A Modelica-Based Execution and Simulation Engine for Automated Power System Model Validation

Francisco Gómez
KTH Royal Institute of Technology, Sweden

Luigi Vanfretti
KTH Royal Institute of Technology, Sweden
Statnett SF - Research & Development, Norway

Svein Harald Olsen
Statnett SF - Research & Development, Norway

***Abstract*—Power systems model information exchange and the simulation of user defined models is a challenge issue for TSOs due to the use of different simulation software with specific data format and models strongly coupled to the simulation software numerical routines. The use of Open Source software for simulation and modelling can help to decouple these models. This work describes the design of a simulation tool following the Free/Libre Open Source Software philosophy to simulate equation-based models and implement a standard data format for model information exchange.**

***Index Terms*—Simulation Software, Open Source Software, OpenModelica, JModelica.**

I. Introduction

Future operations of the pan-European electricity transmission network, thus favouring increased coordination and harmonization processes for the operation procedures of transmission network operators. The presence of renewable development and integration is increasing. The development of power grids for an efficient security assessment, in particular regarding new devices FACTS, HVDC, wind generation, solar generation. The increasing cross boarder connections and energy market integration of the TSOs power grids requiring more coordination.

The FP7 iTESLA Project provides a tool to coordinate the design of system protection schemes between different TSOs and also provide recommendations on the use of renewable generation in defence plans. The coordination of different TSOs requires providing support for a wide range of relevant formats. Data models used in simulation may be different, using different data formats and different models are used for on-line and for off-line simulations. However the data may be the same. Three important facts are identified in terms of information exchange coordination: First, TSOs should be able to easily import the data to be assessed. the FP7 iTESLA Project context suggests that the use of CIM format should play a major role in the definition of static data. Second, The use and acceptance of user defined models in an open format in order to allow the modelling of specific and new devices is also expected. Design, validation an improvement of common models could then be facilitated. The use of Modelica language for modelling these user defined models provides the choice of make this exchange of model information open for TSOs. Third, TSOs also expect openness to plug in their own modules. Clear interfaces need to be defined to be able to launch private computations modules.

This work describes a software architecture developed with Open Source software, such as Modelica, JAVA and Python, for the simulation and analysis of power systems components and models. The architecture also includes the FMI standard technology in order to simulate models developed in other simulation tools. The proposed work is organized as follows: in Section II the use of Open Source software in simulation studies is explored. In Section III the requirements and the design of an open source simulation software are described. Section IV presents a detailed description of the simulation core of the proposed architecture and in Section V results of the application of the simulation core are shown. Finally, Section VII explores new requirements for improving model simulations, within the FP7 iTESLA Project context.

II. Background on Simulation and Software Development models

A. Simulation Software

According to [1], modern simulation software should be conceived to work with discrete-event and continuous systems. Such software will consider constructs with which to build models, a simulation engine to calculate a model's dynamic trajectories and means for control and observation of the simulation as it progresses. Moreover, a simulator of continuous systems approximates the solution to a set of differential equations, the choice of integration method and requirements for accuracy and precision. A discrete-event simulation executes events scheduled by its components in the order of their event times. The simulation engine produces dynamic behaviour from an assemblage of components. When talking about wide-area power system (WAPS) simulation, is important to clearly define which data and how it should be exchanged between components to produce the general

dynamic behaviour. In particular, the exchange of data may be performed within parallel simulation of smaller power system models. In this case, good scheduling techniques are required to ensure that the memory allocation shared by models are accessed in the right way, avoiding conflicts when writing in memory the piece of data produced by one model that another model will read afterwards [ref Mike Zhou].

For continuous-time simulation, software development is easier than for discrete-event simulation. Continuous-time simulation deals exclusively with relationships between real variables, allows simple programming when the equations are in proper form and suitable numerical algorithms are available. However, the software should be built keeping a clear separation between the model and the solution algorithm. In many proprietary power system simulation software is common practice to have the integration of the solution algorithm strongly connected to their own models. This approach has several back draws, such as limitations and extensibility and portability of the model for execution in other simulation engines. In contrast, the equation-based object-oriented Modelica language offers a strict separation between the model and the solver, as well as model portability thanks to its standardized language definition [ref Modelica MSL]. This allows to develop power systems simulation tools with the ability to add new algorithms to existing model structures without the need of modifying the models.

B. The choice of Free and Open Source Software

Proprietary software, being the de facto choice in power system simulations, is perceived as well-tested and more computationally efficient with respect to other tools. Some of these software are powerful in one particular type of analysis or only focused in few analysis methods, with the use of particular data format and data models. However, there are several drawbacks of proprietary software. The first to consider are license agreements restrict the use of proprietary software by imposing economic or other limitations [2]. Free/Libre and Open Source Software (FLOSS) has developed as an alternative to the proprietary software development model. Through different experiences, the FOSS model has shown to produce reliable, secure and efficient code. This fact has drawn recently some attention from the power systems community, resulting in the GOOSA effort [ref GOOSA]. However, previous to this, FLOSS has been more limited to education and learning activities and generally conceived for being used in power system courses [3] even though they provide almost the same functionalities as commercial software. FLOSS can be exploited in the power system field to provide tools allowing engineers to change or implement new code, evaluate new techniques and algorithms and guaranteeing freedom and liberty to exchange information [2]. Emerging challenges in power systems may be better approached through the FLOSS model than the traditional techniques found in proprietary software.

The Modelica language allows engineers to exchange models down to the equation-level found in different Modelica or FMI compliant simulation tools.The Open Source Modelica Consortium (OSMC) provides OSS librares of different fields, e.g.: thermal models and mechanical models. The FP7 iTESLA Project is developing a library of power system components compliant with Modelica. This creates a choice for power system model exchange of information, even those that may be the same in a huge variety of proprietary tools. Simulation tools and API based on Modelica, e.g.: OpenModelica, JModelica [ref modelica tools] facilitates the development of well-known industry models. The use of Modelica also permits developers and engineers to modify or add new components and equations and share these modifications, within the same model. The proposal architecture exploits FLOSS and will adapt the FLOSS development model upon release.

III. Simulation Software Architecture

A. Software Functionality Requirements

The software should provide a solution for simulating together and analysing power system component models that are part of wide-area power systems. To achieve this solution the software tool has to be able to:

- Initialize of the model a steady-state snapshot or a power flow solution.
- Change the configuration of the time-domain simulation solvers or the numerical algorithm itself.
- Simulate power systems models developed by different software tools,
- Use the Modelica and FMI standards [4] compliant simulation compilers. Moreover, the software should be open to plug-in other proprietary or FLOSS simulation compilers,
- Use Modelica as the base modelling language. The tool also has to be able to work with other models stored as Functional Mock-up Units (FMU) from FMI technology.
- Use state-of-the-art algorithms for signal processing, validation of models and calibration of parameters.
- Be able to operate using scripting functionality and also provide an easy to use Graphical User Interface (GUI) to perform the same functionality.
- Store simulation outputs and statistics metrics in an open source format, that can be used by commercial and non-commercial analysis tools (e.g.: MATLAB and Mathematica).

B. Architecture Design

The proposed architecture deals with input data from Modelica models or FMU files containing models from other software. The architecture is designed keeping modularity in mind through different modules or engines that interact with each other with the exchange of data, which can also be used separately from each other (Fig. 1).

1) The Model Execution Engine (MEE) provides different scripts to implement the requirements of running time-domain simulations with different simulation compilers. As inputs the engine gets the models, the simulation options and a set of output variables that should be

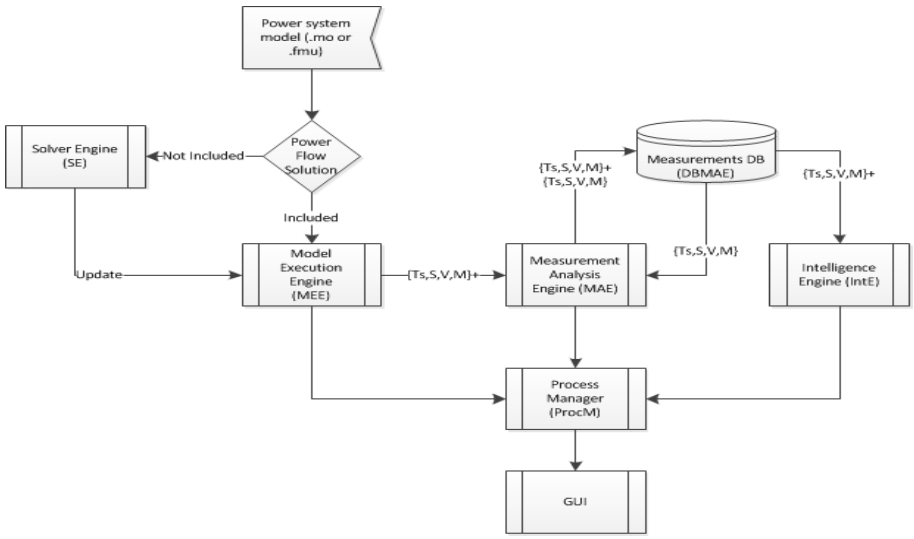

Fig. 1. General view of the software architecture. The main modules are MEE, MAE and IntE. The exchange of data between these modules deals with information about simulation outputs as time series (Ts) and other values such us arrays (V), matrices (M) or scalar (S)

stored and kept for analysis purposes. The outputs of the engine will be a set of time series (Ts), vectors (V), matrices (M) an scalar values (S). In this sense, we need to implement a standard data format that can handle these data and also be supported by other tools. The data format chosen has to follow the FLOSS philosophy (see section IV-B).

2) The Measurement Analysis Engine (MAE) implements different functions to analyse real measurements and compare them with the results of the simulations. Different statistics are calculated and stored in a Measurement DB, where the simulation outputs, measurements and the metrics that the engine has calculated are stored.
3) The Intelligence Engine (InteE) analyses which variables of the model should be calibrated in order to improve the simulation results to better match with the real measurements available.
4) The Steady-State Solver Engine (SSSE) implements mathematical solvers for computing power flow solutions, in order to set the proper initial values of the models we want to simulate. It gets as inputs power system models and update this models with the power flow solution, if needed. This is an important step of the simulation process because proper initial conditions help the MEE Modelica compilers to find a suitable equilibrium. In turn, this allows obtaining a valid response from the simulation of the models.
5) The purpose of the Process Manager (PM), is to provide easy to use commands and a GUI in order to perform all the functionalities described in the requirements.

IV. Implementation of the MEE

Behind the software architecture there is a large effort in the design and implementation of the modules and their functionality and data transfer between the engines and PM. The first engine implemented is the MEE, which is described below. It has been implemented using principles from Model-View-Controller design [ref].

A. Model Execution Engine

This engine performs time-domain simulations of wide-area power system models. Simulations are performed through the use of non-proprietary compilers: the OpenModelica Compiler (OMC) [5] and JModelica [6], however the design of the engine allows using other compilers as plug-ins into the MEE. We choose JModelica to simulate models from .fmu files. With this, the requirement of being able to simulate both Modelica models and FMU models is fulfilled.

From Fig 2, the JAVA language is used for implement the GUI and the PM engine. With this, the user can select the models he wants to simulate, and also manage and change configurations of the models and the simulation compilers. This data is stored in properties files. Fig. 2 shows how JAVA module and Python module interact with data, storing and loading results from the corresponding files. The interaction with Jython [7] allows the JAVA module to use Python scripts and classes, meanwhile both JAVA and Python modules use the HDF5 API to handle data. We use the Python [ref Python] language as the main programming language for the MEE. Python has a variety of libraries that allowing to implement data processing from files and specific libraries for running and executing the functionalities of the simulation compilers are available in this language. The MEE implements different python scripts specific for each compiler.

A set of classes have been developed to manage all the data needed. With the use of Jython it is possible to use the same class structure both in JAVA and Python. The results of the simulation are stored in files, using the HDF5 format, and this data is read by the PM engine for building any kind of reports of the simulation. With the use of JAVA and Python, PM

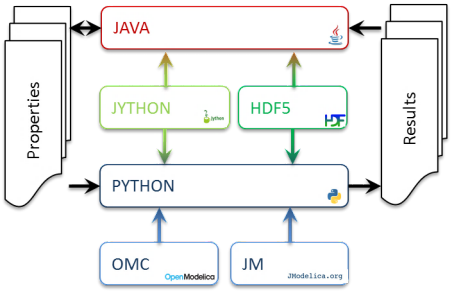

Fig. 2. Detail of the implementation of the Model Execution Engine. Here, the interaction between modules and simulation compilers and data is shown

and MEE engines are implemented following Object Oriented Programming. This allows to decouple the functionality of the engines and to build scalable and modular software. Fig 5 shows the internal work flow of the engine. Managing the outputs from simulation compilers requires some effort in developing functionalities to unify these outputs. To fulfil the requirements of our software, we use the HDF standard, in its latest revision HDF5, to create and store definitely all the simulation outputs. HDF5 is a standard data format, supported by different commercial tools as Matlab [8], for storing scientific data.

With JModelica, retrieving outputs values is straightforward because its API provides functions to handle data directly. But, managing OpenModelica outputs is more challenging, it requires extra programming to extract values from resulting .mat files. ModelicaRes [9] is an open-source tool, developed in Python, which provides a good API for managing the outputs produced by OpenModelica and Dymola result files.

The use of ModelicaRes and the HDF5 API will helps to store data as proposed in our conceptual model (Fig. 3) in a well organize tree structure within datasets and groups of data [10]. The data to be stored in the HDF5 files should be the outputs that the user wants to analyse later on with the use of the functionality of the MAE. In this case, the main outputs that need to be stored are the most common in power System analysis: Active Power (P), Reactive Power (Q), Voltage (V), Current (I) and angles. Nevertheless the design of the software allows to store other kind of data.

B. Formating Outputs

V. Proof of Concept

First, the performance of the proposed implementation is assessed by simulating of a very simple model. This test was performed to ensure that the implementation of different functionalities are properly functioning with a small-scale model, before simulating larger models

A. Test Power System Model

A Single Machine Infinite Bus (SMIB) model is used for testing. One can find different representations of the this simple model. A representation composed by one generator connected to an infinite bus through a transmission line and two buses where will be used (Fig. 4).

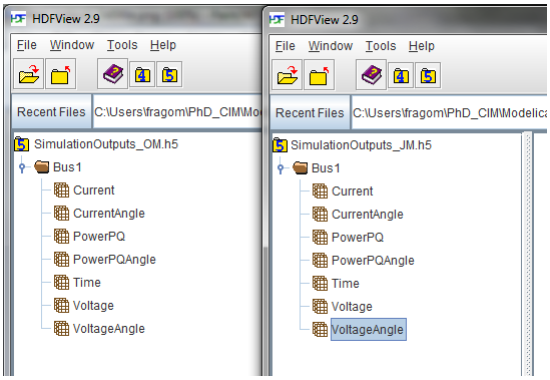

Fig. 3. HDF5 viewer screen shot of simulation outputs.

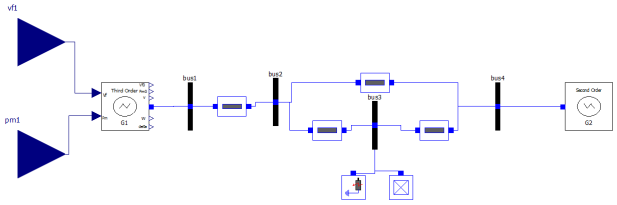

Fig. 4. OpenModelica representation of the Single Machine Infinite Bus Model for testing the simulation environment.

B. Engine Workflow

Two scripts in Python have been implemented for simulating the same model, in parallel. Both scripts implement the work flow detailed in Fig. 5. On one hand, the OMC simulates directly the SMIB model from a Modelica file (.mo). Meanwhile, JModelica first translates the model into a FMU and then simulates. JModelica is also capable simulating a FMU model which has been previously translated by another tool (e.g.: Dymola translate the model to a FMU and then JModelica simulates it).

C. Results

For this experiment the initial conditions of the model have been set manually, directly into the model, from a power flow solution calculated with the same model implemented in PSAT [ref PSAT]. To prove that the results from open source compilers are acceptable, the model has been simulated with Dymola [11] (Fig. 6). The resultant outputs are stored in the HDF5 format. From JModelica the storage is straightforward using the JModelica API to obtain results. From OpenModelica ModelicaRes API was used to parse the results into the HDF5 file.

VI. Discussion on Architectural Impact and Adoption

This section is introduced to discuss important aspects of the proposal architecture when talking about performance of the simulations and how TSOs can adapt the architecture in their systems.

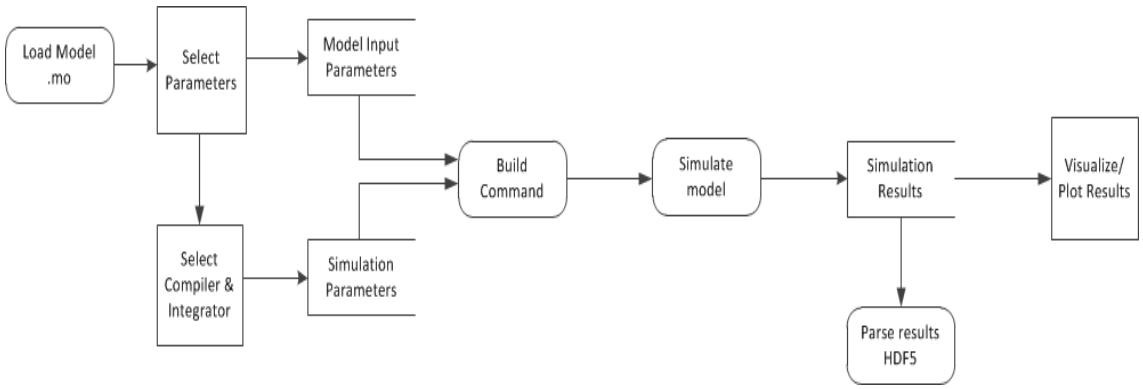

Fig. 5. Gane-Sarson diagram showing the data flow of the MEE. Rounded boxes are processes, squared boxes represent user interaction and open boxes represent data storage in memory or disk

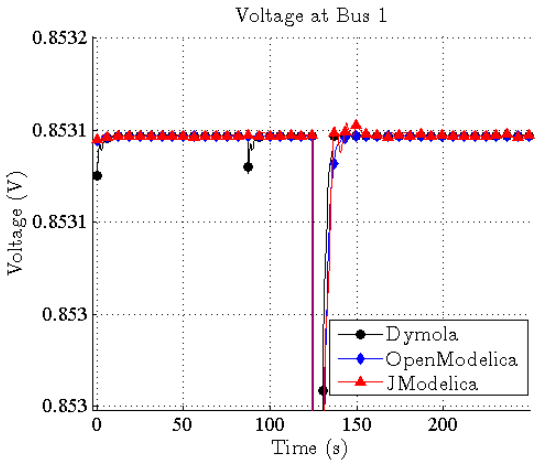

Fig. 6. Simulation results from JModelica and OpenModelica. The outputs are compared with the output produced by Dymola.

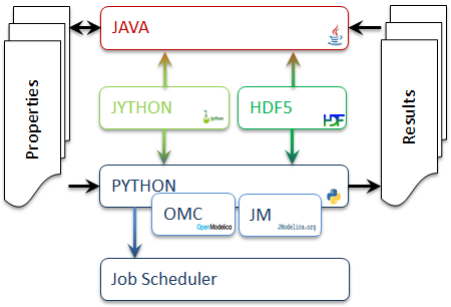

Fig. 7. Detail of the implementation of the Model Execution Engine, with a job scheduler. With the scheduler, the engine is able to execute different simulation compilers among available cores.

A. Architectural Impact on Simulation Performance

As a self-contained software implementation, the architecture will consider the implementation of a process scheduler so that the software can exploit all available cores in a single computer (Fig. 7). Simulations can be ran concurrently through multiple processors in a multi-core deployment, this is because only the Model Execution Engine would be implemented within a scheduler. For example, in an 8 core machine, 8 simulations can be run concurrently; this can be used when assessing different contingencies for a specific operation condition. However note that the overall speed of each simulation itself will be limited to the available hardware in a specific machine. Finally, observe that all other architectural components in Fig. 1 do not affect simulation speed, because the different modules are decoupled from the simulation. To adopt the proposed approach in this paper within the iTesla platform it would require to adapt the Model Execution Engine to the Computation Manager of the iTesla platform (Fig. 9). The best way to achieve this adoption is to use the iTESLA computation API within the Python module of the MEE (Fig. 2)and include the modelica compilers in computation modules layer.

B. Architectural Adoption by the TSOs

As a self-contained software implementation, in order to minimize the effort on the adoption of the suggested approach by a TSO, the straightforward route would be to develop a self-contained translator from the TSO specific format into Modelica. It is difficult to determine how much effort is required to build a specific translator, because this depends on the complexity of the TSO specific format. However, the iTesla project proposes an integrated approach and offers translation facilities.

Commonly, TSOs use proprietary SW tools for their simulation needs and are not familiar with Modelica. For an industrial adoption of the approach proposed in this paper, the iTesla platform offers an integrated solution where no previous knowledge of Modelica is required. Observe from 8 that the iTesla platform has a Data Manager. One of the functionalities of the Data Manager is to provide translators from common proprietary tools such as Eurostag and PSS/E, into a Dynamic Data Manager that links the proprietary models to a Modelica definition. This allows to translate an entire model, for example in Eurostag, into the Modelica language to be used for simulation. The data manager also takes care of building the contingencies and actions that should affect

the model during simulation, and configures the Modelica model to execute these actions at run time. Thus, there would be substantially low effort needed from a TSO to adopt this approach, provided that the iTesla platform is utilized.

VII. Conclusions and Further Work

This work has explored requirements for a scalable and modular simulation software architecture and how open source tools can be used for its implementation. The design of the architecture presented was conceived for complete analysis of power system models: running model simulations, validating simulation outputs from real measurements, and identification and calibration of particular models parameters that will adjust the model to match the measurements.

Towards the implementation of this architecture the Model Execution Engine has been presented which is responsible for performing time domain simulations. The design allows parallel simulation, through the execution of different simulation compilers at the same time. To fully exploid this benefit the simulation engine will be extended with a scheduling technique to fully utilize multiple compilers and to manage different processing time of both compilers. The OpenModelica compiler is suitable for simulating Modelica models and JModelica allows to simulate Modelica models or FMU models translated from other simulation tools.

Next steps include the use of HDF format for storing real measurements. This will facilitate the implementation of the validation techniques within the Measurement Analysis Engine and the Intelligence Engine. However, several aspects of power system simulations using Modelica tools need to be addressed first. The first issue is to automatically and consistently provide the initial conditions, from power flow solution or a steady-state snapshot, in order to obtain valid simulation outputs. To automate this, two tasks will be carried out: a first task will consists on the implementation of the SSSE to solve the power flow of the models or import existing power flow solution from other sources, i.e.: CIM, PSS/E, Eurostag. The second and most important will be the use the UML representation of the CIM standard to translate the CIM data model into a Modelica model. CIM components for power systems have attributes for storing the values from power flow solution [12]

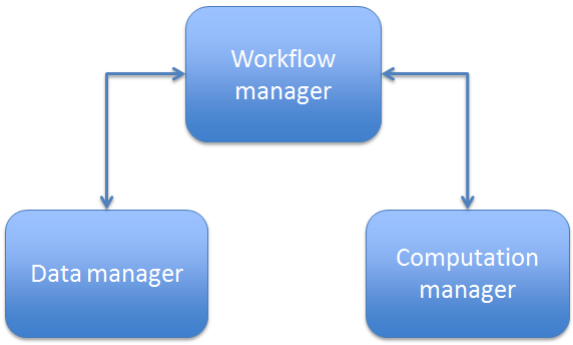

Fig. 8. Main modules of the iTESLA architecture for power systems models simulation and validation.

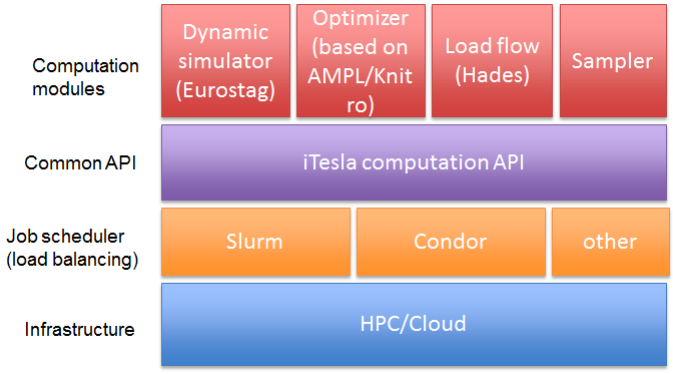

Fig. 9. Detail of the modules part of the iTESLA Computation Manager.

References

[1] J. Nutaro, "Building software for simulation: Theory and algorithms, with applications in c++," Wiley, Ed. Wiley, 2011.

[2] R. M. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Boston, Massachusetts: GNU Press, 2002.

[3] L. Vanfretti and F. Milano, "Facilitating constructive alignment in power systems engineering education using free and open-source software," *Education, IEEE Transactions on*, vol. 55, no. 3, pp. 309–318, Aug 2012.

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

[5] P. Fritzson, P. Aronsson, H. Lundvall, K. Nystrm, A. Pop, L. Saldamli, and D. Broman, "The OpenModelica Modeling, Simulation, and Software Development Environment," *Simulation News Europe*, vol. 44, no. 45, Dec. 2005.

[6] 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," *Computers and Chemical Engineering*, vol. 34, no. 11, pp. 1737–1749, Nov. 2010.

[7] J. Juneau, J. Baker, F. Wierzbicki, L. Soto, and V. Ng, *The Definitive Guide to Jython: Python for the Java Platform*, 1st ed. Berkely, CA, USA: Apress, 2010.

[8] MathWorks. (2014) Hdf5 files. [Online]. Available: http://www.mathworks.se/help/matlab/hdf5-files.html

[9] K. Davies, "Declarative modeling of coupled advection and diffusion as applied to fuel cell," PhD Dissertation, Georgia Institute of Technology, 2014.

[10] M. Poinot, "Five good reasons to use the hierarchical data format," *Computing in Science Engineering*, vol. 12, no. 5, pp. 84–90, Sept 2010.

[11] Dymola. Dynamic modeling laboratory. Dynasim AB Lund, Sweden. [Online]. Available: http://www.dynasim.org

[12] M. Uslar, M. Specht, S. Rohjans, J. Trefke, and J. Vasquez González, *The Common Information Model CIM*. Springer Berlin Heidelberg, 2012.

View publication stats