A component approach for DEVS

Thomas Paris, Laurent Ciarletta, Vincent Chevrier

A component approach for DEVS

Thomas Paris[*], Laurent Ciarletta[*], and Vincent Chevrier[*]

[*] Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
firstname.name@loria.fr

July, 2018

Abstract

In this article, we are interested in the reuse of models to allow the design of complex system multi-models. For instance, smart-grid modeling needs models from at least three domains (electrical grid, IP network, control system). Many existing M&S tools are already dedicated to the design of such models. Then, it would be profitable to be able to reuse the models written in these tools to design complex system multi-models. The FMI standard, based on the export and import of models as software components, supports the exchange of models and the co-simulation between ODE and DAE based M&S software. The success of the FMI standard leads us to study the interests of a similar approach based on the DEVS formalism.

Keywords: Standardization, Co-simulation, DEVS, Reuse

1 Introduction

One of the main challenges in Modeling and Simulation (M&S) is the modeling of complex systems. A complex system is a system composed of several heterogeneous entities whose global behavior emerges from the interaction of these entities [Ramat, 2007].

Cyber-Physical Systems (CPS), such as smart-grids, are complex systems composed of multiple heterogeneous entities from several domains (e.g. power grids, IP networks and control systems for a smart-grid) [Vaubourg et al., 2016].

Modeling such systems requires the collaboration of experts from various domains. Each expert will bring a different yet complementary view on the system under study according to her specific skills. However, new multi-perspective and multi-domain approaches are needed for that purpose [Seck and Honig, 2012].

Multi-modeling and co-simulation seem to be promising ways to deal with this complex system M&S challenge [Gomes et al., 2017]. Multi-modeling consists in designing models for each entity of the complex system and in making these models interact to build a multi-model. Co-simulation consists in performing synchronization and data exchange between several simulators, each one handling the simulation of its own model. Although these methods are promising, they need: (1) to open the models, i.e. to allow their coupling and the design of the multi-model, and (2) to open the simulators, i.e. to enable the interaction of the simulators during a co-simulation.

In the hypothesis that the co-simulation and the reuse of existing models will be a widespread method in the future, the M&S community has to agree on a norm. In this article we propose a first thought to build a norm suited to discrete event system co-simulation. This idea comes from our experience on

MECSYCO (Multi-agent Environment for Complex SYstem CO-simulation)[Camus, 2015] which uses both DEVS (*Discrete EVent System specification*)[Zeigler et al., 2000] and FMI (*Functional Mockup Interface*)[Blochwitz et al., 2011].

On the one hand, we have the FMI standard for the particular case of dynamic systems designed with ODE (*Ordinary Differential Equation*) and DAE (*Differential Algebraic Equation*). FMI is a tool independent standard that supports model exchange and co-simulation between dynamic system M&S software.

On the other hand, DEVS formalism has many benefits for multi-modeling and co-simulation, notably its ability to rigorously integrate heterogeneous components.

Regarding the success of the FMI standard, and the benefits of the DEVS formalism, we propose the development of a standard similar to FMI but formally based on DEVS. As the development of a DEVS-based standard cannot be done without a common reflection within the DEVS community, in this article we only explain the motivations and the advantages of such a norm. An example based on the MECSYCO middleware illustrates our idea.

This document is organized as follow. The Section 2 reminds the multi-modeling and co-simulation requirements. The Sections 3 and 4 present respectively the FMI standard and the DEVS formalism, their benefits and limits. Then, the Section 5 sums up the previous sections to motivate the development of a DEVS-based component approach (in the sense of software component). The Section 6 presents a simple example based on the MECSYCO middleware. Finally, the proposal will be discussed and confronted to related works in Section 7.

2 Complex system M&S requirements

The M&S of complex systems requires the collaboration of experts from several domains which use different formalisms and M&S software. To design a multi-model and to perform a co-simulation, we need to be able to put in interaction models written in different formalisms, and simulators with heterogeneous dynamics.

At the technical and software level, numerous specialized tools exist to design and simulate particular models. Each domain specific expert already masters a specific tool. It is important to enable the reuse of these tools and of the models designed in them. To do that, we need opened models, i.e. models that can accept input events and which produce output events, and we need opened simulators, i.e. simulators which enable enough the control of their simulation loop to make possible the integration of events. To reuse the models and the simulators available in the actual M&S tools, standards have to be defined.

To sum up, at the formal level we need methods to integrate heterogeneous formalisms, and at the technical level we need methods to enable the reuse of models written in heterogeneous M&S software, i.e. we need formal and software wrappings.

3 FMI

3.1 Overview

The development of the FMI standard was initiated by Daimler to ease the exchange of models between industrials [“FMI”, 2014]. The purpose is mainly practical: to ease the collaboration between industrial engineers by enabling the exchange of their models whatever the M&S tool they use.

The FMI standard leans on software architecture principles, notably the use of software components. Models are stored in black boxes called FMU (*Functional Mockup Unit*). An FMU is a zip file which contains compiled code and the FMI standard describes the methods to interact with this code.

FMI supports two ways to interact with the C-compiled code: *Model-Exchange (ME)* and *Co-simulation (CS)*. In the Model-Exchange mode, the FMU contains a dynamic system model, designed with ODE or DAE, and an interface that enables to evaluate these equations when needed. A model can be reused in another M&S software which just has to implement a solver to compute the dynamic of the FMU.

In the Co-Simulation mode, the dynamic system model is exported with a solver that computes its evolution. The interface of an FMU CS offers methods to interact with the solver linked to the model, making it suitable for co-simulation.

It is important to note that the FMI standard does not specify a master algorithm to synchronize the FMU. It is the responsibility of the integrators to choose (or design) the master algorithm.

3.2 Architecture

An FMU is a zip file containing:

- An XML file called ”modelDescription.xml” which describes the variables (parameters, inputs, outputs) used in the model.
- C-compiled files (”.dll” or ”.so” libraries depending on the operating system) with a standard interface to handle them.
- C-files and headers can be present too.

We want to insist on the fact that the difficulty is not the use of an XML file to describe a software component (it is a generic approach in software engineering). The hard part is to define the methods to interact with the software component and the definition of the XML structure (these elements are specific to dynamic system model exchange and co-simulation). It is hard because it must be generic enough and well accepted by the dynamic system M&S community.

3.3 Limit

The FMI standard focuses on the software API and does not propose any solution for formal integration. Moreover, the norm is primarily thought for ODE and DAE based dynamical systems, it is not well suited for event systems. This limit concerning discrete event handling was already pointed out and technical solutions to enhance the standard are proposed in [Tavella et al., 2016].

4 DEVS

4.1 Overview

DEVS [Zeigler et al., 2000] is a discrete event formalism proposed by Bernard P. Zeigler in the 70s [Zeigler, 1976]. Thanks to its universality property, DEVS is seen as a pivot formalism for heterogeneous formalism integration [Vangheluwe, 2000]. Several works have shown the integration of event formalisms like Petri Nets [Jacques and Wainer, 1998] in DEVS, but also the integration of a hybrid formalism like DEV&DESS [Zeigler, 2006] which specifies the interaction between discrete and continuous models.

Many works have pointed out the benefits of DEVS-based standard development [Wainer et al., 2010b]. Most of these benefits are linked to the formal properties of this formalism.

Universality and unicity: Every event system can be described in DEVS, and this description is unique.

Closure under coupling and abstract algorithm: DEVS defines a coupled model structure (a model composed of several DEVS submodels). It is shown that each coupled model is equivalent to an atomic model (allowing hierarchical design), and the chosen hierarchy does not impact the simulation results obtained with the abstract algorithm.

Separation between the model, the simulator and the experimental frame: The definition of a DEVS model is independent of the simulator and of the experimental frame.

4.2 Abstract simulator

The DEVS model simulation is done thanks to five methods [Zeigler et al., 2000] :

- *initialize*: initialization of the model.
- *processInternalEvent*: compute an internal event of the model.
- *processExternalEvent*: compute an external event entering the model.
- *getOutputEvent*: return an output external event to send it in another model.
- *getNextInternalEventTime*: return the time of the next internal event time.

These five methods are sufficient to ensure the synchronization and the data exchange between several DEVS models. Moreover, the universality property tells us that this simulation protocol can simulate every event-based model. This supports the interest of choosing this interface for the co-simulation of this kind of models.

4.3 Limit

Since the creation of DEVS, numerous DEVS-based tools have been developed. However, these M&S tools are not interoperable due to software heterogeneities [Wainer et al., 2010b]. This means that despite of a strong formal link between these tools, the lack of a technical consensus prevent us from using DEVS-based tools in collaboration.

5 Proposal

5.1 Motivations

On the one hand, we have the DEVS formalism which has many advantages for multi-modeling and co-simulation, notably its integrative power, i.e. its ability to integrate other formalisms. But DEVS does not provide a software norm to gather its community.

On the other hand, we have the FMI standard which offers an API for the ODE and DAE based dynamic system modeling community. This API lets the experts of this community collaborate by exchanging models and performing co-simulation while keeping their own tools (maintaining the independence between tools). But FMI does not provide any solution for formal integration and is currently not suited for discrete event model.

5.2 Principle

We propose to define simple rules to ease, and eventually to systematize, the ability to exchange models in DEVS-based M&S tools. These rules are organized around 3 axis.

1) Isolate the description of the model to allow its export. This rule is already supported by the DEVS formalism itself thanks to the separation between the model and its simulator.

2) Define the signatures of the methods that will enable the interaction with the simulators. Finding an agreement on these signatures provides us a homogeneous view, at the formal level, of the models during the co-simulations.

3) Define an XML structure which describes the model interface (input and output ports, parameters, acceptable events, etc).

We want to emphasize that these rules must be defined by the DEVS community. In the rest of this paper, we will just illustrate the benefits and the feasibility of their implementation.

5.3 Benefits of a DEVS-based component approach

In addition to the good properties of DEVS, the software component approach bring the following main properties.

5.3.1 Technology and tool independence

A software component approach raises several technical issues, depending on the languages. In return, it does not impose the use of another technology (communication solution, middleware, other standards, etc) or other software. It also provides a kind of simplicity if we consider the number of elements to develop (few methods have to be implemented).

5.3.2 M&S tool independence

Each tool can develop the export and the import of model to handle co-simulation in its own environment. This means that each expert will be able to launch the co-simulation in the software that provides her the most appropriate features she needs (visualization, analysis, etc).

5.3.3 Collaboration

The definition of a standard interface eases the use of co-simulations. Indeed, it lets each expert develop her model in the most appropriate tool, while ensuring that this model will be usable inside a multi-model during co-simulations.

5.3.4 Verification and reusability

Allowing the simulation of a model independently of the environment where it was designed eases also its verification. Indeed, by exporting her model in a format compliant with the tools of her peers, an expert lets them check the repeatability of her experiment and explore the behavior of the model with other parameters and initial conditions.

5.3.5 Flexibility and opening to non DEV-based tools

The software component approach is also a solution to let experts with non DEVS-based tools collaborate with the DEVS community. Even if a tool is not based on DEVS, a formal wrapping can be performed before the export to make it compliant with DEVS-based tools and to benefit from the good formal properties of DEVS for co-simulations. The simulation interface based on the DEVS simulation protocol is particularly well suited for event-based simulator. In addition, the use of a component based approach enables to respect the Intellectual Property which can be requested by industrial users.

6 Example

This section aims to highlight the needs and requirements that occur when exporting a DEVS model outside its source environment.

In MECSYCO, we integrate models from other M&S software using a DEVS-based wrapping strategy. In this example, we want to illustrate the process needed to obtain a DEVS software component from MECSYCO, i.e. how to produce a model (piece of code) which can be manipulated by another M&S software.

For the example, we are going to (1) define a classic DEVS simulator in Java, (2) export the models from MECSYCO (as Jar files) and (3) simulate these models in the simulator written at the step (1). The signatures of the interaction methods are set in a Java interface, XML descriptions of the models are provided with the Jar files.

In our case, the example is the Lorenz system where each equation is represented by one model. We reuse models written in NetLogo [Tisue and Wilensky, 2004] and available in MECSYCO.

6.1 MECSYCO

MECSYCO[Camus, 2015], available on `mecsyco.com`, is a DEVS-based co-simulation middleware which is based on a wrapping approach, i.e. its purpose is not the design but the integration of models made in other software to co-simulate them. This integrative procedure is formally based on a DEVS-wrapping strategy.

The middleware is implemented in Java and in C++, and wrappers are available for FMI [Vaubourg et al., 2015], NetLogo [Paris et al., 2017], NS3 and Omnet++ [Vaubourg et al., 2016].

For now, the DEVS-wrapping part is done in MECSYCO. The development of a software component approach would allow to perform this work directly in the software where the models are designed. The reuse of models integrated in MECSYCO will be possible in other M&S tools.

6.2 Steps

1. The DEVS interface is written in Java (see Figure 1), a second Java class *Event* which only contains a time and a data is also written (it represents the events exchanged between the models).

```java
public void initialize();
public void processInternalEvent(double time);
public void processExternalInputEvent(String port, Event<?> anEvent);
public double getNextInternalEventTime();
public Event<?> getExternalOutputEvent(String port);
```

Figure 1: Signatures of the methods used for the DEVS interface.

2. The basic DEVS simulator/coordinator is written, it is compliant with the interface described in **1**.

3. Definition of the XML structure which describes a DEVS model (ports, parameters). A simplified scheme of the **entities** and **attributes** used to describe models is shown below:

- **ModelDescription** *name=”LorenzX”*
    - **ModelInterface**
        - **Parameter** *name=”a”*
          -**Real** *default=”1.0”*
        - ...
        - **Input** *name=”y”*
          -**Real** *start=”1.0”*
        - ...
        - **Output** *name=”x”*
          -**Real** *start=”1.0”*
    - ...
    - **ModelSimulator**
        - **SimulVariables**
            - **SimulVariable** *name=”timeStep”*
              -**Real** *default=”0.01”*
            - ...

The idea is to specify a basic structure which contains the data that can be modified for each co-simulation. But the document must also be able to contain extra data required by the model. For example, NetLogo does not have the concept of input or output ports, instead it provides a command interpretor. That is why in [Paris et al., 2017], the XML document is used to define input/output ports and parameters associated to NetLogo commands.

4. Export of the models from MECSYCO. In MECSYCO, the models are represented as model artifacts: Java objects that enable to handle the model from another M&S software as if it is was a DEVS model (thanks to the 5 methods of the DEVS protocol). The models are already separated from the simulators, the work just consisted in writing a class to link the methods of the model artifact with the methods of the interface defined in **1**. Then this class is exported in a Jar file with the XML description document and the model (in our example it is a NetLogo model).

5. We use the simulator/coordinator to perform a co-simulation between 3 NetLogo models which represent a Lorenz system (we also launch these example with three FMU).

6.3 Comments

The implementation of the 5 steps mentioned above raises several interesting points:

1) Easy coding: In our case, the models integrated in MECSYCO already have a DEVS-based interface (thanks to the model artefact). The code adaptation was then relatively easy. From our experience, the DEVS wrapping step can be much more complicated [Vaubourg et al., 2016].

2) Data type choice: During the example, we have used basic data types (real, integer, string and boolean) because they are present in all platforms. This is a limiting choice, more complex data types (tab, list, map) have to be defined and standardized.

3) XML structure: In our example, the NetLogo model description files contain more information than just the description of ports and parameters. We want to point out that the XML structure must be flexible enough to accept extra information needed by particular tools.

4) Initialization: In our case, the XML description document is also used to configure the model during the initialization (by setting the parameters to the default values). It will be necessary to define a standard method to set the parameters (for instance, in MECSYCO we use Java maps).

5) Dependency: Each model has some dependencies requested to compute it, these dependencies must be accessible. For NetLogo model, we need several Jar files linked to the NetLogo API. This aspect was not considered during the export (we manually added the dependencies).

7 Discussion and related works

7.1 Technical constraints

The M&S tools are coded in different languages (C, C++, Java, Python, etc), this will cause many technical constraints at the technical/software level, i.e. the handling of language (Java, C++, etc) and operating system (Window/Unix) heterogeneities. However, there are several existing ways to bridge the gap between software written in different languages (e.g. JavaFMI which enables to simulate FMU from Java, PyFMI which enables to simulate FMU from Python, etc). For these reasons, we assume there are already existing solutions at the technical level.

7.2 DEVS-based standardization

Several works have already highlighted the benefits of DEVS-based standard development [Wainer et al., 2010b], two approaches are proposed.

1. Standardization of DEVS representation [Wainer et al., 2010c]
The idea is to find a way to represent a DEVS model (interface and behavior) independently of any generic languages like C or Java, and to transform this standard representation in a representation a simulator can compute. For instance, DEVSML [Mittal and Risco-Martín, 2016] uses an XML representation of a DEVS model which can be interpreted on several platforms. In our case, we do not focus on the way to represent the behavior of a DEVS model, but only on the way to interact with it, i.e. only on the interface of the model. This allows notably to integrate already existing models without writing then again.

2. Standardization of DEVS interface [Wainer et al., 2010a]
The proposal of this article is in the scope of this approach where the focus is on the interface of interaction with the exported models and not on the representation of the DEVS model behavior. Most of the existing solutions (such as DEVS/SOA) use technologies and approaches linked to web services to enable the interaction between several DEVS simulators. The idea behind these solutions is to create a middleware to coordinate different DEVS-based M&S tools in co-simulations. Unlike these solutions, the approach proposed in this article does not enable the co-simulation of several DEVS-based tools, it enables to exchange models between these M&S tools. Each DEVS-based M&S software can play the role of the coordinator between DEVS models (exported by other software or not) and use its own technology and features.

7.3 Other standardization approaches

FMI standard
Our proposal is greatly inspired by the FMU standard. Even though these approaches are similar in term of software architecture and deal with the same collaboration and model exchange needs, they are not conflicting but complementary. Indeed, whereas the FMI standard offers an interface adapted to dynamic systems based on ODE and DAE, the DEVS interface is naturally more suited to discrete event models. Moreover, while the evolution of FMI is leaded by the practice, the evolution of a DEVS-based standard can be supported by the numerous extensions of DEVS that already offer solutions to various issues (Parallel DEVS, DEVS&DESS, etc). For example, a previous work has shown and discussed the rigorous integration of FMU in DEVS using DEVS&DESS [Camus et al., 2018].

High Level Architecture
The High Level Architecture is a software architecture specification proposed by the US Department of Defense in the 90s. Its objective is to put in interaction several simulation components (distributed simulation) [Dahmann, 1999]. The purposes of HLA are also to enhance the reusability and the interoperability of simulation components. Within HLA, the simulation components interact with each other through the RTI (Run-Time Infrastructure) thanks to a specified interface, i.e. there is a central component to handle the interactions. We can mention that the RTI of HLA can be used as a master for FMU co-simulations [Awais et al., 2013]. The comparison between HLA and DEVS-based standardization approaches is already done in [Wainer et al., 2010b]. The authors state that unlike HLA, DEVS-based standardization solutions can benefit from the DEVS formal properties (such as closure under coupling). Moreover, the HLA specifications are quite complex to follow and there is still limitations (for example, two different implementations of the RTI are not interoperable).

There are other technical standardization approaches inspired by software engineering techniques, web service architecture etc. They are mainly supported by industrials for specific purpose. Among them, we can mention the Simulation Model Portability standard [Nemeth and Demarest, 2010], supported by the European Space Agency, which is dedicated to spaceflight model reuse. However the development of a generic standard for model reuse should be based on a formal M&S theory rather than just technical guidelines [Davis and Anderson, 2004].

8 Conclusion

The purpose of this article is to start a reflection on the creation of DEVS-based development rules to ease the use of existing M&S software for the rigorous design of multi-models and their co-simulations. These rules will include the definition of a software interface compliant with a DEVS coordinator, and the definition of an XML structure to detail the interface of models. Finding an agreement on common rules within the DEVS community will enable to create reusable DEVS-based software components.

This work is a proposal that echoes the different existing works around DEVS-based standardization. This approach has two main advantages: it keeps the independence of M&S tools while providing a way to exchange models, and it benefits from the formal properties of DEVS. We want to emphasize that the different DEVS-based standardization approaches are complementary in the sense that they can be used simultaneously during a co-simulation.

The short-term perspective of our proposal is to try the export of several models from different DEVS-based software and to test their co-simulation. This will enable to face more technical issues and to lead the formal definition of what a DEVS-component should be. A long-term perspective could be to study the existing DEVS extensions to enhance the approach with new features.

Acknowledgments

We would like to thank the organizers and the participants of the ”Journées DEVS Francophones 2018” workshop for their feedback and the many interesting discussions we had around the topic of this article.

References

[Awais et al., 2013] Awais, M. U., Palensky, P., Elsheikh, A., Widl, E., and Matthias, S. (2013). The high level architecture RTI as a master to the functional mock-up interface components. pages 315–320. IEEE.

[Blochwitz et al., 2011] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., and others (2011). The Functional Mockup Interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference*, pages 105–114. Linköping University Press.

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes*. PhD thesis, Université de Lorraine.

[Camus et al., 2018] Camus, B., Paris, T., Vaubourg, J., Presse, Y., Bourjot, C., Ciarletta, L., and Chevrier, V. (2018). Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware. *SIMULATION*.

[Dahmann, 1999] Dahmann, J. S. (1999). The High Level Architecture and beyond: technology challenges. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 64–70. IEEE Computer Society.

[Davis and Anderson, 2004] Davis, P. K. and Anderson, R. H. (2004). Improving the composability of DoD models and simulations. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1(1):5–17.

[“FMI”, 2014] “FMI”, M. A. P. (2014). FMI for Model-Exchange and Co-Simulation v2.0.

[Gomes et al., 2017] Gomes, C., Casper, T., Gorm Larsen, P., and Vangheluwe, H. (2017). Co-simulation state of the art. Technical report.

[Jacques and Wainer, 1998] Jacques, C. J. D. and Wainer, G. A. (1998). Using the CD++ DEVS Tookit to Develop Petri Nets. In *Summer Computer Simulation Conference*, pages 51–56. Society for Computer Simulation International.

[Mittal and Risco-Martín, 2016] Mittal, S. and Risco-Martín, J. L. (2016). DEVSML studio: a framework for integrating domain-specific languages for discrete and continuous hybrid systems into DEVS-based m&s environment. In *Proceedings of the Summer Computer Simulation Conference*, page 41. Society for Computer Simulation International.

[Nemeth and Demarest, 2010] Nemeth, S. and Demarest, P. (2010). Research and Development in Application of the Simulation Model Portability Standard. American Institute of Aeronautics and Astronautics.

[Paris et al., 2017] Paris, T., Ciarletta, L., and Chevrier, V. (2017). Designing co-simulation with multi-agent tools: a case study with NetLogo. In *Proceedings of the 15th European Workshop on Multi-Agent Systems. EUMAS*.

[Ramat, 2007] Ramat, E. (2007). Introduction to Discrete Event Modelling and Simulation. In *Agent Based Modelling and Simulations in the Human and Social Sciences*, Oxford, The Bardwell Press, pages 35–62. Denis Phan & Frédéric Amblard.

[Seck and Honig, 2012] Seck, M. D. and Honig, H. J. (2012). Multi-perspective modelling of complex phenomena. *Computational & Mathematical Organization Theory*, pages 1–17.

[Tavella et al., 2016] Tavella, J.-P., Caujolle, M., Tan, C., Plessis, G., Schumann, M., Vialle, S., Dad, C., Cuccuru, A., and Revol, S. (2016). Toward an Hybrid Co-simulation with the FMI-CS Standard.

[Tisue and Wilensky, 2004] Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA.

[Vangheluwe, 2000] Vangheluwe, H. L. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 129–134. IEEE.

[Vaubourg et al., 2016] Vaubourg, J., Chevrier, V., Ciarletta, L., and Camus, B. (2016). Co-simulation of IP network models in the Cyber-Physical systems context, using a DEVS-based platform. In *Proceedings of the 19th Communications & Networking Symposium*, page 2. Society for Computer Simulation International.

[Vaubourg et al., 2015] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., and Morais, H. (2015). Multi-agent Multi-Model Simulation of Smart Grids in the MS4sg Project. In Demazeau, Y., Decker, K. S., Bajo Pérez, J., and de la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection*, volume 9086, pages 240–251. Springer International Publishing, Cham.

[Wainer et al., 2010a] Wainer, G., Al-Zoubi, K., Dalle, O., Hill, D. R. C., Mittal, S., Martin, J. L. R., Sarjoughian, H., Touraille, L., Traoré, M. K., and Zeigler, B. P. (2010a). Standardizing DEVS Simulation Middleware. In *Discrete-Event Modeling and Simulation: Theory and Applications*, page 459.

[Wainer et al., 2010b] Wainer, G., Al-Zoubi, K., Hill, D., Mittal, S., Martín, J., Sarjoughian, H., Touraille, L., Traoré, M., and Zeigler, B. (2010b). An Introduction to DEVS Standardization. In Wainer, G. and Mosterman, P., editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 393–425. CRC Press.

[Wainer et al., 2010c] Wainer, G., Al-Zoubi, K., Hill, D., Mittal, S., Martín, J., Sarjoughian, H., Touraille, L., Traoré, M., and Zeigler, B. (2010c). Standardizing DEVS Model Representation. In Wainer, G. and Mosterman, P., editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 427–458. CRC Press.

[Zeigler, 1976] Zeigler, B. P. (1976). *Theory of Modelling and Simulation*. Wiley Interscience, New York, 1st edition.

[Zeigler, 2006] Zeigler, B. P. (2006). Embedding DEV&DESS in DEVS. In *DEVS Integrative Modeling & Simulation Symposium*, pages 125–132.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic press, New York, 2nd edition.