Semantics for an Interdisciplinary Computation

Justyna Zander
MathWorks
3 Apple Hill Dr., Natick, MA, USA
Gdansk University of Technology, Poland
justyna.zander@mathworks.com

Pieter J. Mosterman
MathWorks
3 Apple Hill Dr., Natick, MA, USA
pieter.mosterman@mathworks.com

Keywords: Model-Based Design, Approximations, Computer Science, Engineering, Solver, Execution Engine, Simulation, Computation, Numerical Integration

Abstract

Semantics for an interdisciplinary computation is becoming increasingly difficult to capture while dealing with multi-domain problems. Expertise from Computer Science, Computer Engineering, Electrical Engineering, and other disciplines merges as engineering challenges in modern systems, such as, Cyber-Physical Systems, Smart Cities, and Bionic Systems must be tackled in a methodological manner.

In this paper, a paradigm for formalization of such a computational semantics is proposed. It is based on the experience of a Computer Scientist, Mathematician, and an Engineer when their efforts converge to design a complex system. The design is embedded in the Model-Based Design process and a simulation aids to build the actual system implementation. To let the experts achieve the same understanding level, a declarative layer for simulation semantics is defined. This introduces abstraction and helps understand the behavior of the interacting elements in a system design using mathematical principles that are easier to embrace for more diverse scientific communities.

1. INTRODUCTION

In the design of engineered systems many different stakeholders from a broad range of disciplines have come together. However, even the involved technical disciplines such as Computer Science, Computer Engineering, Electrical Engineering, and Mechanical Engineering often are treated in isolation from a scientific perspective. As a consequence, there is a growing disconnect with practitioners, as modern engineered systems such as networked embedded systems, ultra-large systems, and cyber-physical systems require solving design problems at the intersection of the various disciplines.

Model-Based Design (MBD) is a testimony to the value of true interdisciplinary design. Raising the level of abstraction of a system under design allows for a smooth and elegant convergence of disciplines by providing selected respective information. Particular values are derived at the increasingly important analysis of system-level behavior, which allows for quick design space exploration unencumbered by implementation detail. Here, simulation of all the elements plays a significant role in a rapid evaluation of the system at the early system design stages. The main challenge at the system level is designing the integration of executable artifacts while accounting for their embedding physics. As it becomes unavoidable to include both, the system model and models of the environment for a simulation to occur, the communities must understand the system design (in a horizontal sense), but also the technology stack that the system executes on (in a vertical sense).

In this context, the simulation of a system under design relies on a simulation engine that itself constitutes a complex software system. This engine is implemented directly in a programming code by skilled engineers who are familiar with computational algorithms and numerical integration schemes underlying the semantics of the simulation formalism. Computer Scientists often become the users of MBD but they lack a proper understanding of this underlying simulation machinery. To a large extent, the reason for this is that the expertise about the semantics of a simulation engine is rarely imparted to the Computer Science (CS) specialists as part of their training. As consequence, MBD tools may be used outside of their assumptions and beyond their proper configurations. Furthermore, as developers of MBD support artifacts, this deficiency delays building methods, technologies, and tools for computationally-executable models and systems.

To enable an integrated approach of the various disciplines in engineered system design, a conceptual framework for formalizing the semantics of computational simulation from the perspective of both, a computer scientist and an engineer is provided. It serves as a means to harness existing practical computational methods used for simulation of engineered systems. In particular, a CS perspective is delivered on top of the computational and mathematical theories applied in today's simulation practice to synthesize MBD-underlying semantics. This opens up opportunities for a dialogue between the MBD user building an executable model specification and the MBD-tool designer who creates and maintains the actual execution for such a model.

1.1. Cyber-Physical System Notion

An example of a system where MBD derives great utility is a Cyber-Physical System (CPS). A CPS is frequently characterized as a smart system that includes digital cyber technologies, software, and physical components. CPS is intelligently interacting within its own confines across information and physical interfaces. It may further engage other CPS to collaborate and so the cyber-physical systems *paradigm* includes a framework where multiple systems operate in concert to achieve desired functionality as an emerging behavior. The corresponding context mandates continuous responsiveness and live functionalities of CPS during the deployment (PCAST, 2007; Zander, 2012).

1.2. Computer Science Context for CPS

Great progress has been achieved in programming and software engineering as such. Software capabilities are growing and Computer Scientists have transformed to Information Technologists who practice coding at times (Ferguson, 2004). As the IT industry matures, software is coming to be viewed as a commodity and computing as a utility. Further, the industry is seeking the lowest labor costs for implementation, maintenance, support, and operations by outsourcing (Carr, 2004). Still, approaches such as structured programming, modularization, object-oriented programming, and others concentrate on an information rather than a dynamics perspective. As a consequence, these approaches benefit the CPS or embedded systems progress in a limited sense. The lack of support for capturing the dynamics in IT centric approaches has created an inherent gap in support for CPS design. In a CPS, software and the dynamics of the computations merge with the dynamics of physical components, and all embedded in a network-centric environment. To study the effects of embedding computation in a physical world with characteristics such as resource allocation is critical, where resource may be, for example, the response time of a sequence of computations, the amount of memory necessary, and the energy consumed (e.g., Lee, 2000; Lee, 2010).

For successful and efficient CPS design, it is imperative to understand the semantics underlying the computational representation behind CPS components, in particular to formally represent the corresponding dynamics of these components. For CPS, this presents a formidable challenge as the semantics is situated at the nexus of many domains, from Computer Science (CS), through Electrical Engineering (EE), to Mathematics and Physics.

1.3. Technical Computation Convergence

To solve complexities at the intersection of these different disciplines, abstraction enables a domain independent representation of the complexities such that they can be studied from the various disciplines. Because of the multi-disciplinary nature of CPS, abstraction is paramount in designing the necessary embedded computational features. In the system development process, abstraction allows for specification and analysis of critical characteristics by providing high-level formalisms (e.g., UML®, Modelica (Elmqvist et al., 1999), Simulink® (Simulink 2012a), SCADE®, UML® Testing Profile). These formalisms relate mostly to the problem space (business processes, control algorithms, signal processing algorithms, test suite creation, but also mechanical linkages, electrical signal conditioning, hydraulic/pneumatic effects, etc.), while the solution space is nowadays typically understood to be code, frequently automatically generated.

The abstraction results in models of the underlying logical, software, and hardware components and so, the corresponding approach tends to be of a multiparadigm modeling nature as it enables the study of challenges that arise from applying models at multiple time scales, with multiple levels of detail, and using a combination of multiple formalisms (Mosterman et al., 2004). In this context, it is essential to concentrate on the formalization of the meaning of models that represent the physical world so as to correspond to formalizations typical for embedded computation. This formalization is discussed in Section 2. In particular, previous and related work on advancing Model-Based Design with modeling approximation of computational semantics is introduced in more detail. Section 3 outlines the challenges that have emerged from this innovation and explores the process and opportunities of using explicit models of execution for a Computer Scientist. An explanation that merges CS, Math, and EE expertise is communicated in Section 4. Section 5 concludes the paper.

2. REPRESENTATION OF COMPUTATIONAL APPROXIMATIONS

Model-Based Design (MBD) opens up a space to specify the requirements as graphical models in addition to textual documents (e.g., Sander, 2003). With the emergence of computer supported office applications, such models became available in a digital form. As a natural progression, the digital models became executable thanks to further exploiting computational facilities. To simulate such computational models of a specification, in particular in case of continuous-time behaviors, numerical methods are applied that solve a system of differential and algebraic equations combined with inequalities on a computational device and these methods ultimately introduce approximations.

To make an early specification of a CPS executable, in particular at the ‘system level’ (i.e., including physical as well as computing and network effects), models across a breadth of disciplines must be combined and reconciled to

enable consistent simulation results. One way to do this is to reduce the semantics definition to a formalization that applies to the models of computation across disciplines. Here, a challenge lies on the cusp of software simulation and physics simulation, as the latter typically relies on approximations of numerical algorithms (rather than abstractions, which are the focus in software systems). The challenge that developers who further the actual MBD technologies are nowadays facing is to understand how and what a simulation performs when executing a specific model (i.e., how the execution engine operates). As reported by Mosterman et al. (2011) understanding the computational semantics has become particularly important now that computational models have become prime design deliverables (in addition to other parts of the specification), and so the approximations that these models introduce must be carried consistently through to the eventual implementation.

Previous work by Mosterman et al. (2009) introduced the advances of MBD as a means to formalize the computational approximations themselves. In today's implementations of simulation engines extensive testing is necessary because the semantics are defined by the engine code base of the simulator/solver and so may depend on the code path taken through this code base. With testing requiring the majority of system development resources (Zander et al., 2011), alternatives must be explored to find less expensive solutions. For example, if the execution engine code base can be modeled, then the classes of behavior in the design, as opposed to the specific code path, can give different semantics.

To model the execution engine a declarative notion to analyze the approximations is applied, as illustrated in the middle row in Figure 1. Here, the logic of the numerical integration is represented in a model, whereas the implementation of that model is part of the imperative specification layer (e.g., the execution engine of a MBD application tool). Hence, the semantics of the behavior of the logic part for a particular numerical integration scheme should not change between different implementations, whereas the implementation part can be improved (e.g., using heuristics, implementation workaround, or human interpretation). Such a declarative model of the execution engine enables analyses of the models while accounting for the inherent computational approximations. In prototypes, a stream-based approach in Haskell (Denckla et al., 2008; cf. Benveniste et al., 2003), and selected Simulink blocks subset (Zander et al., 2011) have served as examples of the realization.

3. EXPLORING THE EXECUTION

In this section, a perspective of a CS expert on MBD is delivered. The process of using MBD engineering methods for a CPS design is elaborated. It is presented how a CS expert is provided the opportunity to understand EE and applied mathematics in more detail, and ultimately integrate this knowledge into their own expertise to benefit the CPS-like designs.

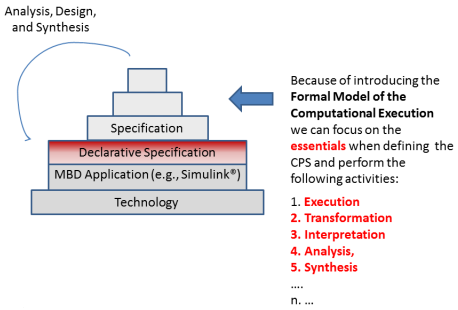

Figure 1: Layered specification of computational approximations

For a CS expert it becomes inevitable to start being confronted with the semantics of computational approximations when dealing with CPS composed of multiple components. Now, matters complicate even further when the various development paradigms are used for various components. For example, one component is defined using the layered specification concept proposed in Section 2, whereas the other component has been developed in a previous product cycle and must be integrated as legacy functionality into a desired architecture. The situation is presented in Figure 2.

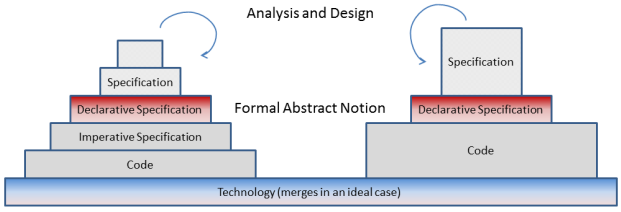

Figure 2: Exchange of declarative specifications to amend the simulation

For a somewhat 'old-school' CS expert the first attempt would be to study the code base and possible interfaces to adjust the application programming interface of the 'outdated' component. However, knowing the implications from the previous Section, an advanced practice makes use of the declarative notion. This approach then allows understanding the simulation logic for each component, and at the design level leaves the implementation details and the functionality out of the scope of considerations. The semantics of the declarative specification are not implementation dependent, whereas those based on code typically are. The integration of such components requires EE expertise. One way to overcome this problem is to add a declarative layer that introduces abstraction and helps understand the behavior of both interacting elements.

Further, the simulation logic is hidden in the numerical solver of differential equations. From a CS viewpoint, a solver (e.g., a differential equation solver) is a component that determines the time of the next simulation step and applies a numerical method to locally solve the set of ordinary differential equations that represent the model (Simulink Documentation, 2012a).

There exist a multitude of solvers and combinations of solvers (e.g., differential equation solver, algebraic loop solver, zero-crossing solver, etc.) and a selection of the proper combination for the desired model execution is a trade-off between simulation targets, such as, stability, accuracy, time available for simulation, etc. Typically, it is recommended to use a highly precise (e.g., a computationally expensive variable-step) solver to generate a reference solution. This allows investigating the model sensitivity (i.e., improve the model robustness) to the use of more approximative (but of lower computational complexity) solvers and solver settings. For example, the choice of a differential equation solver depends on its behavioral characteristics, which can be classified as explicit or implicit, one step or multi step, and single order or variable order, one stage or multi stage.

Solvers handle such issues as stiffness of the system, nonlinearities, zero-crossing, algebraic loops, root finding, and discontinuities (cf. numerical challenges discussed by Moler, 1997). Furthermore, if only considering the Simulink product family, different blocksets use different algorithms that are implemented in separated code bases for solving certain types of equations. For example, the Simscape® solver is based on differential algebraic equations, the PDE® Toolbox on partial differential equations, SimEvents® on discrete-events, and Simulink itself on multiple ordinary differential equations (ODE) algorithms. In the following, ODE solvers are explored in more detail.

Simulation expertise is required to match the model characteristics with a proper solver to obtain desired analysis conditions. It is frequently a challenge for industry experts to match a model with an appropriate solver. For example, if, on the one hand, the system model must have a predictable execution time, however, the execution duration is not restricted, a fixed-step differential equation solver may well serve as a means to experiment with integration order, step size, and accuracy of the simulation. On the other hand, if the system model must be simulated in a time duration that is as short as possible, a variable-step solver often is a better approach because the step size adjusts to the dynamics of the model and the simulation typically executes faster with higher accuracy. In the face of discontinuities in a model, good practice is to also employ a zero-crossing solver to obtain results even quicker and more accurate.

Another significant consideration in industry is code generation for models (Halbwachs et al., 1991). If it is planned to generate code from a model and run it in real-time on a computer system, a fixed-step solver must be applied because the variable-step size cannot be universally mapped to the real-time clock (Simulink Documentation, 2012a).

4. MERGE OF THE DISCIPLINES

In this section the process of creating a declarative specification of the selected execution algorithm known as Heun's method (Renteln, 1995) is presented to highlight the merge of multiple disciplines. These are mathematics, EE, and CS.

In mathematics and computational science, Heun's two-stage method is used for solving ordinary differential equations with a given initial value and it refers to the improved or modified one-step Euler's method.

In previous work, the approximations of a variable-step numerical integration algorithm have been formally modeled in a declarative sense by a strict subset of Simulink blocks (Mosterman et al., 2009; Zander et al., 2011). There, a variable step scheme employs two stages. In the first stage a forward Euler approximation is computed while in the second stage the Euler estimate is used to compute a trapezoidal approximation (*i.e.*, a Heun approximation). The Euler and Heun approximations are then employed in a variable-step numerical integration algorithm that compares the two approximations and in case a given predefined tolerance is exceeded, the step size is reduced by a factor two. Because of the Taylor series expansion, the difference between Euler and Heun's approximations provides an estimate of the error term of the Euler approximation. The user thus sets a tolerance for this error term and if that tolerance is exceeded, a smaller step size is necessary.

The iteration formulas for Heun's method are the following:

$t_{n+1} = t_n + h$, *where h is the step size*
$x_{n+1} = x_n + \Delta x$
average slope $= \Delta x/h$
slope at $t_{n+1} = f(t_n+h, t_n+h*f(t_n, x_n))$
slope at $t_n = f(t_n, x_n)$
$x_{n+1} = x_n + h*$*average slope*
$x_{n+1} = x_n + (h/2)\,(f(t_n, x_n) + f(t_n + h, x_n + h*f(t_n, x_n)))$

These are projected in Figure 3, where the prediction line resulting from the forward Euler integration based on the slope of the left tangent line (i.e., tangent at t_n) is presented on the right-hand side, and the prediction line resulting from Trapezoidal integration based on the slope of the right-hand tangent line (*i.e.*, tangent at t_{n+1}) is presented on the left. The prediction line based on the average slope of the two lines represents the solution and is marked in blue on the actual curve.

Defining a set of equations like the above as a dynamic system in its own right and in a formalized manner is not a trivial task. The engineer reaches out to the role of a mathematician to formulate the proper iteration formulas.

Now, to implement them in code (i.e., in an imperative manner) is yet another task. Moreover, once encoded in an imperative formulation, analysis of the implementation is severely impeded. As a result, the computational semantics of the implementation become fully obscured by the details of the imperative code and often only useful for behavior generation. In effect, the solver implementation is regarded as a black box and, therefore, it prohibits analysis of models that necessitate the use of such a black box solver. As a consequence, the opportunity for analysis of CPS, where continuous-time behavior is inherently present, does not apply directly on the original formulation of the corresponding models that include a differential equation solver.

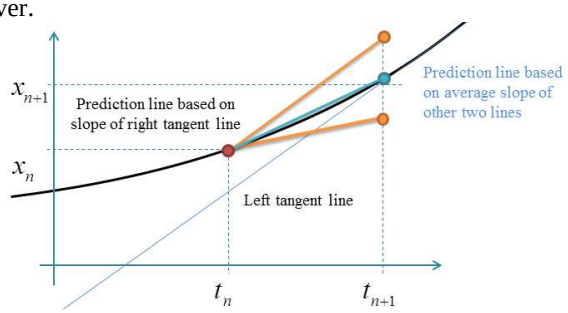

Figure 3: Heun's method in a graph

Modeling in a stream-based notion comes to the rescue as illustrated in Figure 4. Because of the modern modeling language capabilities, the formulas can be designed in a declarative manner and this opens up the space for detailed analyses of the computational semantics. For example, the resulting approximations themselves can be studied in an easier manner, analyses of variable step solver behavior across discontinuities are enabled as documented by Mosterman et al. (2009) and control force synthesis is possible for model-checking as elaborated by Mosterman et al. (2012)[1].

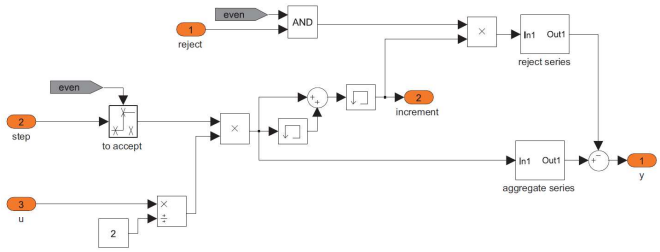

Figure 4: Illustration of the trapezoidal approximation in the Heun's integration scheme exemplified in Simulink notion

Figure 5 presents a simplified set of basic building blocks for complex solvers that are implemented in a Simulink library to enable easy design of an execution engine with a selection of solvers. The solver building blocks are customized according to principles known from CS. A two-stage process is applied here: (i) using a stream-based approach inspired by lambda calculus and (ii) formalizing the solver by restricting the elementary Simulink blocks that are allowed in the design of the building blocks such that they are amenable to a stream based semantic analysis. In this manner the *modeled solver* can be applied for simulating any Simulink model to allow for explicit execution analyses and further syntheses.

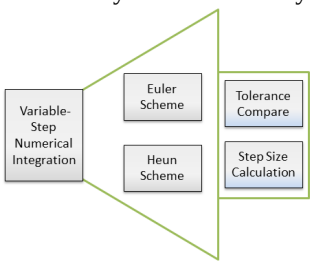

Figure 5: Simplified blockset representation of a Simulink® library for an instantiation of a solver

5. VISION AND CONCLUDING REMARKS

As programmed digital devices continue to shrink in size and cost the computer per se will disappear just as the electric motor disappeared into hundreds of niches in automobiles and households. The essence of CS and EE integration is still there and those who study it are rewarded not just with the appliances but with understanding (Ferguson, 2004; Zander, 2012).

In the following paragraphs, a brief summary of the paper is presented, related paradigms are mentioned, and future outlook is envisioned.

5.1. Summary

In this paper, a vision on further advances in MBD is presented. In particular, a CS perspective is delivered on top of the computational theories applied in today's simulation practice. Reasoning patterns are provided to synthesize MBD-underlying semantics. This opens up the dialog between the MBD user building an executable model specification and MBD-tool designer who creates and maintains the actual execution for such a model.

Conceptually and technically, a layered representation of computational approximations along the MBD process is provided. The declarative specification approach is underlined to define a solver. The computational simulation process is explored and selected solvers are discussed from a system engineering perspective. Finally, the variable step solver based on the combination of Heun and Euler illustrates the vision and is explained from a mathematical, EE, and CS viewpoints in detail.

[1] The model can be downloaded from MATLABCentral: http://www.mathworks.com/matlabcentral/fileexchange/33531-control-synthesis-for-a-surface-mount-device-with-simulink-design-verifier.

5.2. Future Work

Work on formalizing the semantics has been performed before. For example, for Modelica Kågedal and Fritzson (1998) report about shortages in description of modeling the execution engine. Similarly for Ptolemy (Lee and Sangiovanni-Vincentelli, 1996) the dynamics of the numerics are provided as a mere code base.

In future work, more attention is going to be drawn to the benefits and shortcomings of the proposed approach by studying a comprehensive CPS case study (as demonstrated by Zander and Mosterman, 2012). Related work on building the solvers in a declarative manner is going to be explored emphasizing the time aspect. With respect to this, time monotonicity (Zander et al., 2011) and nested time concept are going to be included in the studies.

Technically, it is planned to build a Simulink library including multitude of ODE solvers to let the engineer have insights into the execution at the very early stages of the development.

Recently, converging technologies for behavioral model analysis, simulation, big data extrapolation, and human computation is gaining momentum (Levytskyy, 2009; Zander, 2012). Such integration promises new advances in CPS development that ultimately allows for raising human awareness on many unexplored and unexploited phenomena. Modeling computational approximations and sharing the expertise with other domain colleagues is certainly one of the manners to further improve the growth of the discussed technologies with a mutual and simultaneous acceptance of the human factor that is involved.

References

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and de Simone., R.. The synchronous languages twelve years later. IEEE, 91(1):64–83, 2003.

Carr, N. G. "IT Doesn't Matter," Harvard Business Review, May 2003.

Cellier, F.E., Kofman, E., Continuous System Simulation, ISBN-10: 0387261028, Springer, March 15, 2006.

Denckla, B. and Mosterman, P.J., Stream- and State-Based Semantics of Hierarchy in Block Diagrams, 17th IFAC World Congress, Korea, 2008.

Elmqvist, H. et al. Modelica™ - A Unified Object-oriented Langauge for Physical Systems Modeling: Language specification, 1999. Version 1.3.

Ferguson, E., Impact of offshore outsourcing on CS/IS curricula, Journal of Computing Sciences in Colleges, Volume 19, Issue 4, 2004.

Halbwachs, N., Raymond, P., and Ratel, C. Generating efficient code from data-flow programs. Third International Symposium on Programming Language Implementation and Logic Programming, Germany, 1991.

Kågedal, D. and Fritzson, P. Generating a Modelica Compiler from Natural Semantics Specifications. Summer Computer Simulation Conf., 1998.

Krasner, J. Comparing embedded design outcomes with and without model-based design. Technical report, Embedded Market Forecasters, Framingham, MA, October 2010.

Lee, E.A., Disciplined Heterogeneous Modeling, EECS Department, UC Berkeley, Invited Keynote Talk, MODELS 2010, Norway, 2010.

Lee, E.A. and Sangiovanni-Vincentelli, A. The tagged signal model—a preliminary version of a denotational framework for comparing models of computation. Technical Report UCB/ERL M96/33, University of California, Berkeley, CA, June 1996.

Levytskyy, A., Vangheluwe, H., Rothkrantz, L.J.M., and Koppelaar, H., MDE and customization of modeling and simulation web applications. Simulation Modelling Practice and Theory, 17(4):408-429, 2009.

Moler, C. Are we there yet? Zero crossing and event handling for differential equations. EE Times, 1997. Simulink 2 Special Edition.

Mosterman, P.J., Vangheluwe, H., Computer Automated Multi-Paradigm Modeling: An Introduction, in SIMULATION: Transactions of The Society for Modeling and Simulation International, Vol. 80, Nr. 9, 2004.

Mosterman, P.J. and Justyna Zander, J., Advancing Model-Based Design by Modeling Approximations of Computational Semantics, 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2011), François E. Cellier, David Broman, Peter Fritzson, Edward A. Lee (editors), pp. 3-7, Zürich, Switzerland, September 5, 2011.

Mosterman, P.J., Zander, J., Hamon, G., Denckla, B., Towards Computational Hybrid System Semantics for Time-Based Block Diagrams, In Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09), A. Giua, C. Mahulea, M. Silva, J. Zaytoon (editors), pp.: 376–385, plenary paper, Zaragoza, Spain, September, 2009.

Mosterman, P.J., Zander, J., Hamon, G., Denckla, B., A Computational Model of Time for Stiff Hybrid Systems Applied to Control Synthesis," in Control Engineering Practice, vol. 20, nr. 1, pp. 2-13, January 2012.

President's Council of Advisors on Science and Technology (PCAST). Leadership Under Challenge: Information Technology R&D in a Competitive World. An Assessment of the Federal Networking and Information Technology R&D Program, August 2007.

Renteln, M. von, Karl Heun - his life and his scientific work, in Ronveaux, A., Heun's differential equations, Oxford Science Publications, The Clarendon Press Oxford University Press, pp. XVII-XX, ISBN 978-0-19-859695-0, MR1392976, 1995.

Sander, I. System Modeling and Design Refinement in ForSyDe. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, April 2003.

Simulink®, Using Simulink®, The MathWorks®, Inc., Natick, MA. R2012a Documentation.

Zander, J., Mosterman, P. J., Hamon, G., Denckla, B., On the structure of time in computational semantics of a variable-step solver for hybrid behavior analysis. In Proceedings of the 18th IFAC World Congress, Milan, Italy, September 2011.

Zander, J., Schieferdecker, I., Mosterman, P. J., A Taxonomy of Model-based Testing for Embedded Systems from Multiple Industry Domains, in Model-Based Testing for Embedded Systems, J. Zander, I. Schieferdecker, and P.J. Mosterman (editors), CRC Press, September 2011.

Zander, J., A Vision on Collaborative Computation of Things for Personalized Analyses (talk), PDF, 3rd IEEE Track on Collaborative Modeling & Simulation–CoMetS'12 in WETICE 2012, 21st IEEE International Conference on Collaboration Technologies and Infrastructures, Toulouse, France, June 25-27, 2012.

Zander, J., Mosterman, P.J., Technical Engine for Democratization of Modeling, Simulations, and Predictions, In proceedings of the Winter Simulation Conference, Berlin, Germany, December 9-12, 2012.

Biographies

Justyna Zander is a Senior Technical Education Evangelist at MathWorks in Natick, MA and a Visiting Professor at Gdansk University of Technology in Poland, https://sites.google.com/site/justynazander/.

Pieter J. Mosterman is a Senior Research Scientist at MathWorks in Natick, MA and an Adjunct Professor at McGill University, http://msdl.cs.mcgill.ca/people/mosterman/.