Implementation of a Continuous Integration Workflow for a Power System Modelica Library

Tin Rabuzin, Maxime Baudette, Luigi Vanfretti
SmarTS Lab, KTH Royal Institute of Technology, Stockholm, Sweden
Email: {rabuzin, baudette, luigiv}@kth.se

***Abstract*—Traditional simulation tools for power system studies are, in general, shipped with built-in and closed model libraries. Typically, the models' implementation is not thoroughly documented, preventing the user to gain a full understanding of their implemented behavior. Previous efforts from the authors have focused on the development of an open source software library of power system components developed using Modelica: the Open-Instance Power System Library (OpenIPSL), which provides models that can easily be accessed and studied by the user. Recent developments have focused on the implementation of a software architecture facilitating collaborative developments on OpenIPSL. Employing the latest technologies available in the software development community, this paper details the implementation of a continuous integration workflow, providing automated testing and behavior verification of the library's models. This platform seeks to increase the library's stability and to provide more reliable models developed collaboratively by multiple individuals. Moreover, this software architecture only utilizes open source software, which can be fully tailored to the specific needs of users and other library developers.**

***Index Terms*—Modelica, continuous integration, testing, power system modeling, power system simulation, model validation**

I. Introduction

Today's power systems are expanding, as grids of different areas are interconnected to form larger geographically-spread systems. The system complexity also increases as more renewable energy sources and power electronic-based devices are connected to the grid. This sets higher requirements on power systems dynamic studies using computer simulations.

The European Network of Transmission System Operators for Electricity (ENTSO-E) is responsible for ensuring secure and reliable operation of the interconnected European transmission system [1]. They identified a need for coordination between the TSOs, and made a substantial effort to comply with the Common Information Model (CIM) and Common Grid Model Exchange Standard (CGMES), to facilitate model exchange between TSOs. However, in CIM v14 and v15, dynamic models are not adequately exchanged as only pictorial block diagrams and their associated parameters are included.

A. Drawbacks of Conventional Power System Simulation Tools

The users of power system simulation tools have widely accepted and used a few proprietary tools for phasor time-domain simulations (i.e. "transient stability" simulations), such as PSS/E, PSLF, DigSILENT PowerFactory and Eurostag [2]. These tools ship with pre-compiled model libraries, and thus, much of the information related to their actual implementation is inaccessible [3]. The modeling philosophy of these tools is rarely questioned, and often overlooked when analyzing simulation results [4].

Moreover, these traditional tools rely on legacy computational code that was written decades ago. Such function-oriented code makes both models and solvers unnecessarily complex, and hard to maintain and extend to fulfill new requirements [5]. Most users of such tools often cannot extend the model library by themselves without specialized skills in dedicated and/or complex modeling language, e.g. DIgSILENT Simulation Language [6] and PowerWorld's dynamic link libraries (DLLs) [7]. These drawbacks make power system models virtually inaccessible for proper scrutiny and assessment [8].

B. OpenIPSL: Modelica for Power Systems

In an attempt to alleviate the aforementioned drawbacks, a power system component library was developed as part of the EU funded FP7 *iTesla* project [9]: the iTesla Power System Library (iPSL) [3]. This library was developed using Modelica [10], and was forked[1] by the authors into the Open-Instance Power System Library (OpenIPSL), currently under further development in the OpenCPS ITEA3 project [11].

Modelica was adopted for these libraries, because it is an open-source, object-oriented, multi-domain, and standardized modeling language [10]. This equation-based language provides a framework where each model is defined through an explicit mathematical representation of its behavior or by graphical block diagrams[2]. It further guarantees code scrutiny, and decoupling from integration solvers [12], letting the user pick a solver of choice from the ones available in a range of Modelica-compliant Integrated Development Environments (IDEs). Some of these IDEs are even free and open source software (e.g. OpenModelica [13], JModelica [14]), encouraging the adoption of modern development practices into the power system area.

The OpenIPSL provides a set of power system components for phasor time-domain simulations. These components have been validated against reference tools, mainly PSS/E and PSAT [15], to facilitate the adoption of the library by overcoming stringent social aspects of resistance to change [16].

[1]Forks are copies of a project, independently developed into a new software

[2]In Modelica, graphical models are only "a mask" to an underlying equation-based definition that specifies the behavior and interconnection rules.

C. Software Development Practices

Software engineering has been advancing substantially offering constantly evolving and new technologies, and improved development methods when compared to those of the power system community. These improvements seek to ensure the traceability of code changes, code quality and continuous shipment of revisions to the users. In particular, in recent years, software testing has emerged as a core component of the development of high quality code by systematically carrying out different types of trials [17]. It is these technologies and practices, which are not currently adopted in the power system community, that the authors seek to leverage in the development of the OpenIPSL.

D. Paper Contributions and Organization

This paper presents the methodology, software architecture, and prototype implementation that are used in the proposed continuous integration workflow in OpenIPSL. The paper is organized as follows: Section II discusses the development process and the issues encountered that motivated the work presented herein. Section III presents a brief overview of the tools comprising the software architecture and the detailed workflow for model testing. A use case using an exciter model is presented in Section IV. Conclusions are drawn in Section V.

II. Development Workflow and Identified Issues

The development of the iPSL began with the *iTesla* project. At the time, the development group had very little knowledge about Modelica, it grew organically, and there was no well-defined collaboration strategy. Most of the team members worked independently, delivering the work upon completion; there was no version control system in place, which resulted in many copies of the library, all at different development stages. Each newly developed model was validated only once through a software-to-software (SW-to-SW) validation procedure and integrated into the library [15]. Considering the simultaneous developments by different members, some models or subcomponents could be changed, without the team noticing it. Overall, the development process was slow and cumbersome.

As the library grew in terms of its number of implemented models and their complexity, and as the Modelica knowledge of the development team improved, it was decided that this library would be used in several other research projects that required power system simulation. Thus, if the library was to continue growing with additional models, the code would need re-factorization, and further developments would need to be done following an open source approach. Rapidly, the need for forking the original iPSL project appeared, and the OpenIPSL project was created to facilitate independent development of the original library with a research oriented focus.

The development workflow for the OpenIPSL was fully revamped to focus more on incremental changes to the library. As such the following practices were introduced:

- Version control system: To facilitate collaborative and simultaneous development within the team and with external actors, the library was moved to a git repository (explained later in the text) hosted on Github.
- Feature-branch: The development of new features in the library (new models, code re-factorization, etc.) is done in separate branches. Upon completion, the branch can be merged back into the stable branch (master branch).
- SW-to-SW validation: The library will now include a test system for each component to check the validation when changes are made to it or its subcomponents.

The new workflow allowed to speed up the library development by involving more people, and by delivering smaller, incremental changes, their integration was facilitated. The continuous integration of new features brings, however, the challenge of systematically testing the models in order to assess the impact, of proposed changes, on the models' validity. The aim of this paper is to present the solution implemented by the authors to address this point.

III. Continuous Integration Solution

The facilities that perform automated checking and validation code tests are referred to as *continuous integration* services. The solution implemented by the authors sought to automate the methodology previously used in manual procedures, and depicted in Fig. 1. The task automation allows the project members to merge changes in confidence that these won't negatively impact the library's stability.

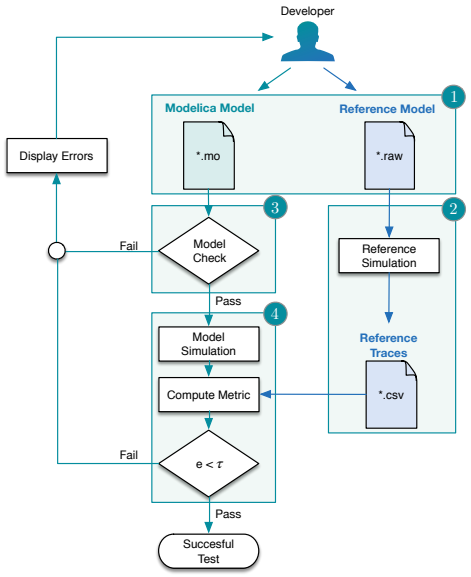


Fig. 1. CI Methodology

A. Testing and Validation Methodology

The procedure for testing and validating the models in the OpenIPSL is comprised by the following steps (see Fig. 1):

1) **Test system:** A small-scale test system using the component model to be tested is implemented in both Modelica and in a reference tool (e.g. PSS/E).

2) **Reference trace:** The model in the reference tool is simulated and the simulation output (i.e. trace) is recorded and stored for later assessment of the Modelica model.

3) **Model check:** In the first stage of the procedure, compliance with the Modelica syntax is checked for each model. The procedure also checks whether a model is under/over-determined (unbalanced in the number of independent variables and equations) [18]. In case a model fails at this stage, the procedure is aborted and an error message is returned.

4) **Model validation (behavior verification):** In the second stage of the procedure, a SW-to-SW validation is performed by comparing the simulation output of the Modelica model to its reference trace. The two datasets are time-aligned and resampling is applied w.r.t. the reference trace. The comparison is carried out on a preselected set of variables by computing the Root Mean Square Error (RMSE), see Eq. 1, where n is the number of data points, x_i and y_i are the simulation and reference data points at the time instant i. The RMSE serves as a quantitative metric to validate the model: below a chosen threshold, the model is considered validated.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1}$$

B. Architecture

The goal was to setup an automated system that would carry out the aforementioned methodology. To achieve the desired functionality, several free open source tools and services were combined. The resulting architecture is depicted in Fig. 2.

1) Technologies : A brief description and context for each employed technology are given below.

Git is the distributed version control system used for the development of the OpenIPSL. It allows to keep track of changes made to the source code, even in scenarios with developments made in separate branches, and provides the facility to merge several development branches together. **GitHub** is a hosting platform for git repositories used for the OpenIPSL project. It facilitates collaborative developments by providing a central facility accessible to team members and the rest of the GitHub community. Moreover, such environment provides additional facilities for documentation and issue tracking also accessible to the entire Github community. **Travis CI** is a web-based continuous integration (CI) platform integrated with GitHub. It can be configured to trigger testing routines every time a developer pushes [3] to the repository, or creates a pull-request [4] to the master branch. The result of the test is reported back to GitHub. **Docker** is an open-source program that lets users package an application and its dependencies into a standardized unit (i.e. container). With Docker, developers can create, run and ship applications in a lightweight package that will execute on any Linux machine. The authors have been using the "Docker Hub" service to host a pre-configured image with Python and OpenModelica installations used to generate the container necessary to execute the testing routines.

[3]operation by which a developer uploads a set of changes to the repository.

[4]operation to request the merge of a set of changes in the chosen branch.

2) Workflow: It was decided to build a CI service that would trigger for every commit and pull-request sent to the master branch of the OpenIPSL repository. The automated process depicted on Fig. 2 is described as follows:

- α) A pull-request is created on the OpenIPSL repository, triggering the code testing script on Travis CI.
- β) Upon start, the Travis CI platform clones the submitted code from the GitHub repository.
- γ) The Travis CI also pulls the image from Docker Hub and the reference traces to be used later for validation purposes from a local FTP server at SmarTS Lab.
- δ) Within the Docker container, Python scripts will start OpenModelica, execute the model checks, and carry out the SW-to-SW validation procedure.
- ϵ) In the case that a model does not pass the tests, the Python script and Docker container will exit with the flag "1", meaning that the test failed. Vice versa, they exit with the flag "0" when all of the models pass the tests.
- ζ) Test results are reported back to GitHub and, depending on the test result, the pull request will be allowed or blocked. Travis CI also preserves a snapshot of the process, which can be used to diagnose potential failures.

IV. Illustrative Example

In this Section, errors will be intentionally introduced in the code of one exciter model of the library to demonstrate how the CI can support the code reviewing steps by automatically detecting errors in the code submitted.

A. Test Model and Testing Case

The errors will be introduced in the Modelica code of the IEEEX1 excitation system. This component was originally implemented following the specifications given in the PSS/E manual [19], and in the IEEE standard [20]. The implementation of the excitation system is shown on Fig. 3.

The model was originally validated against PSS/E by using a small-scale power system network[5] depicted on Fig. 4. In this Single Machine Infinite Bus (SMIB) test power system model, the IEEEX1 excitation system is connected to a generator. In order to excite the dynamics of the system, a three phase-to-ground fault is applied at the *FAULT* bus at $t = 2$ s for 150 ms. The following set of variables will be evaluated: generator excitation and terminal voltage, and active and reactive powers produced by the generator.

B. Synthetic Testing

For illustrative purposes, a set of errors will be introduced intentionally in the IEEEX1 model. First, a **syntax error** is introduced in the form of erroneous parameter naming. In the instantiation of the IEEEX1 model, the code of the modifiers will erroneously attempt to declare the value of the exciter's gain using the name `KA` (the correct name is `K_A`). This error is expected to be detected in the first stage of the workflow.

[5]Observe that power system tools can only simulate components when embedded in a network. Note that this is not required for Modelica models.

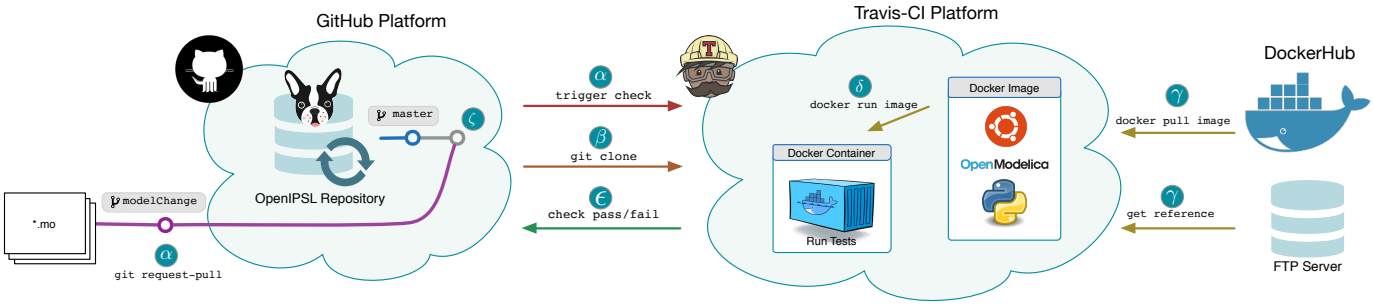


Fig. 2. Architecture of the proposed continuous integration methodology

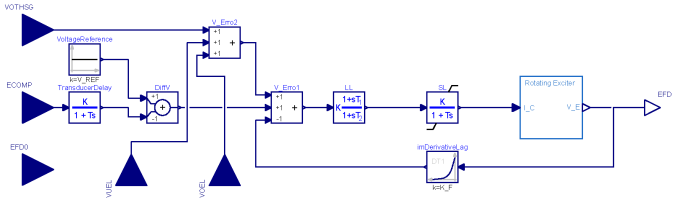


Fig. 3. IEEEX1 excitation system model

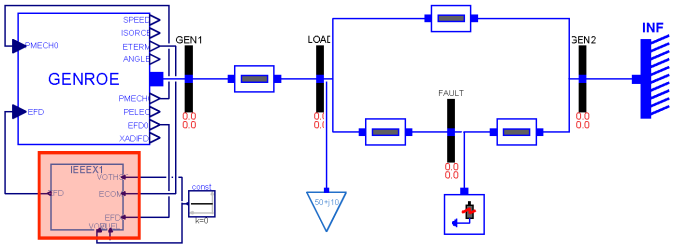


Fig. 4. SMIB test system for the IEEEX1 exciter model (highlighted in red)

In a second test, a **model error** is introduced in the IEEEX1 model by changing the operator of the feedback loop in the model. In the original model, the excitation field voltage (EFD) is subtracted to the reference voltage, instead, it will be added (a "+" will replace the "-") . This error is expected to only be detected at the second stage of the validation workflow.

C. Synthetic Testing Results

The errors described in the previous Section were successively pushed to the OpenIPSL repository to trigger Travis CI. The results received from the CI server are shown next.

In the first stage of the workflow, all the models of the library are checked. When the CI server checks the IEEEX1 model with a **syntax error**, it produces the console output shown in List. 1. Note that the error can easily be identified from the error message displayed in the console output.

In the second test, the IEEEX1 model containing a **model error** is pushed to the repository. The first validation stage is passed, as there are no syntax errors. In the second stage of the workflow, the model does not pass the SW-to-SW validation test. As shown in List. 2, the RMSE values exceed the chosen threshold of 10^{-3}. In particular, the RMSE of the excitation

Listing 1. CI output for a syntax error test

```
[/OpenIPSL/OpenIPSL/Examples/Controls/PSSE/ES/IEEEX1.mo
     :26:3-41:80:readonly] Error: Variable iEEEX1_1: In
     modifier (KA = 75), class or component KA not found
     in <OpenIPSL.Electrical.Controls.PSSE.ES.
     IEEEX1$iEEEX1_1>.
==================== Check Summary ====================
Number of models that passed the check is: 265
Number of models that failed the check is: 1
```

Listing 2. CI output for a modelling error test

```
[TEST]: OpenIPSL.Examples.Controls.PSSE.ES.IEEEX1
Simulation time: 1.092331523
Signal RMSE values follow:
Q - 0.283342688434
P - 0.100032711858
ETERM - 0.190655371519
EFD - 3.94786543302
```

field voltage stands out in terms of magnitude. This type of metric can help developers locate the source of errors.

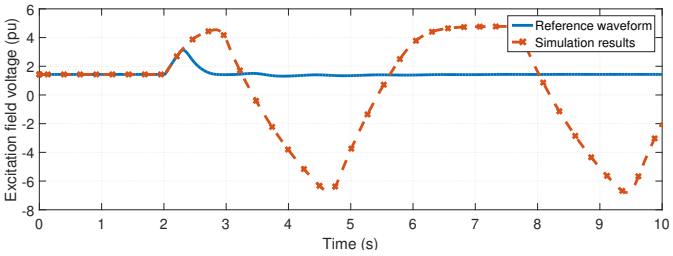
For illustration purposes, a comparison between the reference trace and the Modelica simulation was plotted from the available signals (see Fig. 5). Figure 5a shows the traces associated with the model error test. It can be observed that the large difference in the two signals is reflected in both RMSE value and their plot. Especially, it is very far from the results obtained when the model is free from errors (see Fig. 5b).

When the tests are carried out on the CI platform, the results are reported back to GitHub. In the current configuration, a successful test is required to merge the changes in the master branch. Therefore, in case of unsuccessful test, GitHub will prevent the merging and encourage the developer to submit corrections to the faulty code, as illustrated in Fig. 6a.

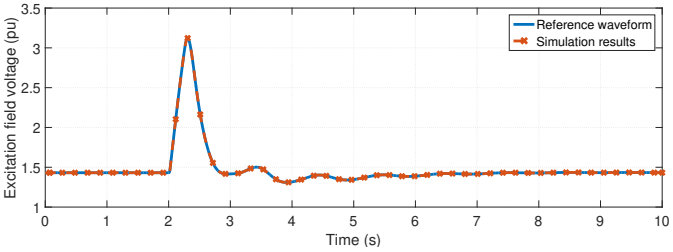
On the other hand, when all changes submitted to the repository pass the validation tests, the CI platform will send a green light back to GitHub, as illustrated on Fig. 6b. The developer can then merge the changes in confidence.

V. Conclusions

The increasing size of power systems and complexity of their component models require new modeling and simulation technologies that will aid engineers gain common understanding of the overall system behavior. Currently available simulation tools do not allow for unambiguous dynamic model exchange. In addition, they do not explicitly ensure repeatability of simulation results nor a consistent behavior of their models from one version to the next.



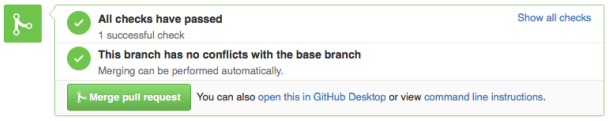
(a) Unsuccessful test



(b) Successful test

Fig. 5. Simulation results and the reference trace comparisons

(a) Unsuccessful test



(b) Successful test

Fig. 6. GitHub interface with the Travis CI showing test results

The methodology and software architecture for continuous power system model development presented in this paper is an attempt to bridge the gap in software development practices found in software engineering and those commonly used for power system modeling and simulation. The systematic approach for model checking, verification and validation will not only contribute to eradicating common development mistakes, but also allow to engage the community in contributing to the development of the OpenIPSL library by facilitating the integration of other's work.

In the future, the development and testing of Modelica models may become of great importance because it could affect simulations of power systems on multi-TSO, regional and even continental level, especially now that Modelica has been adopted as the language to be used in the definition of dynamic models in the CGMES v2.5 standard [21].

A. Future Work

The proposed architecture is a prototype implementation yet to be fully deployed on the OpenIPSL. Upon deployment, other functionalities will be considered, e.g. automated test case generation, new validation metrics, etc. The configuration files for the testing infrastructure are stored in the same repository, inviting further developments from the community.

Acknowledgment

The work was supported in part by Vinnova through the ITEA3 Project 14018-OpenCPS; Statnett SF, the Norwegian TSO through the SYMPTOM project; the StandUP for Energy Collaboration Initiative; and the EU funded FP7 project *iTesla*.

References

[1] "Regulation (EC) No 714/2009 of The European Parliament and of the Council of 13 July 2009." [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:211:0015:0035:EN:PDF

[2] F. Milano and L. Vanfretti, "State of the art and future of OSS for power systems," in *2009 IEEE Power & Energy Society General Meeting*. Institute of Electrical and Electronics Engineers (IEEE), jul 2009.

[3] L. Vanfretti, T. Rabuzin, M. Baudette, and M. Murad, "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations," *SoftwareX*, may 2016.

[4] F. Milano, *Power System Modelling and Scripting*, ser. Power Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[5] E. Z. Zhou, "Object-oriented programming, C++ and power system simulation," *IEEE Transactions on Power Systems*, vol. 11, no. 1, pp. 206–215, Feb 1996.

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

[7] "PowerWorld Documentation - Transient Stability: User Defined Models." [Online]. Available: http://www.powerworld.com/knowledge-base/user-defined-model-documentation

[8] F. Milano, L. Vanfretti, and J. C. Morataya, "An Open Source Power System Virtual Laboratory: The PSAT Case and Experience," *IEEE Trans. Educ.*, vol. 51, no. 1, pp. 17–23, Feb 2008.

[9] "iTesla: Innovative Tools for Electrical System Security within Large Areas." [Online]. Available: http://www.itesla-project.eu/

[10] Modelica Association, "Modelica ® - A Unified Object-Oriented Language for Systems Modeling - Language Specification," 2014. [Online]. Available: www.modelica.org

[11] "OpenCPS: Open Cyber-Physical System Model-Driven Certified Development." [Online]. Available: https//www.opencps.eu/

[12] P. Fritzson, *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Hoboken, NJ, USA: Wiley-Blackwell, sep 2011.

[13] P. Fritzson, P. Aronsson, H. Lundvall, K. Nystrom, A. Pop, L. Saldmli, and D. Broman, "The OpenModelica Modeling, Simulation, and Software Development Environment," *Simul. Notes Eur.*, no. 45, 2005.

[14] J. Åkesson, K. E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with Optimica and JModelica.org Languages and tools for solving large-scale dynamic optimization problems," *Comput. Chem. Eng.*, vol. 34, no. 11, pp. 1737–1749, Nov. 2010.

[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 *56th Conf. Simul. Model. (SIMS 56)*, nov 2015, pp. 105–112.

[16] L. Vanfretti, T. Bogodorova, and M. M. Baudette, "A Modelica Power System Component Library for Model Validation and Parameter Identification," in *Proc. 10th Int. Model. Conf., Lund, Sweden*, mar 2014, pp. 1195–1203.

[17] Atlassian. Agile development. [Online]. Available: https://www.atlassian.com/agile/developer

[18] H. Lundvall, P. Bunus, and P. Fritzson, "Towards automatic generation of model checkable code from modelica," 2004.

[19] "PSS/E 30.2 Program Application Guide Volume II," November 2005.

[20] "IEEE Recommended Practice for Excitation System Models for Power System Stability Studies," *IEEE Std 421.5-2005 (Revision of IEEE Std 421.5-1992)*, 2006.

[21] ENTSO-E, "Common Grid Model Exchange Standard (CGMES)." [Online]. Available: https://www.entsoe.eu/major-projects/common-information-model-cim/cim-for-grid-models-exchange/standards/Pages/default.aspx