# Advancing Model-Based Design by Modeling Approximations of Computational Semantics

Pieter J. Mosterman[1,2] Justyna Zander[3]

[1]Design Automation Department, MathWorks, USA, pieter.mosterman@mathworks.com
[2]School of Computer Science, McGill University, Canada
[3]Harvard Humanitarian Initiative, Harvard University, USA, justyna.zander@googlemail.com

## Abstract

Over the past decades, engineered systems have increasingly come to rely on embedded computation in order to include advanced and sophisticated features. The unparallelled flexibility of software has been a blessing for implementing functionality with a complexity that could not have been imagined heretofore. One important manifestation of this is in the use of software as the universal system integration mechanism. With the increasing use, however, has come a suite of difficulties in effectively employing software engineering practices because (i) C (the language of choice in embedded software implementation) is very close to the hardware implementation and (ii) software engineering methods typically only consider logical correctness, irrespective of critical characteristics for embedded computation (e.g., response time). To address these problems, Model-Based Design helps raise the level of abstraction while accounting for such critical characteristics. The corresponding models are designed using high-level formalisms such as block diagrams and state transition diagrams whose meaning is particularly intuitive because of their executable nature. The necessity to support increasingly complicated language elements, however, has caused the underlying execution engine to explode in complexity. As a result, the meaning of the high-level formalisms exists almost exclusively by merit of simulation. This paper attempts to present the challenges faced by the current state of Model-Based Design tools and outlines a solution approach by modeling the execution engine.

***Keywords*** Model-Based Design, Cyber-Physical Systems, Modeling, Simulation, Computation, Numerical Integration, Hybrid Systems

4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. September, 2011, ETH Zürich, Switzerland.
 Proceedings available at:
http://www.ep.liu.se/ecp/056/
EOOLT 2011 website:
http://www.eoolt.org/2011/

## 1. Introduction

With the advent of solid state transitors, computation has been on a steady curve of making ever increasing computational power available. Because of the flexibility of software in the design of functionality this trend has been a boon for engineered system designers. Embedding powerful processors into most any engineered system has allowed including features that are close to being limited only by imagination.

However, while miniaturization has been driving an exponential increase in transistor count per surface area, design methods that enable exploitation of the corresponding computing power have been lagging [23]. This problem not only manifests in hardware but also in software. For example, because of integration issues in the avionics software, the F/A-22 fighter continuously struggled to meet its projected deployment dates [25].

The challenges in software producibility [8][1] are perhaps especially curious in the face of the great strides that have been made in programming and software engineering. However, approaches such as *structured programming*, *modularization*, *object-oriented programming*, etc. all concentrate on an *information* rather than a *dynamics* perspective. Consequently, these approaches do not sufficiently benefit the embedded systems or Cyber-Physical Systems (CPS) communities. For example, when it comes to the effects of *embedding* computation in a physical environment the response time of a sequence of computations is critical, yet software engineering approaches typically consider *time* a 'nonfunctional' characteristic (e.g., [10]) even though computational complexity is a standard consideration.

Abstraction is a natural approach to attempt to overcome these difficulties in designing embedded computational features. In CPS design, abstraction attempts to facilitate critical characteristics such as computation timing and data representation by providing high-level formalisms that align well with the problem space (control algorithms, signal processing algorithms, etc.), rather than the solution

[1] In the automotive industry, the extensive use of software has been responsible for distinct quality issues such as documented in "Sync sinks Ford's J.D. Power quality ratings" (http://money.cnn.com/2011/06/22/autos/ford_jd_power_initial_quality).

space (typically C code). The abstraction results in *models* of the underlying software and so the corresponding approach to design is called *Model-Based Design* (e.g., [20]).

Now, to enable Model-Based Design of CPS it is not only necessary to abstract the embedded computation but also the physical world to a representation that is most appropriate for solving the design problem. The use of the most appropriate formalism at the most appropriate level of abstraction constitutes the underlying premise of the field of Computer Automated Multiparadigm Modeling (CAMPaM), which highlights and studies the complications that arise from models at multiple time scales, with multiple levels of detail, and using a combination of multiple formalisms [15, 16, 17]. 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.

In Section 2, Model-Based Design is introduced in more detail. Section 3 outlines the challenges that have emerged from the increasing use of Model-Based Design. An outline of a possible solution approach is communicated in Section 4. Section 5 concludes the considerations.

## 2. Model-Based Design

Design of engineered systems is a process in multiple stages where each stage has a set of requirements as input and a specification as output (e.g., [7]). A produced specification then serves as requirements for the next design stage where additional information necessary to implement a specification is included. This information comprises additional implementation detail as well as functionality. For example, a control law may be specified by a declarative block diagram (e.g., [1]). This specification serves as the requirements for the embedded software of the control law. The block diagram specification then requires additional detail such as data types of its signals so that the specification can be implemented in embedded software. Furthermore, additional functionality such as a scheduler to execute the blocks in the block diagram specification is added.

The rise of Information Technology (IT) has supported digital documents such as created with rich text, presentation, and spreadsheet applications. Likewise, graphical models of dynamic systems such as block diagram models have become available in digital form. Such digital models are at the core of Model-Based Design. This is depicted in Figure 1 where a system specification is represented by a triangle. The computer application that is used to create the digital form of a specification is represented by the rectangle at the base of the triangle. This computer application executes on a host technology, as depicted by the rectangle at the bottom.

Because a digital form of the specification exists, any models that it contains can now be processed automatically. Model-Based Design has exploited this substantially by enabling the (i) execution of dynamic models, (ii) transformation of models to include implementation detail (or even complete C code), and (iii) integration of models with further design artifacts such as requirements and tests. In turn, the availability of all these specific features has formed the catalyst to developing an entire test and verification ecosystem around the digital models through the various design stages.

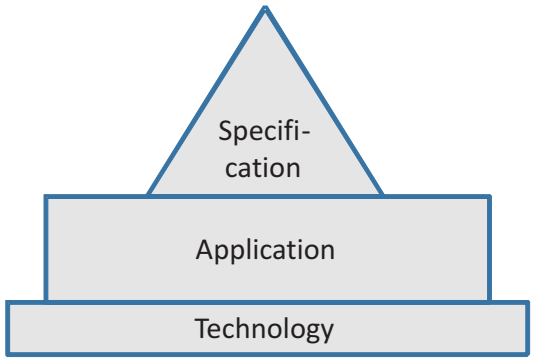


**Figure 1.** Digital representation of models.

## 3. The Challenge

By making the digital representation of a model executable, Model-Based Design introduces approximations necessary for simulation algorithms. For example, the behavior of a model as a differential equation system is typically approximated by numerical integration algorithms that are implemented in software modules (e.g., [21]). In case such a continuous-time model includes discrete mode changes, a *hybrid dynamic system* [3, 12] emerges that comprises further approximations (e.g., because of the root-finding algorithms that detect the point in time when a discrete mode change occurs [13] and the reinitialization algorithms that may follow such a mode change [14]). These approximations are depicted in Figure 2 by representing the original system specification triangle of Figure 1 as a discretized triangle. Often the approximations are specific to the particular Model-Based Design (MBD) application that is used (e.g., Simulink® [24]).

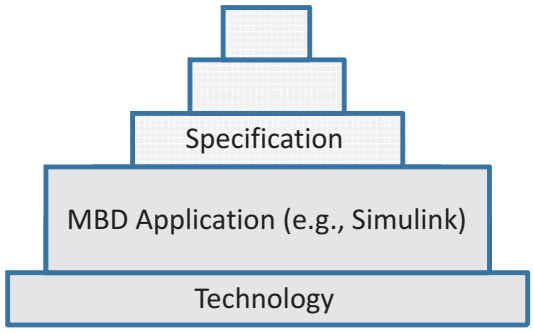


**Figure 2.** Computational approximation of executable models.

In spite of these approximations, Model-Based Design has shown to be very successful [9, 11]. To a large extent this is precisely because an executable model exists. For example, a computational model of a physical system can be fit to constitute a precise representation by automatic system identification methods that rely on a multitude of simulations to search for the optimal underlying model. As the

digital models have become prime design deliverables between design stages, the approximations are carried consistently through to the eventual implementation. As a result, the computational models can be successfully exploited, though at the expense of a necessity for extensive testing throughout. Moreover, any analysis other than simulation based is prohibitively complex, as it requires analysis of the entire execution engine code base.

To further mature and advance Model-Based Design, the challenge that developers of Model-Based Design applications face is then one of defining the computational approximations. Specifically, the computational approximations are best *modeled* at a declarative level, as illustrated in Figure 3. Such a model of the execution engine enables analyses of the models that the execution engine executes and that are used in design, while accounting for the inherent computational approximations. Moreover, the declarative representation enables synthesis methods based on computational approaches such as *model checking* (e.g., [19]).

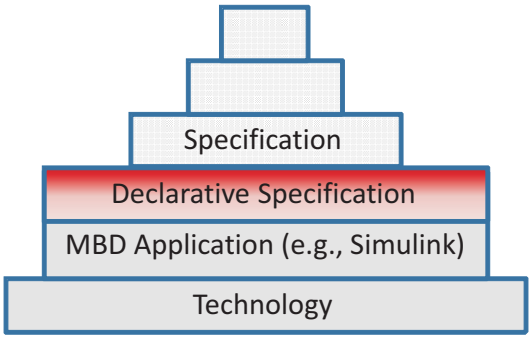


**Figure 3.** Declarative specification of computational approximations.

## 4. A Solution Approach

An approach to tackle the challenge of defining the computational approximations must be declarative and sufficiently formal so as to serve as a specification or reference implementation. The approach must support analysis and synthesis while being sufficiently powerful to apply to a broad range of models of computation.

In previous work, Haskell was employed as the definition language [5] which satisfies the declarative requirement. Moreover, the functional nature supported a stream-based approach (i.e., 'state-less behavior' of functions) which aids greatly in analyzability (e.g., [2]). Furthermore, work on synchronous languages [4, 6] has demonstrated the breadth of applicability of such a stream-based declarative approach while enabling synthesis of an implementation in terms of a state machine or software code.

More recently, the approximations of a variable-step numerical integration algorithm have been formally modeled in a declarative sense by a strict subset of Simulink blocks [18, 19]. This subset includes only 'delay' and 'latch' operations as so-called 'sequential' operations [22] that communicate between sample times. For example, Figure 4 shows a model of the *Euler* and Figure 5 of the *trapezoidal* integration algorithms as they are used by a Heun numerical integration scheme. Such a scheme employs two stages, where 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.

The Euler and Heun approximations are then employed in a variable-step numerical integration algorithm that compares the two approximations and in case a tolerance is exceeded, the step size is reduced by a factor two. Because in this case the integration has to undo the original approximation as produced by the *aggregate series* facility, the Euler (Figure 4) and Trapezoidal approximation (Figure 5) include a *reject series* facility to selectively subtract the approximations of failed time steps.

Because of the declarative model of a variable-step solver, detailed analyses of the semantics in the face of zero-crossings are enabled [26]. Moreover, it allows a feedforward control to be synthesized for a stiff hybrid dynamic system by using computational model checking methods [19].

## 5. Conclusions

In this paper, a vision for continuous improvement of Model-Based Design has been outlined. It was argued that Information Technology ultimately enabled the development of Model-Based Design principles that can be applied for developing computable and executable graphical models. Such models introduce the necessity to properly understand the semantics of the computation because the power of computational execution amplifies numerical approximations. This highlights the challenge that Model-Based Design is facing today. Whenever a specification of a system is considered, its computational meaning comprises the approximation of the computation method used for simulation. In this manner, the semantics of a system design permeates down to the implementation detail of the simulation engine (in some cases, it is even affected by host technology specifics). This, in turn, renders activities such as system analysis, design refinement, and synthesis very complex and prone to human mistakes or errors, even (or especially) in case automatic processing is exploited.

To mitigate the complexity, a separate conceptual and simultaneously technical layer is introduced to the existing Model-Based Design artifacts. This layer is a formal declarative specification that is applicable to a broad range of models of computation. Its functional nature is supported by a stream-based approach, which lends itself well to analysis. As an example, a declarative stream-based implemention is outlined for a variable-step numerical integration algorithm (based on a forward Euler and Heun approximation).

The discussion above applies in particular to Cyber-Physical Systems (CPS) where both embedded computation and the dynamics of the physical world must be represented in a unified computational manner to allow for comprehensive analyses. Neglecting the computational approximation and execution semantic for such a combination is prone to improper results and unpredictable conclusions.

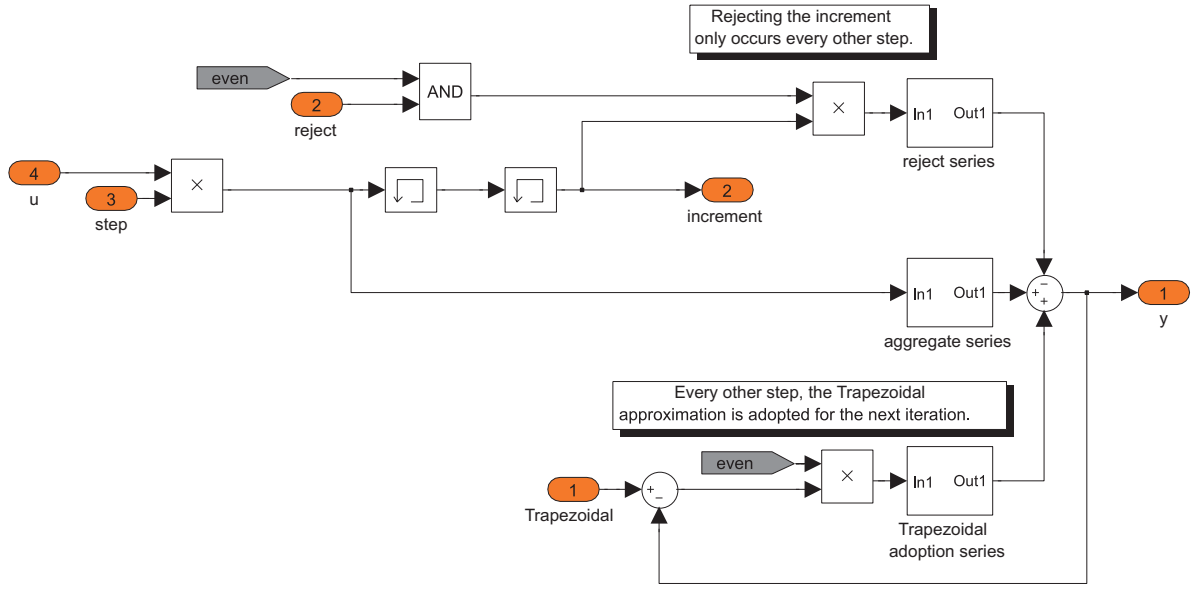


**Figure 4.** Euler approximation of the Heun numerical integration algorithm.

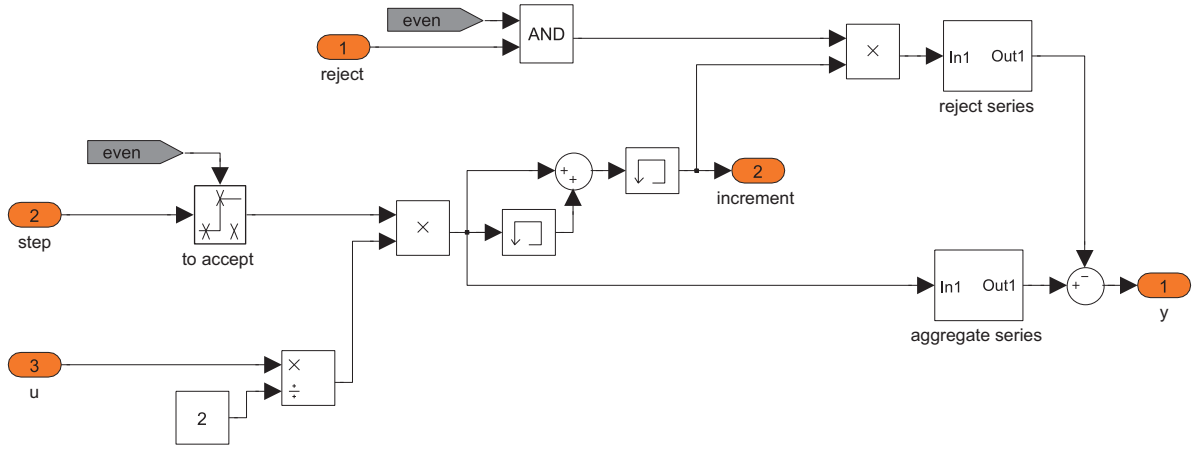


**Figure 5.** Trapezoidal approximation of the Heun numerical integration algorithm.

In the context of future directions, the formal model of the execution engine represented by the declarative layer proposed in this paper introduces the following benefits over the current approach. The analysis and specific design of a CPS component can now be performed at a high abstraction, without involving an imperative specification or even implementation. Because the computation is described in an abstract manner, it is easier for engineers to understand. In turn, this illustrates how Model-Based Design is not only of value for designing engineered systems, but the principles also apply to Model-Based Design tools themselves, thereby unlocking the potential for an autocatalytic trend



## References

[1] Karl J. Åström and Björn Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[2] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[3] Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2001.

[4] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[5] Ben Denckla and Pieter J. Mosterman. Stream- and state-based semantics of hierarchy in block diagrams. In *Proceedings of the 17th IFAC World Congress*, pages 7955–7960, Seoul, Korea, July 2008.

[6] Nicolas Halbwachs, Pascal Raymond, and Christophe Ratel. Generating efficient code from data-flow programs. In *Third International Symposium on Programming Language Implementation and Logic Programming*, Passau, Germany, August 1991.

[7] Derek J. Hatley and Imtiaz Pirbhai. *Strategies for Real-Time Systems Specification*. Dorset House Publishing Co., New York, New York, 1988.

[8] High Confidence Software and Systems Coordinating Group. High-confidence medical devices: Cyber-physical systems for the 21st century health care. Technical report, Networking and Information Technology Research and Development Program, feb 2009.

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

[10] Edward A. Lee. What's ahead for embedded software. *Computer*, 33(9):18–26, September 2000.

[11] Joy Lin. Measuring return on investment of model-based design. *EE Times Design*, May 2011.

[12] Nancy Lynch and Bruce Krogh, editors. *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2000.

[13] Cleve Moler. Are we there yet? zero crossing and event handling for differential equations. *EE Times*, pages 16–17, 1997. Simulink 2 Special Edition.

[14] Pieter J. Mosterman. HYBRSIM—a modeling and simulation environment for hybrid bond graphs. *Journal of Systems and Control Engineering*, 216(1):35–46, 2002.

[15] Pieter J. Mosterman, Janos Sztipanovits, and Sebastian Engell. Computer automated multi-paradigm modeling in control systems technology. *IEEE Transactions on Control System Technology*, 12(2):223–234, March 2004.

[16] Pieter J. Mosterman and Hans Vangheluwe. Guest editorial: Special issue on computer automated multi-paradigm modeling. *ACM Transactions on Modeling and Computer Simulation*, 12(4):249–255, 2002.

[17] Pieter J. Mosterman and Hans Vangheluwe. Computer automated multi-paradigm modeling: An introduction. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 80(9):433–450, September 2004.

[18] Pieter J. Mosterman, Justyna Zander, Gregoire Hamon, and Ben Denckla. Towards computational hybrid system semantics for time-based block diagrams. In *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 376–385, Zaragoza, Spain, September 2009. plenary paper.

[19] Pieter J. Mosterman, Justyna Zander, Gregoire Hamon, and Ben Denckla. A computational model of time for stiff hybrid systems applied to control synthesis. *Control Engineering Practice*, 19, 2011.

[20] Gabriela Nicolescu and Pieter J. Mosterman, editors. *Model-Based Design for Embedded Systems*. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, Boca Raton, FL, 2009. ISBN: 9781420067842.

[21] Linda R. Petzold. A description of DASSL: A differential/algebraic system solver. Technical Report SAND82-8637, Sandia National Laboratories, Livermore, CA, 1982.

[22] Ingo Sander. *System Modeling and Design Refinement in ForSyDe*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, April 2003.

[23] Semiconductor Industry Association. International technology roadmap for semiconductors: 1999 edition—design. Technical report, Sematech, Austin, TX, 1999.

[24] Simulink®. *Using Simulink®*. MathWorks®, Natick, MA, March 2011.

[25] Michael Sullivan. TACTICAL AIRCRAFT–F/A-22 and JSF acquisition plans and implications for tactical aircraft modernization. Technical Report GAO-05-519T, United States Government Accountability Office, April 2005.

[26] Justyna Zander, Pieter J. Mosterman, Gréegoire Hamon, and Ben Denckla. 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.