Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL

Giuseppe Laera[1] Luigi Vanfretti[1] Marcelo de Castro Fernandes[1] Sergio A. Dorado-Rojas[1] Fernando Fachini[1] Chetan Mishra[2] Kevin D. Jones[2] R. Matthew Gardner[2] Hubertus Tummescheit[3] Stéphane Velut[3] Ricardo J. Galarza[4]

[1]ECSE, Rensselaer Polytechnic Institute, Troy (NY), USA,
{laerag,vanfrl,decasm3,dorads,fachif}@rpi.edu
[2]Dominion Energy, Richmond (VA), USA,
{chetan.mishra,kevin.d.jones,matthew.gardner}@dominionenergy.com
[3]Modelon, Glastonbury (CT), USA & Lund, Sweden,
{hubertus.tummescheit,stephane.velut}@modelon.com
[4]PSM Consulting, Inc., Guilderland (NY), USA, rgalarza@psm-consulting.com

Abstract

This paper offers systematic guidelines for modeling power systems components in the phasor time-domain using the Modelica language and their verification. It aims to share the authors' experience in power system modeling with Modelica and the approaches used to meet the high expectations of the power industry w.r.t. to the models' simulation results. While the modeling guidelines are generic, the verification procedure includes the validation against a domain-specific commercial software tool called PSS®E that is the *de facto* tool used for power system transmission planning and analysis. To formalize the proposed approaches, a schematic description of the processes of model implementation and validation is elicited through flowcharts. Challenging use cases are presented to point out some of the major difficulties that can be faced in the modeling steps because of unclear or missing documentation of the models' dynamics in the reference tool. Finally, unique features of the Modelica language that allow for power system modeling and verification unavailable in traditional tools are illustrated.

Keywords: Modelica, OpenIPSL, PSS®E, Dymola, Modelon Impact, SystemModeler, OpenModelica

1 Introduction

This paper aims to formalize the process of power systems dynamic modeling and model verification using the Modelica language. The mail goal is to provide, for the first time, a formal description of the steps necessary for complete re-implementation of power systems components of closed-source commercial software like PSS®E in the Modelica language. A generic methodology for re-implementing existing models from different software tools and domains can be derived from the proposed steps. Key use cases that have been challenging to implement and verify are presented to highlight the value of the proposed approaches for model implementation and validation. Moreover, it is shown how the self-documenting nature of object-oriented equation-based modeling offered by Modelica provides unique advantages for human and computer readable model implementation, as compared to existing modeling tools that do not always offer transparency regarding the dynamic behavior implemented in their tools. In this collection of examples, models for the OpenIPSL library[1] have been considered and validated against PSS®E. Finally, we emphasize the value of the open-access standardized Modelica specification as a key enabler of open-access standards-based interoperability (Gómez et al. 2020), showing how the newly implemented models can be re-utilized in multiple Modelica-standard-compliant tools without a need for re-implementation.

The reminder of this paper is organized in four sections: *Introduction* describing some motivations and contributions, *Guidelines* formalizing in flowcharts the process of models implementation and validation, *Use Cases* illustrating the proposed approaches with examples of implemented models and *Future Work and Conclusions* with some final comments.

1.1 Motivations

Modeling of power systems has always been fundamental for the design, operation and planning of electric networks. To help all the players of the electric power systems sector to perform their studies and analyses, over the last decades several software tools have been developed. The *de facto* tools used by industry include proprietary software like PSS®E, PSCAD, EMTP-RV, PowerFactory, etc., that require the user to become an expert of their functionalities and poses intimate tool-and-domain specific knowledge to be productive. In addition, each tool has its own way of defining the data used to characterize their discretized model (i.e. "data format") mak-

[1]https://github.com/OpenIPSL/OpenIPSL

 10.3384/ECP21186146 DOI

ing it inflexible when attempting to share models and data between tools (Hongesombut et al. 2005). Another drawback is represented by the inconsistency of dynamic simulation results between different simulation platforms. This is due to the need to re-implement models in each simulation platform, which has tremendous costs. As reported by the Australian Energy Market Commission [2], the order of magnitude costs for model implementation in each individual tool can reach almost $500,000.00 in the case of existing components with power electronic interfaces, such as wind and solar, thus making it challenging to study the effects of the integration of renewable energy resources. Therefore, the idea to remove ambiguity in power systems modeling was suggested in (Vanfretti et al. 2013) by using the object-oriented equation-based modeling language Modelica (Tiller 2001). Following this innovative idea a power systems library called OpenIPSL has been developed (Baudette et al. 2018). The library continues to be maintained and expanded based on the concepts of regression testing and Continuous Integration (CI) (Rabuzin, Baudette, and Vanfretti 2017), that allow the models to be verified against a traditional commercial software tool like PSS®E giving the user of the power systems community the confidence about its reliability for performing dynamic simulations and studies.

1.2 Previous Works

The proposed process for testing newly implemented models in Modelica and their verification make use of a Single Machine Infinite Bus (SMIB) equivalent system model (Zhang et al. 2015). This model is typically used to test the implementation of new models and designs of control strategies, which has been emphasized in the literature (Chaudhary and Singh 2014; Kumar 2018; Jayapal and Mendiratta 2010; Wang et al. 2015). This small network is useful in describing the key behavior of a power plant within a power system for most practical purposes. It offers a good framework for studying basic power systems stability concepts and the application of different control techniques to understand their effects on the network.

Regarding the development of new open-source software for power systems studies several examples based on different languages have been reported in the literature. During the 1990's and 2000's, MATLAB saw an exponential adoption in academia, resulting in a number of power system simulation software, such as *MATPOWER* (Zimmerman, Murillo-Sánchez, and Thomas 2010), focusing on steady state computations and the Power System Analysis Toolbox that offers an array of analysis types, both steady state or dynamic (F. Milano 2005). With the rise of Python, a new generation of Python-based tools for modeling, analysis and optimization of electric power systems have been developed. Among these, *pandapower* (Thurner et al. 2018) which aids with steady state computations. *GridCal*[3] is another platform for power systems research and simulation based on Python, again, focusing on steady state computations. Meanwhile, another Python-based tool for power system simulation is *ANDES* (Cui, Li, and Tomsovic 2020). It is an open-source Python library for power system modeling, computation, analysis, and control and it uses a hybrid symbolic-numeric framework for numerical analysis. What these software have in common is that they define both their models in discretized form, interlinking a specific numerical solver (e.g. Newton solver for steady state analysis or trapezoidal integration for dynamic simulation) during implementation.

In recent years, the Julia language has been gaining popularity by the modeling and simulation community (Elmqvist, Henningsson, and Otter 2016). Naturally, Julia packages for power system modeling and simulation of power systems have also emerged, with *PowerSimulationsDynamics.jl* for power system dynamics and for power systems operations called *PowerSimulations.jl* (Henriquez-Auba et al. 2021). While these packages do require the user to specify their models in discretized form and target a specific solver, as in the case of the MATLAB and Python-based tools described above, they do require the user to specify models with pre-defined data structures and the resulting models can only be used with the solvers available within the Julia ecosystem (Henriquez-Auba et al. 2021), not to mention the models have not been validated against any other reference software tool.

Regardless, as each of the afromentioned tools define their own approach for model implementation, the means to define parameter data and support only specific solvers, the results obtained will be different.

In contrast, Modelica facilitates the re-use of models among different Modelica-compliant tools by defining interoperable libraries, with the models of each component implemented separately and without the need of a numerical solver. There are several power system analysis libraries based on Modelica, (Winkler 2017) gives an overview of all available open-source libraries for power system dynamic analysis and highlights the advantages and disadvantages of each library. Note that most libraries listed are for positive sequence phasor-based dynamic simulation. In addition, it is worth mentioning *MSEMT*: an advanced Modelica library for power system electromagnetic transient studies (Masoom et al. 2021). This paper enhances and expands the OpenIPSL library, which focuses on power system models in the phasor-domain that have been validated against PSS®E. In that regard, some recent examples of models validation for renewable energy sources and battery energy storage systems are given in (Fachini et al. 2021).

[2]Online: `https://tinyurl.com/aemc-rule2017`

[3]`https://github.com/SanPen/GridCal`

1.3 Contributions

The use of the object-oriented equation-based modeling language, Modelica, allows for unambiguous power systems modeling. The flexibility of the language offers options to model a component, either by using typical block diagram representations or by defining mathematical equations that describe its dynamic behavior. The classical approach to validate a component's dynamics is by creating a small reference power network model, like an SMIB as used herein. As shown in Section 3, using the Modelica language it is also possible to validate each individual sub-system component within a model by using the reference tool's output signals and connecting them to the inputs of the newly implemented model. This simplifies the modeling process as it allows to isolate the specific individual part of the model that is being implemented in Modelica, instead of having to use it in a small reference power network as it is necessary for the traditional software tools to perform simulations.

Explicitly, the contributions of this paper are as follows:

- To formalize the model implementation approach used for component model development in Modelica to meet the requirements of the power industry.
- To formalize the software-to-software model validation process for Modelica model validation against reference domain-specific tools, illustrated with the *de facto* standard in the power industry, PSS®E software.
- To propose an approach to validate sub-system model components without the need to define entire system models in Modelica by replaying the reference tool simulation results into the Modelica model.
- To illustrate the proposed approach with challenging implementation and validation use cases.
- Synthesizing the know-how described above into guidelines which fully elicit the model implementation and validation process using flowcharts.

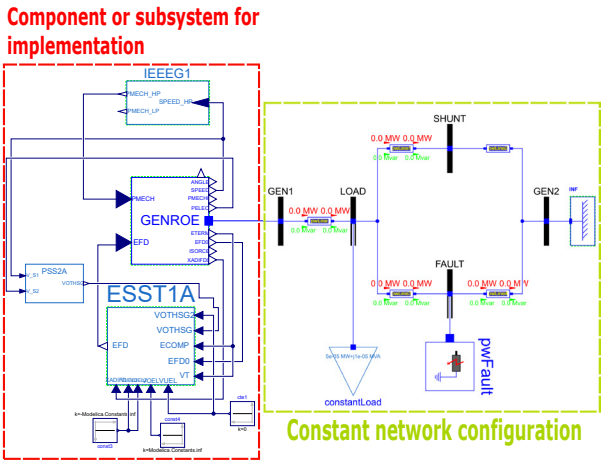

Figure 1. SMIB template in Modelica.

2 Guidelines

In this section the approach of modeling power systems components in Modelica is described.

2.1 Template Models for Modeling and Validation

A basic example of electric power system network is the SMIB. Its block diagram in Modelica is given in Figure 1. This system is used to model a power plant with its controls connected to the rest of the grid through transmission lines and substations represented by buses. This small network also defines the model to standardize the testing of different device models implemented in Modelica and having PSS®E models as reference.

In Figure 1, in the red block, a complete power plant is modeled with a generator, connected to bus GEN1, and its controls. The grid is represented by the generator connected to bus GEN2. Between GEN1 and GEN2 other power systems components are considered like lines, a load and a ground fault. The bus SHUNT also allows to connect other components to this small network. Depending on the type of power systems component to model, the configuration of the generating unit (red block) connected to bus GEN1 varies whereas the configuration of the rest of the network (green block) remains the same.

In the sequel, components in the red block of Figure 1 will be implemented and validated. Note that for all the tests, the remainder of the power system in the green block of Figure 1 remains unchanged.

2.2 Model Implementation Guide

The modeling implementation process is defined in Figures 2 and 3. While the approach is generic, the process depicted considers PSS®E the reference software tool. In Figure 2 the process of implementation starts with the assignment of a model to implement (a). The identification of available technical information (b) about the model is necessary to the model implementation in Modelica. If the PSS®E manuals do not present sufficient information about the model's dynamics then it is necessary to find additional literature (c) that can help understanding how the models were implemented within the reference tool. Once the documents describing the model have been identified, collected and analyzed by finding the block diagrams and/or the equations of the model (d), it is possible to determine if the building blocks of the model are already present in the OpenIPSL library (e). It might be required to implement missing blocks or functions (f) before building the entire component model with the appropriate initialization of its sub-blocks (g). To assess the validity of the model to implement, a small test network (SMIB) is used in both Modelica and the reference software, and generating the reference results from PSS®E (h). That means the SMIB network with the target model needs to be assembled both in PSS®E (i) and Modelica (l). After that the software-to-software validation can be performed

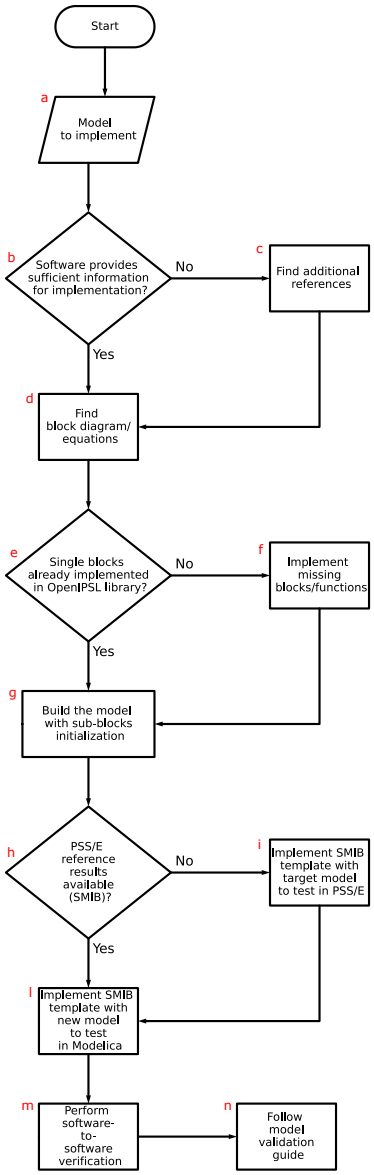

Figure 2. Flowchart of the process of implementation of power systems models.

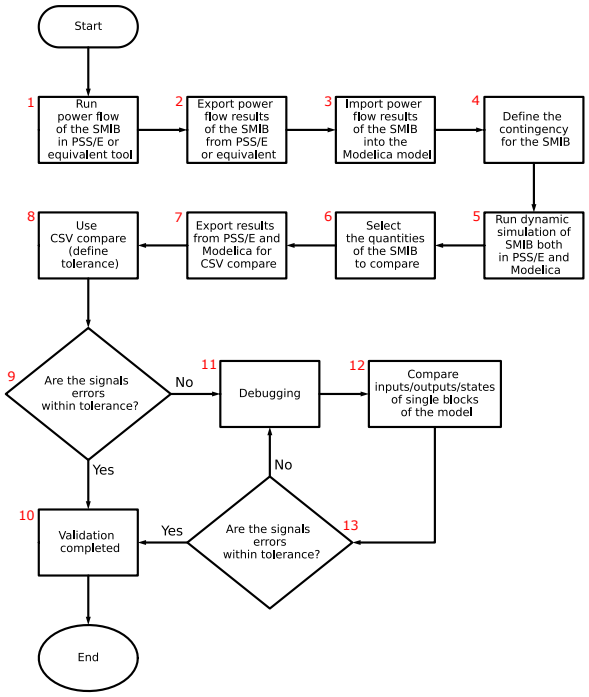

Figure 3. Flowchart of the process of software-to-software validation of power systems models.

(m) following the model validation guide (n) (see Figure 3).

2.3 Model Validation Guide

The process of model validation is defined through the flowchart in Figure 3 and it comes after the process of Figure 2. After assembling the SMIB in PSS®E it is necessary to obtain the steady state computation results of a "power flow" (1), export them (2) and provide them as initial guess values to solve the initialization problem of the corresponding SMIB in the Modelica compliant software tool (3). The OpenIPSL library used in this validation process describes the dynamic behavior of power system components therefore it relies on external tools for the power flow calculations necessary for initializing the models. The import of the power flow results can be made manually or automatically (Dorado-Rojas et al. 2021), the latter increasing numerical accuracy and reducing human errors. Next, the scenario for the dynamic simulation in both tools can be defined (4) and a dynamic simulation of the SMIB in both softwares can be run (5). Once the results are generated the quantities to compare can be chosen (6) and exported in the appropriate format (7) to be used in another tool, for example CSV Compare[4] (8). Tools like CSV Compare allow to quantify the discrepancies between the simulation software tools after defining an acceptable tolerance level (see Figure 4). If the errors between the quantities to compare are within the tolerance band (9) then the validation is complete (10). If the er-

[4]https://github.com/modelica-tools/csv-compare

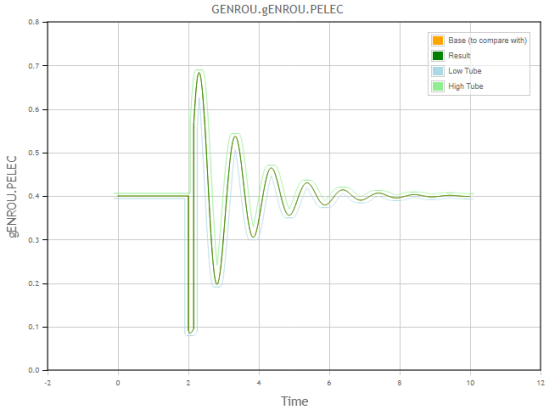

Figure 4. Example of the use of the CSV Compare tool.

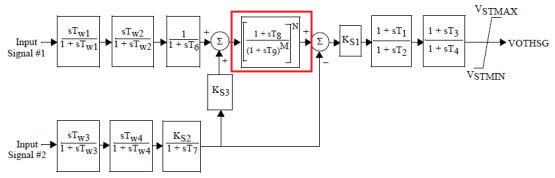

Figure 5. Block diagram of PSS2A from PSS®E manual (Siemens Industry 2013).

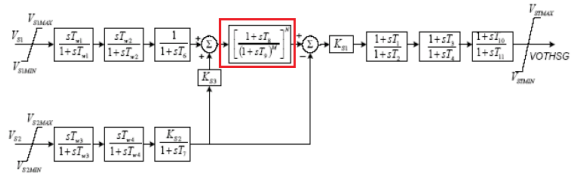

Figure 6. Block diagram of PSS2B from PSS®E manual (Siemens Industry 2013).

rors are bigger than the defined tolerance then more debugging (11) of the model is required. A better insight of the model dynamics can be obtained by comparing subcomponent inputs, outputs and states of the implemented model (an example is given in Section 3) with the analogous signals from the SMIB in PSS®E (12). The iterative process continues until the difference between signals is lower than the tolerance (13), meaning that the validation is completed. This process can be part of a continuous integration and regression methodology, such as the one described in (Baudette et al. 2018).

3 Use Cases

In this section some key use cases of the implementation of power system components are described following the steps in Section 2. The verification illustrated through the plots of some quantities includes a software-to-software validation between PSS®E and different Modelica compliant platforms.

3.1 Power System Stabilizer PSS2A model

3.1.1 Implementation

An example of the difficulties faced when implementing a standard power system model like those of power system stabilizers (PSSs) PSS2A and PSS2B is given in this section. The reference software tool PSS®E comes with several manuals to help understanding the implementation and behavior of the different components present in its libraries. In some cases, like for the aforementioned PSSs models, the documentation is insufficient and not helpful to understand the behavior of one of the blocks of the models. In particular, this block is the ramp tracking filter highlighted in Figures 5 and 6.

An initial implementation of the ramp tracking filter in Modelica revealed to be not accurate compared to PSS®E implementation. This result led to additional investigations about the ramp tracking filter block. Because the PSS®E documentation does not offer more details, then the idea of analysing the continuous time trajectories of the states of this block was used. This idea was derived considering the available information, that needs to be appropriately selected, included in the simulation results of a SMIB test system in PSS®E with one of the PSSs. So from PSS®E it is possible to analyze the dynamics of the ramp tracking filter block through its input, output and states. A visual illustration of the implemented corrections when the filter model was debugged is given in Figure 7.

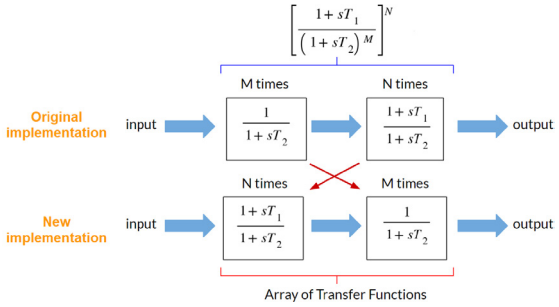

Figure 7. Original and new block diagram representation of the ramp tracking filter for the Modelica implementation.

The Modelica code for the ramp tracking filter model is given in Listing 1. With this kind of implementation several features of the Modelica language have been used. The language is object-oriented and, in this case, it allows for the creation of arrays of transfer functions to build the model. This implementation is transparent and self-documented without leaving any uncertainty about the dynamics of the component as opposed to the *de facto* standard proprietary tools of the power systems domain that are not always well documented, or as in this case, the documentation of the software was erroneous. In addition to that, the level of abstraction and portability with the Modelica language is superior.

```
1 model RampTrackingFilter "Ramp-tracking filter"
2   extends Modelica.Blocks.Interfaces.SISO;
3   import Modelica.Blocks.Continuous;
4   parameter Real T_1;
5   parameter Real T_2;
6   parameter Integer M = 5 ">=0, M*N<=8";
```

```
  parameter Integer N = 1 ">=0, M≥N<=8";
  parameter Real y_start = 0 "Output start value";
  final parameter Boolean bypass = if M == 0 or N == 0 then true
      else false "Boolean parameter" annotation(Evaluate =
      true);
  Continuous.TransferFunction TF1[M](b=fill({1},M), a=fill({T_2
      ,1},M),each y_start=y_start) if bypass == false "
      Conditional component";
  Continuous.TransferFunction TF2[N](b=fill({T_1,1},N), a=fill({
      T_2,1},N), each y_start=y_start) if bypass == false "
      Conditional component";
equation
  if M == 0 or N == 0 then
    u = y;
  elseif M == 1 then
    connect(u, TF2[1].u);
    for i in 1:N-1 loop
      connect(TF2[i].y, TF2[i+1].u);
    end for;
    connect(TF2[N].y, TF1[1].u);
    connect(TF1[1].y, y);
  elseif N == 1 then
    connect(u, TF2[1].u);
    connect(TF2[1].y, TF1[1].u);
    for i in 1:M-2 loop
      connect(TF1[i].y, TF1[i+1].u);
    end for;
    connect(TF1[M-1].y, TF1[M].u);
    connect(TF1[M].y, y);
  elseif M == 1 and N ==1 then
    connect(u, TF2[1].u);
    connect(TF2[1].y, TF1[1].u);
    connect(TF1[1].y, y);
  else
    connect(u, TF2[1].u);
    for i in 1:N-1 loop
      connect(TF2[i].y, TF2[i+1].u);
    end for;
    connect(TF2[N].y, TF1[1].u);
    for i in 1:M-2 loop
      connect(TF1[i].y, TF1[i+1].u);
    end for;
    connect(TF1[M-1].y, TF1[M].u);
    connect(TF1[M].y, y);
  end if;
end RampTrackingFilter;
```

Listing 1. Modelica code for *Ramp Tracking Filter* model

The implementation of the ramp tracking filter has been tested first with the component alone as in the system of Figure 8. A ramp signal has been applied as input and the following parameters: $T_1 = 0.5$, $T_2 = 0.1$, $M = 5$, $N = 1$ for the filter have been used. The choice of the parameters of the ramp tracking filter has been derived from the reference example of Figure 11. By varying those parameters it is possible to obtain the desired ramp tracking behavior (Bérubé and Hajagos 2007; Berube, Hajagos, and Beaulieu 1999).

The results of the test in Figure 8 are given in Figure 9.

3.1.2 Validation

The new implementation of the ramp tracking filter has been introduced in the PSS2A component model. The Modelica model of PSS2A corresponding to the one in Figure 5 is given in Figure 10.

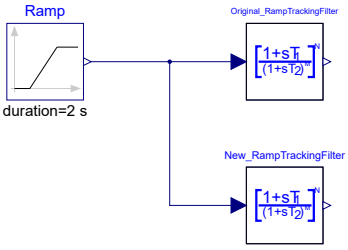

Figure 8. Ramp tracking filter test with the original and final version (see Figure 7).

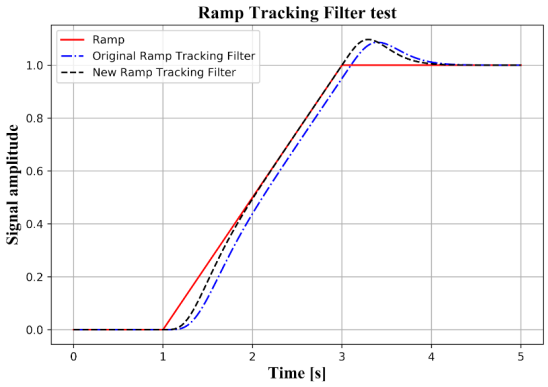

Figure 9. Results of the ramp tracking filter test: reference ramp signal (red), original Modelica implementation of ramp tracking filter (blue) and final new implementation of ramp tracking filter (black) (see Figure 7).

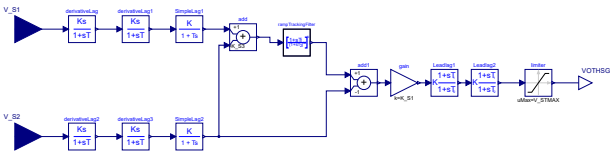

Figure 10. Block diagram of PSS2A in Modelica.

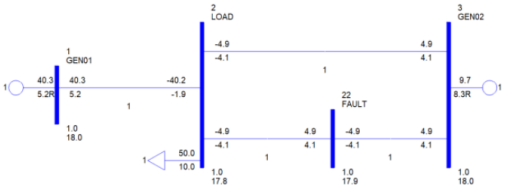

Figure 11. SMIB test system for PSS2A in PSS®E.

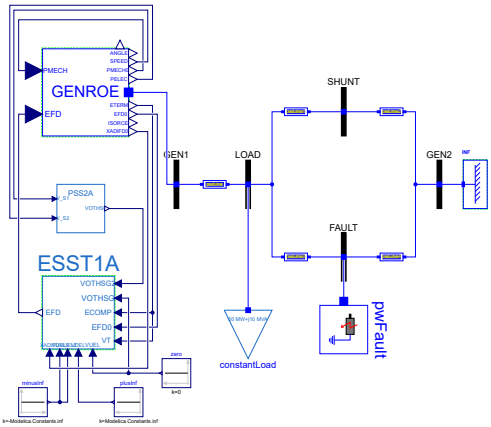

Figure 12. SMIB test system for PSS2A in Modelica.

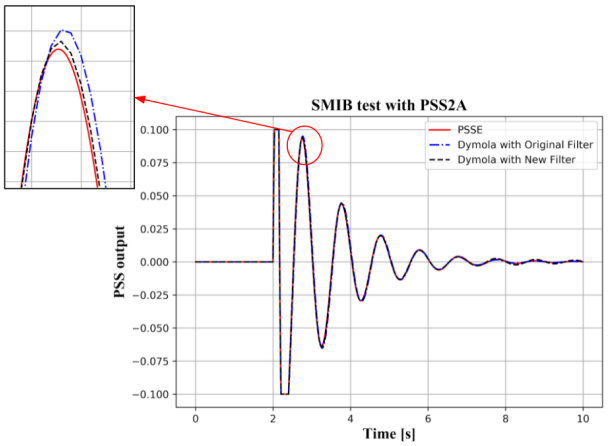

Figure 13. Results comparison between the Modelica SMIB with the original (see Figure 7) and final implementation (see Listing 1) of the ramp tracking filter and the SMIB from PSS®E.

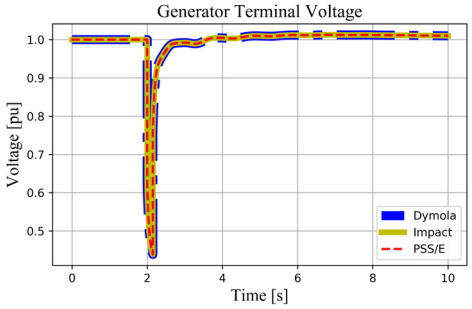

Figure 14. Generator terminal voltage at bus GEN1 of the system in Figure 12.

The component PSS2A has then been tested in a small network like a SMIB. The PSS®E benchmark system is illustrated in Figure 11.

The corresponding SMIB implementation in Modelica is given in Figure 12.

The tested scenario for the SMIB consists of a 3 phase fault to ground applied at bus FAULT at $t = 2s$ for $0.15s$. The results are plotted in Figure 13.

A multi-platform software-to-software validation of PSS2A using the same SMIB system in Figure 12 with the same scenario is given in Figures 14, 15, 16 and 17. The simulations in the different tools have been performed with the same tolerance (1e-06) and the same output interval length ($0.001s$).

Another feature of the Modelica language is the possibility to implement models in the traditional way using block diagrams, like PSS2A, or with coding, like the ramp tracking filter. The graphic layer and the text layer of a model are linked to each other without the need to work on both separately like in the standard software tools, which is illustrated next.

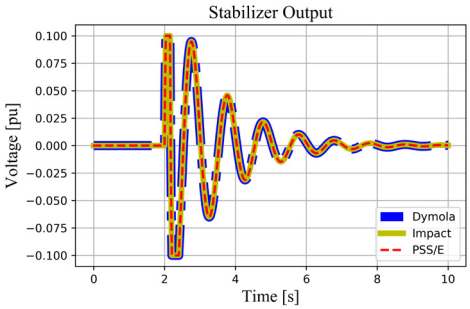

Figure 15. PSS2A output of the system in Figure 12.

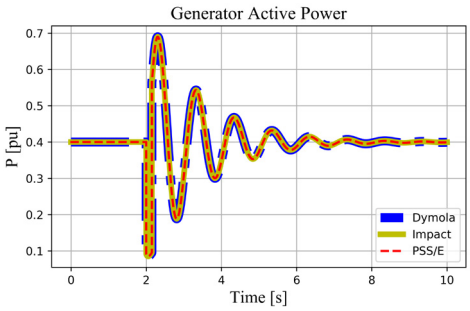

Figure 16. Active Power of the generator at bus GEN1 of the system in Figure 12.

3.2 IEEE 421.5 2005 DC4B Excitation System model

3.2.1 Implementation

Another example of model implementation is the exciter model DC4B. In Figure 18 the block diagram of the component from a PSS®E manual is given.

The PID with non-windup limits included in the model in Figure 18 is represented in Figure 19.

The challenge of the implementation of this component was represented by the integrator block inside the PID block. From (Murad and Federico Milano 2019) it is clear that for the same component, like a PI, there can be different implementations. In our case, to find the right representation of the dynamics of the PID block it has been necessary to check the state of the integrator. The interpretation has been facilitated by observing the output of the PID block together with the state of the integrator of the PID during a dynamic simulation in PSS®E of the SMIB including the DC4B model. The considered scenario is a 3-phase fault applied at bus FAULT at $t = 2s$ for $0.15s$. The corresponding SMIB in Modelica is given in Figure 20. The time trajectories of the PID output and its integrator state from PSS®E for the bus fault scenario are illustrated in Figure 21.

From Figure 21 it is possible to see that when the output of the PID reaches its limits the state of the integrator freezes to avoid windup effects (the so called *conditional integrator*). This observation led to the Modelica imple-

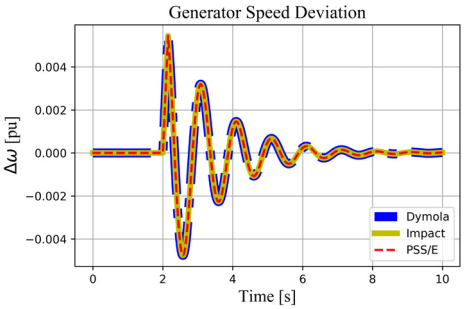

Figure 17. Speed Deviation of the generator at bus GEN1 of the system in Figure 12.

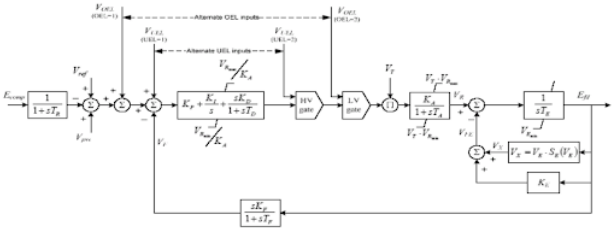

Figure 18. Block diagram of DC4B from PSS®E manual (Siemens Industry 2013).

mentation of the PID with non-windup limits as in Figure 22.

The Modelica text layer of the model in Figure 22 explains the behavior of the integrator of the PID (see Listing 2). To make the code more easily readable, in Listing 2 all the annotations, the lines indicating the blocks and additional parameters for the initialization of the model have been removed.

```
1  model PID_No_Windup
2    import      Modelica.Units.SI;
3    parameter SI.PerUnit K_P "Voltage regulator proportional gain
         (pu)";
4    parameter SI.TimeAging K_I "Voltage regulator integral gain (
         pu)";
5    parameter SI.PerUnit K_D "Voltage regulator derivative gain (
         pu)";
6    parameter SI.Time T_D "Voltage regulator derivative channel
         time constant (sec)";
7    parameter SI.PerUnit V_RMAX "Maximum regulator output (pu)";
8    parameter SI.PerUnit V_RMIN "Minimum regulator output (pu)";
9  equation
10   reset_switch.u2 =
```

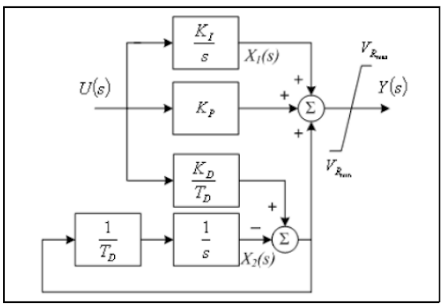

Figure 19. Block diagram of PID with non-windup limits from PSS®E manual (Siemens Industry 2013).

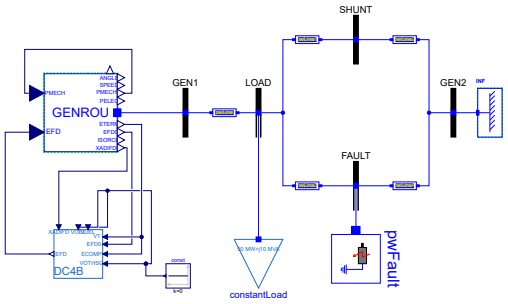

Figure 20. SMIB with DC4B in Modelica.

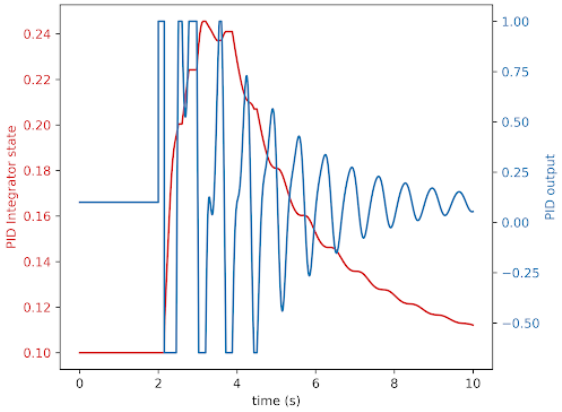

Figure 21. Plot of the output of the PID and the state of its integrator from a bus fault simulation in PSS®E.

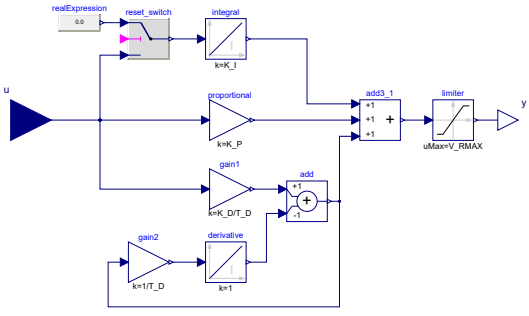

Figure 22. PID with non-windup limits implemented in Modelica.

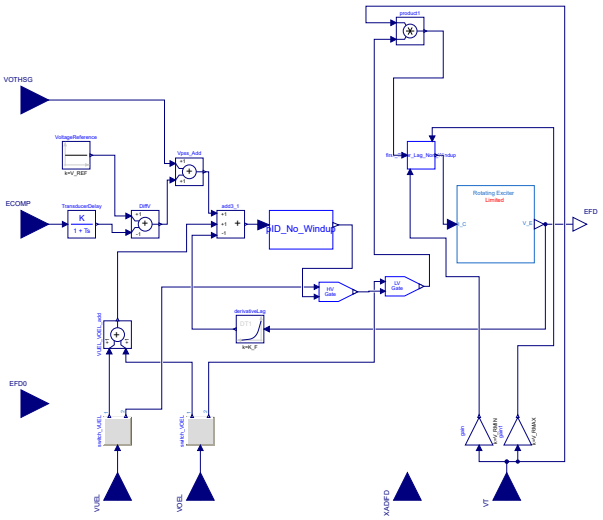

Figure 23. IEEE 421.5 2005 DC4B Excitation System model in Modelica.

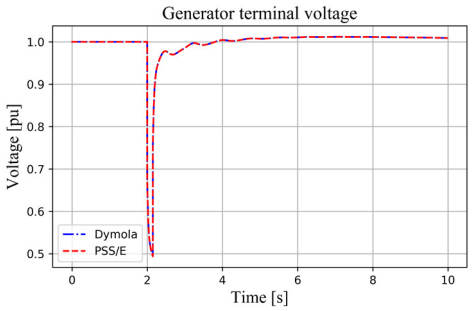

Figure 24. Generator terminal voltage at bus GEN1.

```
    if (abs(V_RMAX - y) <= Modelica.Constants.eps and der(integral
        .y)>0)
      then true
    else if (abs(V_RMIN - y) <= Modelica.Constants.eps and der(
        integral.y)<0)
      then true
    else false;
end PID_No_Windup;
```

Listing 2. Modelica code for *PID with non-windup limits* model

Then the implementation of the DC4B model in Modelica has been completed as in Figure 23.

3.2.2 Validation

The software-to-software validation of the DC4B model against PSS®E has been performed to check the validity of the adopted modeling approach. For the verification the SMIB in Figure 20 has been considered with a bus fault applied at bus FAULT at $t = 2s$ for $0.15s$. Some results of the validation are given in Figures 24, 25 and 26.

An alternative approach for the validation of the DC4B excitation system, that Modelica flexibility allows for, consists of simulating only the block representing the DC4B component driven by external input signals collected from the reference SMIB network in PSS®E. This means that the Modelica model of DC4B can also be run

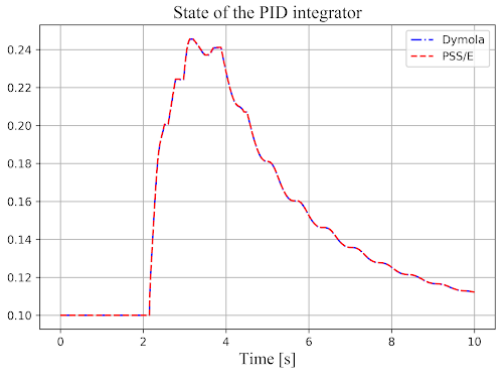

Figure 25. Time trajectory of the state of the integrator of the PID with non-windup limits included in the DC4B model.

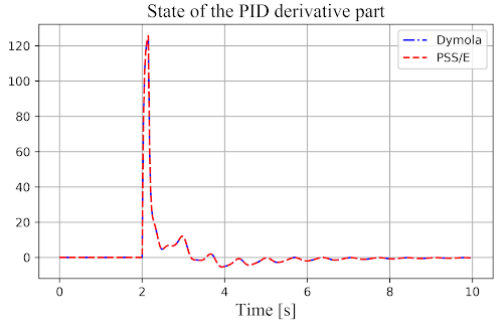

Figure 26. Time trajectory of the state of the derivative part of the PID with non-windup limits included in the DC4B model.

using different input signals coming, for example, from real measurements.

To prove this new approach the PID model in Figure 22 has been modified replacing the integrator blocks with an input so to use the integrator state trajectory as input to the model (see Figure 27).

Then a new test system in Modelica has been created as in Figure 28. Instead of assembling an SMIB to test the new implemented model it is possible to build a test system with only the target model with all the inputs depending on the available signals and a table collecting the external signals driving the model. To clarify the link between the external signals and the model to test, the text layer of the Modelica model in Figure 28 is given in Listing 3 keeping only the *equation* section.

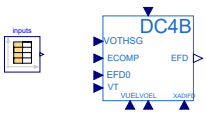

Figure 28. New Modelica test system for DC4B.

```
model DC4B_state_input_test
equation
  dC4B_state_input.VOTHSG = inputs.y[1];
  dC4B_state_input.ECOMP = inputs.y[2];
  dC4B_state_input.EFD0 = inputs.y[3];
```

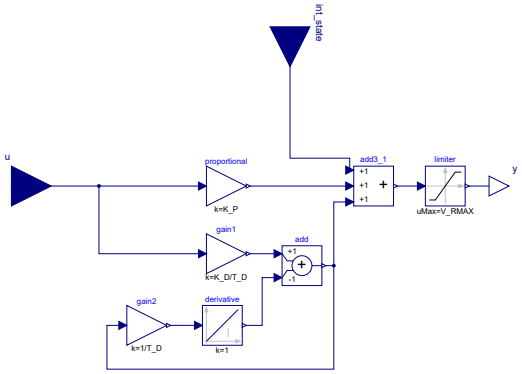

Figure 27. Modified PID with non-windup limits model.

```
    dC4B_state_input.VUEL = inputs.y[4];
    dC4B_state_input.VOEL = inputs.y[5];
    dC4B_state_input.XADIFD = inputs.y[6];
    dC4B_state_input.VT = inputs.y[7];
    dC4B_state_input.int_state = inputs.y[8];
end DC4B_state_input_test;
```

Listing 3. Modelica code for the DC4B test system in Figure 28

Then the validation of the model has been performed comparing the DC4B exciter output (EFD) in a scenario of SMIB system, as in Figure 20, with a bus fault applied at bus FAULT at $t = 2s$ and duration $t = 0.15s$. The results are given in Figure 29.

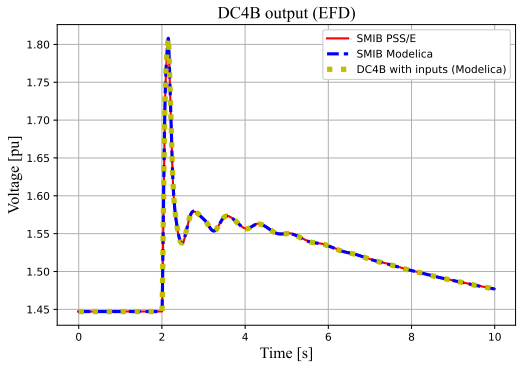

Figure 29. DC4B exciter output (EFD).

In Figure 29 the output of DC4B for the SMIB in Figure 20, the corresponding SMIB in PSS®E and the new test system in Figure 28 has been plotted. The system in Figure 28 takes the required input signals exported from the SMIB in PSS®E simulated with a bus fault as described before.

3.3 Examples of interoperability: portable system modeling with unambiguous and homogeneous results

Finally, to demonstrate an additional value of implementing and validating models with the Modelica language, we highlight the portability of system models using validated components. This can be shown by running the simulation of an example of the OpenIPSL library in different software environments. The example of a SMIB with the exciter EXST1 has been chosen (see Figure 30).

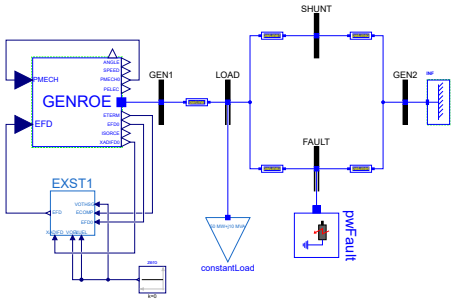

Figure 30. SMIB example for EXST1 from the OpenIPSL library.

Some results of the simulation of the network in Figure 30 are reported in Figures 31 and 32. The plots show that the model gives the same results regarless of the tool being used, which means that tool-specific re-implementation can be avoided.

Another example is illustrated with the network model IEEE14 in Figure 33.

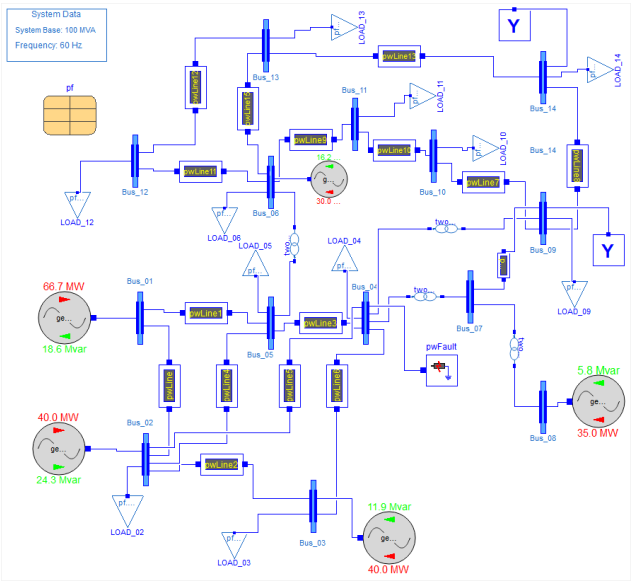

Figure 33. Modelica model of the power systems network IEEE14.

A simulation of the network has been run in the reference tool PSS®E and three different platforms that are Modelica based. They are Dymola, Modelon Impact and SystemModeler. The same scenario has been considered in all three software and it consists of applying a three-phase fault to ground at Bus 4 at time $t = 3s$ for $0.01s$. The fault to ground has an impedance $Z = R + jX = (0.01 + j0.02)pu$. The same simulation settings about the output interval length ($0.01s$) and the tolerance (1e-06) have been used. Results of the

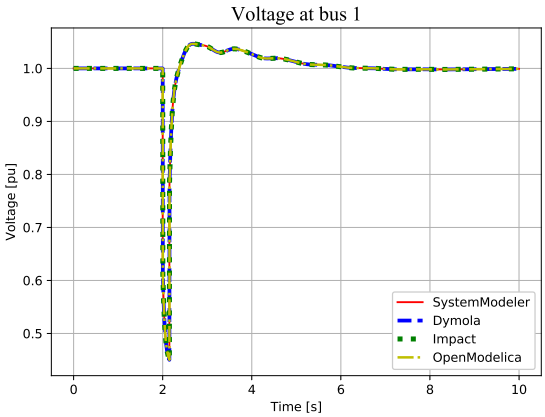

Figure 31. Voltage at bus GEN1 of the system in Figure 30.

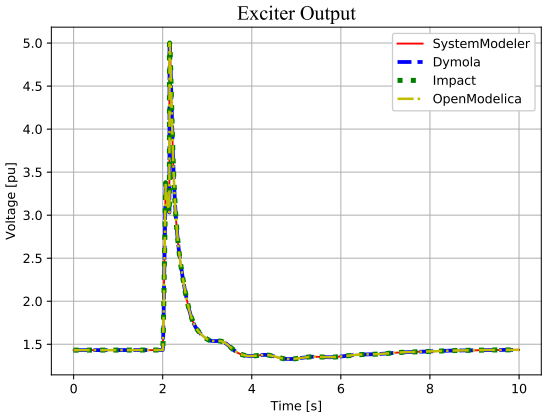

Figure 32. EXST1 output of the system in Figure 30.

simulation are given in Figure 34, where it is possible to see that once the models are validated they can give the same results as the reference tool with the same simulation settings even for larger networks.

4 Conclusions and Future Work

This paper formalizes and illustrates a procedure to implement power system models in Modelica with a software-to-software validation methodology that is an important step for developing and maintaining a Modelica library using the concepts of regression testing and continuous integration. The guidelines illustrated in this paper are indispensable for the initial debugging of new models addressing all possible challenges of the time consuming re-implementation process in a systematic way. The guidelines were illustrated through key use cases that proved to be challenging to implement, this gives an idea of the difficulties faced when developing power system models from a reference software tool. Once the results of an initial validation are satisfactory then the software-to-software

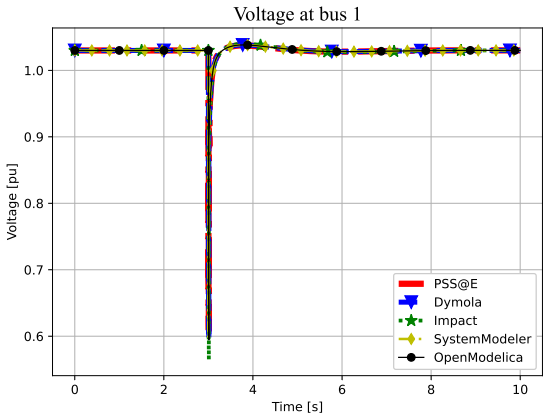

Figure 34. Voltage at bus 1 of the IEEE14 model with a fault to ground at bus 4.

validation process can be automated using scripts to test different scenarios in the different simulation platforms. This work will be presented in a future publication.

Acknowledgements

This material is based upon work supported in whole or in part by Dominion Energy, New York State Energy Research and Development Authority (NYSERDA) and the New York Power Authority (NYPA) under agreement numbers 137940, 37951, and by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Berube, GR, LM Hajagos, and Roger Beaulieu (1999). "Practical utility experience with application of power system stabilizers". In: *1999 IEEE Power Engineering Society Summer Meeting. Conference Proceedings (Cat. No. 99CH36364)*. Vol. 1. IEEE, pp. 104–109.

Bérubé, GR and LM Hajagos (2007). "Accelerating-power based power system stabilizers". In: *Year not known*, p. 10.

Chaudhary, Rekha and Arun Kumar Singh (2014). "Transient stability improvement of power system using non-linear controllers". In: *Energy and Power Engineering* 2014.

Cui, Hantao, Fangxing Li, and Kevin Tomsovic (2020). "Hybrid symbolic-numeric framework for power system modeling and analysis". In: *IEEE Transactions on Power Systems* 36.2, pp. 1373–1384.

Dorado-Rojas, Sergio A. et al. (2021-09). "Power Flow Record Structures to Initialize OpenIPSL Phasor Time-Domain Simulations with Python". In: *Proceedings of the 14th International Modelica Conference*. Ed. by Martin Sjölund

et al. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 147–154. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181147.

Elmqvist, Hilding, Toivo Henningsson, and Martin Otter (2016). "Systems Modeling and Programming in a Unified Environment Based on Julia". In: *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*. Ed. by Tiziana Margaria and Bernhard Steffen. Cham: Springer International Publishing, pp. 198–217. ISBN: 978-3-319-47169-3.

Fachini, Fernando et al. (2021). "Modeling and Validation of Renewable Energy Sources in the OpenIPSL Modelica Library". In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6. DOI: 10.1109/IECON48115.2021.9589148.

Gómez, Francisco J. et al. (2020). "Software requirements for interoperable and standard-based power system modeling tools". In: *Simulation Modelling Practice and Theory* 103, p. 102095. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2020.102095.

Henriquez-Auba, Rodrigo et al. (2021). "Transient Simulations With a Large Penetration of Converter-Interfaced Generation: Scientific Computing Challenges And Opportunities". In: *IEEE Electrification Magazine* 9.2, pp. 72–82. DOI: 10.1109/MELE.2021.3070939.

Hongesombut, K. et al. (2005). "Object-oriented modeling for advanced power system simulations". In: *2005 IEEE Russia Power Tech*, pp. 1–6. DOI: 10.1109/PTC.2005.4524823.

Jayapal, R and JK Mendiratta (2010). "H_∞ Controller Design for a SMIB-Based PSS Model 1.1." In: *Journal of Theoretical & Applied Information Technology* 11.

Kumar, Ajit (2018). "Damping enhancement for smib power system equipped with partial feedback linearization avr". In: *2018 20th National Power Systems Conference (NPSC)*. IEEE, pp. 1–6.

Masoom, Alireza et al. (2021). "MSEMT: An Advanced Modelica Library for Power System Electromagnetic Transient Studies". In: *IEEE Transactions on Power Delivery*.

Milano, F. (2005). "An open source power system analysis toolbox". In: *IEEE Transactions on Power Systems* 20.3, pp. 1199–1206. DOI: 10.1109/TPWRS.2005.851911.

Murad, Mohammed Ahsan Adib and Federico Milano (2019). "Modeling and simulation of PI-controllers limiters for the dynamic analysis of VSC-based devices". In: *IEEE Transactions on Power Systems* 34.5, pp. 3921–3930.

Rabuzin, Tin, Maxime Baudette, and Luigi Vanfretti (2017). "Implementation of a continuous integration workflow for a power system Modelica library". In: *2017 IEEE Power Energy Society General Meeting*, pp. 1–5. DOI: 10.1109/PESGM.2017.8274618.

Siemens Industry, Inc. (2013-03). *MODEL LIBRARY*. English. Siemens Power Technologies International. 748 pp.

Thurner, Leon et al. (2018). "pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems". In: *IEEE Transactions on Power Systems* 33.6, pp. 6510–6521.

Tiller, Michael (2001). *Introduction to physical modeling with Modelica*. Springer Science & Business Media.

Vanfretti, Luigi et al. (2013). "Unambiguous power system dynamic modeling and simulation using Modelica tools". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Wang, Xiaodong et al. (2015). "Nonlinear dynamic analysis of a single-machine infinite-bus power system". In: *Applied Mathematical Modelling* 39.10-11, pp. 2951–2961.

Winkler, Dietmar (2017). "Electrical Power System Modelling in Modelica - Comparing Open-source Library Options". In: *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58)*, pp. 263–270.

Zhang, Mengjia et al. (2015). "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*. Linköping University Electronic Press, pp. 105–112.

Zimmerman, Ray Daniel, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas (2010). "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education". In: *IEEE Transactions on power systems* 26.1, pp. 12–19.