Towards Consistent Model Exchange and Simulation of VSC-HVdc Controls for EMT Studies

Robert Rogersten[1], Luigi Vanfretti[1,2], and Wei Li[1]

[1]School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden
[2]Statnett SF, R&D, Oslo, Norway

***Abstract*—Design of voltage source converters (VSC) for high-voltage direct current (HVdc) systems requires extensive simulation tools that are accurate and reliable. Before the installation of VSC-HVdc links, different simulation-based studies need to be performed by different parties and using different electromagnetic transient (EMT) simulation platforms. Previous work shows that even with a pedantic re-implementation of the models from the ground up, simulation results are still inconsistent between different EMT tools.This paper demonstrates a methodology, which provides a higher consistency between the simulation tools Simulink and PSCAD. This methodology is complex and exposes several limitations. Hence, the is a need for the adoption of a standardized equation-based modeling language for EMT model exchange.**

I. Introduction and Motivation

Design and tuning of controls for voltage source converter (VSC) high-voltage direct current (HVdc) installations need to be assessed using simulation tools. Hence, research on new control methods and techniques also requires the use of simulation tools prior to prototype implementation and to reduce development costs. Currently, there are several tools available for electromagnetic transient (EMT) modeling and simulation of power systems. PSCAD and MATLAB/Simulink are two of such tools, which are used both in industry and academic research.

An issue arises when the model of a VSC-HVdc system is available only for a specific tool. This limits a systematic assessment of the VSC-HVdc design and controller performance by different parties. This is particularly critical when a transmission system operator (TSO) needs to perform studies in tool A and the VSC-HVdc models from the manufacturer are only available in tool B. Given that today's EMT tools do not offer a facility for model exchange (specially at the equation level) [1], this leads to differences in simulation results between tools. Previous work [2] demonstrates such simulation differences. The consequences are either speculation on the 'correctness' of the model or the adequacy of a simulation tool. A quantitative method for measuring discrepancies has been presented in [2]. In this paper, the method will be used to demonstrate how differences in simulation results between Simulink and PSCAD can be minimized through a stricter model exchange mechanism.

PSCAD has the possibility to interface with Simulink through a special PSCAD-Simulink interface included in PSCAD. The work in [3] demonstrates simulations of a classical HVdc system using PSCAD alone, Simulink alone, and using a PSCAD-Simulink interface. The conclusion from this work is that PSCAD outperforms in speed. The work in [3] demonstrates a simulation of 2 s, for which PSCAD took 30.5 s and Simulink took 72.2 s. In comparison to these, the PSCAD-Simulink interface took 12503.58 s using a 50 μs sampling period in both PSCAD and Simulink environments. Even though differences in simulation results can be minimized using the PSCAD-Simulink interface, this approach leads to lengthy design cycles due to a significant increase in simulation time. As an alternative, this paper demonstrates how controls designed in Simulink can be integrated with PSCAD using C code generated from Simulink so to allow for modeling and simulation of VSC-HVdc controls.

The remainder of the article is organized as follows. Modeling of the VSC-HVdc system and its controls is briefly introduced in Section II. In Section III, the proposed model exchange methodology is presented. In order to compare results systematically, two implementations and eight test scenarios were simulated and a quantitative assessment methodology was applied in Section IV. Conclusions are drawn in Section V.

II. Modeling of the VSC and its Control

This paper utilizes a vector current control strategy as presented in [2] to simulate a point-to-point VSC-HVdc system. We omit the detailed introduction due to limited space and refer the readers to [2]. The use of a detailed model of the VSC increases simulation time compared to an average value model (AVM). Therefore, an AVM which represents the average response of the converter by using controlled sources and averaged functions is used in this work. The AVM implementations in PSCAD and Simulink are based on the model proposed in [4].

III. Control Implementation in PSCAD using C-Code Generation

This section explains how the controls in PSCAD can be tailored to replicate the behavior of Simulink more constantly. To implement the controls, C-code was generated for each component in the controller, block-by-block. This approach was preferable in order to facilitate debugging. Within the graphical implementation, each component was replaced piece-by-piece with C-code, which ensures a functioning model after each

978-1-4673-8040-9/15/$31.00 ©2015 IEEE

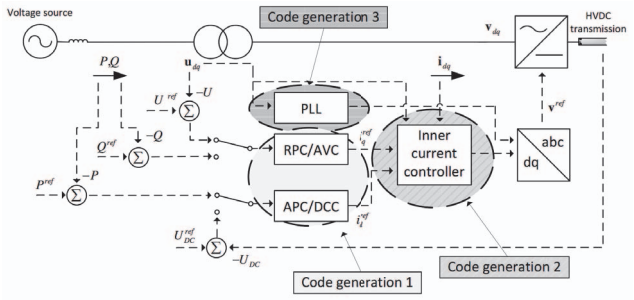


Fig. 1: Steps for Code Generation

component exchange. Therefore, only the components that behave differently between simulation platforms have been exchanged by C-code from the graphical implementation.

Figure 1 illustrates the order of how the controllers were generated and implemented. In the first step everything within the outer control loop was generated in C code from Simulink and implemented in PSCAD. In the second step, everything within the inner control loop was generated in C code and implemented in PSCAD. In the final step, the PLL was generated in C code and implemented in PSCAD. Also, four low-pass filter blocks used in the Simulink implementation were generated on each side of the point-to-point VSC-HVdc model.

A. Graphical Implementation

Screen-shots of the overall model, pedantically implemented in each tool, are not shown here due to space limitations. However, screen-shots of the point-to-point model in PSCAD and in Simulink are shown in [2].

B. Simulink C Code Generation

In computing, C is a general-purpose programming language. The controls from the generic Simulink model can be readily generated in C code. This can be achieved using the Simulink Coder in MATLAB. The Simulink Coder generates C code from Simulink block diagrams, Stateflow charts, and MATLAB functions. After the C code has been generated, it is possible to run and interact with the code outside MATLAB and Simulink.

In this paper, PSCAD has been used to execute the C code generated from Simulink. PSCAD executes the C code on a fixed time-step interval. Therefore, it was necessary to use the Embedded Coder, which extends the Simulink Coder. When the Embedded Coder is used, the Simulink model is configured so it maintains a constant (fixed) step size and also it applies a fixed-step integration technique for computing the state derivatives of the model. The fixed-step time used in this work is 20 ms. Some essential files generated from the Embedded Coder are shown in Table I.

The controller behavior depends on the values of the control parameters. It is possible to modify these values even after the controller has been generated in C code. Modifying the `ModelName_data.c` file changes the parameter values for the controller.

TABLE I: Files generated from the Simulink Coder.

`ert_main.c`	Main file to execute step function
`ModelName.c`	C file that contains the controller
`ModelName_data.c`	C file that assigns values to data structures
`ModelName.h`	Header file that defines data structures
`ModelName_private.h`	Header file that defines data structures
`ModelName_types.h`	Header file that defines the model data structure
`rtGetInf.c`, `rtGetNaN.c`, `rtwtypes.c`, `rtGetInf.h` `rtGetNaN.h`, `rtwtypes.h`	Other C files and header files that are generated. These files serve a general purpose and can be used by other models.

```
#include <stdio.h>
#include "ModelName.h"
#include "rtwtypes.h"
void rt_onestep(double* Ud, double* Uq,
  double* d, double* q, double* Ivq,
  double* Ivd, double* Id_ref,
  double* Iq_ref)
     {ModelName_U.I_ref_q=*Iq_ref;
ModelName_U.I_ref_d=*Id_ref;
ModelName_U.Iv_d=*Ivd;
ModelName_U.Iv_q=*Ivq;
ModelName_U.U_q=*Uq;
ModelName_U.U_d=*Ud;
ModelName_step();
*Id=ModelName_Y.d;
*Iq=ModelName_Y.q; }
```

Fig. 2: The main function, which is manually written in C. The code assign values to the inputs and outputs along with the execution of the step function.

One of the most important files generated from Simulink is the file that contains the step function (called `ModelName.c` in Table I). The step function should be executed each fixed time step during run-time in PSCAD.

The files generated from the Embedded Coder in Simulink contains a set of global variables along with the model step, initialize, and terminate functions. The global variables `ModelName_U` and `ModelName_Y` correspond to the input and output structure of the model, respectively. These can be used to set the inputs prior to each execution of the step function and to receive the outputs after each execution of the step function. Therefore, by the use of a main execution file (`ert_main.c`), the inputs are set, then the step function is executed, and finally the outputs are set. The procedure can be illustrated for the inner control in Figure 5b. This corresponds to the code generation 2 from Figure 1. The inner control loop has 6 inputs and 2 outputs. The inputs are here called U_d, U_q, i_d, i_q, i_d^{ref}, and i_q^{ref}. The outputs are called d and q and represent the voltages that will be transformed to three-phase quantities and converted into line voltages by the VSC. As recently explained, the generated C code for the controller needs to be executed every 20 ms in PSCAD. Figure 2 illustrates how the file, which is executed every 20 ms, can look like. Values are assigned for the inputs, the step function is executed, and values are assigned for the outputs. The code in Figure 2 needs to be manually written by the user.

C. Fortran Integration with C code

Fortran is a general-purpose programming language that is especially suited for numerical computations and scientific computing. Fortran can call existing code that is written in another language. This is commonly referred to as mixed-

```
SUBROUTINE FUN(Ud,Uq,d,q,Ivq,Ivd,
            Id_ref,Iq_ref)
REAL Ud,Uq,d,q,Ivq,Ivd,Id_ref,Iq_ref
INTERFACE
  SUBROUTINE RT_ONESTEP (Ud,Uq,d,q,Ivq,
                    Ivd,Id_ref,Iq_ref)
    !DEC$ ATTRIBUTES C :: RT_ONESTEP
    !DEC$ ATTRIBUTES REFERENCE :: Ud
    !DEC$ ATTRIBUTES REFERENCE :: Uq
    !DEC$ ATTRIBUTES REFERENCE :: d
    !DEC$ ATTRIBUTES REFERENCE :: q
    !DEC$ ATTRIBUTES REFERENCE :: Ivq
    !DEC$ ATTRIBUTES REFERENCE :: Ivd
    !DEC$ ATTRIBUTES REFERENCE :: Id_ref
    !DEC$ ATTRIBUTES REFERENCE :: Iq_ref
    REAL Ud,Uq,d,q,Ivq,Ivd,Id_ref,Iq_ref
  END SUBROUTINE
END INTERFACE
CALL RT_ONESTEP(Ud,Uq,d,q,Ivq,Ivd,
            Id_ref,Iq_ref)
END
```

Fig. 3: Fortran code which calls the C function in Figure 2. Note the call of the C procedure on line 17.

```
CALL FUN($Ud,$Uq,$d,$q,$Ivq,$Ivd,
        $Id_ref,$Iq_ref)
```

Fig. 4: Single line Fortran code in the PSCAD module which calls the subroutine in Figure 3.

language programming. In mixed-language programming, a routine written in Fortran can call a function written in C code. Mixed language programming between these languages is relatively straightforward because of some key similarities between the languages.

PSCAD is a graphical front-end to EMTDC (Electromagnetic Transients including DC) for creating models and analyzing results. EMTDC solves the model's DAEs in the time domain and uses a fixed time-step algorithm. The blocks in PSCAD are actually Fortran code, which call the EMTDC code library to combine them into an executable file. When this file is executed, the simulation performed and the results can be picked up by PSCAD. In PSCAD, custom (i.e. user defined) components can be implemented by using Fortran code. Therefore, several of these components have been created using Fortran code that interface with the C code generated from Simulink. Such components will thus inherit the same behavior of the Simulink model. Figure 3 illustrates the Fortran code that can call the C procedure in Figure 2. A custom designed component needs to be designed in PSCAD to call the code in Figure 3. The code contained in the component for calling the subroutine is listed in Figure 4.

Additionally, library (.lib) and object (.obj) files can be linked in PSCAD. Therefore, all C and Fortran files are compiled into one library file that is linked to PSCAD. The procedure to compile all files is lengthy. Therefore, this is preferably achieved with a script of some kind. The authors of this paper merged together the files with a bash script.

Unfortunately, it is not possible to use multiple instances of the Fortran components that can call a C function in PSCAD, which is the greatest limitation of this approach. Therefore, two inner controllers, and two outer controllers are generated from Simulink. Also the PLL and low pass filters have been implemented using C code. This was done in order to maintain a stricter modeling consistency with the Simulink model.

D. Overall Integration for the Code Implementation

The previous sections described how to generate the C code of the controllers from the Simulink model and implement them in PSCAD. To illustrate the whole process, a flowchart is shown in Figure 5. The upper left of the flowchart shows how the code is automatically generated from Simulink. The automatically generated files here are those from Simulink, which includes header (`.h`) files and `.c` files. A main file (`ert_main.c`) needs to be manually written or modified by the user for the purpose of reading and assigning values for variables in PSCAD. Therefore, the `ert_main.c` file is shown in the upper right of the flowchart, along with the manually written Fortran files.

The process to merge the automatically generated and manually written files into a `control.lib` is lengthy. Therefore, the files are merged together with a script. The bottom of the flowchart illustrates how the `control.lib` file finally is linked together with PSCAD during run-time.

The procedures within the grey box in Figure 5 need to be manually written or modified by the user. This is the most time consuming and complicated part regarding the C code implementation. Particularly, the procedure needs to be repeated for each component that calls any C code because multiple instances are not supported. There exists a possibility to write a script that automatically writes the Fortran and C code needed for each module. However, this is considered to be out of the scope for this paper.

The grey box can be considered as a customized interface in order to get the C code implementation to work. This interface is rather difficult to build and maintain. It also lacks generality. There exists a standardized interface to be used in computer simulations, called the functional mock-up interface (FMI) [5]. FMI is an independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml files and compiled C code. With such interface there is no need for any manually or automatically generated written script. Unfortunately, PSCAD lacks support for this interface. This serves as evidence on the importance of the adoption of standardized interfaces, in particular those that are native to equation-based modeling language (i.e. Modelica [6]), such as FMI, between a broader set of simulation platforms.

E. Scalability of the Proposed Implementation

Scalability is the ability of a computer application or product to function well as it is changed in volume or size [7]. In other words, scalability is the concept of a system to accommodate an increasing number of elements or objects. Therefore, scalability is often a desirable attribute for almost any system.

In a multi-terminal VSC-HVdc system, scalability becomes an important issue when the number of converters increases. In PSCAD, it is normally possible to use multiple instances of modules. Therefore, it is easy to extend a point-to-point system to any number of terminals linked together to form a dc grid. A multi-terminal VSC-HVdc system can be built by a

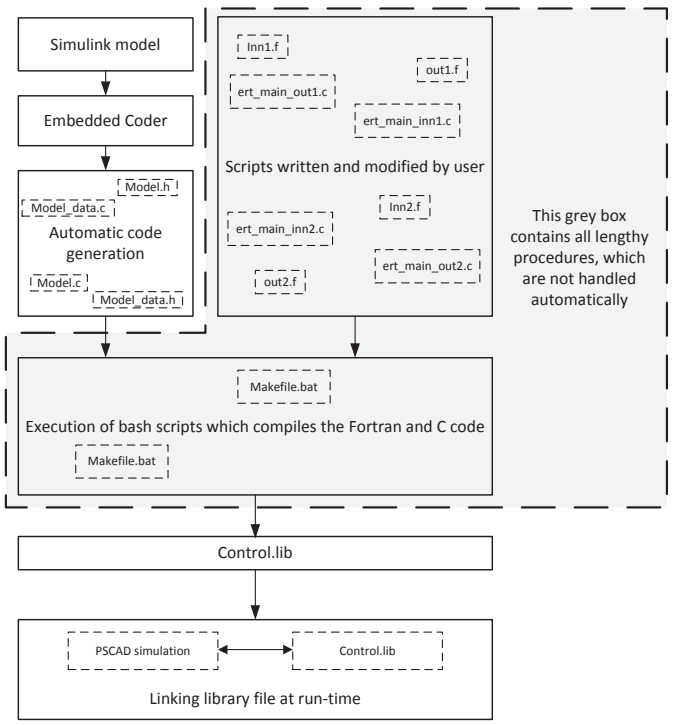
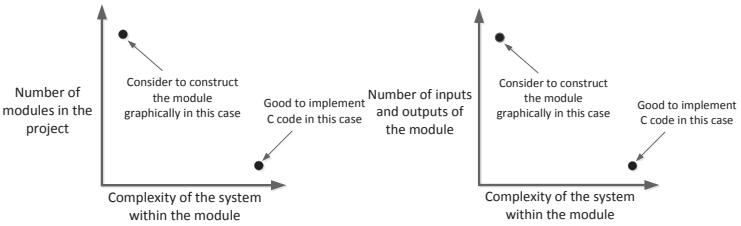


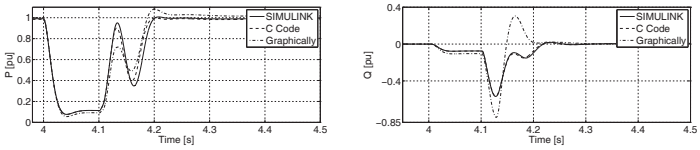
Fig. 5: The flowchart describes how the `control.lib` file is generated and linked as a library file in PSCAD during run-time. The grey box contains all procedures (script writing and compilation), which are not handled automatically. The other boxes are more or less automated.



(a) Illustration of when to use C code from Simulink to construct a module.

(b) Illustration of when to use modules of C code to build up a project.

Fig. 6: Illustration of good and bad scenarios of when to use C code from Simulink to construct modules and when it is good or bad to use these modules to build up a project. Note that the y axis denotes the total number of inputs and outputs in Figure 10a and that the y axis denotes the number of modules within the whole project in Figure 10b.



(a) Active-power measurement.

(b) Reactive-power measurement.

Fig. 7: Illustration of the powers during a three-phase fault with 10% remaining voltage. Figure 7a presents the active-power and Figure 7b the reactive-power. Note that axes have different limits for Figure 7a and Figure 7b. The curves represent the Simulink implementation, the PSCAD C code implementation, and the PSCAD implementation when everything has been implemented graphically.

copy and paste procedure. Normally, the limiting factors will be the increase in simulation time when the system increases in complexity and computer memory.

One of the most major drawbacks with the C code implementation in PSCAD is the lack of support to use multiple instances of user defined modules. This constraint forces the user to go through all the relevant steps in the previous sections for each terminal. This is a rather lengthy procedure compared to the simple copy and paste procedure. How lengthy this procedure depends on the number of inputs and outputs of the system, because each signal needs some lines of code when the Fortran and C code scripts are made.

On the other hand, if the system in Simulink is complex, only has a few set of inputs and outputs, and only a few instances of the same module will be used, then, this approach may be appealing. There will be a trade-off between the number of modules, the total number of inputs and outputs, and the complexity of the system within the module. See Figure 6 for an illustration of this trade-off.

As previously discussed, there exist no script that automatically writes the Fortran and C code for the customized interface shown in figure 5. Therefore, with n modules, i inputs, and j outputs, there will be $n(i+j)/2$ more steps to consider in the Fortran and C code in comparison to one module with one input and one output.

The example in section III-B illustrates how the inner controller is extracted in C code. The inner controller has in total 8 inputs and outputs while the complexity is rather simple. The inner controller consists of two PI-controllers and an additional decoupling term. Therefore, it can be built graphically rather quickly in PSCAD. The author has found that it is easier to construct such module graphically. Particularly, it will facilitate to construct modules graphically if several modules are used in the project.

IV. Comparison and Analysis

In this section, the same power system configuration and simulation scenarios as in [2] are applied. Figure 7 presents the comparison of powers during a three-phase fault with 10% remaining voltage. It gives a hint that the C code implementation provides the best match.

Paper [2] also proposes a quantitative method to determine the discrepancy between different model implementations. This methodology utilizes the root mean square (RMS) of the difference between simulation outputs. The RMS value is calculated according to

$$X_{\text{RMS}} = \sqrt{\frac{1}{n}[(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2]}, \qquad (1)$$

where x_i is the discrete measurement point at time t_i for software (a), and y_i is the discrete measurement point at time t_i for software (b). For more information see [2]. We applied this quantitative assessment metric and methodology to measure discrepancies between graphical implementation and C code implementation.

Table II and III confirm that the C code implementation provides the best match. The mean values (here denoted $\overline{X}$)

of all active- and reactive power RMS measurements are significantly lower for the C code implementation compared to the graphical implementation. The mean values for the graphical implementation are

$$\overline{P}^{\text{gr}}_{\text{RMS}} = 0.0340, \;\; \overline{Q}^{\text{gr}}_{\text{RMS}} = 0.0388.$$

Further, the mean values for the C code implementation are

$$\overline{P}^{\text{cc}}_{\text{RMS}} = 0.0081, \;\; \overline{Q}^{\text{cc}}_{\text{RMS}} = 0.0012.$$

It follows that the active power match is

$$\frac{\overline{P}^{\text{gr}}_{\text{RMS}}}{\overline{P}^{\text{cc}}_{\text{RMS}}} \approx 4$$

times better for the C code implementation. The reactive power match is

$$\frac{\overline{Q}^{\text{gr}}_{\text{RMS}}}{\overline{Q}^{\text{cc}}_{\text{RMS}}} \approx 33$$

times better. Note that these calculations are only representative for the specific test scenarios, here tested on the rectifier side. Other scenarios might yield different results. In addition to the active- and reactive power measurements, quantities i_d^{ref}, i_q^{ref}, and $|\mathbf{i}_{dq}|$ are almost identical for the C code implementation compared to the Simulink implementation.

V. Conclusions

It does not take much effort to implement models graphically in PSCAD. However, for unambiguous modeling and simulation, one drawback with the standard PSCAD library components is that it is difficult to know which mathematical model is implemented within the graphical block. The phase-locked loop (PLL) can be used to illustrate this issue. If a replica of the PLL in Simulink is applied in PSCAD, it is difficult to know what are the implementation differences. Therefore, when a model is graphically built in PSCAD, its behavior does not make a perfect match to the Simulink model.

This paper has suggested a methodology to minimize discrepancies in simulation outputs between PSCAD and Simulink. In particular, this approach achieves 4 times lower value of the mean RMS value difference of active-power and 33 times lower value of the mean RMS value of reactive-power for the specific test scenarios used. In addition, this approach does not significantly increase the simulation time.

Although this approach provides a higher consistency of simulation results between different platforms, the implementation process with generated C code is complex and exposes several limitations. Hence, the results presented in this paper serve as evidence of the need for a standardized equation-based modeling language for model exchange which strictly separates the model from the numerical solver, and highlight the value of standardized simulation interfaces. Standardized equation-based modeling languages have been adopted by other industries [5, 6], leading to a substantial decrease of costs for model development, engineering design and software maintenance; as well as helping users to avoid vendor lock-in from specific tools.

TABLE II: RMS comparison of graphical implementation

Test scenario	1	2	3	4
P_{RMS}	0.0030	0.0144	0.0435	0.0386
Q_{RMS}	0.0143	0.0026	0.0628	0.0573
Test scenario	5	6	7	8
P_{RMS}	0.0421	0.0557	0.0542	0.0208
Q_{RMS}	0.0513	0.0587	0.0555	0.0081

TABLE III: RMS comparison of C code implementation

Test scenario	1	2	3	4
P_{RMS}	0.0076	0.0003	0.0198	0.0125
Q_{RMS}	0.0003	0.0002	0.0031	0.0020
Test scenario	5	6	7	8
P_{RMS}	0.0061	0.0061	0.0062	0.0063
Q_{RMS}	0.0003	0.0019	0.0011	0.0004

References

[1] E. Lambert, X. Yang, and X. Legrand, "Is CIM suitable for deriving a portable data format for simulation tools?" in *IEEE PES General Meeting*, July 2011, pp. 1–5.

[2] R. Rogersten, L. Vanfretti, W. Li, L. Zhang, and P. Mitra, "A Quantitative Method for the Assessment of VSC-HVdc Controller Simulations in EMT Tools," in *IEEE PES ISGT Europe*, 2014.

[3] M. Faruque, Y. Zhang, and V. Dinavahi, "Detailed modeling of CIGRE HVDC benchmark system using PSCAD/EMTDC and PSB/SIMULINK," *IEEE Transactions on Power Delivery*, vol. 21, no. 1, pp. 378–387, Jan 2006.

[4] J. Peralta, H. Saad, S. Dennetiere, J. Mahseredjian, and S. Nguefeu, "Detailed and averaged models for a 401-level MMC-HVDC system," *IEEE Transactions on Power Delivery*, vol. 27, no. 3, pp. 1501–1508, 2012.

[5] T. Blochwitz *et al.*, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models," in *Proceedings of the 8th International Modelica Conference*, Mar. 2011.

[6] L. Vanfretti, W. Li, T. Bogodorova, and P. Panciatici, "Unambiguous power system dynamic modeling and simulation using Modelica tools," in *IEEE PES General Meeting*, July 2013, pp. 1–5.

[7] A. B. Bondi, "Characteristics of Scalability and Their Impact on Performance," in *International Workshop on Software and Performance*, ser. WOSP '00. New York, NY, USA: ACM, 2000, pp. 195–203. [Online]. Available: http://doi.acm.org/10.1145/350391.350432