Modelica-Enabled Rapid Prototyping of Cyber-Physical Energy Systems Via The Functional Mockup Interface

Atiyah Elsheikh, *Member, IEEE,* Muhammed Usman Awais, Edmund Widl, *Member, IEEE,* Peter Palensky, *Senior Member, IEEE*

Austrian Institute of Technology, Energy Department, Vienna, Austria

{givenname.surname}@ait.ac.at

***Abstract*—Modelica has achieved a great success in the last decade. Universal modeling concepts, object-oriented facilities and large set of libraries in several physical domains allow for rapid prototyping of multidisciplinary applications. A larger community can benefit from these capabilities if Modelica-based components can be integrated into their favourite simulation tools. This work addresses the impact of transferring Modelica prototyping capabilities into different classes of simulation tools: general-purpose modeling tools, domain-specific tools and academical research-oriented simulation environments. In particular, it shows that the realization of model-based research of cyber-physical systems shall benefit from the convergence of such efforts using the functional mockup interface.**

***Index Terms*—functional mockup interface FMI, functional mockup unit FMU, cosimulation , Modelica, agent-based modeling, GridLAB-D, HLA, TRNSYS**

I. Introduction

CYBER-PHYSICAL ENERGY SYSTEMS are facing rapid developments concerning energy resources, communication technologies, sensor devices and others. The variety of components out of which such complex systems are composed makes model-based research a challenging task. Components can be not only of physical types like e.g. power generation, transportation, networks, power consumption, cities and buildings but can be also of virtual nature like communication, statistically random behavior, controls with sensors, national politics, oil prices etc [1]. Many actual challenges can be viewed from two perspectives: specification and implementation perspectives.

From the model specification perspective, models need to be described using well-established specification covering many aspects like continuous-time, discrete events, statistics and artificial intelligence domains [2], [3]. From implementation perspective, these specifications need to get translated to simulation code adequately. Nevertheless, while a wide set of highly-advanced simulation tools exists each providing sophisticated functionalities covering a specific aspect of energy systems [4], [5], a tool that covers all possible mentioned aspects does not exist for practical reasons.

Consequently, investigating future cyber-physical systems of new technologies would certainly benefit from a comprehensive modeler-oriented methodology that is capable of performing the following tasks for the components of such complex systems that are modeled with various tools:

- rapid prototyping
- arrangement within well-defined hierarchies possibly according to hybrid paradigms
- distributed cosimulation in different (hardware) platforms
- exploiting high-performance computing resources

One step towards this goal is to enable the combination of many types of tools within a cosimulation platform. These kinds of tools, can be classified, among others, as follows:

1) General-purpose modeling languages
2) Domain-specific tools capable of modeling highly sophisticated aspects
3) Self-developed tools and methodologies from academia by which research-oriented questions are investigated

A natural approach is to exploit standardized cosimulation capabilities for enabling the usage of desired tools and hence combining the advantages of all underlying approaches. Additionally, flexible prototyping capabilities of new components would certainly enhance this approach. Here, we constructively suggest the Modelica language as a rapid prototyping platform for other simulation tools. In Elsheikh et al. [2], many features of the Modelica language were primarily examined in the context of complex energy systems. The main elements of such systems were abstracted in a model example. The prototyping capabilities of the Modelica language as well as were the advantages of employing Modelica were primarily emphasized in the context of complex energy systems.

In this work, the impact of transferring Modelica capabilities into other tools is addressed. This is discussed along three tools categorized according to the mentioned tools classification. The integration of Modelica into other tools is done via the functional mockup interface (FMI) [6]. In this aspect, FMI can be viewed as a medium for enabling Modelica technologies for other arbitrary simulation tools and hence extend the scope of their possible applications.

The rest of this work is structured as follows: Section II gives an overview of the Modelica language followed by Section III for introducing the FMI technology. Sections IV, V and VI demonstrate the advantages of integrating Modelica-based components into GridLAB-D, TRNSYS and HLA, respectively. Finally, Section VII provides a brief summary.

II. Rapid prototyping with Modelica

A. Background

Modelica [7] is one of the modern state of the art equation-based modeling and simulation languages. It is based on universal domain-independent modeling concepts adequate for multidisciplinary applications [8], [9]. Object-oriented facilities enabling encapsulation for component reuse and object inheritance for hierarchical modeling are fundamental characteristics of Modelica. The object oriented modeling philosophy relies on the fact that any system, no matter how complex it is, can be decomposed into a finite set of smaller components. Modelica can describe components corresponding to differential algebraic equations (DAEs) with intuitive language constructs for implicit equations.

The Modelica language was initiated 1997 as a specification language for exchanging models among different working groups. Many well-established and promising features of many existing modeling languages and developed concepts have been adopted. It was and is still subject to intensive discussions concerning its maintenance and development within the Modelica association (www.modelica.org). An international conference is organized every 18 months with ever continuously increasing number of participants from industry and academia.

B. Modeling concepts

The key concept behind Modelica is that it employs non-causal modeling concepts by which input/output relationship among components is usually absent. That is, data flow among model components needs not to be explicitly defined. Component variables are not necessarily supposed to be declared as inputs or outputs of each others[1]. A component is typically encapsulated with well-defined interfaces to the external world. These interfaces, called connectors, work as communication ports enabling the connection to other components. The connectors can be viewed as energy carriers usually characterized by two types of variables, flow variables (say E) and potential variables (say P). By connecting two identical connectors together, two types of equations are generated, sum to zero equations for flow variables and identity equations for potential variables, see Figure 1 [2]. The former type of equations emulates conservation laws present in physical domains, that is the sum of all flows of conserved quantities (e.g. energy) into and out of a component must be equal to zero. In this way, the modeling of large-scale systems becomes the task of dragging, dropping component icons and connecting them together. Resulting models are visually one-to-one map to the conceptual reality.

[1]Nevertheless, some causal standard constructs similar to those present in classical procedural languages are also supported. Modelica specification is flexible enough that also the classical block-diagram approach (e.g. as in Simulink) can be emulated, if needed

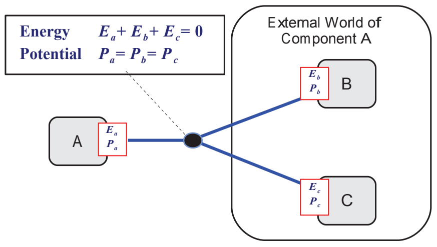
Energy $E_a + E_b + E_c = 0$
Potential $P_a = P_b = P_c$
External World of Component A
A
E_a P_a
B
E_b P_b
C
E_c P_c

Fig. 1. Connecting components in Modelica assembles common conservation laws of Physics

C. Modeling and simulation with Modelica

Components from different physical domains are implemented by identifying the right quantities for potential and flow variables. For example, typical quantities for flow variables are current, particles flow rate and heat flow rate for the electrical, hydraulic and thermal fields, respectively. The corresponding potential quantities are voltage, pressure and temperature, respectively. A significant advantage of Modelica is the presence of a large-set of free and commercial libraries in many physical domains. The Modelica association continuously integrates further libraries into an ever growing Modelica standard library (MSL). By developing sophisticated new applications, there is no need to start development from scratch. Many already implemented components and types can be reused.

Having constructed Modelica models, many Modelica-like simulation environments like Dymola [10], MapleSim (www.maplesoft.com), OpenModelica [11], Wolfram SystemModeler (www.wolfram.com) and others exist for modeling, compiling and simulating Modelica models. Such Modelica simulation environments are responsible for transforming graphical models into efficient simulation code using sophisticated symbolic algorithms for manipulating and simplifying resulting large-scale equation systems. Such generated equations are usually sparse, i.e. in each equation, only few variables are present. Consequently, modern DAE solvers exploiting the Jacobian sparsity are usually utilized for efficient and robust numerical integration. Moreover, additional capabilities for detecting and handling events are present. More features of Modelica are explicitly emphasized in [2].

III. Transferring Modelica via FMI

A. Background

FMI, a standardized unified model interface, is a result of the MODELISAR project (www.modelisar.com) aiming at improving the design and interoperability of simulation tools for embedded system applications. FMI is becoming the current trend for tools interoperability among simulation-based software. Currently, more than 40 simulation tools are supporting FMI (www.fmi-standard.org/tools).

A simulation tool supporting FMI is capable of exporting its models as functional mock-up units (FMU)s. A FMU is a zip file containing the following components:

1) The implementation of a model as a compiled C shared

libraries following a specific API in C, optionally accompanying the source code
2) A model-description XML file, containing among others a description of the model inputs, parameters, outputs, used types and physical units
3) Other optional data e.g. icon images, user interface specification and relevant documentation

Such FMUs can be simulated as stand-alone programs or imported by other FMI-supporting simulation tools.

B. *Supported FMI operations*

A FMU implements a model mathematically described by a hybrid ordinary differential equations system (ODE) as a mixture of continuous state and discrete variables, see Figure 2. Additionally, a set of algebraic equations depending on state variables x could be also present making the underlying model conceptually equivalent to a DAE of index one [12]. A typical cosimulation master importing FMUs as a slave black-box model would typically perform the following FMI operations on a FMU in the same order emphasized by Figure 2:

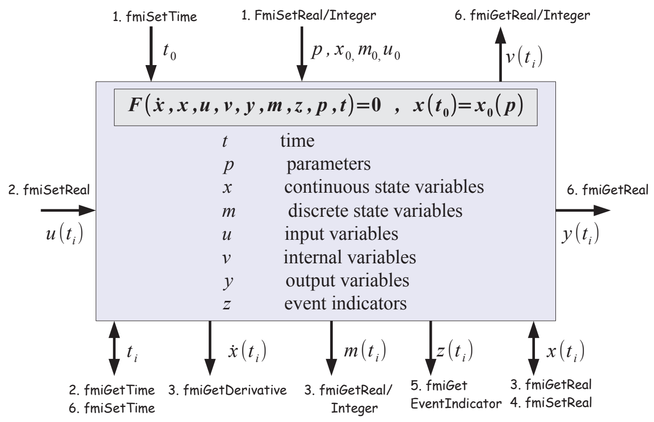

Fig. 2. Mathematical description of a FMU. Contents of FMUs can be retrieved and updated using naturally ordered elementary FMI calls

1) Initialization step: setting up start time t_0, model parameters p, start values $x(t_0)$ and input variables $u(t_0)$

Having an initialized FMU, the numerical integration is iteratively performed at discrete time steps $t_i, i = 1, 2, \dots$ as follows:

2) Pre-processing step: set step t_i and set the inputs $u(t_i)$
3) Integration step: update values of $\dot{x}(t_i), x(t_i)$ and $m(t_i)$ and perform numerical integration
4) Post-processing step: update state variables $x(t_i)$
5) Event handling step: report the presence of events if $z_j(t_i) * z_j(t_{i-1}) < 0$ and handle them adequately
6) Output steps: Compute the outputs $y(t_i)$ and other intermediate variables $v(t_i)$, if needed

and finally:

7) Finalization step: memory deallocation and processing of results

Step number 3 is done using either an external solver or an internal solver that comes with the FMU. The former case is performed if the FMU only supports FMI for model exchange (FMI-ME) while the latter is referred to as FMI for cosimulation (FMI-CO). In FMI-CO, additional FMI routines are provided for performing self numerical integration.

C. *Common advantages of FMI*

There are many drivers for considering the FMI technology within common as well as own self-developed simulation tools. Firstly, simulation tools can import model components implemented by other simulation environments capable of exporting their models as FMUs. Secondly, self exported FMUs can be imported by other modeling, simulation and analysis tools capable of importing FMUs such as PySimulator [13] and JModelica [12]1. Many tools for assisting the implementation, validation and simulation of FMUs exist such as the FMI library (www.jmodelica.org/FMILibrary), FMI SDK development kit (www.qtronic.de/en/fmusdk.html) and FMU compilance checker (www.fmi-standard.org/downloads).

In summary, although the original motivation behind FMI is to support Modelica-based developments for embedded systems, nevertheless, FMI can be viewed as a medium for transferring the Modelica capabilities into other modeling and simulation tools. Overall, this definitely enhances the multidisciplinary collaboration potentials among several working groups and specialists in several domains. In particular, several tools based on long decades of experiences and developments are definitely not easily reproducible in favourite languages.

IV. Modelica-based FMUs for Universal simulation languages: GridLAB-D

A. *Background*

GridLAB-D (www.gridlabd.org) is one of the leading developer-oriented open-source modeling environment for simulating discrete event-based systems [14]. This specification language is typically used for describing several aspects of energy systems like weather, market, power systems, grids, distribution automation modules and many others for power system analysis applications. Already a large repositories of such modules exist and are developed, utilized and exchanged among different research groups.

GridLAB-D adopts an agent-based simulation approach. Each agent type is characterized by a set of property states and actions for updating states. The specification of an agent type is realized by either one of the following:

1) An intuitive GridLAB-D specification languages (or)
2) A C++ class according to templates

Using the GridLAB-D language, a *module* of interacting agents can be specified. Each instantiated agent dynamically calculates next time point for updating its states. The simulation engine of GridLAB-D is responsible for tracking the states of agents, managing the communication among agents by advanced scheduling algorithms and controlling the progress of time through a universal global clock. The modeler can construct and efficiently simulate large-scale complex systems with thousands of interacting agents by organizing the agents into hierarchies of subsystems and working subgroups.

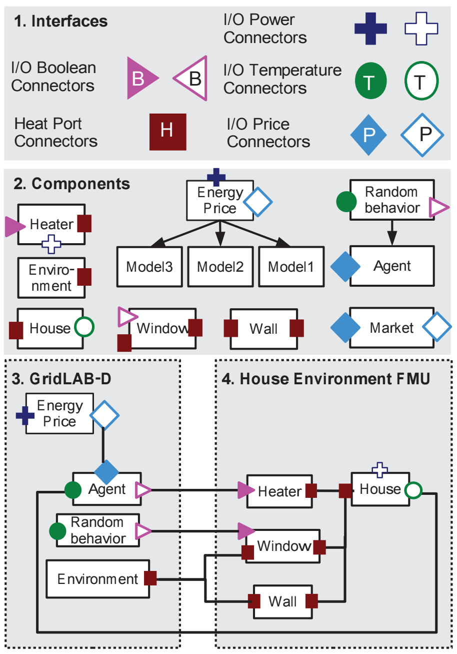

Fig. 3. Coupling GridLAB-D modules with a FMU component. The continuous-time part is modeled with Modelica and the discrete part is modeled with GridLAB-D

B. *Advantages*

Many advantages for integrating FMU-based Modelica components within GridLAB-D are visually demonstrated in Figure 3. The physical part of the model describes the thermal behaviour of an energy consumption unit (a house with a heater, a window and an isolation wall), cf. [2] for implementation details in Modelica. The discrete part implemented by GridLAB-D describes the external conditions influencing the power consumption. This includes a weather module, statistically random behaviour agent (opening/closing windows), an intelligent agent controlling the power consumption of a house (e.g. heater settings), influenced by a market agent supplying the energy prices.

1) Benefits for the GridLAB-D community: The benefits of the presented coupling can be directly extracted from this simple model. Both Modelica and GridLAB-D together can be used for efficient prototyping and simulation of complex hybrid systems. The benefits to the GridLAB-D community can be viewed as follows: when modeling cyber physical systems, the continuous part can be rapidly prototyped with Modelica. Moreover, despite of the fact that GridLAB-D is a simulation language, it does not inherit any capabilities for numerical integration of ODEs. On the other side, FMUs can be numerically integrated in a straightforward way, even if FMI-CO is not supported. FMI-ME design is adequately relevant for interfacing with common ODE solvers.

2) Benefits for the Modelica community: On the side, GridLAB-D is also a useful tool for FMU-based modeling. GridLAB-D can be considered as an easy-to-use user interface for setting and instantiating FMU-based components. Existing powerful statistical tools can be used for initializing FMUs using e.g. uniform, normal and exponential distribution among many others. In this way, statistical studies based on parameter variations can be performed on the fly. Another significant aspect is the powerful capabilities of GridLAB-D for high performance simulation of very large-scale systems that cannot be simulated with Modelica alone due to their complexity [3]. Easy-coupled large-scale systems can be decomposed into thousands of smaller interacting FMUs on a discrete simulation basis. The interactions among FMUs, their execution order and the exchange of data can be specified with GridLAB-D.

In summary, adjoining the non-causal modeling approach with the discrete agent-based approach results in a powerful modeling environment with a larger scope of applications, as done in Stifter and et al. [15].

V. Modelica-based FMUs for domain specific tools: TRNSYS

A. *Background*

TRNSYS [16] is a domain-specific tool for simulating the thermal behaviour within energy-efficient buildings among many other domain-related applications. It provides high-level facilities for constructing building models with comprehensive details including multi-zoning description (e.g. internal structure and rooms), exact architectural specifications and geothermal conditions (e.g. typical weather conditions, sun light directions and duration). TRNSYS additionally provides a wide extensive set of model components for controllers, electrical storages, HVAC, solar thermal collectors and many others. A sophisticated highly-detailed GUI is available for the modeling task.

TRNSYS is based on a block-diagram approach for modeling technical systems. Each component computes output variables from given input variables and parameters. Systems are assembled by connecting components together by which the inputs of some components are the outputs of others. For components implementing ODEs, the numerical integration can be performed with the TRNSYS simulation engine, the *kernel*. The kernel additionally employs a block-wise successive iteration method suitable for handling potential loops in the specified model.

B. *Developing new components*

The underlying modular architecture of TRNSYS separates between a model component specification interface, its implementation and the numerical integration process. The components are implemented as dynamically loaded libraries (DLLs) to be loaded by the kernel. This gives the opportunity for supplying user-implemented components not provided by the TRNSYS standard library. This is done by implementing standard template routine in C++ or Fortran. A typical routine implements a specific set of operations like:

- setting initial values and model parameters
- computing model outputs and state derivatives at given step-sizes
- post-processing and a finalization step

These operations are then queried by the kernel for performing the numerical integration[2]. While low-level implementation of small-size models could be a straightforward task, realizing large-scale physical models within a procedural language is a tedious task. Firstly, the developer needs to translate the model into a set of equations. This is not a straightforward task for complex large-scale models.

C. Advantages

Alternatively, the physical modeling can be rather done by Modelica and integrated into TRNSYS via FMI, see Wetter [17] for a comparison between Modelica and TRNSYS w.r.t. model development time. The required operations for realizing user-defined types in TRNSYS are ideally compatible with typical FMI operations. That is, it is possible to build FMU-based types for TRNSYS as done in Elsheikh et al. [18]. In this work, a single wrapper FMI-based code file is provided for integrating TRNSYS-conform[3] FMUs into TRNSYS. This has a lot of advantages briefly demonstrated in Figure 4. While TRNSYS already contains a comprehensive subset of

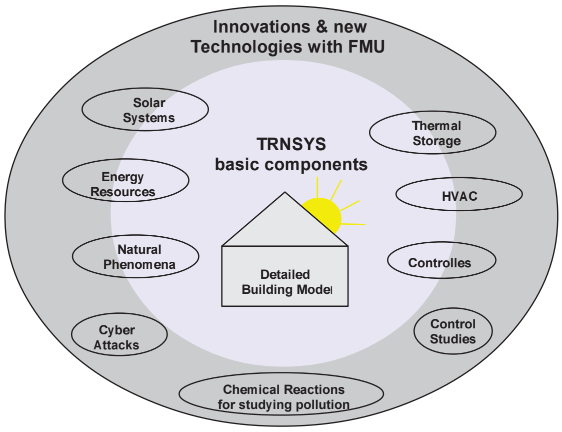

Fig. 4. Rapid prototyping of innovations and new technologies with TRNSYS and FMI. The internal region corresponds to applications that can be implemented with TRNSYS . The external regions corresponds to model components that are rapidly prototyped with Modelica.

model components, model-based investigation for innovative research studies of new concepts and emerging technologies can most benefit from the prototyping capabilities of Modelica. Advanced complex Modelica models can be integrated into basic TRNSYS types without low-level explicit implementation of the mathematical details. Specialized features exclusively present in some Modelica building libraries [19] can be utilized. Meanwhile, many sophisticated extensively tested model components in TRNSYS, that are not easily constructable with Modelica, can be exploited. That is, the horizon of applications scope by combining both sides certainly gets extended.

[2]The numerical integration can be also performed internally within the component

[3]Complete declaration for the public items are specified, the inputs and the outputs of the FMU and a parameterized start values for state variables, if required. Any Modelica model can be rewritten in a way to make the resulting FMU to be TRNSYS-conform

VI. Modelica-based FMUs for the HLA cosimulation environment

A. Background

The high level architecture (HLA) is a simulation interoperability cosimulation standard [20], designed to support distributed simulations with various different synchronization schemes. It has been used in large-scale applications of industry and defence simulations for more than a decade. Dynamic management, incremental design and development support are fundamental features of the HLA. In the HLA terminology a simulation component is called a ``Federate˝, while the whole distributed simulation is called a ``Federation˝. The most important functional aspect of the HLA is its runtime infrastructure (the RTI). The RTI works as a central communication server for all the individual simulations. The RTI is responsible for synchronizing the federates, providing them timely updates of the federation shared data structures. It also allows the federates to manage the creation, deletion and ownership of data structures. Hence, the HLA can be viewed as a platform supporting parallel discrete event simulations (PDES).

B. Advantages

By combining FMI into the HLA framework, a PDES platform enhanced with continuous-time simulations is established. The resulting framework is ideal for performing efficient simulations of massively large-scale applications and intensively expensive computations, e.g. energy consumption of a city with various energy resources. Such large simulations would also need flexible-demand of computational power to be assimilated. The idea of extending HLA with continuous-time simulation has been presented in Müller et al. [21] where discrete event network simulators were combined with electricity domain continuous simulators. Awais et al. [22] has presented a preliminary design of using the HLA with the FMI.

Figure 5 presents a corresponding hypothetical scenario of hardware configuration along with the RTI, which may enable a large simulation like this. Figure 5 emphasizes that

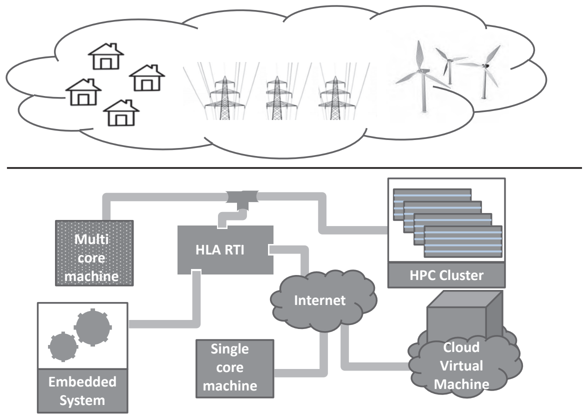

Fig. 5. A distributed simulation for analyzing the energy requirements of a big city

an HLA based simulation is completely network independent. Several machines within super-computer clusters are dedicated to simulate the energy producers, like wind mills, hydro electricity and others. Communication of data is simulated by discrete event based network simulators on a parallel computing machine. Actual devices as embedded systems are assisting the simulation of transfer of power and load regulation. The houses as energy consumption units are simulated on cloud virtual machines. In this topology, cloud-based computing enables dynamical computational power resources based on time-varying demands.

To summarize; combining FMU components within the HLA gives many advantages over simulations running as single processes. An efficient exploitation of parallel distributed simulation environment is more scalable, robust and flexible. Currently HLA is one of the reliable standardized interface for parallel and distributed simulation interoperability. Stipulating its flexibility with the power of FMU-based modeling enhances reusability and brings great benefits for model-based investigation of large-scale systems.

VII. Outlook

This work addresses the impact of transferring Modelica rapid prototyping capabilities into other simulation tools via the FMI technology. For that purpose, three different tools has been chosen as representatives of different classes of modeling and simulation environments. GridLAB-D, a universal modeling language, certainly benefits from enhanced prototyping capabilities from FMU-based Modelica components. Additionally, more descriptive power enhances the underlying agent-based discrete modeling approach. Numerical integration capabilities of FMUs are provided by defaults. Large-scale applications can be efficiently simulated by decomposing complex systems at weak-coupled points into hierarchical subsystems of interacting agents communicating on discrete-time basis. The scope of applications is extended by exploiting the best of both languages.

Enabling Modelica rapid prototyping to TRNSYS, a domain-specific tool, gives a promising approach for implementing new innovative technologies. FMU-based prototyped components can be independently tested and analyzed by a large set of FMU-based tools. Capabilities of domain-specific Modelica libraries can be integrated. These statements are also applicable to many domain-specific tools. HLA, a co-simulation environment for parallel discrete event simulators, provides a perfect ground for establishing high performance computing environment for massively large-scale applications of cyber physical systems. Continuous-time simulations can be integrated by enabling Modelica-based technologies within the HLA platform via FMI.

References

[1] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, “Seamless model-based development: From isolated tools to integrated model engineering environments,” in *Proceedings of the IEEE*, vol. 98, no. 4, 2010, pp. 526 – 545.

[2] A. Elsheikh, E. Widl, and P. Palensky, “Simulating complex energy systems with modelica: A primary evaluation,” in *DEST’2012, the 6th IEEE International Conference on Digital Ecosystems and Technologies*, Campione d’Italia, Italy, 2012.

[3] E. Widl, P. Palensky, and A. Elsheikh, “Evaluation of two approaches for simulating cyber-physical energy systems,” in *IECON’2012, the 38th Annual Conference of the IEEE Industrial Electronics Society*, Montreal, Canada, October 2012.

[4] C. Macana, N. Quijano, and E. Mojica-Nava, “A survey on cyber physical energy systems and their applications on smart grids,” in *Innovative Smart Grid Technologies (ISGT Latin America), 2011 IEEE PES Conference on*, October 2011.

[5] M. Ilic, L. Xie, U. Khan, and J. Moura, “Modeling future cyber-physical energy systems,” in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, 2008.

[6] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, “The functional mockup interface for tool independent exchange of simulation models,” in *Modelica’2011: The 8th International Modelica Conference*, Dresden, Germany, 2011.

[7] H. Elmqvist and S. E. Mattsson, “Modelica - the next generation modeling language: An international design effort,” in *ESS97: The 9th European Simulation Symposium*, Passau, Germany, 1997.

[8] F. Casella and P. Colonna, “Dynamic modeling of IGCC power plants,” *Applied Thermal Engineering*, vol. 35, pp. 91 – 111, 2012.

[9] F. Casella, F. Donida, and M. Lovera, “Beyond simulation: Computer-aided control system design using equation-based object-oriented modelling for the next decade,” *Simulation News Europe*, vol. 19, no. 1, pp. 29 – 41, 2009.

[10] D. Brück, H. Elmqvist, H. Olsson, and S. E. Mattsson, “Dymola for multi-engineering modeling and simulation,” in *Proceedings of the 2nd International Modelica Conference*, Munich, Germany, 2002.

[11] 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.

[12] C. Andersson, J. Åkesson, C. Führera, and M. Gäfvert, “Import and export of functional mock-up units in JModelica.org,,” in *Modelica’2011: The 8th International Modelica Conference*, Dresden, Germany, 2011.

[13] A. Pfeiffer, M. Hellerer, S. Hartweg, M. Otter, and M. Reiner, “PySimulator – A simulation and analysis environment in Python with plugin infrastructure,” in *Modelica’2012, the 9th International Modelica Conference*, Munich, Germany, September 2012.

[14] D. Chassin, K. Schneider, and C. Gerkensmeyer, “Gridlab-d: An open-source power systems modeling and simulation environment,” in *Transmission and Distribution Conference and Exposition, 2008. IEEE/PES*, april 2008, pp. 1 –5.

[15] M. Stifter, E. Widl, F. Andrén, A. Elsheikh, T. Strasser, and P. Palensky, “Co-simulation of components, controls and power systems based on open source software,” in *2013 IEEE Power & Energy Society General Meeting*, Vancouver, Canada, July 2013, accepted.

[16] S. A. Klein, J. A. Duffie, and W. A. Beckman, “TRNSYS: A transient simulation program,” *ASHRAE Transactions*, vol. 82, pp. 623 – 633, 1976.

[17] M. Wetter and C. Haugstetter, “Modelica versus trnsys – a comparison between an equation-based and a procedural modeling language for building energy simulation,” in *The 2nd SimBuild Conference*, Cambridge, MA, USA, August 2006.

[18] A. Elsheikh, E. Widl, P. Pensky, F. Dubisch, M. Brychta, D. Basciotti, and W. Müller, “Modelica-enabled rapid prototyping via TRNSYS,” in *BS’2013, The 13th International Conference of the International Building Performance Simulation Association*, Chambéry, France, August 2013, submitted.

[19] M. Wetter, “Modelica-based modelling and simulation to support research and development in building energy and control systems,” *Journal of Building Performance Simulation*, vol. 2, pp. 143 – 161, 2009.

[20] Simulation Interoperability Standards Committee *et al.*, “IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-IEEE std 1516-2000, 1516.1-2000, 1516.2-2000. new york: Institute of electrical and electronics engineers,” *Inc., New York*, 2000.

[21] S. C. Müller, H. Georg, C. Rehtanz, and C. Wietfeld, “Hybrid simulation of power systems and ICT for real-time applications,” in *3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Berlin, Germany, 2012.

[22] M. Awais, P. Palensky, A. Elsheikh, E. Widl, and M. Stifter, “The high level architecture RTI as a master to the functional mock-up interface components,” in *ICNC 2013 International Workshop on Cyber-Physical System (CPS) and its Computing and Networking Design*, San Diego, USA, January 2013.