage angle in deg.
Gen1	10.0019	-42.8841	1.04	0
Gen2	20	20	1.04416	0.0450432
Gen3	20	20	1.04465	0.0727758
LoadA	10	10	1.0425695	-0.0369459
LoadB	10	10	1.04263043	-0.0318971
LoadC	10	10	1.04278260	-0.0314053
Bus4	-	-	1.042474	-0.031714
Bus5	-	-	1.042571	-0.036946
Bus6	-	-	1.042626	-0.031897
Bus7	-	-	1.042783	-0.030731
Bus8	-	-	1.042784	-0.031405
Bus9	-	-	1.042812	-0.028207

A similar approach is followed as shown in Figure 5-18 to obtain the CGMES-CIM files. The Python based CGMES-CIM to Modelica converter converts the CGMES-CIM profiles directly into Modelica file. The test system in PowerFactory is shown in Figure 5-24. The grid parameters are shown in Appendix.

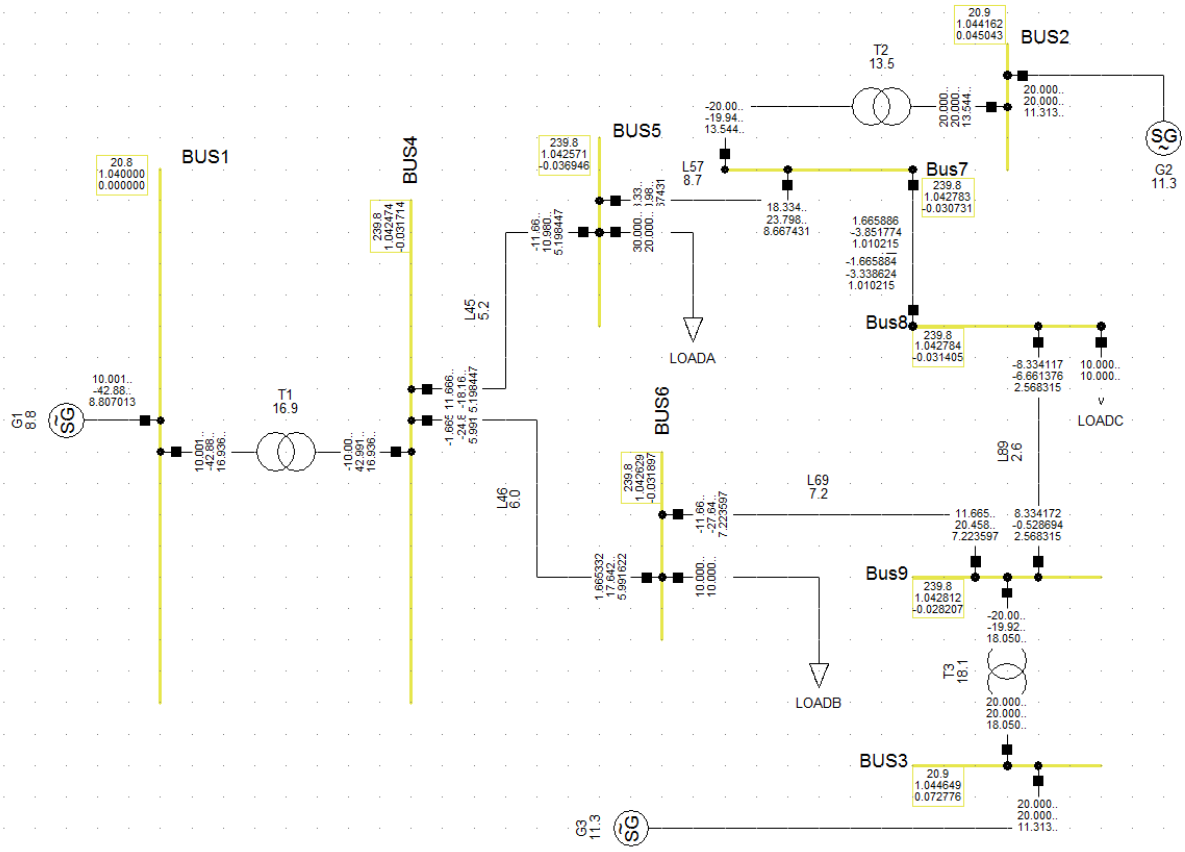


Figure 5-24: The test system modelled in PowerFactory

The proposed CGMES-CIM to Modelica converter uses SV, TP, EQ, DL and DY profiles. The information regarding the element coordinates are obtained from the DL profile. After placing the elements in their respective co-ordintes, the details regarding their connections with other elements in the grid is obtained from the TP profile. However, the elements are approximately (scaling is done to accommodate the grid model within the boundaries allowed in OpenModelica.) placed according to the original PowerFactory model.

The converter successfully converts the CGMES-CIM into Modelica file (.mo) with exact names and parameter values as in the original PowerFactory model (Figure 5-24). The Modelica script in OpenModelica is as shown in Figure 5-27. The corresponding generated Modelica script is shown in Appendix C-2.

Note-1 : The proposed converter works well with transformers, transmission lines, generators, loads and buses. It can be further extended to work with other elements in the grid.

Note-2 : The perturbations were added within the converter script. Also, PowerFactory allows CGMES export of standard IEEE models only, therefore even the modified ST1A exciter was added within the converter script and not from the CGMES-CIM files. However, the user can also opt to add the perturbations manually after loading the .mo script in OpenModelica.

The process used in this thesis for obtaining the converted script is as shown in Figure 5-25.

1. The grid model is developed in PowerFactory and powerflow is carried out (SV profile is obtained only if powerflow is carried out). Also, make sure that the poweflow is converged and limits of the power system elements are not violated.

2. Using the 'CGMES Tools' option, first the grid information needs to be converted to CIM. Later on, it is exported clicking 'CIM Data Export' option.

3. These exported CGMES-CIM files are accessed by the developed Python based converter and internal dictionaries/lists are generated that stores the information of the grid that is later used to make the Modelica script (.mo) file.

4. The generated .mo file from the converter is loaded into OpenModelica and dynamic simulations are carried out.

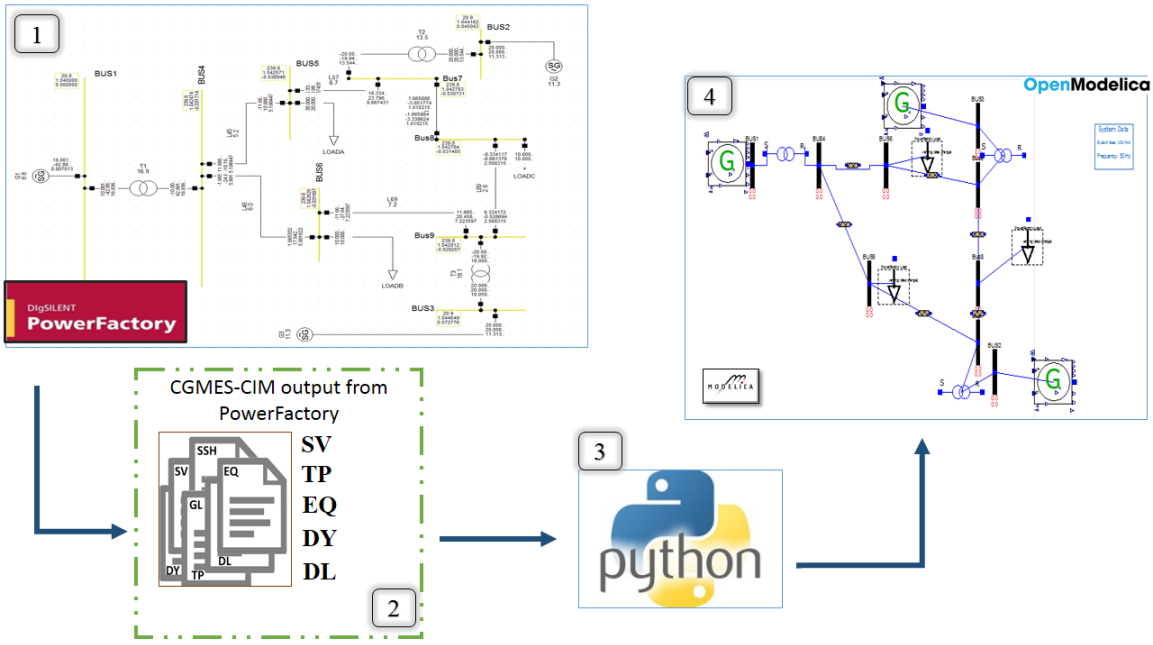


Figure 5-25: Process followed in this thesis to obtain the Modelica model directly from CGMES-CIM

The Dynamic parameters of the generators are obtained from DY profile. Figure 5-26 shows the DY profile with the generator (Gen1) parameters. Similar values are obtained for other generators in the system.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with PowerFactory 16.0.4 (digcimdb.dll ServicePack 0) -->
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2013/CIM-schema-cim16#" xmlns:entsoe="http://entsoe.eu/CIM/SchemaExtension/3/1#" xmlns:md=
"http://iec.ch/TC57/61970-552/ModelDescription/1#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <md:FullModel rdf:about="urn:uuid:_22375a1d-bf20-446c-8d34-730f79f4266d">
    <md:Model.DependentOn rdf:resource="urn:uuid:_3a7ab671-bb90-4eff-b25d-3adbc755dbf8" />
    <md:Model.created>2017-07-25T10:38:57</md:Model.created>
    <md:Model.modelingAuthoritySet>9bus</md:Model.modelingAuthoritySet>
    <md:Model.profile>http://entsoe.eu/CIM/Dynamics/3/1</md:Model.profile>
    <md:Model.scenarioTime>2011-03-07T14:18:25</md:Model.scenarioTime>
  </md:FullModel>
  <cim:SynchronousMachineTimeConstantReactance rdf:ID="_d7357eb6-3f33-4e28-9248-60ff26369ec3">
    <cim:DynamicsFunctionBlock.enabled>true</cim:DynamicsFunctionBlock.enabled>
    <cim:IdentifiedObject.name>Plant1</cim:IdentifiedObject.name>
    <cim:RotatingMachineDynamics.damping>0.</cim:RotatingMachineDynamics.damping>
    <cim:RotatingMachineDynamics.inertia>3.5</cim:RotatingMachineDynamics.inertia>
    <cim:RotatingMachineDynamics.saturationFactor>0.</cim:RotatingMachineDynamics.saturationFactor>
    <cim:RotatingMachineDynamics.saturationFactor120>0.</cim:RotatingMachineDynamics.saturationFactor120>
    <cim:RotatingMachineDynamics.statorLeakageReactance>0.2</cim:RotatingMachineDynamics.statorLeakageReactance>
    <cim:RotatingMachineDynamics.statorResistance>0.031</cim:RotatingMachineDynamics.statorResistance>
    <cim:SynchronousMachineDetailed.efdBaseRatio>1.</cim:SynchronousMachineDetailed.efdBaseRatio>
    <cim:SynchronousMachineTimeConstantReactance.tpdo>9.1</cim:SynchronousMachineTimeConstantReactance.tpdo>
    <cim:SynchronousMachineTimeConstantReactance.tppdo>0.03</cim:SynchronousMachineTimeConstantReactance.tppdo>
    <cim:SynchronousMachineTimeConstantReactance.tppqo>0.2</cim:SynchronousMachineTimeConstantReactance.tppqo>
    <cim:SynchronousMachineTimeConstantReactance.tpqo>2.3</cim:SynchronousMachineTimeConstantReactance.tpqo>
    <cim:SynchronousMachineTimeConstantReactance.xDirectSubtrans>0.25</cim:SynchronousMachineTimeConstantReactance.xDirectSubtrans>
    <cim:SynchronousMachineTimeConstantReactance.xDirectSync>2.1</cim:SynchronousMachineTimeConstantReactance.xDirectSync>
    <cim:SynchronousMachineTimeConstantReactance.xDirectTrans>0.3</cim:SynchronousMachineTimeConstantReactance.xDirectTrans>
    <cim:SynchronousMachineTimeConstantReactance.xQuadSubtrans>0.256</cim:SynchronousMachineTimeConstantReactance.xQuadSubtrans>
    <cim:SynchronousMachineTimeConstantReactance.xQuadSync>2.1</cim:SynchronousMachineTimeConstantReactance.xQuadSync>
    <cim:SynchronousMachineTimeConstantReactance.xQuadTrans>0.73</cim:SynchronousMachineTimeConstantReactance.xQuadTrans>
  </cim:SynchronousMachineTimeConstantReactance>
```

Figure 5-26: Dynamic profile showing the generator parameters required for dynamic simulations

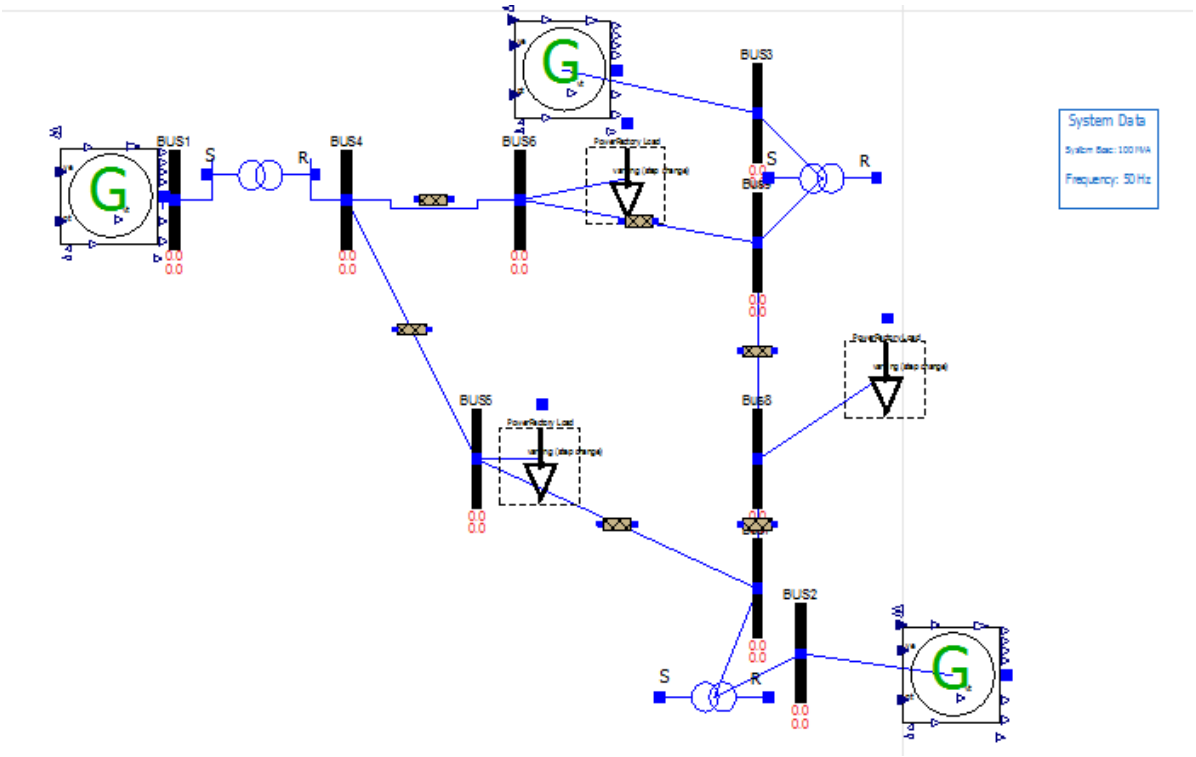


Figure 5-27: Automatically generated test system in OpenModelica using the proposed converter

The responses of the Modelica based test system was compared with the responses of the same test system in PowerFactory. A perturbation of load variation at LoadC was applied between 5-7 s. The responses of voltage at Bus1 is shown in Figure 5-28.

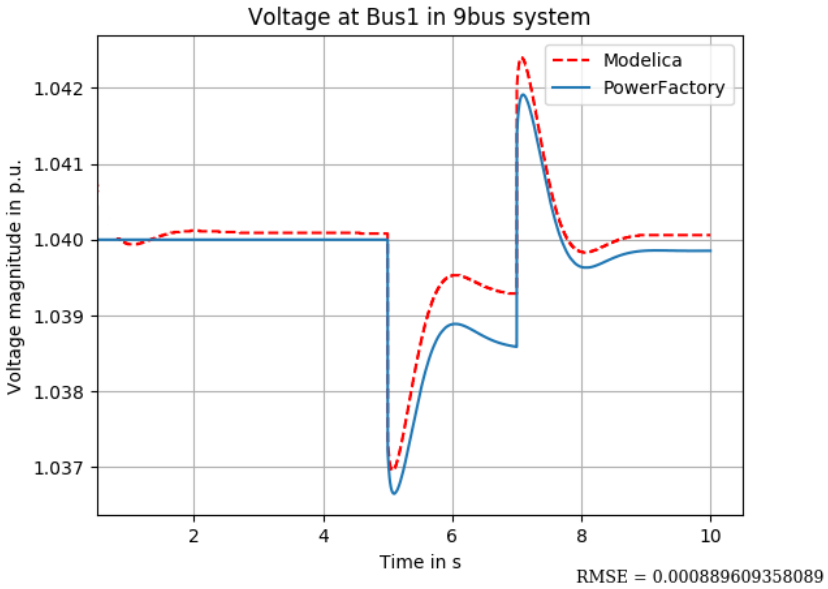


Figure 5-28: Volatge at Bus1 for a load increase of 100% at LoadC between 5-7 sec.

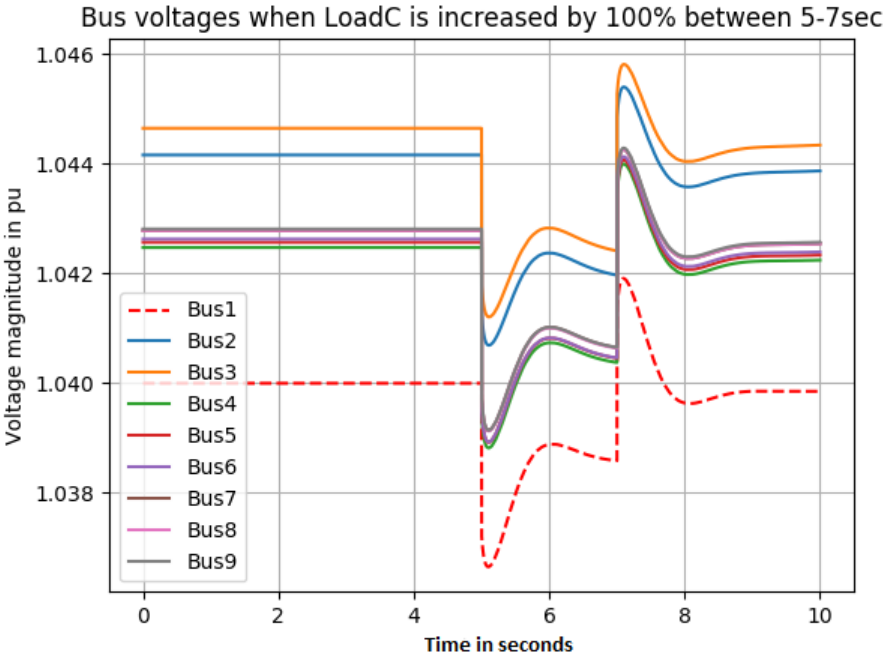


Figure 5-29: Voltages at all the buses when a perturbation is applied

Therefore, if the grid model consists only generators, lines, loads, buses and transformers, then the user can directly convert the CGMES-CIM files into Modelica (.mo file) with a click of a button and perform dynamic simulations in the Modelica environment using the proposed converter.

Chapter 6

Conclusions and future scope

This chapter aims to briefly describe the findings obtained during this thesis project and the future scope of the CGMES-CIM to Modelica Converter.

6-1 Conclusion

PowerFactory based Modelica models were successfully developed and validated for a test case of modified BPA system. The validation results show a high degree of similarity between dynamic performance of Modelica model and that of the reference PowerFactory model.

The literature survey helped understand the importance of a common language that supports interoperability between power system tools with respect to dynamic simulations. Chapter one gives the introduction about the background of this thesis project and the motivation to use Modelica with the CGMES-CIM for dynamic simulations. Within the same chapter, it was concluded that the current CGMES does not fully support interoperability, with respect to dynamic simulations. Modelica based simulations are going to be crucial in the near future for dynamic model exchanges if implemented through the latest version of CGMES (v2.5).

With this as a backdrop, Chapter 2 was dedicated for describing the Modelica language and OpenIPSL. Extending the PowerFactory based models within OpenIPSL was one of the sub-goals of this thesis project. Therefore, Chapter 3 describes the modelling philosophy used by PowerFactory. All the necessary equations and parameter descriptions of the required component models are shown in Chapter 3 and used for model development in OpenModelica. Another sub-goal of this thesis is using the CGMES-CIM for initialization of Modelica grid models, Chapter 4 was dedicated to describe the CGMES-CIM in detail.

Finally, each component model (Generator, load, exciter etc) was validated using test system that was identical to that of PowerFactory. Modelica based dynamic simulations were carried out and compared with reference results (signal records obtained through time domain simulations) from PowerFactory to validate each component model. With individual models validated, the test system considered in this thesis (Modified BPA system) was

validated. The results of this test system did not contain the dynamics offered by the infinite bus, as the infinite bus from PSAT package is utilized in this thesis.

This thesis also proposes a new method for initialization of Modelica models using CGMES-CIM files (Method-1). This method is an efficient substitution for manual input of powerflow values and helps to minimize the human-error caused during input of powerflow values into the Modelica model. Furthermore, the shortcomings and the rules for using this method is mentioned in Chapter 4. This thesis further uses CGMES-CIM to obtain Model to Model transformation or direct conversion of CGMES-CIM to Modelica file (Method-2). The Method-2 was shown to be better than Method-1 and is well described in Chapter 5. Example of 9 bus system was chosen to test this proof of concept. The proposed CGMES-CIM converter worked successfully for a grid model with generators, loads, buses, transformers and transmission lines.

Therefore, this thesis successfully provides a proof of concept for an open-source converter that directly converts the CGMES based CIM to Modelica script file with just a click of a button. All the data concerning the validation tests are documented in the appendix section of this report. The procedure for validating a model is explained wherever possible and would be easy to follow up on the work if there is any need to carry out more tests.

6-2 Future scope and recommendations

Further development of PowerFactory based models:

- Either due to the scope of the thesis or due to the time constraint for the thesis, not all the electrical models required for the analysis of modified BPA test system are complete. Furthermore, Modelica models can be developed based on PowerFactory for all kinds of standard test systems like IEEE 9 bus, 14 bus or 39 bus systems. Alos, models for power electronic interfaced devices and FACTs devices need to be developed.
- Once all the models are developed, project specifically targeting the study of Modified BPA test system or the all-in-one system can be initiated. Small signal stability, long-term voltage stability and frequency stability could be carried out on the test system.
- The solver used in this thesis for Modelica simulation is not the actual solver that PowerFactory uses. Further studies/ analysis can be done with respect to the solvers.
- Investigate the use of Rapid Parameter Identification toolbox (RaPId), developed within the EU FP7 iTesla project. The toolbox does parameter identification on models developed using the Modelica language, focusing in particular on power system model identification needs.

Further development of CGMES-CIM to Modelica converter:

This thesis emphasized on achieving a proof of concept with CGMES based initialization of Modelica models. The CGMES-CIM converter proposed in this thesis worked successfully for a grid model with generators, loads, buses, transformers and transmission lines.

- Further improvements can be made to the converter by adding routines to read other electrical elements from CGMES-CIM like the shunts, tap changers, governors, PSS, exciters etc. The converter could be developed to an extend, where no specialized knowledge is required from the user.
- Geographic location (GL) profile can also be utilized based on the logic used in the converter for real grid models. This way the user can also find the geographic information of the element in the grid.
- The Python based converter can be further developed to accommodate any model from the OpenIPSL based on the origin of CGMES-CIM files. (eg: if the CGMES-CIM is from PSSE, then all the required models are chosen from the available PSSE package in OpenIPSL). This way the converter works with CGMES-CIM from any source and generates a Modelica model that is source specific. This plays a crucial role in the co-ordination among energy players.

Few precautions to take while working with Modelica based simulations; check if the model has been initialized properly. Always verify the initialization values and then only proceed to dynamic simulations. Check if the faults/events in both systems (OpenModelica and PowerFactory) are the same. Verify the model equations and parameter values.

Bibliography

[1] L. Vanfretti, T. Rabuzin, M. Baudette, and M. Murad, "itesla power systems library (ipsl): A modelica library for phasor time-domain simulations," *SoftwareX*, vol. 5, pp. 84–88, 2016.

[2] C. European, "A policy framework for climate and energy in the period from 2020 to 2030," tech. rep., 2014.

[3] F. Gonzalez-Longatt and J. L. Rueda, *PowerFactory applications for power system analysis*. Springer, 2014.

[4] P. Aristidou, F. Plumier, T. Van Cutsem, and C. Geuzaine, "Power system simulation challenges," 2013.

[5] P. Kundur, N. J. Balu, and M. G. Lauby, *Power system stability and control*, vol. 7. McGraw-hill New York, 1994.

[6] L. Qi, *Modelica Driven Power System Modeling, Simulation and Validation*. PhD thesis, Master's thesis, Royal Institute of Technology (KTH), 2014.

[7] J. Britton, P. Brown, J. Moseley, and M. Bunda, "Optimizing operations with cim: Today's grid relies on network analysis (and a lot of data)," *IEEE Power and Energy Magazine*, vol. 14, no. 1, pp. 48–57, 2016.

[8] ENTSO-E, "Common Grid Model Exchange Standard, Version 2.4," Tech. Rep. August, 2014.

[9] "ENTSO-E common grid model exchange standards." https://www.entsoe.eu/major-projects/common-information-model-cim/cim-for-grid-models-exchange/standards/Pages/default.aspx. Accessed: 2016.

[10] "CGMES Annex F common grid model exchange standards v2.5." https://www.entsoe.eu/Documents/CIM_documents/IOP/CGMES_2_5_TechnicalSpecification_61970-600_Part%201_Ed2.pdf. Accessed: 2016.

[11] G. León, M. Halat, M. Sabaté, J.-B. Heyberger, F. J. Gómez, and L. Vanfretti, "Aspects of power system modeling, initialization and simulation using the modelica language," in *PowerTech, 2015 IEEE Eindhoven*, pp. 1–6, IEEE, 2015.

[12] F. J. Gómez, L. Vanfretti, and S. H. Olsen, "Binding cim and modelica for consistent power system dynamic model exchange and simulation," in *Power & Energy Society General Meeting, 2015 IEEE*, pp. 1–5, IEEE, 2015.

[13] L. Vanfretti, T. Bogodorova, and M. Baudette, "A modelica power system component library for model validation and parameter identification," in *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, no. 096, pp. 1195–1203, Linköping University Electronic Press, 2014.

[14] T. Øyvang, D. Winkler, B. Lie, and G. J. Hegglid, "Power system stability using modelica," 2014.

[15] M. Zhang, M. Baudette, J. Lavenius, S. Løvlund, and L. Vanfretti, "Modelica implementation and software-to-software validation of power system component models commonly used by nordic tsos for dynamic simulations," in *Proceedings of the 56th Conference on Simulation and Modelling (SIMS 56), October, 7-9, 2015, Linköping University, Sweden*, no. 119, pp. 105–112, Linköping University Electronic Press, 2015.

[16] R. Leelaruji and L. Vanfretti, "" all-in-one" test system modelling and simulation for multiple instability scenarios," 2011.

[17] C. T. Force, "38-02-10, modeling of voltage collapse including," *Dynamic Phenomena, CIGRE Brochure*, no. 75, 1993.

[18] OpenCPS, "CIM/UML and Modelica Electrical Power System Models and Tooling for MST Testing and Demonstration.".

[19] E. Widl, P. Palensky, P. Siano, and C. Rehtanz, "Guest editorial modeling, simulation, and application of cyber-physical energy systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2244–2246, 2014.

[20] "Modelica org. modelica and the modelica association." https://www.modelica.org/. Accessed: 2017.

[21] D. Zimmer, "Towards improved class parameterization and class generation in modelica," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools; Oslo; Norway; October 3*, no. 047, pp. 33–42, Linköping University Electronic Press, 2010.

[22] L. Vanfretti, W. Li, T. Bogodorova, and P. Panciatici, "Unambiguous power system dynamic modeling and simulation using modelica tools," in *Power and Energy Society General Meeting (PES), 2013 IEEE*, pp. 1–5, IEEE, 2013.

[23] "openModelica open-source modelica-based modeling and simulation environment." https://www.openmodelica.org/. Accessed: 2017.

[24] M. ZHANG, "Mango classes-modelica classes of the norvegian grid for itesla and sw-to-sw validation," 2014.

[25] B. Karlsson, “Comparison of psse & powerfactory,” 2013.

[26] D. PowerFactory, “Digsilent powerfactory 16 user manual,” 2016.

[27] R. Leelaruji and L. Vanfretti, “" all-in-one" test system modelling and simulation for multiple instability scenarios,” 2011.

[28] D. PowerFactory, “Digsilent powerfactory 16 technical reference documentation, synchronous machine,” 2016.

[29] D. PowerFactory, “Digsilent powerfactory 16 technical reference documentation, general load,” 2016.

[30] D. PowerFactory, “Digsilent powerfactory 16 technical reference documentation, two winding transformer (3phase),” 2016.

[31] D. PowerFactory, “Digsilent powerfactory 16 technical reference documentation, asynchronous machines,” 2016.

[32] C. Ivanov, T. Saxton, J. Waight, M. Monti, and G. Robinson, “Prescription for interoperability: Power system challenges and requirements for interoperable solutions,” *IEEE Power and Energy Magazine*, vol. 14, no. 1, pp. 30–39, 2016.

[33] “Common information model primer: Third edition.” https://www.epri.com/. Accessed: 2016.

[34] R. V. S. M. L. María, Z. G. L. F. Beaude, S. Petitrenaud, and J.-B. Heyberger, “A tool to ease modelica-based dynamic power system simulations,”

Appendix A

Implementing the PowerFactory based models in Modelica is a challenging task. The following shows the steps to achieve a successful implementation:

- Try to understand the conceptual background of each individual model. Read the PowerFactory manual/documentation of the model.
- Identify the state equations and other main equations that define the dynamic behaviour of the model. Figure out the initialization equations.
- Develop a model in Modelica based on the information gathered (usually, a test system needs to be constructed in both simulation platforms to test a new developed model).
- Perform a software-to-software validation of the Modelica model against the PowerFactory model (comparison of the behavior by using the same test system in both Modelica and PowerFactory).

A-1 Synchronous generator

Table A-1: Input Definition of the RMS-Model

Input Signal	Symbol	Description
ve	ve	Excitation voltage
pt		Turbine power

Table A-2: State Variables

Parameter	Symbol	Description
psifd	ψ_{fd}	Excitation flux
psi1d	ψ_{1d}	Flux in 1d-damper winding, d-axis
psi1q	ψ_{iq}	Fluxin 1q-damper winding, q-axis
psi2q	ψ_{2q}	Flux in 2q-damper winding, q-axis
speed	n	Speed
phi	ϕ	Rotor position angle

Table A-3: Output Definition

Parameter	Symbol	Description
ut	$\|ut\|$	Terminal Voltage, Magnitude
utr		TerminalVoltage, Real Part
uti		Terminal Voltage, Imaginary Part
pgt		Electrical Power (based on rated active power)
ie	ie	Excitation Current (in non-reciprocal p.u. system)
xspeed		Speed (xspeed = speed)
xme	t_e	Electrical Torque
xmt	t_m	Mechanical Torque
P1		Positive-Sequence, Active Power
Q1		Positive-Sequence, Reactive Power

A-1-1 Parameters and equations

Name in PF	Symbol	Unit	Description
$rstr$	r_{str}	$p.u.$	Stator resistance
xl	x_l	$p.u.$	Stator leakage reactance
xrl	x_{rld}	$p.u.$	Coupling reactance between field and damper winding
$xrlq$	x_{rlq}	$p.u.$	Coupling reactance between q-axis damper windings
xad	x_{ad}	$p.u.$	Mutual (magnetising) reactance, d-axis
xaq	x_{aq}	$p.u.$	Mutual (magnetising) reactance, q-axis
xfd	x_{fd}	$p.u.$	Reactance of excitation (field) winding (d-axis)
$x1d$	x_{1d}	$p.u.$	Reactance of 1d-damper winding (d-axis)
$x1q$	x_{1q}	$p.u.$	Reactance of 1q-damper winding (q-axis)
$x2q$	x_{2q}	$p.u.$	Reactance of 2q-damper winding (q-axis)
rfd	r_{fd}	$p.u.$	Resistance of excitation winding (d-axis)
$r1d$	r_{1d}	$p.u.$	Resistance of 1d-damper winding (d-axis)
$r1q$	r_{1q}	$p.u.$	Resistance of 1q-damper winding (q-axis)
$r2q$	r_{2q}	$p.u.$	Resistance of 2q-damper winding (q-axis)

Figure A-1: The parameters of round rotor 2.2 generator in PowerFactory

A-1-2 Generator code

Listing A.1: Modelica code of the Base Machine and synchronous generator 2.2

```
partial model BM "Developed during a master thesis project at TU Delft"
import Modelica.Constants.pi;
import Complex;
import Modelica.ComplexMath.arg;
import Modelica.ComplexMath.real;
import Modelica.ComplexMath.imag;
import Modelica.ComplexMath.'abs';
import Modelica.ComplexMath.conj;
import Modelica.ComplexMath.fromPolar;
import Modelica.ComplexMath.j;
import Modelica.Blocks.Interfaces.*;
extends OpenIPSL.Electrical.Essentials.pfComponent;
//Machine parameters
parameter Real M_b = 100 "Nominal Power rating (MVA)" annotation(
```

```
Dialog(group = "Machine parameters"));
parameter Real ugn = 20 "Nominal Voltage rating (kV)" annotation(
Dialog(group = "Machine parameters"));
parameter Real cosn = 0.8 "power factor (pu)" annotation(
Dialog(group = "Machine parameters"));
parameter Real xd = 2.1 "d-axis transient reactance (pu)" annotation(
Dialog(group = "Machine parameters"));
parameter Real xq = 2.1 "q-axis transient reactance (pu)" annotation(
Dialog(group = "Machine parameters"));
parameter Real h = 3.5 "Inertia time constant rated to apparent power"
    annotation(
Dialog(group = "Machine parameters"));
parameter Real pgini = 80 "Nominal Active power in MW" annotation(
Dialog(group = "Machine parameters"));
parameter Real qgini = 15 "Nominal Reactive power in Mvar" annotation(
Dialog(group = "Machine parameters"));
parameter Real usetp = 1 "Voltage rating of the machine in (pu)" annotation(
Dialog(group = "Machine parameters"));
parameter Real phiini = 0 "machine angle in degrees" annotation(
Dialog(group = "Machine parameters"));
parameter Real rstr = 0.031;
parameter Real xl = 0.2;
parameter Real xrl = 0.0;
parameter Real td01 = 9.1 "tds0";
parameter Real tq01 = 2.3 "tqs0";
parameter Real td011 = 0.03 "tdss0";
parameter Real tq011 = 0.2 "tqss0";
parameter Real xd1 = 0.3;
parameter Real xq1 = 0.73;
parameter Real xd11 = 0.25;
parameter Real xq11 = 0.256;
//Initialization of pins
OpenIPSL.Connectors.PwPin p(vr(start = vr0), vi(start = vi0), ir(start = ir0
    ), ii(start = ii0)) annotation(
Placement(transformation(extent = {{100, -10}, {120, 10}}),
    iconTransformation(extent = {{100, -10}, {120, 10}})));
//Output phi
RealOutput phi(start = phi0) "Rotor position angle (rad)" annotation(
Placement(transformation(extent = {{100, 60}, {120, 80}}),
    iconTransformation(extent = {{100, 82}, {116, 98}})));
//Input and initial ve;
RealInput ve "Excitation voltage (pu)" annotation(
Placement(transformation(extent = {{-114, 40}, {-94, 60}}),
    iconTransformation(extent = {{-108, 40}, {-88, 60}})));
//Output ie;
RealOutput ie "Machine excitation current" annotation(
Placement(transformation(extent = {{100, -40}, {120, -20}}),
    iconTransformation(extent = {{100, 42}, {116, 58}})));
//Input turbine power(pt)
RealInput pt "Generator turbine power (pu)" annotation(
Placement(transformation(extent = {{-114, -62}, {-94, -42}}),
    iconTransformation(extent = {{-108, -60}, {-88, -40}})));
Modelica.Blocks.Interfaces.RealOutput pt0 "Initial generator turbine power (
    pu)" annotation(
Placement(visible = true, transformation(extent = {{100, -60}, {120, -40}},
    rotation = 0), iconTransformation(extent = {{-42, -104}, {-26, -88}},
    rotation = 0)));
//OutofStep indicator
RealOutput OutOfStep "outofStep indicator" annotation(
```

```
Placement(transformation(extent = {{100, 20}, {120, 40}}),
    iconTransformation(extent = {{100, -58}, {116, -42}})));
//Output torques
RealOutput te "electrical torque (pu)" annotation(
Placement(transformation(extent = {{100, -80}, {120, -60}}),
    iconTransformation(extent = {{100, 62}, {116, 78}})));
RealOutput tm "Machine mechanical torque (pu)" annotation(
Placement(transformation(extent = {{100, 78}, {120, 98}}),
    iconTransformation(extent = {{100, 22}, {116, 38}})));
//Output initial terminal voltage
RealOutput ut0 "Machine mechanical torque (pu)" annotation(
Placement(transformation(extent = {{110, 78}, {130, 98}}),
    iconTransformation(extent = {{110, 22}, {136, 38}})));
//Output speed
RealOutput n(start = n0) "Machine field current (pu)" annotation(
Placement(transformation(extent = {{-10, -10}, {10, 10}}, rotation = 0,
    origin = {110, -90}), iconTransformation(extent = {{-8, -8}, {8, 8}},
    rotation = 0, origin = {108, -90})));
//Output Voltages and currents
Modelica.Blocks.Interfaces.RealOutput Vt(start = V_0) "Bus voltage magnitude
     (pu)" annotation(
Placement(visible = true, transformation(extent = {{120, 50}, {120, 50}},
    rotation = 0), iconTransformation(extent = {{10, -54}, {26, -38}},
    rotation = 0)));
Real anglev(start = anglev_rad) "Bus voltage angle (deg.)";
RealOutput I(start = sqrt(ir0 ^ 2 + ii0 ^ 2)) "Terminal current magnitude (
    pu)" annotation(
Placement(transformation(extent = {{80, 20}, {90, 40}}), iconTransformation(
    extent = {{90, -118}, {106, -132}})));
Real anglei(start = atan2(ii0, ir0)) "Terminal current angle (deg.)";
RealOutput utr "Real part of Terminal voltage" annotation(
Placement(transformation(extent = {{80, 20}, {100, 40}}), iconTransformation
    (extent = {{-80, -118}, {-96, -128}})));
RealOutput uti "Imaginary part of terminal voltage" annotation(
Placement(transformation(extent = {{80, 50}, {100, 70}}), iconTransformation
    (extent = {{-80, -98}, {-90, -112}})));
RealOutput cur1r "Real part of current" annotation(
Placement(transformation(extent = {{80, -60}, {100, -80}}),
    iconTransformation(extent = {{-100, 118}, {-120, 132}})));
RealOutput cur1i "imaginary part of current" annotation(
Placement(transformation(extent = {{80, -20}, {100, -40}}),
    iconTransformation(extent = {{-100, 128}, {-120, 142}})));
//Output powers in pu : NOTE:convert to MW/MVAr for further use!
RealOutput P(start = P_0 / S_b) "Active power (p.u. on S_b)";
RealOutput Q(start = Q_0 / S_b) "Reactive power (p.u. on S_b)";
//protected
Real id "d-axis armature current (pu)";
Real iq "q-axis armature current (pu)";
Real ud "d-axis terminal voltage (pu)";
Real uq "q-axis terminal voltage (pu)";
//protected
parameter Real w_b = 2 * pi * fn "System base speed (rad/s)";
parameter Real anglev_rad = angle_0 * pi / 180 "initial value of bus voltage
     angle in rad";
parameter Real CoB = M_b / S_b;
parameter Real vr0 = V_0 * cos(anglev_rad) "Real component of initial
    terminal voltage";
parameter Real vi0 = V_0 * sin(anglev_rad) "Imaginary component of intitial
    terminal voltage";
```

```
parameter Real p0 = P_0 / M_b "initial active power generation in pu
    machinebase";
parameter Real q0 = Q_0 / M_b "initial reactive power generation in pu
    machinebase";
parameter Complex VT = V_0 * cos(anglev_rad) + j * V_0 * sin(anglev_rad) "
    Complex terminal voltage";
parameter Real angleVolt = arg(VT);
parameter Real magVolt = sqrt((V_0 * cos(anglev_rad)) ^ 2 + (V_0 * sin(
    anglev_rad)) ^ 2);
parameter Complex S = p0 + j * q0 "Complex power on machine base";
parameter Complex It = real(S / VT) - j * imag(S / VT) "Terminal current";
parameter Real ir0 = real(It);
parameter Real ii0 = imag(It);
parameter Real angleCurr = arg(It);
parameter Real magCurr = sqrt(real(It) ^ 2 + imag(It) ^ 2);
parameter Real angg = acos(p0 / (magVolt * magCurr)) "initial Power Factor
    angle";
parameter Real Tan1 = xq * magCurr * cos(angg) - rstr * magCurr * sin(angg);
parameter Real Tan2 = 1 + rstr * magCurr * cos(angg) + xq * magCurr * sin(
    angg)
//load angle
parameter Real TOTangle2 = atan(Tan1 / Tan2);
parameter Real phi0 = TOTangle2 - pi / 2 "initial state variable phi";
parameter Real n0 = 1 "initial speed";
parameter Real tag = 7 "acceleration time constant";
Real fref;
equation
ut0 = V_0;
[p.ir; p.ii] = CoB * [sin(TOTangle2), cos(TOTangle2); -cos(TOTangle2), sin(
    TOTangle2)] * [id; iq];
[p.vr; p.vi] = [sin(TOTangle2), cos(TOTangle2); -cos(TOTangle2), sin(
    TOTangle2)] * [ud; uq];
P = ud * id + uq * iq;
Q = uq * id - ud * iq;
Vt = sqrt(p.vr ^ 2 + p.vi ^ 2) "Terminal Volatage";
anglev = atan2(p.vi, p.vr);
utr = p.vr;
uti = p.vi;
I = sqrt(p.ii ^ 2 + p.ir ^ 2) "Terminal current";
anglei = atan2(p.ii, p.ir);
cur1r = p.ir;
cur1i = p.ii;
OutOfStep = 0 "can't be detected,as this is the reference machine";
der(n) = (tm - te) / tag;
fref = n "The voltage source is considered as reference";
der(phi) = w_b * (n - fref) "Considered as slack machine,fref=n,der(phi)=0";
end BM;

model PowerFactory_6order "Developed during a master thesis project at TU
    Delft and TenneT"
import Modelica.Constants.pi;
import Complex;
import Modelica.ComplexMath.arg;
import Modelica.ComplexMath.real;
import Modelica.ComplexMath.imag;
import Modelica.ComplexMath.'abs';
import Modelica.ComplexMath.conj;
import Modelica.ComplexMath.fromPolar;
import Modelica.ComplexMath.j;
import Modelica.ComplexMath.'sqrt';
```

```
import Modelica.Blocks.Interfaces.*;
Modelica.Blocks.Interfaces.RealOutput ve0(start = ve00) "Initial Excitation
    voltage (pu)" annotation(
Placement(visible = true, transformation(extent = {{100, 40}, {120, 60}},
    rotation = 0), iconTransformation(extent = {{-8, 90}, {8, 106}},
    rotation = 0)));
extends OpenIPSL.TestFiles.BM(id(start = id0), iq(start = iq0), ud(start =
    ud0), uq(start = uq0), te(start = te0), tm(start = tm0), ie(start = ie0)
    , pt(start = pt00), ve(start = ve00));
//protected
parameter Real ud0 = magVolt * sin(TOTangle2) "q-axis component of intitial
    current";
parameter Real uq0 = magVolt * cos(TOTangle2) "d-axis component of intitial
    current";
parameter Real id0 = magCurr * sin(TOTangle2 + angg) "d-axis component of
    intitial voltage";
parameter Real iq0 = magCurr * cos(TOTangle2 + angg) "q-axis component of
    intitial voltage";
protected
Real i1d(start = i1d0) "current in 1d damper";
Real i1q(start = i1q0) "current in 1q damper";
Real i2q(start = i2q0) "current in 2q damper";
Real ifd(start = ifd0) "excitation current";
Real psid(start = psid0);
Real psiq(start = psiq0);
Real psifd(start = psifd0) "excitation flux";
Real psi1d(start = psi1d0) "flux in 1d damper";
Real psi1q(start = psi1q0) "flux in 1q damper";
Real psi2q(start = psi2q0) "flux in 2q damper";
//NOTE:Assumption based on Kundur;
parameter Real i1d0 = 0;
parameter Real i1q0 = 0;
parameter Real i2q0 = 0;
//reactances that are required for rotor current calculations
protected
parameter Real xdetd = (xad + xrl) * (x1d + xfd) + xfd * x1d;
parameter Real xdetq = (xaq + xrlq) * (x2q + x1q) + x2q * x1q;
parameter Real xfdloop = xad + xrl + xfd;
parameter Real x1dloop = xad + xrl + x1d;
parameter Real x1qloop = xaq + xrlq + x1q;
parameter Real x2qloop = xaq + xrlq + x2q;
parameter Real kfd = xad * x1d / ((xad + xrl) * (x1d + xfd) + xfd * x1d);
parameter Real k1d = xad * xfd / ((xad + xrl) * (x1d + xfd) + xfd * x1d);
parameter Real k1q = xaq * x2q / ((xaq + xrlq) * (x2q + x1q) + x2q * x1q);
parameter Real k2q = xaq * x1q / ((xaq + xrlq) * (x2q + x1q) + x2q * x1q);
parameter Real XD11 = xad + xl - (k1d + kfd) * xad;
parameter Real XQ11 = xaq + xl - (k2q + k1q) * xaq
//Initial conditions for rotor flux linkages
parameter Real ifd0 = ((xad + xl) * id0 + uq0 + rstr * iq0) / xad;
parameter Real psid110 = kfd * psifd0 + k1d * psi1d0;
parameter Real psiq110 = k1q * psi1q0 + k2q * psi2q0;
parameter Real psid0 = (-xd11 * id0) + psid110;
parameter Real psiq0 = (-xq11 * iq0) + psiq110;
parameter Real psifd0 = (-xad * id0) + (xad + xrl + xfd) * ifd0;
parameter Real psi1d0 = (-xad * id0) + (xad + xrl) * ifd0;
parameter Real psi1q0 = -xaq * iq0;
parameter Real psi2q0 = -xaq * iq0;
//{START}d axis model parameters//
protected
parameter Real td11 = td011 * (xd11 / xd1);
```

```
parameter Real tq11 = tq011 * (xq11 / xq1);
parameter Real td1 = td01 * (xd1 / xd);
parameter Real tq1 = tq01 * (xq1 / xq);
parameter Real xad = xd - xl;
parameter Real xaq = xq - xl;
parameter Real x1 = xad + xrl;
parameter Real x2 = x1 - (xd - xl) ^ 2 / xd;
parameter Real x3 = (x2 - x1 * (xd11 / xd)) / (1 - xd11 / xd);
parameter Real T1 = xd / xd1 * td1 + (1 - xd / xd1 + xd / xd11) * td11;
parameter Real T2 = td1 + td11;
parameter Real a = (x2 * T1 - x1 * T2) / (x1 - x2);
parameter Real b = x3 * td1 * td11 / (x3 - x2);
parameter Real Tsfd = (-a / 2) + sqrt(a ^ 2 / 4 - b);
parameter Real Ts1d = (-a / 2) - sqrt(a ^ 2 / 4 - b);
parameter Real xfd = (Tsfd - Ts1d) / ((T1 - T2) / (x1 - x2) + Ts1d / x3);
parameter Real x1d = (Ts1d - Tsfd) / ((T1 - T2) / (x1 - x2) + Tsfd / x3);
parameter Real rfd = xfd / (w_b * Tsfd);
parameter Real r1d = x1d / (w_b * Ts1d);
//{END}d axis model parameters//
//q axis model parameters//
protected
parameter Real x11 = xq - xl + xrlq;
parameter Real x22 = x11 - (xq - xl) ^ 2 / xq;
parameter Real x33 = (x22 - x11 * (xq11 / xq)) / (1 - xq11 / xq);
parameter Real T11 = xq / xq1 * tq1 + (1 - xq / xq1 + xq / xq11) * tq11;
parameter Real T22 = tq1 + tq11;
parameter Real a1 = (x22 * T11 - x11 * T22) / (x11 - x22);
parameter Real b1 = x33 * tq1 * tq11 / (x33 - x22);
parameter Real Ts2q1 = (-a1 / 2) + sqrt(a1 ^ 2 / 4 - b1);
parameter Real Ts1q1 = (-a1 / 2) - sqrt(a1 ^ 2 / 4 - b1);
parameter Real x2q = (Ts2q1 - Ts1q1) / ((T11 - T22) / (x11 - x22) + Ts1q1 /
    x33);
parameter Real x1q = (Ts1q1 - Ts2q1) / ((T11 - T22) / (x11 - x22) + Ts2q1 /
    x33);
parameter Real r2q = x2q / (w_b * Ts2q1);
parameter Real r1q = x1q / (w_b * Ts1q1);
//{END}q axis model parameters//
//Assumption{start};
protected
parameter Real xrlq = xrl;
parameter Real te0 = (iq0 * psid0 - id0 * psiq0) / cosn;
parameter Real tm0 = pt00 / n0;
parameter Real xadu = 1.9;
//Assumption{end};
Real ufd(start = ufd0), ud11(start = ud110), uq11(start = uq110), psid11(
    start = psid110), psiq11(start = psiq110);
protected
parameter Real ie0 = xadu * ifd0;
parameter Real ve00 = ie0;
parameter Real ufd0 = rfd / xadu * ve00;
parameter Real pt00 = te0 * n0;
parameter Real ud110 = -n0 * psiq110;
parameter Real uq110 = n0 * psid110;
equation
//____state equations___//
der(psifd) = (ufd - rfd * ifd) * w_b;
der(psi1d) = -r1d * i1d * w_b;
der(psi1q) = -r1q * i1q * w_b;
der(psi2q) = -r2q * i2q * w_b;
////_______stator voltage eqns______//
```

```
//Electromechanical simulation (RMS)
ud11 = -n0 * psiq11;
uq11 = n0 * psid11;
ud = ud11 - rstr * id + n0 * xq11 * iq;
uq = uq11 - rstr * iq - n0 * xd11 * id;
////__________stator flux linkages____//
psid11 = kfd * psifd + k1d * psi1d;
psiq11 = k1q * psi1q + k2q * psi2q;
psid = (-xd11 * id) + psid11;
psiq = (-xq11 * iq) + psiq11;
//_______rotor current eqns______//
ifd = kfd * id + (x1dloop * psifd - (xad + xrl) * psi1d) / xdetd;
i1d = k1d * id + (xfdloop * psi1d - (xad + xrl) * psifd) / xdetd;
i1q = k1q * iq + (x2qloop * psi1q - (xaq + xrlq) * psi2q) / xdetq;
i2q = k2q * iq + (x1qloop * psi2q - (xaq + xrlq) * psi1q) / xdetq;
te = (iq * psid - id * psiq) / cosn;
pt0 = pt00;
tm = pt / n;
ie = xadu * ifd;
ve0 = ve00;
ufd = rfd / xadu * ve;
end PowerFactory_6order;
```

A-2 Load

Listing A.2: Modelica code for PowerFactory non-linear dynamic load

```
model Complete_Load "Developed during a master thesis project at TU Delft -
(100% non-linear dynamic load)"
import Modelica.Constants.pi;
extends OpenIPSL.Electrical.Essentials.pfComponent;
OpenIPSL.Connectors.PwPin p annotation(
Placement(visible = true, transformation(origin = {132, 0}, extent = {{-56,
    -10}, {-36, 10}}, rotation = 0), iconTransformation(origin = {70, 100},
    extent = {{-80, 0}, {-60, 20}}, rotation = 0)));
Real V "Voltage magnitude (pu)";
Real Angle_V "voltage angle (rad)";
Real P(start = P_0 / S_b) "Active power (pu)";
Real Q(start = Q_0 / S_b) "Reactive power (pu)";
Real I;
parameter Real t_start_1 "Start time of first load variation (s)" annotation
    (
Dialog(group = "Variation 1"));
parameter Real t_end_1 "End time of first load variation (s)" annotation(
Dialog(group = "Variation 1"));
parameter Real dP1 "First active load variation  (MW)" annotation(
Dialog(group = "Variation 1"));
parameter Real dQ1 "First reactive load variation (MVAr)" annotation(
Dialog(group = "Variation 1"));
parameter Real t_start_2 "Start time of first load variation (s)" annotation
    (
Dialog(group = "Variation 1"));
parameter Real t_end_2 "End time of first load variation (s)" annotation(
Dialog(group = "Variation 1"));
parameter Real dP2 "First active load variation  (MW)" annotation(
Dialog(group = "Variation 1"));
parameter Real dQ2 "First reactive load variation (MVAr)" annotation(
```

```
Dialog(group = "Variation 1"));
Real pout;
Real qout;
parameter Real umin = 0.8"Lower Voltage Limit";
parameter Real umax = 1.2"upper Voltage Limit";
parameter Real t1 = 0.05 "Dynamic Time constant";
parameter Real kpf, tpf, tpu, kqf, tqf, tqu = 0 "Frequency Constants";
parameter Real ap = 1 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real bp = 0 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kpu0 = 1.5 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kpu1 = 1 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kpu2 = 2 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real aq = 1 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real bq = 0 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kqu0 = 2.5 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kqu1 = 1 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kqu2 = 2 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real cp, cq = 0 annotation(
Dialog(group = "Volatge dependency constants "));
parameter Real kput = ap * kpu0 + bp * kpu1 + cp * kpu2;
parameter Real kqut = aq * kqu0 + bq * kqu1 + cq * kqu2;
parameter Real S_b = SysData.S_b"system base MVA";
Real k;
Real dV;
Real dF;
iPSL.NonElectrical.Continuous.LeadLag f1(K = 0, T1 = 0, T2 = 0, y_start = 0)
    annotation(
Placement(visible = true, transformation(origin = {-52, 60}, extent = {{-10,
    -10}, {10, 10}}, rotation = 0)));
iPSL.NonElectrical.Continuous.LeadLag f2(K = 0, T1 = 0, T2 = 0, y_start = 0)
    annotation(
Placement(visible = true, transformation(origin = {-52, 24}, extent = {{-10,
    -10}, {10, 10}}, rotation = 0)));
iPSL.NonElectrical.Continuous.LeadLag v1(K = 1, T1 = 0, T2 = t1, y_start =
    0) annotation(
Placement(visible = true, transformation(origin = {-52, -40}, extent =
    {{-10, -10}, {10, 10}}, rotation = 0)));
iPSL.NonElectrical.Continuous.LeadLag v2(K = 1, T1 = 0, T2 = t1, y_start =
    0) annotation(
Placement(visible = true, transformation(origin = {-50, -76}, extent =
    {{-10, -10}, {10, 10}}, rotation = 0)));
equation
-P = p.vr * p.ir + p.vi * p.ii;
-Q = p.vi * p.ir - p.vr * p.ii;
V = sqrt(p.vr ^ 2 + p.vi ^ 2);
Angle_V = atan2(p.vi, p.vr);
f1.u = dF;
f2.u = dF;
dV = V_0 - V;
dF = 0;
```

```
v1.u = dV;
v2.u = dV;
pout = (f1.y + 1) * (P_0 / S_b) * (v1.y + V_0) * (ap * (V / V_0) ^ kpu0 + bp
    * (V / V_0) ^ kpu1 + cp * (V / V_0) ^ kpu2);
qout = (f2.y + 1) * (Q_0 / S_b) * (v2.y + V_0) * (aq * (V / V_0) ^ kqu0 + bq
    * (V / V_0) ^ kqu1 + cq * (V / V_0) ^ kqu2);
I = sqrt(p.ii ^ 2 + p.ir ^ 2) "Terminal current";
if V >= umin and V <= umax then
k = 1;
elseif V > 0 and V < umin / 2 then
k = 2 * abs(V) ^ 2 / umin ^ 2;
elseif V > umin / 2 and V < umin then
k = 1 - 2 * ((abs(V) - umin) / umin) ^ 2;
else
k = 1 + (V - umax) ^ 2;
end if;
if time >= t_start_1 and time <= t_end_1 then
P = k * (pout + dP1 / S_b);
Q = k * (qout + dQ1 / S_b);
elseif time >= t_start_2 and time <= t_end_2 then
P = k * (pout + dP2 / S_b);
Q = k * (pout + dQ2 / S_b);
else
P = k * pout;
Q = k * qout;
end if;
end Complete_Load;
```

Listing A.3: Equation part of PowerFactory static load

```
equation
P = p.vr * p.ir + p.vi * p.ii;
Q = p.vi * p.ir - p.vr * p.ii;
V = sqrt(p.vr ^ 2 + p.vi ^ 2);
Angle_V = atan2(p.vi, p.vr);
I = sqrt(p.ii ^ 2 + p.ir ^ 2) "Terminal current";
if V >= umin and V <= umax then
k = 1;
elseif V > 0 and V < umin / 2 then
k = 2 * abs(V) ^ 2 / umin ^ 2;
elseif V > umin / 2 and V < umin then
k = 1 - 2 * ((abs(V) - umin) / umin) ^ 2;
else
k = 1 + (V - umax) ^ 2;
end if;
if time >= t_start_1 and time <= t_end_1 then
P = k * ((P_0 + dP1) / S_b);
Q = k * ((Q_0 + dQ1) / S_b);
elseif time >= t_start_2 and time <= t_end_2 then
P = k * ((P_0 + dP2) / S_b);
Q = k * ((Q_0 + dQ2) / S_b);
else
P = k * (P_0 / S_b);
Q = k * (Q_0 / S_b);
end if;
```

A-3 Transmission Line

```
model PFTransmission_Line "Developed during a master thesis project at TU
    Delft and TenneT:Model for a transmission Line based on the
    pi-equivalent circuit"
outer iPSL.Electrical.SystemBase SysData;
import Modelica.ComplexMath.conj;
import Modelica.ComplexMath.real;
import Modelica.ComplexMath.imag;
import Modelica.ComplexMath.j;
iPSL.Connectors.PwPin p annotation(
Placement(transformation(extent = {{-80, -10}, {-60, 10}}),
    iconTransformation(extent = {{-80, -10}, {-60, 10}})));
iPSL.Connectors.PwPin n annotation(
Placement(transformation(extent = {{60, -10}, {80, 10}}), iconTransformation
    (extent = {{60, -10}, {80, 10}})));
parameter Real R "Resistance in ohms (PowerFactory) in actual units"
    annotation(
Dialog(group = "Line parameters"));
parameter Real X "Reactance in ohms (PowerFactory)in actual units"
    annotation(
Dialog(group = "Line parameters"));
parameter Real G "Conductance from PowerFactory in actual units" annotation(
Dialog(group = "Line parameters"));
parameter Real B "Susceptance from PowerFactory in actual units" annotation(
Dialog(group = "Line parameters"));
parameter Real S_b = SysData.S_b "System base power (MVA)" annotation(
Dialog(group = "Line parameters", enable = false));
parameter Real t1 = Modelica.Constants.inf annotation(
Dialog(group = "Perturbation parameters"));
parameter Real t2 = Modelica.Constants.inf annotation(
Dialog(group = "Perturbation parameters"));
Real P12;Real P21;Real Q12;Real Q21;
//Calculation of sending and recieving end voltage and currents
Complex vs(re = p.vr, im = p.vi);
Complex is(re = p.ir, im = p.ii);
Complex vr(re = n.vr, im = n.vi);
Complex ir(re = n.ir, im = n.ii);
//The inputs in Powerfactory are not in 'pu'
parameter Real V_b "base voltage";
protected
parameter Complex Y(re = G1, im = B1);
parameter Complex Z(re = R1, im = X1);
parameter Real baseZ = V_b ^ 2 / S_b;
parameter Real R1 = R / baseZ;
parameter Real X1 = X / baseZ;
parameter Real G1 = G * baseZ / 2;
parameter Real B1 = B * baseZ / 2;
equation
//Calculations for the power flow display
P12 = real(vs * conj(is)) * S_b;
P21 = -real(vr * conj(ir)) * S_b;
Q12 = imag(vs * conj(is)) * S_b;
Q21 = -imag(vr * conj(ir)) * S_b;
//PI model
(vs) - vr = Z * (is - vs * Y);
(vr) - vs = Z * (ir - vr * Y);
end PFTransmission_Line;
```

A-4 Governor

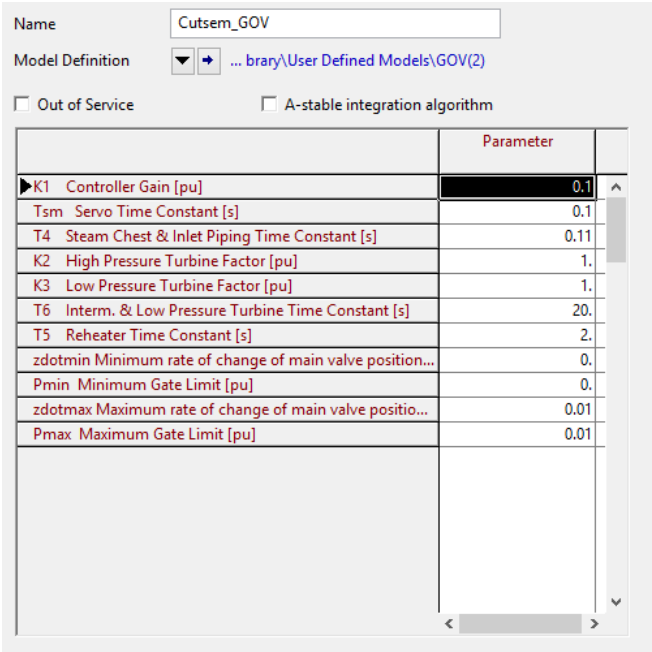


Figure A-2: Governor parameter description

A-5 Exciter

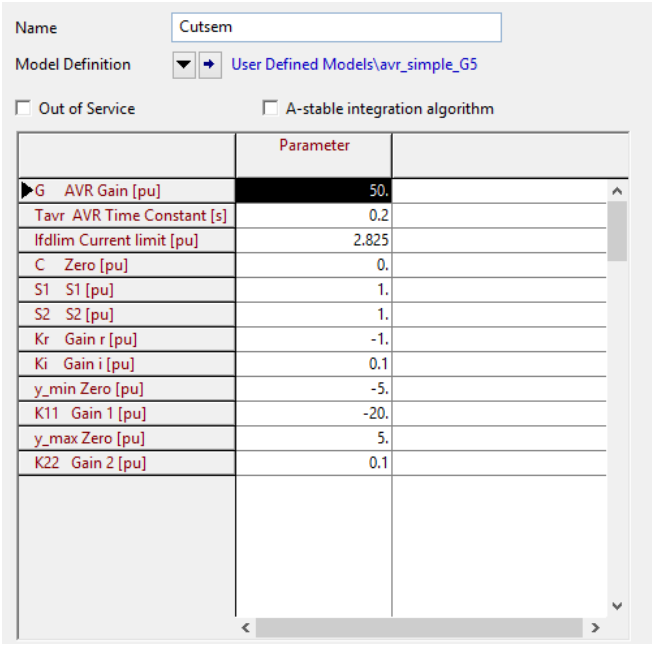


Figure A-3: Exciter parameter description

A-6 OLTC

Asymmetrical tap changer is used in this thesis. The parameters used are explained in section 3-2-9. The dialog box is found in the Load flow tab inside the 'Transformer type' as shown in A-4.

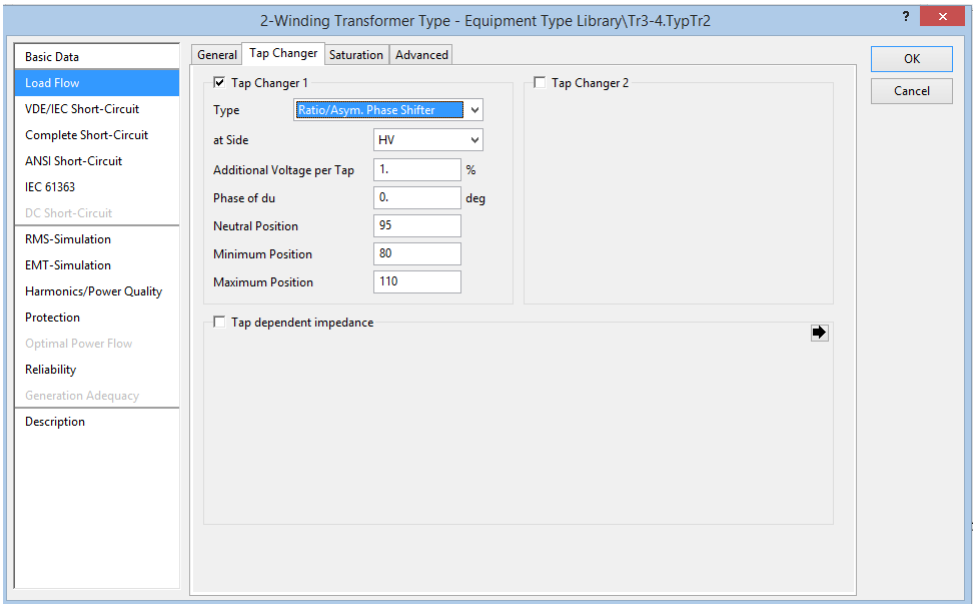


Figure A-4: off-load tap changer in Powerfactory - 1

If off-load tap changer is used, the tap position needs to be defined in the RMS-Simulation tab as shown in Figure A-5.

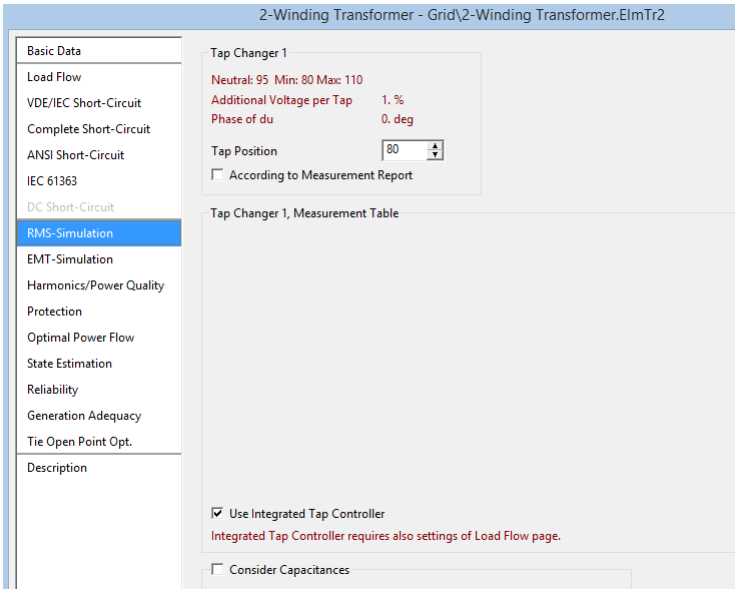


Figure A-5: off-load tap changer block in Powerfactory - 2

In order to use the on-load tap changer, the 'Use Integrated Tap Controller' box needs to be checked as shown in Figure A-5. After which the required set-points need to be filled in the Load flow tab inside the 'Transformer element' as shown in Figure A-6.

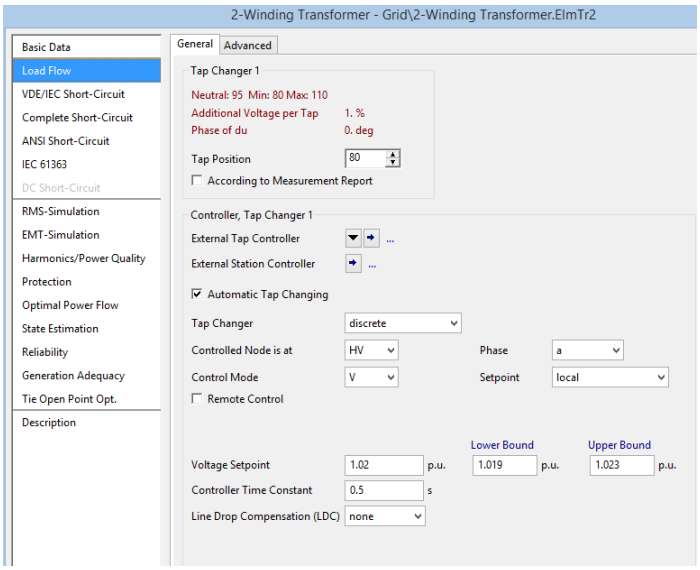


Figure A-6: on-load tap changer block in Powerfactory

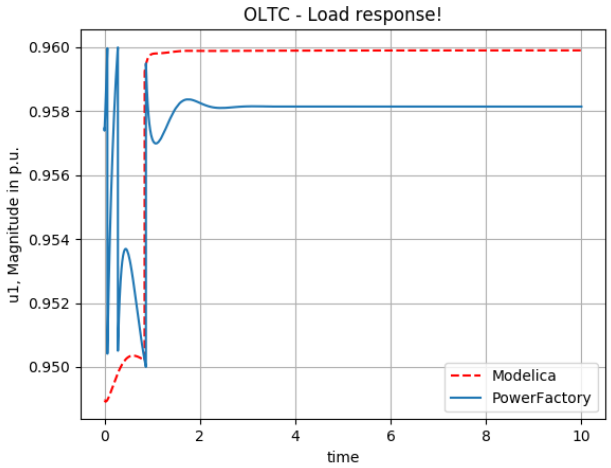


Figure A-7: Voltage at the Load when OLTC is used in OpenModelica

As evident from the Figure A-7, there is quite a large error. The test system used is same as shown in Figure 5-13. The steady state value obtained in OpenModelica after the action of OLTC is different from that of PowerFactory. There are several possible reasons for this error, this thesis doesn't use the exact solver as in PowerFactory, the algorithm used for OLTC is not straight forward in the DIgSILENT manual. Therefore, this model is made inactive in the test system and does not contribute to the analysis of the test system.

Appendix B

B-1 Grid parameters

Table B-1: Synchronous machine parameters used in the validation process

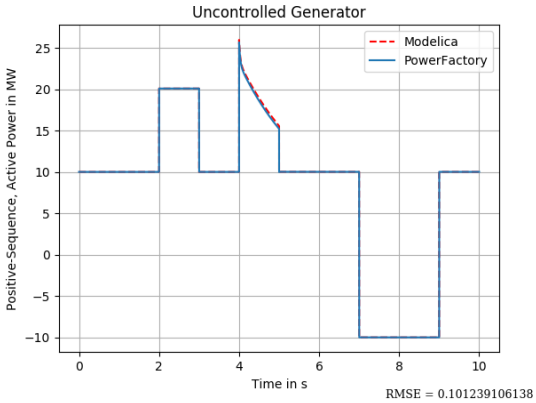
Parameter	Value
h	4.375
rstr	0.031
xl	0.2
$td0^{'}$	9.1
$tq0^{'}$	2.3
$td0^{''}$	0.03
$tq0^{''}$	0.2
$xd^{''}$	0.25
$xq^{''}$	0.256
xd	2.1
xq	2.1
$xd^{'}$	0.3
$xq^{'}$	0.73

Table B-2: Line parameters used in uncontrolled generator model

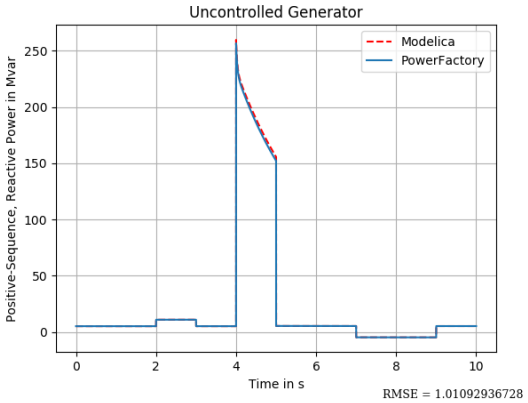
Line Parameters		
parameter	Description	Value
r	resistance	0.04 Ω/Km
x	reactance	0.4 Ω/Km
g	conductance	0.0 μS/Km
b	susceptance	125 μS/Km

B-2 Validation results of uncontrolled generator

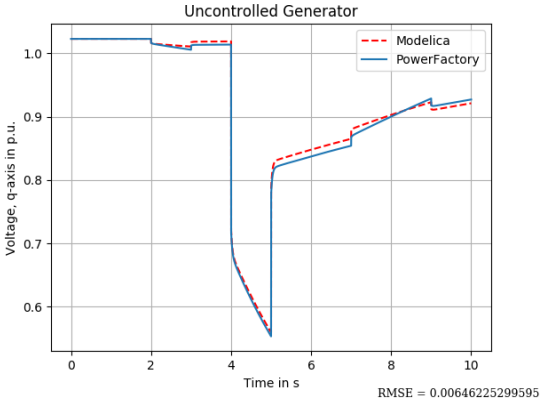
The generator parameter values used in the model are shown in Table B-1.



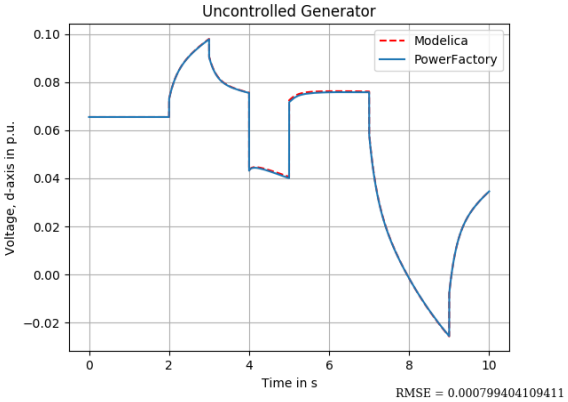
(a) Active power



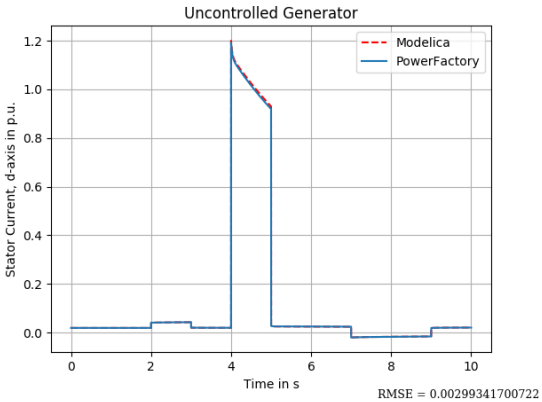
(b) Reactive Power



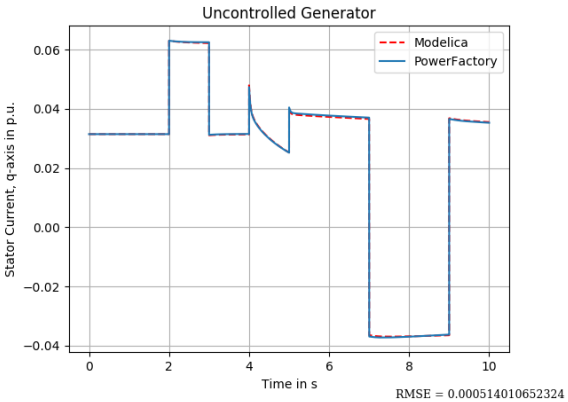
(c) d-axis voltage



(d) q-axis voltage



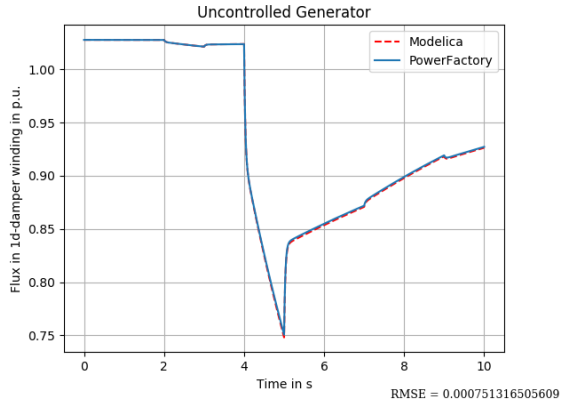
(e) d-axis current



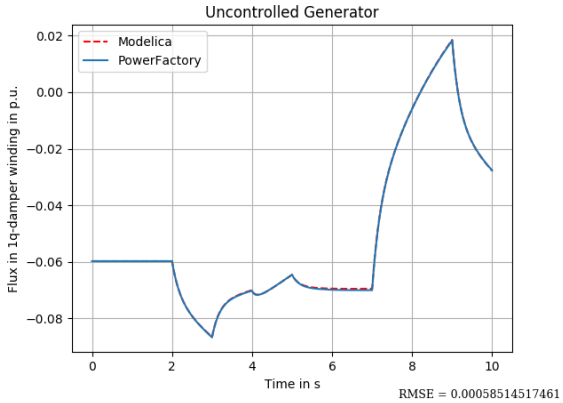
(f) q-axis current

Figure B-1: Comparison graphs for uncontrolled generator - 1

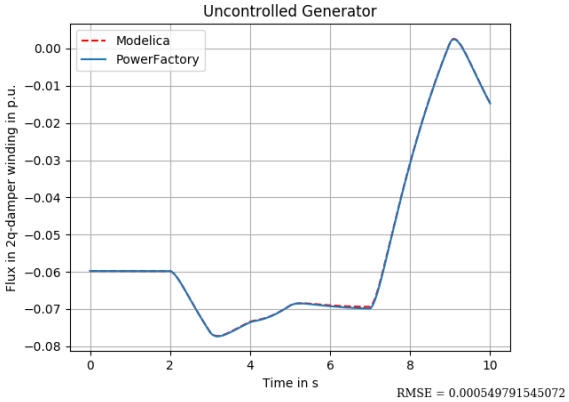
The transmission lines are 1 Km each and have the parameters as shown in Table B-2.



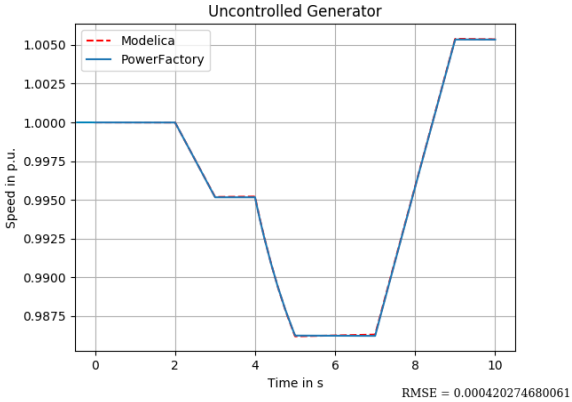
(g) Flux in 1d damper winding



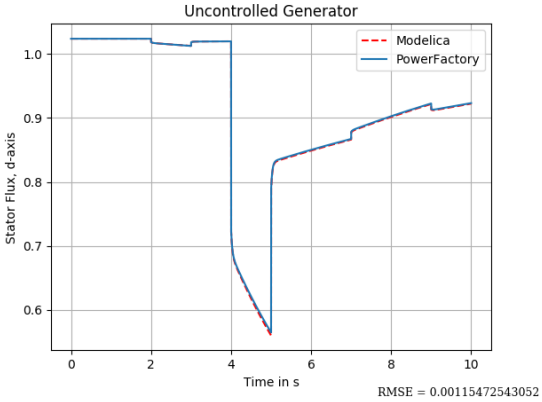
(h) Flux in 1q damper winding



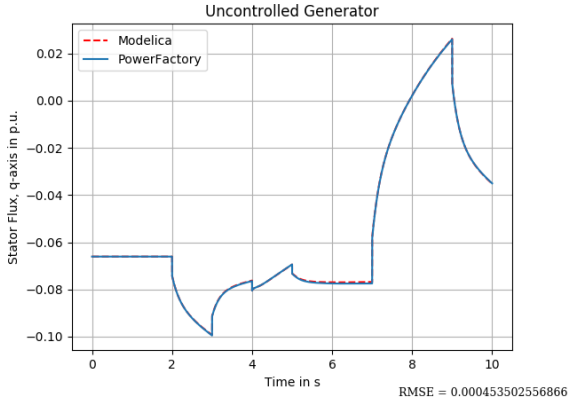
(i) Flux in 2q damper winding



(j) Speed



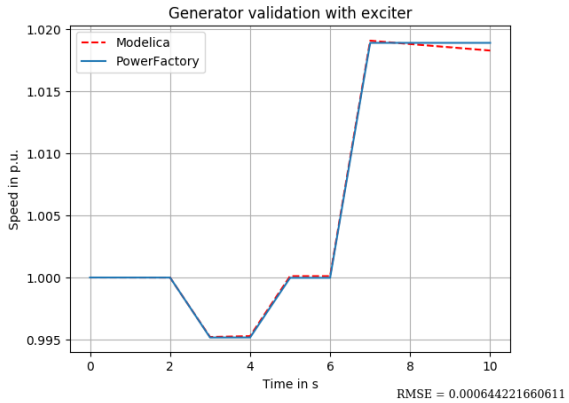
(k) Stator d-axis flux



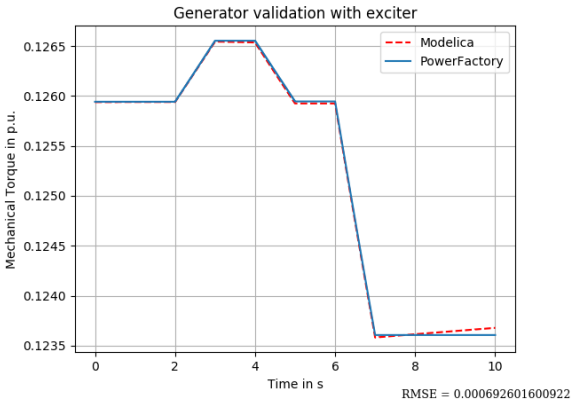
(l) Stator q-axis flux

Figure B-2: Comparison graphs for uncontrolled generator - 2

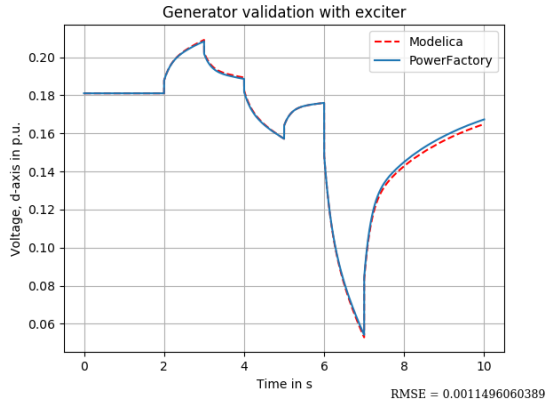
B-3 Generator validation with excitation system



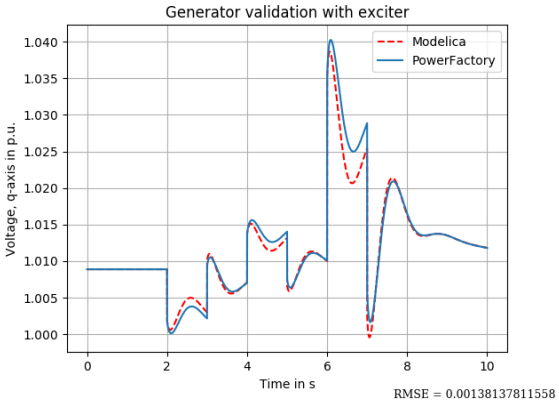
(a) speed



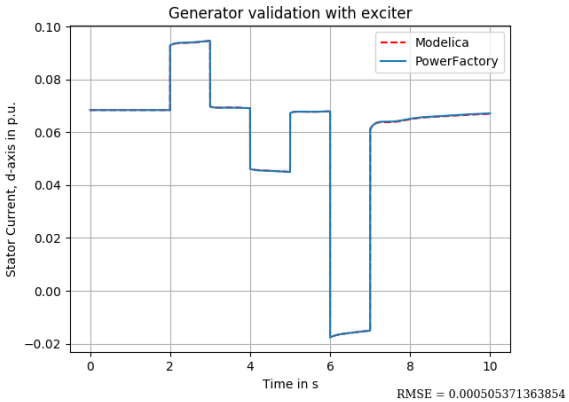
(b) current magnitude



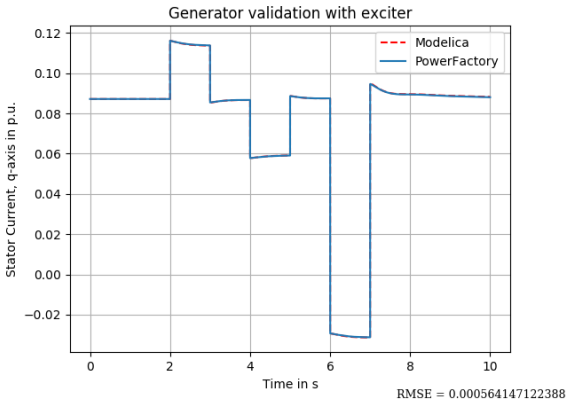
(c) d-axis voltage



(d) q-axis voltage

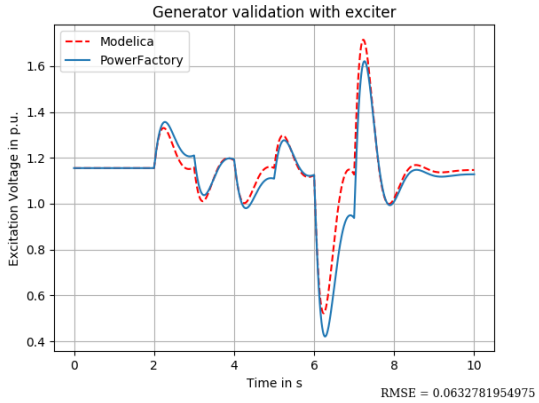


(e) d-axis current

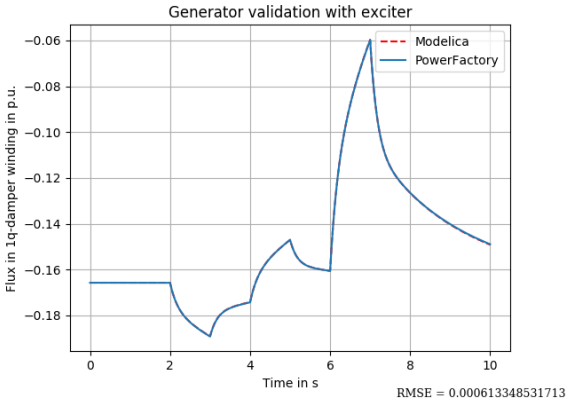


(f) d-axis current

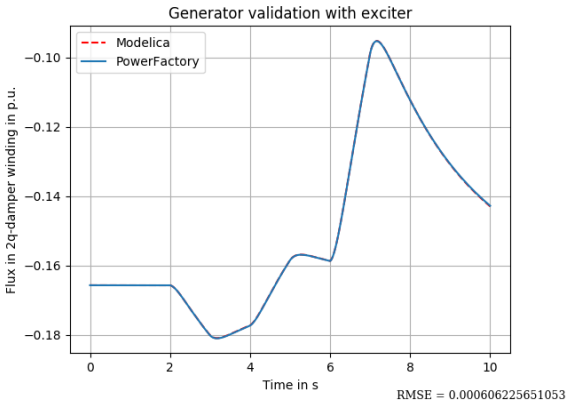
Figure B-3: Comparison graphs for a generator with excitation system-1



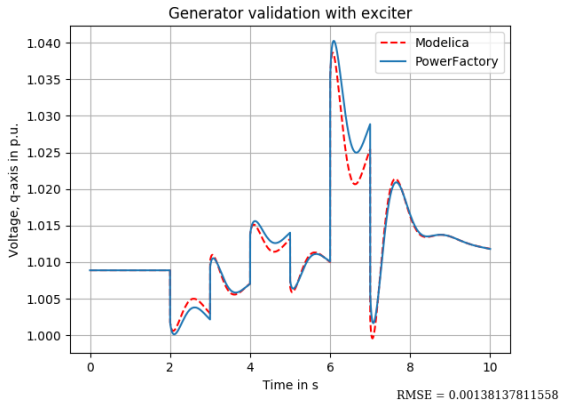
(g) excitation voltage



(h) flux in 1q damper winding



(i) flux in 2q damper winding



(j) q-axis voltage

Figure B-4: Comparison graphs for a generator with excitation system-2

Exciter block is created as shown in Figure 3-19 and connected to the generator.

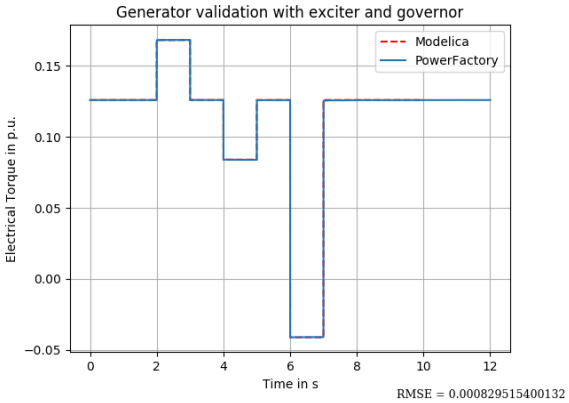
Table B-3: Exciter parameter values used for validation

Exciter system parameter values		
G	AVR gain [pu]	50
y_{min}	Minimum excitation limit [pu]	-5
y_{max}	Maximum excitation limit [pu]	2
T_{avr}	Excitation time constant [s]	0.2
K11	Lower bound of OEL timer [pu]	-20
K22	Upper bound of OEL timer [pu]	0.1
K_r	Reset constant of OEL [pu]	-1
K_i	Integral gain of OEL [pu]	0.1
I_{fdlim}	Max field current enforced by OEL [pu]	2.825

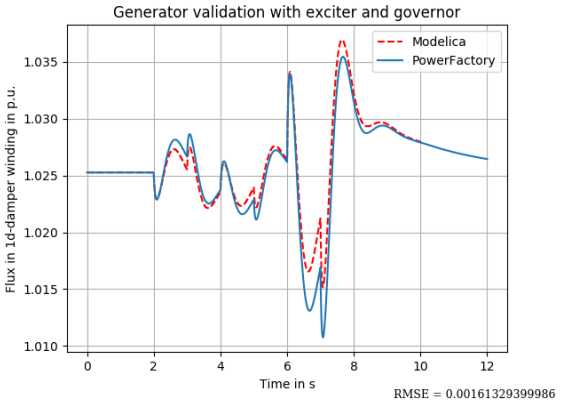
All the transmission lines are 1 km each and have the values shown in Table B-2. The powerflow values are shown within the Figure 5-6.

B-4 Generator validation with exciter and governor system

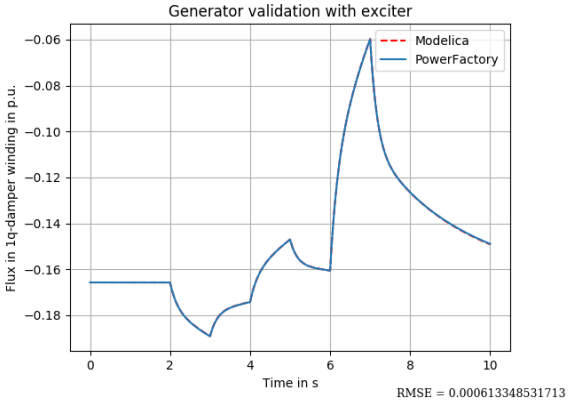
The line parameter values used in the model are shown in Table B-2. The generator parameter values used in the model are shown in Table B-1.



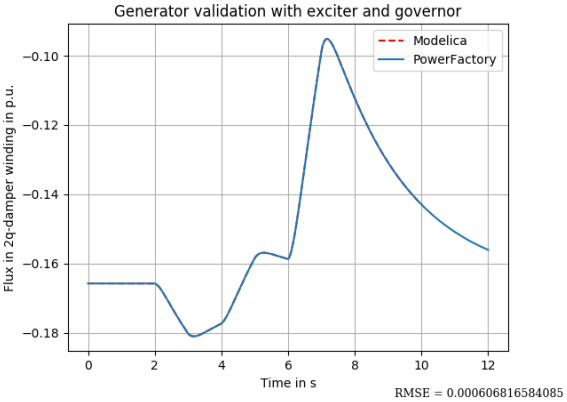
(a) electrical torque



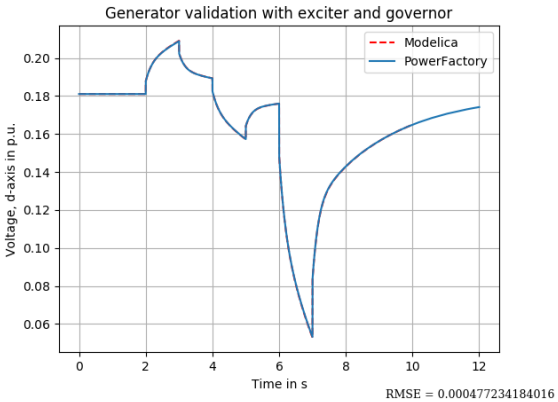
(b) flux in 1D damper winding



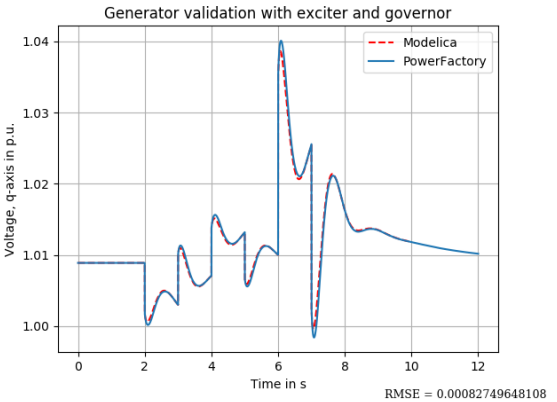
(c) flux in 1Q damper winding



(d) flux in 2Q damper winding

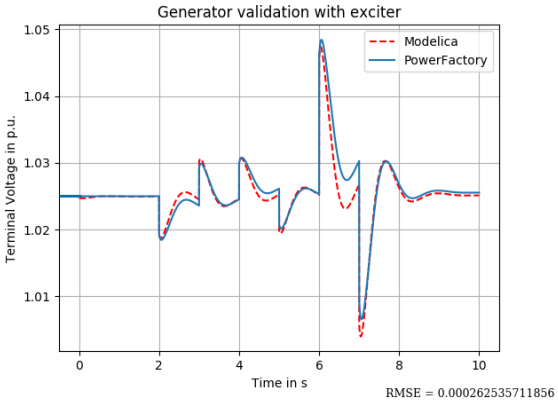


(e) d-axis voltage

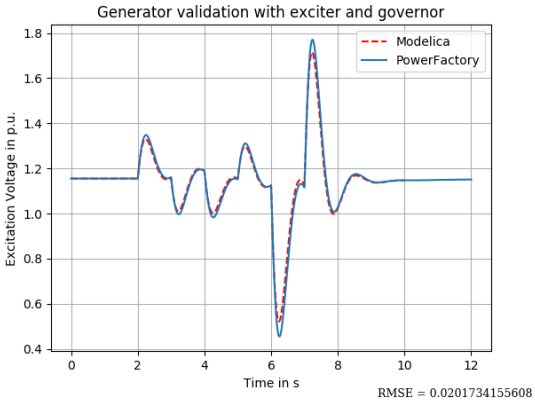


(f) q-axis voltage

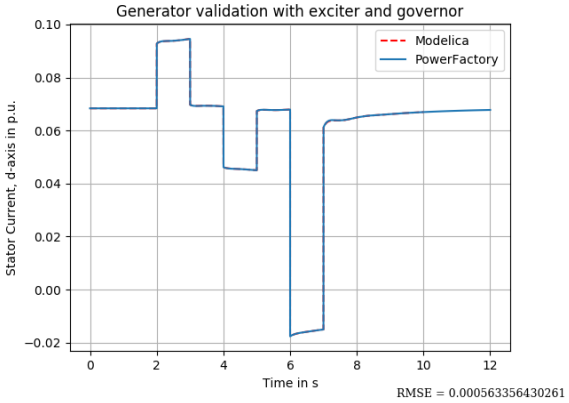
Figure B-5: Comparison graphs for a generator with excitation system and governor system-1



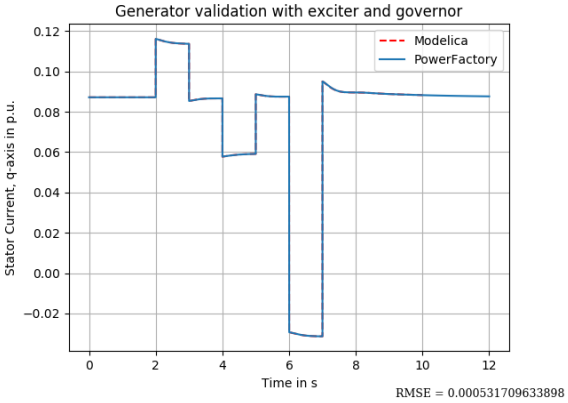
(g) terminal voltage



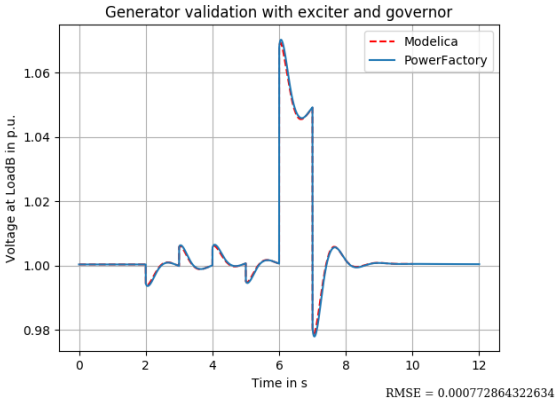
(h) excitation voltage



(i) d-axis current



(j) q-axis current



(k) voltage magnitude at Load B

Figure B-6: Comparison graphs for a generator with excitation system and governor system-2

For the system shown in Figure 5-6, a governor system block is connected and the values used for the Governor system are shown in Table B-4. The exciter values used in the validation is the same as shown in Table B-3.

Table B-4: Governor system parameter values used for validation

Governor system parameter values		
K1	Controller gain [pu]	1
T_{sm}	Servo time constant [s]	0.04
T4	Steam chest & inlet piping time constant [s]	0.1
K2	High pressure turbine factor [pu]	4.5
K3	Low pressure turbine factor [pu]	-4.5
T6	Interm & low pressure turbine time constant [s]	0.01
T5	Reheater time constant [s]	0.06
zdotmin	Minimum rate of change of main valve position	-10
zdotmax	Maximum rate of change of main valve position	10
P_{max}	Maximum gate limit [pu]	4
P_{min}	Minimum gate limit [pu]	-4

B-5 Complete Model

The OLTC and the induction machine are kept inactive in the test system, as they are not completely developed to be used for analysis.

Table B-5: Line parameters used in modified BPA system for Transmission line near the load

Line Parameters		
parameter	Description	Value
r	resistance	0 Ω/Km
x	reactance	0.05776 Ω/Km
g	conductance	0.0 µS/Km
b	susceptance	0 µS/Km
l	length	100 Km

Table B-6: Line parameters used in both of the parallel transmission lines

Line Parameters		
parameter	Description	Value
r	resistance	0 Ω/Km
x	reactance	0.798 Ω/Km
g	conductance	0.0 µS/Km
b	susceptance	0 µS/Km
l	length	100 Km

Table B-7: Exciter parameter values used for validation of modified BPA system

Exciter system parameter values		
G	AVR gain [pu]	50
y_{min}	Minimum excitation limit [pu]	-5
y_{max}	Maximum excitation limit [pu]	5
T_{avr}	Excitation time constant [s]	0.2
K11	Lower bound of OEL timer [pu]	-20
K22	Upper bound of OEL timer [pu]	0.1
K_r	Reset constant of OEL [pu]	-1
K_i	Integral gain of OEL [pu]	0.1
I_{fdlim}	Max field current enforced by OEL [pu]	2.825

Table B-8: Governor system parameter values used for validation for BPA test system

Governor system parameter values		
K1	Controller gain [pu]	129.5
T_{sm}	Servo time constant [s]	0.04
T4	Steam chest & inlet piping time constant [s]	0.1
K2	High pressure turbine factor [pu]	4.5
K3	Low pressure turbine factor [pu]	-4.5
T6	Interm & low pressure turbine time constant [s]	0.01
T5	Reheater time constant [s]	0.06
zdotmin	Minimum rate of change of main valve position	-10
zdotmax	Maximum rate of change of main valve position	10
P_{max}	Maximum gate limit [pu]	4
P_{min}	Minimum gate limit [pu]	-4

Table B-9: Tranformer near G1

Transformer Parameters		
parameter	Description	Value
xT	Transformer reactance (pu)	0.08
rT	Transformer resistance (pu)	0
Vbus1	Sending end Bus nominal voltage (KV)	20000
Vbus2	Receiving end Bus nominal voltage (KV)	380000
S	Power rating (MVA)	200

Appendix C

C-1 CGMES based Initialization

The python based converter was used to verify IEEE 14 bus system (Figure. C-1).

```
Mapping the Generator buses with the name of the generator....$
-------------------------
('the map btw gen and genbus is', {'Gen_0006': 'Bus_0006', 'Gen_0001': 'Bus_0001', 'Gen_0008': 'Bus_0008',
'Gen_0003': 'Bus_0003', 'Gen_0002': 'Bus_0002'})
('the map btw gen and genbus (TUPLE) is', [['Gen_0006', 'Bus_0006'], ['Gen_0003', 'Bus_0003'], ['Gen_0008',
'Bus_0008'], ['Gen_0002', 'Bus_0002'], ['Gen_0001', 'Bus_0001']])
-------------------------
-------------------------
/

Mapping the Load buses with the name of the Load....$
-------------------------
('the map btw load and loadbus is', {'Load_0009': 'Bus_0009', 'Load_0011': 'Bus_0011', 'Load_0003':
'Bus_0003', 'Load_0013': 'Bus_0013', 'Load_0012': 'Bus_0012', 'Load_0006': 'Bus_0006', 'Load_0014':
'Bus_0014', 'Load_0004': 'Bus_0004', 'Load_0005': 'Bus_0005', 'Load_0002': 'Bus_0002', 'Load_0010':
'Bus_0010'})
('the map btw load and loadbus (TUPLE) is', [['Load_0011', 'Bus_0011'], ['Load_0012', 'Bus_0012'],
['Load_0006', 'Bus_0006'], ['Load_0004', 'Bus_0004'], ['Load_0003', 'Bus_0003'], ['Load_0009', 'Bus_0009'],
['Load_0013', 'Bus_0013'], ['Load_0010', 'Bus_0010'], ['Load_0005', 'Bus_0005'], ['Load_0014', 'Bus_0014'],
['Load_0002', 'Bus_0002']])
the total no. of buses is:  14
the bus details are[('Bus_0007', '1.06195', '-13.3682'), ('Bus_0011', '34.8837', '-14.7953'), ('Bus_0012',
'34.8223', '-15.0774'), ('Bus_0006', '35.31', '-14.2226'), ('Bus_0004', '134.458', '-10.3242'), ('Bus_0003',
'133.32', '-12.718'), ('Bus_0009', '34.8594', '-14.9466'), ('Bus_0013', '34.6646', '-15.1589'), ('Bus_0008',
'11.99', '-13.3682'), ('Bus_0010', '34.6938', '-15.1043'), ('Bus_0005', '134.675', '-8.78257'), ('Bus_0014',
'34.1813', '-16.0389'), ('Bus_0002', '137.94', '-4.98095'), ('Bus_0001', '139.92', '0.')]
1.06195
$$$$$$$$$$$$$$$$$$$$  Scripting for Modelica begins.......$$$$$$$$$$$$$$$$$$
--------------
-----------
-------!
PowerFlow values of Gen1 is [('Gen_0006', 0.000289681, 12.24, '35.31', '-14.2226'), ('Gen_0002', 40.0001,
42.3961, '137.94', '-4.98095'), ('Gen_0008', 0.000134477, 17.3565, '11.99', '-13.3682'), ('Gen_0001',
232.386, -16.889, '139.92', '0.'), ('Gen_0003', 0.000133661, 23.3934, '133.32', '-12.718')]

Process finished with exit code 0
```

Figure C-1: Generator output obtained from the python based CGMES reader for IEEE 14 bus system

C-2 Model to model transformation

Listing C.1: Python script to generate the Modelica file

```
#################################################################
import fileinput
import sys
print 'WRITING THE .mo FILE.....................'
print '.....................................\n'
print '_____________________________________'
datafile = open('newfile.mo', 'w')
datafile.write('model Test_system "Developed during a master thesis project
    at TU Delft"')
datafile.write('\n')
datafile.write('OpenIPSL.Electrical.SystemBase SysData annotation(Placement(
    visible = true, transformation(origin = {-2, -7}, extent = {{-1, -1},
    {1, 1}}, rotation = 0)));')
datafile.write('\n')
#write the buses
for i in range(0,len(BUS_Axes)):
if i%5==0:
datafile.write('OpenIPSL.Electrical.Buses.Bus '+ BUS_Axes[i]+' annotation(')
datafile.write('\n')
if i%5==1:
datafile.write('Placement(visible=true, transformation(origin={'+str(float(
    BUS_Axes[i])/10))
if i%5==3:
datafile.write(','+str(float(BUS_Axes[i])/10)+'},extent = {{-1, -1}, {1,
    1}}, rotation = 0)));'+'\n')
#Write the slack generator
GenName=SlackGenData[0]
S=SlackGenData[1]
U=SlackGenData[2]
cosn=SlackGenData[3]
for i in range(len(Total_Gen_PF)):
a=Total_Gen_PF[i]
if GenName==a[0]:
P=a[1]
Q=a[2]
V=a[3]
An=a[4]
print GEN_Axes_tuple
for item in range(len(GEN_Axes_tuple)):
a1=GEN_Axes_tuple[item]
if GenName==a1[0]:
axis1=a1[1]
axis2=a1[2]
for j in range(len(Gen_Dynamics)):
b=Gen_Dynamics[j]
if b[1]==GenName:
h=b[2]
xl=b[3]
rstr=b[4]
td01=b[5]
tq01=b[6]
td011=b[7]
tq011=b[8]
xd11=b[9]
xq11=b[10]
xd=b[11]
```

```
xq=b[12]
xd1=b[13]
xq1=b[14]
datafile.write('OpenIPSL.Multiple.PowerFactory_6orderX ' + GenName+'(h='+h+'
    ,xq1='+xq1+',xd1='+xd1+',xq='+xq+',xd='+xd+',xq11='+xq11+',xd11='+xd11+'
    ,tq011='+tq011+',xl='+xl+',td011='+td011+',tq01='+tq01+',td01='+td01 +',
    rstr='+rstr+',V_b='+str(U)+',V_0='+str(V)+',angle_0='+str(An)+',P_0='+
    str(P)+',Q_0='+str(Q)+',cosn='+cosn+',M_b='+S+')' + ' annotation(')    #
            datafile.write('\n')
datafile.write('\n')
datafile.write('Placement(visible=true, transformation(origin={' + str(float
    (axis1)/10))
datafile.write(',' + str(float(axis2)/10) + '},extent = {{-1, -1}, {1, 1}},
    rotation = 0)));' + '\n')
#Write remaining generators.....
for item in range(len(Remaining_Gen_EQ)):
a=Remaining_Gen_EQ[item]
GenName=a[0]
S=a[1]
U=a[2]
cosn=a[3]
for i in range(len(Total_Gen_PF)):
b=Total_Gen_PF[i]
if GenName == b[0]:
print b[0]
P = b[1]
Q = b[2]
V = b[3]
An = b[4]
for item1 in range(len(GEN_Axes_tuple)):
a1 = GEN_Axes_tuple[item1]
if GenName == a1[0]:
axis1 = a1[1]
axis2 = a1[2]
X=(GenName,S,U,cosn,P,Q,V,An)
print ('the X is',X)
for j in range(len(Gen_Dynamics)):
b = Gen_Dynamics[j]
if b[1] == GenName:
h = b[2]
xl = b[3]
rstr = b[4]
td01 = b[5]
tq01 = b[6]
td011 = b[7]
tq011 = b[8]
xd11 = b[9]
xq11 = b[10]
xd = b[11]
xq = b[12]
xd1 = b[13]
xq1 = b[14]
datafile.write('OpenIPSL.Multiple.PowerFactory_6order ' + GenName +'(h='+h+'
    ,xq1='+xq1+',xd1='+xd1+',xq='+xq+',xd='+xd+',xq11='+xq11+',xd11='+xd11+'
    ,tq011='+tq011+',xl='+xl+',td011='+td011+',tq01='+tq01+',td01='+td01 +',
    rstr='+rstr+',V_b=' + str(U) + ',V_0=' + str(V) + ',angle_0=' + str(An)
    + ',P_0=' + str(P) + ',Q_0=' + str(Q) + ',cosn=' + cosn + ',M_b=' + S +
    ')' + ' annotation(')  # datafile.write('\n')
datafile.write('\n')
```

```
datafile.write('Placement(visible=true, transformation(origin={' + str(float
    (axis1) / 10))
datafile.write(',' + str(float(axis2) / 10) + '},extent = {{-1, -1}, {1,
    1}}, rotation = 0)));' + '\n')
#write the loads (for static loads:IEEE 9bus system)
for item in range(len(Total_Load_details)):
a = Total_Load_details[item]
name=a[0]
vb=a[5]
p=a[1]
q=a[2]
vo=a[3]
an=a[4]
axis1=a[6]
axis2=a[7]
datafile.write('OpenIPSL.TestFiles.Load_CGMES ' + name + '(V_b='+str(vb)+',
    V_0='+str(vo)+',angle_0='+str(an)+',P_0='+str(p)+',Q_0='+str(q)+')'+'
    annotation(')
datafile.write('\n')
datafile.write('Placement(visible=true, transformation(origin={' + str(float
    (axis1)/10))
datafile.write(',' + str(float(axis2)/10) + '},extent = {{-1, -1}, {1, 1}},
    rotation = 0)));' + '\n')
#write the Transmission lines
#(B = 0.0005 / 2, G = 0, R = 0.01, X = 0.1)
TL_Axes=[]
for i in range(0,len(Total_AC_Line_Segment)):
if i%9==2:
Name_of_the_line=Total_AC_Line_Segment[i]
Vbase=Total_AC_Line_Segment[i+1]
r=Total_AC_Line_Segment[i+3]
x=Total_AC_Line_Segment[i+4]
g=Total_AC_Line_Segment[i+5]
b=Total_AC_Line_Segment[i+6]
datafile.write('OpenIPSL.TestFiles.PFTransmission_Line '+Name_of_the_line+'(
    B='+b+',V_b='+Vbase+',G='+g+',R='+r+',X='+x+')'+' annotation('+'\n')
#         datafile.write('OpenIPSL.PowerFactory_models_steadyS.
    Transmission_Line ' + Name_of_the_line+';'+'\n')
BUS1=Total_AC_Line_Segment[i-2]
BUS2=Total_AC_Line_Segment[i-1]
for j in BUS_Axes:
index1=BUS_Axes.index(BUS1)
x1=BUS_Axes[index1+1]
y1=BUS_Axes[index1+3]
index2 = BUS_Axes.index(BUS2)
x2 = BUS_Axes[index2 + 1]
y2 = BUS_Axes[index2 + 3]
datafile.write('Placement(visible=true, transformation(origin={'+str((float(
    x1)+float(x2))/20)+','+str((float(y2)+float(y1))/20)+'},extent={{-0.3,
    -0.2}, {0.3, 0.2}}, rotation=0)));'+'\n')
TL_Axes.append(Name_of_the_line)
X=(float(x1)+float(x2))/20
Y=(float(y1)+float(y2))/20
TL_Axes.append(X)
TL_Axes.append(Y)
#write Transformers.....
for i in range(len(Tf_full)):
a=Tf_full[i]
V1=float(a[2])
V2=float(a[6])
```

```
n=(V1/V2)*(V1/V2)
print n
r=float(a[3])
x=float(a[4])
print x
rt=str(r/n)
xt=str(x/n)
print xt
for j in range(len(Tranformer_values1)):
b1=Tranformer_values1[j]
if a[0]==b1[0]:
Sn=b1[1]
for k in range(len(TF_AXES)):
a1=TF_AXES[k]
if a1[0]==a[0]:
x1=a1[1]
y1=a1[2]
datafile.write('OpenIPSL.TestFiles.Transformer '+a[0]+'(Vbus1='+a[6]+',Sn='+
    Sn+',Vn='+a[6]+',Vbus2='+a[2]+',rT='+rt+',xT='+xt+') '+' annotation('+'\
    n')
datafile.write('Placement(visible = true, transformation(origin = {'+str(
    float(x1)/10)+','+str(float(y1)/10)+'}, extent = {{-1, -1}, {1, 1}},
    rotation = 0)));'+'\n')
datafile.write('equation')
datafile.write('\n')
#Connect the transformer to the buses
for item in range(len(TF_BUS)):
a=TF_BUS[item]
name=a[0]
BUS1 = a[1]
BUS2 = a[2]
datafile.write('connect(' + a[0] + '.n, ' + BUS1 + '.p) annotation(' + '\n')
for i in range(len(TF_AXES)):
tff = TF_AXES[i]
if tff[0]==a[0]:
x1 = tff[1]
y1 = tff[2]
#print(x1,y1)
for j in BUS_Axes:
index1 = BUS_Axes.index(BUS1)
x2 = BUS_Axes[index1 + 1]
y2 = BUS_Axes[index1 + 3]
datafile.write('Line(points = {{' + str(float(x1) / 10) + ',' + str(float(y1
    ) / 10) + '}, {' + str(float(x2) / 10) + ',' + str(float(y2) / 10) + '
    }}, color = {0, 0, 255}));' + '\n')
datafile.write('connect(' + a[0] + '.p, ' + BUS2 + '.p) annotation(' + '\n')
for i in range(len(TF_AXES)):
tff = TF_AXES[i]
if tff[0] == a[0]:
x1 = tff[1]
y1 = tff[2]
for j in BUS_Axes:
index1 = BUS_Axes.index(BUS2)
x2 = BUS_Axes[index1 + 1]
y2 = BUS_Axes[index1 + 3]
datafile.write(
'Line(points = {{' + str(float(x1) / 10) + ',' + str(float(y1) / 10) + '}, {
    ' + str(float(x2) / 10) + ',' + str(
float(y2) / 10) + '}}, color = {0, 0, 255}));' + '\n')
#Connecting the loads to the buses
```

```
#print Map_Load_BusName_Tuple
for item in range(len(Map_Load_BusName_Tuple)):
a=Map_Load_BusName_Tuple[item]
datafile.write('connect('+a[0]+'.p, '+a[1]+'.p) annotation('+'\n')
index_of_load=LOAD_Axes.index(a[0])
cx=LOAD_Axes[index_of_load+1]
cy=LOAD_Axes[index_of_load+2]
index_of_bus = BUS_Axes.index(a[1])
dx=BUS_Axes[index_of_bus+1]
dy=BUS_Axes[index_of_bus+3]
datafile.write('Line(points = {{'+str(float(cx)/10)+','+str(float(cy)/10)+'
    }, {'+str(float(dx)/10)+','+ str(float(dy)/10)+'}}, color = {0, 0, 255})
    );'+'\n')

#Connecting the generators to the buses
#print Map_Gen_BusName_Tuple
for item in range(len(Map_Gen_BusName_Tuple)):
a=Map_Gen_BusName_Tuple[item]
datafile.write('connect(' + a[0] + '.p, ' + a[1] + '.p) annotation(' + '\n')
index_of_load = GEN_Axes.index(a[0])
cx=GEN_Axes[index_of_load+1]
cy=GEN_Axes[index_of_load+2]
index_of_bus = BUS_Axes.index(a[1])
dx = BUS_Axes[index_of_bus + 1]
dy = BUS_Axes[index_of_bus + 3]
datafile.write('Line(points = {{' + str(float(cx) / 10) + ',' + str(float(cy
    ) / 10) + '}, {' + str(float(dx) / 10) + ',' + str(float(dy) / 10) + '
    }}, color = {0, 0, 255}));' + '\n')

#Connecting the lines to the buses.
#print Total_AC_Line_Segment
for i in range(0,len(Total_AC_Line_Segment)):
if i%9==2:
Name_of_the_line=Total_AC_Line_Segment[i]
BUS1=Total_AC_Line_Segment[i-2]

datafile.write('connect(' + Name_of_the_line + '.n, ' + BUS1 + '.p)
    annotation(' + '\n')
index_of_line=TL_Axes.index(Name_of_the_line)
x1=TL_Axes[index_of_line+1]
y1=TL_Axes[index_of_line+2]
index_of_bus = BUS_Axes.index(BUS1)
x2 = BUS_Axes[index_of_bus + 1]
y2 = BUS_Axes[index_of_bus + 3]
datafile.write('Line(points = {{' + str(float(x1)) + ',' + str(float(y1)) +
    '}, {' + str(float(x2) / 10) + ',' + str(float(y2) / 10) + '}}, color =
    {0, 0, 255}));' + '\n')

BUS2=Total_AC_Line_Segment[i-1]
datafile.write('connect(' + Name_of_the_line + '.p, ' + BUS2 + '.p)
    annotation(' + '\n')
index_of_line = TL_Axes.index(Name_of_the_line)
x1 = TL_Axes[index_of_line + 1]
y1 = TL_Axes[index_of_line + 2]
index_of_bus = BUS_Axes.index(BUS2)
x2 = BUS_Axes[index_of_bus + 1]
y2 = BUS_Axes[index_of_bus + 3]
datafile.write('Line(points = {{' + str(float(x1)) + ',' + str(float(y1)) +
    '}, {' + str(float(x2) / 10) + ',' + str(float(y2) / 10) + '}}, color =
    {0, 0, 255}));' + '\n')
```

```
datafile.write('end Test_system;')
datafile.close()
print '_____________________________________'
print '_____________________________________'
print 'FILE Writing finished.....................'
print '.....................................\n'
```

Listing C.2: The corresponding Modelica script for the test system generated

```
\model Test_system "Developed during a master thesis project at TU Delft and
    TenneT"
OpenIPSL.Electrical.SystemBase SysData annotation(
Placement(visible = true, transformation(origin = {26, 17}, extent = {{-1,
    -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS2 annotation(
Placement(visible = true, transformation(origin = {17.9375, 7.0}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus Bus8 annotation(
Placement(visible = true, transformation(origin = {17.0625, 10.9375}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS5 annotation(
Placement(visible = true, transformation(origin = {11.375, 10.9375}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS3 annotation(
Placement(visible = true, transformation(origin = {17.0625, 17.9375}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus Bus9 annotation(
Placement(visible = true, transformation(origin = {17.0625, 15.3125}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS4 annotation(
Placement(visible = true, transformation(origin = {8.75, 16.1875}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS1 annotation(
Placement(visible = true, transformation(origin = {5.25, 16.1875}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus BUS6 annotation(
Placement(visible = true, transformation(origin = {12.25, 16.1875}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Electrical.Buses.Bus Bus7 annotation(
Placement(visible = true, transformation(origin = {17.0625, 8.3125}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Multiple.PowerFactory_6orderX G1(M_b = 500., P_0 = 10.0019, Q_0 =
    -42.8841, V_0 = 1.04, V_b = 20., angle_0 = 0.0, cosn = 1., h = 3.5, rstr
    = 0.031, td01 = 9.1, td011 = 0.03, tq01 = 2.3, tq011 = 0.2, xd = 2.1,
    xd1 = 0.3, xd11 = 0.25, xl = 0.2, xq = 2.1, xq1 = 0.73, xq11 = 0.256)
    annotation(
Placement(visible = true, transformation(origin = {3.9375, 16.25}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Multiple.PowerFactory_6order G2(h = 3.5, xq1 = 0.73, xd1 = 0.3, xq
    = 2.1, xd = 2.1, xq11 = 0.256, xd11 = 0.25, tq011 = 0.2, xl = 0.2, td011
    = 0.03, tq01 = 2.3, td01 = 9.1, rstr = 0.031, V_b = 20., V_0 = 1.04416,
    angle_0 = 0.0450432, P_0 = 20.0, Q_0 = 20.0, cosn = 0.85, M_b = 250.)
    annotation(
Placement(visible = true, transformation(origin = {21.0, 6.5625}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.Multiple.PowerFactory_6order G3(h = 3.5, xq1 = 0.73, xd1 = 0.3, xq
    = 2.1, xd = 2.1, xq11 = 0.256, xd11 = 0.25, tq011 = 0.2, xl = 0.2, td011
    = 0.03, tq01 = 2.3, td01 = 9.1, rstr = 0.031, V_b = 20., V_0 = 1.04465,
    angle_0 = 0.0727758, P_0 = 20.0, Q_0 = 20.0, cosn = 0.85, M_b = 250.)
    annotation(
```

```
Placement(visible = true, transformation(origin = {13.125, 18.8125}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Load_CGMES LOADB(V_b = 230., V_0 = 1.04263043478, angle_0
    = -0.0318971, P_0 = 10.0, Q_0 = 10.0) annotation(
Placement(visible = true, transformation(origin = {14.4375, 16.625}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Load_CGMES LOADC(P_0 = 10.0, Q_0 = 10.0, V_0 =
    1.0427826087, V_b = 230., angle_0 = -0.0314053, dP1 = 10.0, dQ1 = 10.0,
    t_end_1 = 7, t_start_1 = 5) annotation(
Placement(visible = true, transformation(origin = {19.6875, 12.6875}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Load_CGMES LOADA(V_b = 230., V_0 = 1.04256956522, angle_0
    = -0.0369459, P_0 = 30.0, Q_0 = 20.0) annotation(
Placement(visible = true, transformation(origin = {12.6875, 10.9375}, extent
    = {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L57(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {14.21875, 9.625}, extent
    = {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L46(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {10.5, 16.1875}, extent =
    {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L89(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {17.0625, 13.125}, extent
    = {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L78(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {17.0625, 9.625}, extent =
    {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L45(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {10.0625, 13.5625}, extent
    = {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.PFTransmission_Line L69(B = 0.000125, V_b = 230., G = 0.,
    R = 0.04, X = 0.4) annotation(
Placement(visible = true, transformation(origin = {14.65625, 15.75}, extent
    = {{-0.3, -0.2}, {0.3, 0.2}}, rotation = 0)));
OpenIPSL.TestFiles.Transformer T1(Sn = 250., Vbus1 = 20., Vbus2 = 230., Vn =
    20., rT = 0.0, xT = 0.015) annotation(
Placement(visible = true, transformation(origin = {7, 16.6875}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Transformer T3(Sn = 150., Vbus1 = 20., Vbus2 = 230., Vn =
    20., rT = 0.0, xT = 0.0144) annotation(
Placement(visible = true, transformation(origin = {18.375, 16.625}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Transformer T2(Sn = 200., Vbus1 = 20., Vbus2 = 230., Vn =
    20., rT = 0.0, xT = 0.0144) annotation(
Placement(visible = true, transformation(origin = {16.1875, 6.125}, extent =
    {{-1, -1}, {1, 1}}, rotation = 0)));
OpenIPSL.TestFiles.Exciter_ST1A_mod exciter_ST1A_mod1;
equation
connect(G1.n, G3.fref) annotation(
Line);
connect(G1.n, G2.fref) annotation(
Line);
connect(G1.pt0, G1.pt) annotation(
Line);
connect(G1.p, BUS1.p) annotation(
```

```
Line(points = {{5, 16}, {5, 16.1875}, {5.25, 16.1875}}, color = {0, 0, 255})
    );
connect(exciter_ST1A_mod1.vee, G1.ve) annotation(
Line);
connect(G1.ve0, exciter_ST1A_mod1.VEE) annotation(
Line);
connect(G1.ut0, exciter_ST1A_mod1.UTT) annotation(
Line);
connect(G1.Vt, exciter_ST1A_mod1.utt) annotation(
Line);
connect(G1.ie, exciter_ST1A_mod1.ie) annotation(
Line);
connect(L46.n, BUS4.p) annotation(
Line(points = {{11, 16}, {9.625, 16}, {9.625, 16.1875}, {8.75, 16.1875}},
    color = {0, 0, 255}));
connect(L46.p, BUS6.p) annotation(
Line(points = {{10, 16}, {11.375, 16}, {11.375, 16.1875}, {12.25, 16.1875}},
     color = {0, 0, 255}));
connect(T1.n, BUS4.p) annotation(
Line(points = {{8, 17}, {8, 16.1875}, {8.75, 16.1875}}, color = {0, 0, 255})
    );
connect(T1.p, BUS1.p) annotation(
Line(points = {{6, 17}, {6, 16.1875}, {5.25, 16.1875}}, color = {0, 0, 255})
    );
connect(G2.pt0, G2.pt);
connect(G2.ve, G2.ve0);
connect(G3.pt0, G3.pt);
connect(G3.ve0, G3.ve);
connect(T3.n, Bus9.p) annotation(
Line(points = {{18.375, 16.625}, {17.0625, 15.3125}}, color = {0, 0, 255}));
connect(T3.p, BUS3.p) annotation(
Line(points = {{18.375, 16.625}, {17.0625, 17.9375}}, color = {0, 0, 255}));
connect(T2.n, Bus7.p) annotation(
Line(points = {{16.1875, 6.125}, {17.0625, 8.3125}}, color = {0, 0, 255}));
connect(T2.p, BUS2.p) annotation(
Line(points = {{16.1875, 6.125}, {17.9375, 7.0}}, color = {0, 0, 255}));
connect(LOADC.p, Bus8.p) annotation(
Line(points = {{19.6875, 12.6875}, {17.0625, 10.9375}}, color = {0, 0, 255})
    );
connect(LOADA.p, BUS5.p) annotation(
Line(points = {{12.6875, 10.9375}, {11.375, 10.9375}}, color = {0, 0, 255}))
    ;
connect(LOADB.p, BUS6.p) annotation(
Line(points = {{14.4375, 16.625}, {12.25, 16.1875}}, color = {0, 0, 255}));
connect(G2.p, BUS2.p) annotation(
Line(points = {{21.0, 6.5625}, {17.9375, 7.0}}, color = {0, 0, 255}));
connect(G3.p, BUS3.p) annotation(
Line(points = {{13.125, 18.8125}, {17.0625, 17.9375}}, color = {0, 0, 255}))
    ;
connect(L57.n, BUS5.p) annotation(
Line(points = {{14.21875, 9.625}, {11.375, 10.9375}}, color = {0, 0, 255}));
connect(L57.p, Bus7.p) annotation(
Line(points = {{14.21875, 9.625}, {17.0625, 8.3125}}, color = {0, 0, 255}));
connect(L89.n, Bus8.p) annotation(
Line(points = {{17.0625, 13.125}, {17.0625, 10.9375}}, color = {0, 0, 255}))
    ;
connect(L89.p, Bus9.p) annotation(
Line(points = {{17.0625, 13.125}, {17.0625, 15.3125}}, color = {0, 0, 255}))
    ;
connect(L78.n, Bus8.p) annotation(
```

```
Line(points = {{17.0625, 9.625}, {17.0625, 10.9375}}, color = {0, 0, 255}));
connect(L78.p, Bus7.p) annotation(
Line(points = {{17.0625, 9.625}, {17.0625, 8.3125}}, color = {0, 0, 255}));
connect(L45.n, BUS5.p) annotation(
Line(points = {{10.0625, 13.5625}, {11.375, 10.9375}}, color = {0, 0, 255}))
    ;
connect(L45.p, BUS4.p) annotation(
Line(points = {{10.0625, 13.5625}, {8.75, 16.1875}}, color = {0, 0, 255}));
connect(L69.n, Bus9.p) annotation(
Line(points = {{14.65625, 15.75}, {17.0625, 15.3125}}, color = {0, 0, 255}))
    ;
connect(L69.p, BUS6.p) annotation(
Line(points = {{14.65625, 15.75}, {12.25, 16.1875}}, color = {0, 0, 255}));
annotation(
uses(OpenIPSL(version = "1.0.0")));
end Test_system;
```