Towards Stochastic FMI Co-simulations: Implementation of an FMU for a Stochastic Activity Networks Simulator

C. Bernardeschi[1], A. Domenici[1], and M. Palmieri[2][1]

[1] Dept. of Information Engineering, University of Pisa
[2] DINFO, University of Florence

Abstract. The advantage of co-simulation with respect to traditional single-paradigm simulation lies mainly in the modeling flexibility it affords in composing large models out of submodels, each expressed in the most appropriate formalism. One aspect of this flexibility is the modularity of the co-simulation framework, which allows developers to replace each sub-model with a new version, possibly based on a different formalism or a different simulator, without changing the rest of the co-simulation. This paper reports on the replacement of a sub-model in a co-simulation built on the INTO-CPS framework. Namely, an existing co-simulation of a water tank, available in the INTO-CPS distribution, has been modified by replacing the tank sub-model with a sub-model built as a Stochastic Activity Network simulated on Möbius, a tool used to perform statistical analyses of systems with stochastic behavior. This work discusses aspects of this redesign, including the necessary modifications to the Möbius sub-model. In this still preliminary work, the Stochastic Activity Network features related to stochastic models have not been used, but a simple deterministic model has proved useful in indicating an approach to the integration of Stochastic Activity Networks into a co-simulation framework.

1 Introduction

Co-simulation is gaining interest and acceptance as an approach to modeling and simulation of cyber-physical systems (CPS) [1, 30], as it is based on the concept of modeling each part of a large, heterogeneous system with the most appropriate formalism, and simulating each part with a tool fit for the formalism. This requires coordinating the execution of two or more simulators, which usually have been designed as standalone tools, therefore the need arises of standards for the exchange of data and control among different modeling and simulation tools. One such standard is the *Functional Mockup Interface* (FMI), defining the common interface and protocol that must be honored by the simulators involved in a co-simulation.

The simulation of a set of heterogeneous models, called a *multi-model*, is coordinated by a *master* algorithm. Implementing a master algorithm from scratch would be costly and inefficient, but co-simulation usually relies on a framework providing the algorithm together with a user interface to configure and control the co-simulation. An important feature of co-simulation frameworks is modularity: Developers should be able to add simulators and also to replace any simulator with another one, which simulates the same subsystem but with a different modeling technique, with a minimum effort and leaving the other simulators unchanged.

This paper is focused on (i) the development of an FMU for a modeling and simulation tool not yet used in FMI-based co-simulations, and (ii) modifying a previous multi-model by replacing one of its submodels with a new one, expressed in a completely different formalism. These two points exemplify the flexibility and modularity of the co-simulation approach. A further, longer-term goal is investigating the integration of statistical simulation techniques and co-simulation.

The simulation tool considered in this paper is Möbius, an environment for the analysis and simulation of *Stochastic Activity Networks* (SAN). SANs are a wide-ranging extension to Petri nets, oriented to the evaluation of performance and dependability. The Möbius tool has been integrated in a co-simulation built on the INTO-CPS framework. This co-simulation is a case study available in the INTO-CPS distribution, concerning the control of a water tank. The tank controller activates the tank's exhaust valve depending on the water level and is modeled in VDM, and the tank's dynamics are modeled in Modelica. This latter model has been replaced by a SAN and simulated with the Möbius tool.

2 Related Work

The co-simulation of a human heart modeled in Simulink and an implantable pacemaker modeled in PVS [21] has been presented in [5], where the PVSio-web [20] prototyping toolkit provided the communication infrastructure. A PVS model of a controller was also used in the simulation of a semi-autonomous vehicle [22] whose mechanical part was modeled with 20-SIM and OpenModelica, in the INTO-CPS framework.

The Möbius [7, 9, 10] tool can be seen as oriented to co-simulation, as it has been designed to build complex models by integrating submodels in different formalisms [24–26], but it requires the submodels to be developed with tools built-in in the Möbius framework. Another multi-formalism framework is SIMTHESys [14].

SAN models have been used in a large number of application fields, including biology and medicine [28, 29], integrated circuits [2, 3], and railway systems [19].

From the literature on the integration of deterministic and non-deterministic simulation, we may cite [16, 18, 17].

3 Background

This section introduces basic information on the tools and standards referred to in the paper, with an emphasis on Stochastic Activity Networks and the Möbius tool.

3.1 Stochastic Activity Networks

Stochastic Activity Networks [27] are an extension of Petri Nets (PN). SANs are directed graphs with four disjoint sets of nodes: *places*, *input gates*, *output gates*, and *activities*. The latter are an extension of PN *transitions*. The allowed arcs are from places to input gates, from input gates to activities, from activities to output gates, and from output gates to places.

Each SAN activity may be either *instantaneous* or *timed*. Timed activities represent actions with a duration affecting the performance of the modeled system, e.g., message transmission time. The duration of each timed activity is expressed via a *time distribution* function. An activity *completes* when its (possibly instantaneous) execution terminates.

Any instantaneous or timed activity may have mutually exclusive outcomes, called *cases*, chosen probabilistically according to the *case distribution* of the activity. Cases can be used to model probabilistic behaviors.

The state of a SAN is defined by its *marking*, i.e., a function that, at each step of the net's evolution, maps the places to non-negative integers. SANs enable the user to specify any desired enabling condition and firing rule for each activity. This is accomplished by associating an *enabling* (or *input*) *predicate* and an *input function* to each input gate, and an *output function* to each output gate. The enabling predicate is a Boolean function of the marking of the gate's input places. The input and output functions compute the next marking of the input and output places, respectively, given their current marking. If these predicates and functions are not specified for some activity, the standard PN rules are assumed.

The evolution of a SAN, starting from a given marking μ, may be described as follows: (i) The instantaneous activities enabled in μ complete in some unspecified order; (ii) if no instantaneous activities are enabled in μ, the enabled (timed) activities become *active*; (iii) the completion times of each active (timed) activity are computed stochastically, according to the respective time distributions; the activity with the earliest completion time is selected for completion; (iv) when an activity (timed or not) completes, one of its cases is selected according to the case distribution, and the next marking μ' is computed by evaluating the input functions of the input gates and the output functions of the gates connected to the selected case; (v) if an activity that was active in μ is no longer enabled in μ', it is removed from the set of active activities.

Graphically, places are drawn as circles, input (output) gates as left-pointing (right-pointing) triangles, instantaneous activities as narrow vertical bars, and timed activities as thick vertical bars. Cases are drawn as small circles on the right side of activities. Gates with default (standard PN) enabling predicates and firing rules are not shown.

3.2 The Möbius Tool

Möbius [9, 10] is a software tool that provides a comprehensive framework for model-based evaluation of system dependability and performance. The main features of the tool include support for multiple high-level modeling formalisms beyond SANs, such as, among others, PEPA fault trees [13] and the ADVISE security model formalism [12], and statistical characterization of system behavior.

The Möbius tool introduces two extensions to the SAN formalism: *extended places* and *global variables*. Extended places are places whose marking is a numerical value other than non-negative integers, or a complex data structure. Global variables are (possibly complex) data structures that can be accessed by enabling predicates and input and output functions, and can be shared among different SANs.

Enabling predicates and input and output functions of the gates are specified as C++ code.

A *study model* is a set of *experiments*, i.e., assignments to the global variables. Study models enable developers to run simulations for different values of system parameters. Variable assignments can be specified manually or generated by the tool as sequences of values according to various patterns.

The tool generates a *simulation solver*, an executable file that can be run from the Möbius user interface or launched from the command line.

3.3 The FMI Standard

The FMI standard [6] defines a set of C functions to support interaction among heterogeneous simulators coordinated by a master algorithm. The interface includes operations to initialize and configure the simulators, to exchange data with setter and getter operations, and to orchestrate the co-simulation by issuing *doStep* commands to the individual simulators.

A Functional Mockup Unit is a software artifact packaging all components necessary to simulate a single model, including, if needed, a whole simulator application. Some modeling tools can produce an FMU from their user interface, or provide scripts to create it from the command line. Otherwise, a developer can adapt those scripts to modeling tools that do not yet support the FMI standard.

3.4 The INTO-CPS Framework

INTO-CPS [15] is an integrated tool-chain to support model-based development of CPSs using co-simulation according to the FMI standard. The top-level component of the tool-chain is the INTO-CPS *application*, a graphical user interface for the management of co-simulation projects. Developers create FMUs for their models using the respective tools, place them in the INTO-CPS project directory, and define their interconnections with the user interface. Simulations are executed under control of the *Co-Simulation Orchestration Engine* (COE), the core component of the tool-chain. The user interface also provides a graphical output to plot selected quantities.

The reader is addressed to the literature [11] for other important features of the tool-chain, such as design space exploration [8].

4 The SAN Water Tank Co-simulation

As anticipated in Section 1, the main motivations for the present work are the development of an FMU for a new modeling tool and the replacement of a sub-model into an existing multi-model, as described in this section.

4.1 The INTO-CPS Water Tank Example

The INTO-CPS application comes with a set of case studies [23] including a water tank whose level is controlled by an exit valve with two states, fully open or fully closed. The tank is fed at a constant flow and drained (when the valve is open) at a flow rate depending on the instantaneous water level. The valve tank controller reads the water level, then it opens the valve when the maximum allowed level is exceeded and closes it when the water goes below the minimum allowed level. Two models for the tank are available, one in Modelica and one in 20-SIM, while the controller model is in VDM-RT.

4.2 The SAN Model

The tank sub-model has been replaced with a SAN developed on the Möbius tool. In addition to the different modeling language, a different physical model has been chosen, adapted from the one studied in [4], and the main differences from the INTO-CPS case study are the following: (i) the intake flow is variable; (ii) the valve is opened and closed gradually, so that its area varies linearly with time; and (iii) the drain flow depends on the valve area, and not on the water level. The valve actuator, however, accepts the same control inputs as in the original model.

More precisely, the control signal takes the values 0 (*close*) or 1 (*open*) when the lower or upper level limits, respectively, are reached. Otherwise, it maintains the current value, as defined in the original INTO-CPS model. The area of the valve increases when the control signal equals 1 and decreases when the control signal is 0, unless one of the limit positions has been reached. In this case, the valve remains open or closed until a reversing control signal is received. The outgoing flow is proportional to the valve area, and the tank level is the integral of the net flow.

Figure 1 shows the SAN model, where the lighter (orange) circles are extended places, used to store quantities of interest. Let us ignore, for the moment,

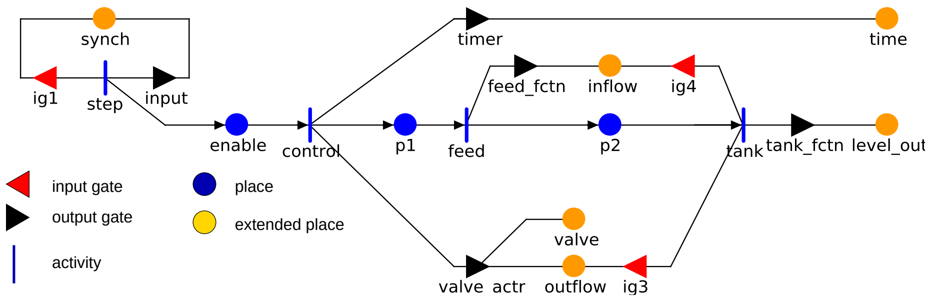

Fig. 1. SAN model of the tank.

Table 1. Gate functions

Gate	Predicate	Function
ig1	*synch* = 1	
input		wait for input, then set *valve_control* and reset *synch*
timer		increase *time*; terminate if max time reached
valve_actr		set valve position; compute outflow
feed_fctn		compute inflow
ig4	true	
ig3	true	
tank_fctn		compute net flow and level; set *synch*

the *step* activity, whose purpose is to synchronize the tank and the controller models. The input gate predicates and output gate functions are summarized in Table 1. Note that an explicit input gate is required between an extended place and an activity, hence the always enabled gates *ig3* and *ig4*. The *control* activity performs the following actions:

1. it increments the simulation time in gate *timer*;
2. it computes the valve area and the drain flow in gate *valve_actr*; and
3. it enables the next activity, *feed*, by marking place p_1.

Gate *timer* increments the marking of place *time* by a fixed amount *dt*. This is a global variable set by the user in the study model.

The valve area and the drain flow are computed by the output function of *valve_actr*, modeling the valve actuator. This function reads a global variable containing the last command from the controller, it increases or decreases the valve area accordingly, and computes the flow. The computed values are stored as the markings of the extended places *valve* and *outflow*.

The *feed* activity models the water source and enables activity *tank*. Gate *feed_fctn* computes the intake flow as a function of time and stores its value in place *inflow*. In this simulation, a sinusoidal function has been used.

Finally, the output gate *tank_fctn* of *tank* computes the net flow and the updated level. The output function executes a simple integration step increasing the current level by the product of the net flow and the time interval.

We may note that all computations are C++ fragments entered through the user interface and inserted by the tool into the functions of the output gates. However, such fragments may also call external user-defined code, for example to implement a more accurate integration method.

4.3 The FMU

Replacing a sub-model in a co-simulation multi-model involves addressing three main concerns: ensuring semantic coherence, complying with the multi-model synchronization mechanism, and translating between different syntactic representations. The third concern is not very important for the case at hand, since the submodels only exchange control and synchronization signals, the only quantitative information being the water level. In fact, the multi-model is composed only of a plant subsystem (the tank, water source, and valve) and a control subsystem. The latter receives the water level from a sensor and returns a binary signal.

The controller's signal concerns the issue of semantic coherence. In the original multi-model, its effect is to cause the valve to fully open or close, whereas in the new model the valve opens or closes gradually. Further, there are other differences between the physical behavior of the two models. However, the meaning of the controller's output remains the same, as in both cases it signals that the water level is not within the allowed limits. Therefore the new multi-model makes sense even if it simulates a system with different properties.

The synchronization mechanism is controlled by the master algorithm, which periodically invokes a *doStep* operation on each FMU to trigger one execution step. This requires the simulators to agree on a common time base and be able to pause between each simulation step.

In the INTO-CPS multi-model, the simulation step is configurable from the user interface, so the common time base is achieved by setting variable *dt* equal to the simulation step.

Pausing the SAN simulator requires adding a simple synchronization mechanism to the model. The output function of gate *input* (Fig. 1) reads the control signal from standard input, stores its value in a global variable, disables activity *step* by zeroing the marking of place *synch*, and enables the *control* activity to start a simulation step. The step terminates when the output function of *tank_fctn* prints the water level to standard output and sets the marking of place *synch* to re-enable activity *step* for the next step.

The final task is providing an FMI-compliant interface to the SAN executable. This is done by a software component that implements the FMI interface, and in particular the operations *fmi2Instantiate* and *fmi2DoStep*. The former spawns

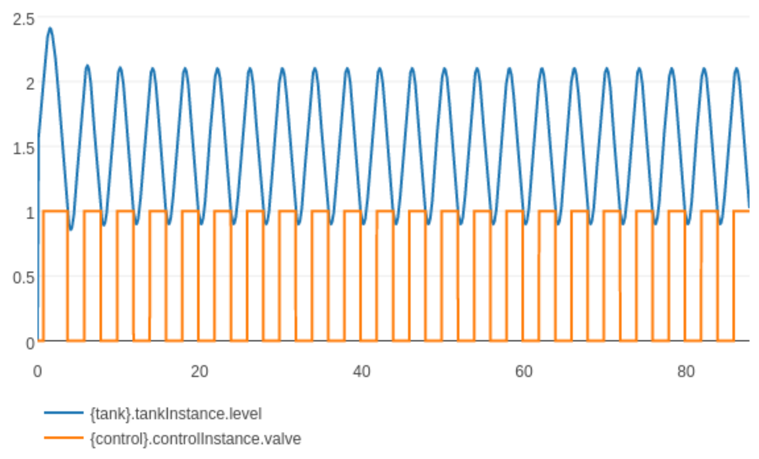

Fig. 2. Results of co-simulation.

the Möbius-generated executable and connects with it through Unix pipes on which the executable's standard input and output are redirected. The module is compiled into a dynamic library that is then packed in the FMU component.

The FMU described above was installed in the INTO-CPS multi-model without changing the FMU for the controller, and simulated. Figure 2 is an example of the output for one the simulations, where the darker (blue) line is the water level and the lighter (brown) one is the controller output. This plot is consistent with the one shown in the INTO-CPS case study [23], except for the different waveform of the water level, due to the different incoming flow.

5 Conclusions

The present work shows, by means of a practical example, the "plug and play" capability of the FMI standard of the co-simulation framework adopted. A pre-existing and independently developed multi-model has been modified by replacing a substantial part with a new version, differing from the original one in the modeling formalism, in the simulation engine, and even in its physical behavior. The replacement has been performed without any change to the rest of the multi-model, and has required only the inclusion of an explicit synchronization mechanism in the SAN model and the development of an FMI-compliant wrapper process to interface the model.

This simple procedure has been possible in spite of the fact that the new model is expressed in a formalism quite different from such languages as Modelica or Bond-Graphs. Making diverse modeling paradigms available gives developers the possibility to explore more aspects of the systems being developed. Stochastic Activity Networks, for example, make it easy to study probabilistic behaviors, although this capability has not been exploited in the present work. In spite of this limitation, this experience has proved useful in finding interesting aspects of the integration of SAN models that will continue to be investigated in further research. In particular, the synchronization with the master algorithm needs more study. In the present work, the straightforward solution of inserting an *ad hoc* sub-network into the SAN model has been adopted, but more modular, less invasive methods should be developed. A more fundamental issue for further work is how to synchronize the co-simulation in presence of stochastic durations of simulation steps. Finally, even if plugging the new model in the simulation "by hand" was rather easy, it should be made easier by providing generic tools that can produce FMUs for new simulator from a purely declarative description of the required interface.

Acknowledgments

The authors wish to thank the anonymous referees for their helpful suggestions.

References

1. Van der Auweraer, H., Anthonis, J., De Bruyne, S., Leuridan, J.: Virtual engineering at work: the challenges for designing mechatronic products. Engineering with Computers 29(3), 389–408 (Jul 2013), https://doi.org/10.1007/s00366-012-0286-6
2. Bernardeschi, C., Cassano, L., Domenici, A., Sterpone, L.: ASSESS: A Simulator of Soft Errors in the Configuration Memory of SRAM-Based FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33(9), 1342–1355 (Sept 2014)
3. Bernardeschi, C., Cassano, L., Domenici, A.: Failure Probability and Fault Observability of SRAM-FPGA Systems. In: International Conference on Field Programmable Logic and Applications (FPL2011). pp. 385–388. IEEE (sep 2011)
4. Bernardeschi, C., Domenici, A.: Verifying safety properties of a nonlinear control by interactive theorem proving with the Prototype Verification System. Information Processing Letters 116(6), 409–415 (2016), http://www.sciencedirect.com/science/article/pii/S0020019016300072
5. Bernardeschi, C., Domenici, A., Masci, P.: A PVS-Simulink Integrated Environment for Model-Based Analysis of Cyber-Physical Systems. IEEE Transactions on Software Engineering PP(99), 1–1 (2017)
6. Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauß, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauß, J., Neumerkel, D., Olsson, H., Viel, A.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In: Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany
7. Buchanan, C., Keefe, K.: Simulation Debugging and Visualization in the Möbius Modeling Framework. In: Norman, G., Sanders, W. (eds.) Quantitative Evaluation of Systems. pp. 226–240. Springer International Publishing, Cham (2014)

8. Christiansen, M., Larsen, P., Nyholm Jørgensen, R.: Robotic Design Choice Overview using Co-simulation and Design Space Exploration. Robotics 4, 398–421 (2015)
9. Clark, G., Courtney, T., Daly, D., Deavours, D.D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.G.: The Möbius modeling tool. In: 9th Int. Workshop on Petri Nets and Performance Models. pp. 241–250. IEEE Computer Society Press, Aachen, Germany (Sept 2001)
10. Deavours, D.D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.G.: The Möbius framework and its implementation. IEEE Trans. Softw. Eng. 28(10), 956–969 (2002)
11. Fitzgerald, J., Gamble, C., Larsen, P., Pierce, K., Woodcock, J.: Cyber-Physical Systems Design: Formal Foundations, Methods and Integrated Tool Chains. In: Proceedings of the 2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering (FormaliSE). pp. 40–46. IEEE (2015)
12. Ford, M.D., Keefe, K., LeMay, E., Sanders, W.H., Muehrcke, C.: Implementing the ADVISE security modeling formalism in Möbius. In: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 1–8 (June 2013)
13. Gulati, R., Dugan, J.B.: A modular approach for analyzing static and dynamic fault trees. In: Annual Reliability and Maintainability Symposium. pp. 57–63. IEEE Computer Society Press (1997)
14. Iacono, M., Gribaudo, M.: Element based semantics in multi formalism performance models. In: 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. pp. 413–416 (Aug 2010)
15. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In: 2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data). pp. 1–6 (April 2016)
16. Lawrence, D.P.Y., Gomes, C., Denil, J., Vangheluwe, H., Buchs, D.: Coupling Petri nets with deterministic formalisms using co-simulation. In: Proceedings of the Symposium on Theory of Modeling & Simulation. pp. 6:1–6:8. TMS-DEVS '16, Society for Computer Simulation International, San Diego, CA, USA (2016), http://dl.acm.org/citation.cfm?id=2975389.2975395
17. Liu, J., Jiang, K., Wang, X., Cheng, B., Du, D.: Improved co-simulation with event detection for stochastic behaviors of cpss. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 209–214 (June 2016)
18. Mancini, T., Mari, F., Massini, A., Melatti, I., Merli, F., Tronci, E.: System level formal verification via model checking driven simulation. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 296–312. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Nelli, M., Bondavalli, A., Simoncini, L.: Dependability modeling and analysis of complex control systems: An application to railway interlocking. In: Hlawiczka, A., Silva, J.G., Simoncini, L. (eds.) Dependable Computing — EDCC-2. pp. 91–110. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
20. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In: FMIS2013, 5th International Workshop on Formal Methods for Interactive Systems, London, UK, June 24, 2013 (2013)

21. Owre, S., Rajan, S., Rushby, J., Shankar, N., Srivas, M.: PVS: combining specification, proof checking, and model checking. In: Alur, R., Henzinger, T. (eds.) Computer-Aided Verification, CAV '96, pp. 411–414. No. 1102 in LNCS, Springer-Verlag (1996)
22. Palmieri, M., Bernardeschi, C., Masci, P.: Co-simulation of semi-autonomous systems: The line follower robot case study. In: Cerone, A., Roveri, M. (eds.) Software Engineering and Formal Methods. pp. 423–437. Springer International Publishing, Cham (2018)
23. Payne, R., Gamble, C., Pierce, K., Fitzgerald, J., Foster, S., Thule, C., Nilsson, R.: Examples Compendium 2. Tech. Rep. D3.5, INTO-CPS Deliverable (Dec 2008), http://projects.au.dk/into-cps/dissemination/publications
24. Peccoud, J., Courtney, T., Sanders, W.H.: Möbius: an integrated discrete-event modeling environment. Bioinformatics 23(24), 3412–3414 (2007), http://dx.doi.org/10.1093/bioinformatics/btm517
25. Sanders, W.H.: Integrated frameworks for multi-level and multi-formalism modeling. In: Proceedings 8th International Workshop on Petri Nets and Performance Models (Cat. No.PR00331). pp. 2–9 (1999)
26. Sanders, W., Courtney, T., Deavours, D., Daly, D., Derisavi, S., Lam, V.: Multi-formalism and multi-solution-method modeling frameworks: The Möbius approach. In: Proc. Symp. on Performance Evaluation – Stories and Perspectives. pp. 241–256. Vienna, Austria (Dec 2003)
27. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: formal definitions and concepts. In: Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science, pp. 315–343. Springer-Verlag New York, Inc., New York, NY, USA (2002)
28. Srivastava, R., Peterson, M.S., Bentley, W.E.: Stochastic kinetic analysis of the Escherichia coli stress circuit using σ^{32}-targeted antisense. Biotechnology and Bioengineering 75(1), 120–129 (2001), https://onlinelibrary.wiley.com/doi/abs/10.1002/bit.1171
29. Tsavachidou, D., Liebman, M.N.: Modeling and simulation of pathways in menopause. Journal of the American Medical Informatics Association 9(5), 461–471 (2002), http://dx.doi.org/10.1197/jamia.M1103
30. Vangheluwe, H.: Foundations of modelling and simulation of complex systems. Electronic Communications of the EASST 10 (2008)