Towards Computational Hybrid System Semantics for Time-Based Block Diagrams

Pieter J. Mosterman [*] **Justyna Zander** [**] **Gregoire Hamon** [*]
Ben Denckla [***]

[*] *The MathWorks, Natick, MA 01760, USA*
(e-mail: pieter.mosterman@mathworks.com, gregoire.hamon@mathworks.com).
[**] *Fraunhofer Institute FOKUS, 10589 Berlin, Germany*
(e-mail: j.zander@fokus.fraunhofer.de)
[***] *Denckla Consulting, Los Angeles, CA, USA*
(e-mail: bdenckla@alum.mit.edu)

Abstract: At the core of Model-Based Design, computational models have caused an autocatalytic trend to use computation in design by unlocking the potential of model transformations. Precisely specifying a computational transformation requires well-defined semantics of the source and target representations. In this regard, continuous-time behavior is an essential aspect of time-based block diagrams that is typically approximated by numerical integration. The corresponding theory, however, is mostly concerned with local error and the *mathematical* semantics of long time behavior fails to be sufficiently precise from a computational perspective. In this work, first a *computational* semantics is developed based on a multi-stage variable-step solver. Next, the computational semantics of the discrete and continuous parts of *hybrid systems* and their interaction are formalized in a unifying framework. The framework exploits a successful functional approach to defining discrete-time and discrete-event behavior established in other work. Unification is then achieved by developing a computational representation of the continuous-time behavior as pure functions on streams.

Keywords: Computational methods, Computer simulation, Computer-aided control system design, Embedded systems, Numerical simulation, Synchronous data flow, Systems design, Variable-structure systems, Verification, Zero crossings

1. INTRODUCTION

Over the past decades, the feature set of engineered systems has rapidly increased. To a large extent this increase has been enabled by merit of the flexible realization of functionality in embedded software. Simultaneously, Model-Based Design (e.g., Friedman and Ghidella (2006); Nicolescu and Mosterman (2009); Potter (2004)) has become essential to competitively, if not just successfully, engineer embedded systems (e.g., Jones (2005)). Computation can then be identified as the main driver that enables (i) the design of modern systems and (ii) an unparallelled feature differentiation. Moreover, in addition to theory and experimentation, the President's Information Technology Advisory Committee (2005) identified computation as having become the third pillar of scientific discovery. These combined observations underscore the importance of, indeed, computational studies of computation.

Concentrating on Model-Based Design, of particular importance is the status of computational models as first-class deliverables to allow:

- An interconnected infrastructure for management of design artifacts with support for fine-grain active links. For example, requirements for production code can be linked to the code lines with automatic tracing between them.
- Generating behavior from specifications to provide early insight into design decisions, share unambiguous information between design teams, and automate tasks such as optimization and experimental design.
- Verification of domain specific constraints that are static as well as dynamic in nature. This includes statically checking not only static semantics (e.g., type checks) but also dynamic semantics (e.g., array out of bound indexing).
- Automated completion of a partial design and generation of an implementation by means of transformations. For example, data types of variables in a design can be automatically determined while imperative source code for an implementation can be generated from a declarative design.

If not solely, design deliverables used to primarily be paper documents. Simulation engineers would code a documented design specification in FORTRAN for design engineers to study its behavior. After several such iterations the design specification would then be implemented by software engineers, for example in ADA.

Computational models have, to a large extent, eliminated the strict boundaries between these different activities. As a result, models are now being shared at an enterprise level as primary elements of communication between groups, departments, and even companies. This trend instilled a desire to establish open compendia (e.g., Adam et al. (2008)) or repositories (e.g., Mosterman et al. (2008)) of models that are well documented, fully tested, and possibly certified or accredited.

In turn, the rapid increase in available computing power has given rise to the *design productivity gap*, documented by the Semiconductor Industry Association (1999) as the discrepancy between the computations that can be manufactured versus the computational functionality that can be designed. To close this gap, design must be enabled at a more abstract level. A related manifestation is the software producibility problem that has been identified by Sullivan (2005) as one of the causes for the F-22 delivery delays and budget overruns. At present, there is isolated evidence of the significance of raising of abstraction levels from the source code to the model level. For example, fixed-point data type determination can now be performed at the model level, which unlocks the benefits of simulating fixed-point behavior before coding an implementation. A more systematic and structured approach, however, is needed with its success contingent upon the availability of compilers and platforms.

Evaluating these observations exposes two computation related trends. In a horizontal sense, models are shared by user communities within and across organizations, often as a container of intellectual property (IP). In a vertical sense, models are exploited in 'lowering' a specification to an implementation, promoting model transformations (e.g., Mens and Gorp (2006)) to critical ingredients in modern engineered system design. As outlined by Mosterman and Vangheluwe (2004), research at the confluence of (i) modeling of model transformations to enable (ii) use of multiple (domain-specific) formalisms at (iii) varying levels of abstraction has given rise to the field of Computer Automated Multiparadigm Modeling (CAMPaM), which attempts to provide a sound reasoning framework across these dimensions, supported by model-based tools.

Common to the computation trends, then, is a distinct need for well-defined semantics of the models so as to:

- be able to decouple the IP captured by a model from its implementation,
- design domain-specific formalisms and raise the abstraction level, and
- enable domain-specific formalisms by providing automation support for lowering.

However, while the abstract syntax of textual formalisms can be well defined in a Backus-Naur Form and of graphical formalisms by a metamodel (e.g., Engstrom and Krueger (2000); Flatscher (2002)), Zhang and Xu (2004) posit that so far there is no like generally accepted solution for rigorously, consistently, and unambiguously capturing semantics.

In previous work, Denckla and Mosterman (2006) defined the computational semantics of discrete-time modeling formalisms such as time-based block diagrams in Simulink$^{\circledR}$ (2008) as a composition of pure functions on streams. Because of the time sampled nature of embedded control and signal processing systems, discrete-time formalisms are often used in modeling. The strict functional approach allows capturing the meaning of a model in a denotational sense (i.e., *what* it does) as opposed to an operational sense (i.e., *how* it does something) (e.g., Nielson and Nielson (1992); Zhang and Xu (2004)). This decouples a specification from its implementation and in general provides a representation that is easier to reason about, for one because with pure functions there is no internal state to account for.

With physics comprising an essential part of embedded systems, there is an interest in also defining the semantics of the corresponding modeling formalisms as denotational composition of functions on streams. Since physics is well modeled by differential equations (e.g., Breedveld (1984); Cellier et al. (1996)), either ordinary differential equations (ODEs) or differential and algebraic equations (DAEs), a unifying framework must encompass the computational semantics of discrete-time and continuous-time models. Moreover, discrete state changes are often part of the otherwise continuous-time models of physics, for example to capture mode changes in models of a component such as a valve or diode. Thus, the framework should further support defining semantics of discrete-event models.

The work presented here attempts to formulate the computational semantics of the continuous-time part of a model when it is specified by differential equations. It recognizes that the numerical integration algorithms employed for computational simulation are key in precisely capturing how a model executes. In order to honor the continuity requirements of differential equations (such as a continuous domain and time-derivative constraints), a multi-stage variable-step solver is studied.

The presentation is structured so that Section 2 first provides a brief background of the challenges that the mathematical theory of numerical integration is facing. In Section 3, two numerical integration schemes are developed to constitute the basis of a computational semantics. Section 4 then presents a functional representation of a variable-step solver based on these integration schemes. In Section 5, a case study combines the variable-step solver with discrete-time and discrete-event model parts. Section 6 evaluates how the presented work advances previous achievements as well as its more general contribution. Section 7 concludes and outlines future work.

2. BACKGROUND

As stated by the High Confidence Software and Systems Coordinating Group (2009), a critical challenge in the design of high-integrity embedded systems, especially when they are of a self-configuring nature, is modeling the environment. An important motivation is the potential role of accurate physics models of the environment in the certification (e.g., SC-167 (1992)) of such variable structure systems. Similarly, in control system design, a model of the physical system under control is indispensable for the synthesis of control algorithms (e.g., Åström and Wittenmark (1984)).

Given the macrophysical principles of *conservation of energy* and *continuity of power* (e.g., Paynter (1961); Falk and Ruppel (1976)), dynamic models of physical systems are often represented by differential equations, possibly supplemented with algebraic constraints (e.g., to formulate balance equations). These models can be designed based on first principles, such as the laws of physics. Parameters are measured or estimated and once a system of equations is arrived at, it is common to further tune the parameters to best fit the model to measured data. In the extreme of empirical models, only the order of a model may be provided as an approximation of required detail.

Generally, the parameters are fit based on computational simulation of the differential equations. As a testimony to the absence of an innate and unique separation between solver and model, the ultimate model then incorporates the computational characteristics of the numerical solver that generates the simulation. So, parts of a model may be moved to the solver and vice versa.

Though the mathematical derivation of solver equations provides an estimate of the error bounds, and as such it can be claimed that the solver semantics are defined at the mathematical level, these error bounds are only local. Guckenheimer (2002) elaborates that the global error of numerical integration is unknown for most practical cases. For example, consider an ideal inductor/capacitor oscillator

$$\begin{cases} p = -\dot{q} \\ q = \dot{p} \end{cases} \tag{1}$$

where p may represent flux, q may represent charge, and where the mass and capacitance parameters are chosen to be 1. The initial flux is chosen $p_0 = 1$ and the initial charge $q_0 = 0$. This system of equations can be solved in Simulink with a variable-step Dormand-Prince integration method (`ode45`) that computes fourth and fifth order Runge-Kutta solutions and adapts the step size based on the difference. The solution exhibits a decrease in energy over time, as shown by the solid line in Fig. 1.

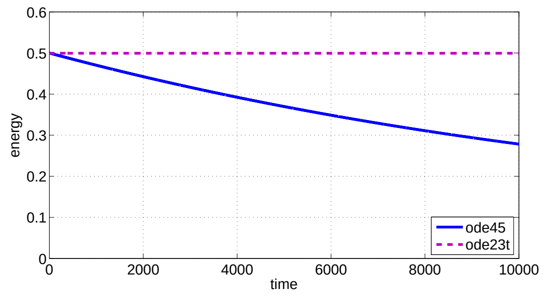


Fig. 1. Numerical simulation of total energy in an ideal oscillator

Numerical dissipation is typical for integration schemes, where the mathematical theory is mostly concerned with the accuracy of the integration algorithm as the discretization tends to 0. Dynamic systems theory in turn, is more concerned with asymptotic behavior but finds it difficult to determine the long time error in general. Qualitative analyses exploit structure in the underlying problem and may be the best approach. For example, Sanz-Serna (1992) provides an overview of how symplectic integrators preserve the properties of Hamiltonian systems such as the ideal oscillator in (1). Figure 1 illustrates this by the dotted line, which is a solution generated by numerical integration with the modified trapezoidal scheme in Simulink (`ode23t`) that according to Shampine et al. (1999) eliminates numerical damping.

Shampine et al. (1999) continue to state that numerically damped behavior at infinity may, in fact, be desirable. Indeed, symplectic integrators have limited applicability in general, which brings about a situation where mathematical semantics of the long time error behavior of general-purpose solvers are still not well developed. This is even more pronounced for solvers with variable step and sophisticated error control (e.g., Bujakiewicz (1994); Petzold (1982)). Heuristics to improve performance for classes of systems further exacerbate the matter. Moreover, to be able to efficiently handle continous-time behavior interspersed with discrete changes, these solvers interact with algorithms to accurately detect and locate when discrete events occur. The delicate interplay between these different algorithms with different error bounds and convergence characteristics makes it difficult to establish a comprehensive mathematical analysis. This holds especially true in the face of potentially infinite sensitivity because of discrete state changes. In order to precisely define the semantics of a model that relies on a given solver, though, an accurate definition of the solver behavior is imperative. A computational semantics then provides a representation for analysis at a useful level without attempting to solve the dynamic systems problems.

Not any less important a ground for developing a computational semantics is the intent to establish a framework that allows integrating continuous-time semantics with discrete-time and discrete-event semantics of various formalisms. In this regards, a computational representation of a solver may facilitate capturing the semantics of combined formalisms, which enables a sound approach to the study of the intricacies that emerge from such combinations. Given that discrete-time behavior is well defined by functions that operate on streams (e.g., Caspi and Pouzet (1997); Reekie (1994)), unification can be achieved by choosing stream-based functions to capture the computational semantics of continuous-time behavior as well. From an analytic point of view, this allows a comprehensive consideration as a functional composition.

3. A COMPUTATIONAL REPRESENTATION OF CONTINUOUS TIME

To obtain a computational representation of time, a fixed-step Euler and trapezoidal integration scheme are first reviewed. Next, a variable-step approach is outlined.

3.1 Fixed-Step Integration

Numerical integration is performed in a number of stages at which the dynamic system is evaluated. A single-stage and a multi-stage approach follow some preliminaries.

Preliminaries Disregarding the forcing function for now, a continuous-time function can often be represented as an ODE

$$\frac{dx}{dt} = \dot{x} = f(x); \quad f : R^n \to R^n \tag{2}$$

where x is the state vector. The vector field (2) that this defines is assumed to be Lipschitz continuous and to satisfy the usual conditions for existence and uniqueness of a solution. In a neighborhood of $R^n \times 0$, this field then has a unique flow $\Phi : R^n \times R \rightarrow R^n$ such that $\Phi(x, 0) = x$ and $\dot{\Phi}(x, t) = f(\Phi(x, t))$. Since there rarely are explicit formulae for Φ in terms of f, iterative numerical integration algorithms are the norm for obtaining solutions over time based on discrete approximations.

Single-Stage Integration To compute the time, t, map for each iteration, a Forward Euler integration scheme approximates the state that results from making a time step of magnitude h as

$$x((k+1)h) = x(kh) + h\dot{x}(kh) = x(kh) + hf(x(kh)) \quad (3)$$

with k the natural numbers representing the iteration step.

To investigate the numerical accuracy of this method, the Taylor series of f can be expanded around kh to determine the value at $(k+1)h$ (note that $(k+1)h - kh = h$)

$$x((k+1)h) = x(kh) + \frac{\dot{x}(kh)}{1!}h + \frac{\ddot{x}(kh)}{2!}h^2 + \dots \quad (4)$$

This agrees to the second degree with the Forward Euler approximation in (3). The estimate of the error per time step (the local error) then becomes $\frac{\ddot{x}(kh)}{2}h^2$ or $O(h^2)$.

The long time error can be investigated by first defining a discrete map of step h, $E_h(x) = x + h \cdot f(x)$. To integrate to a point in time, t_e, this map can be iteratively applied $E_h^{k+1}(x) = E_h(E_h^k(x))$ with $E_h^0(x) = x$. Now, by reducing the step size $h \rightarrow 0$ and taking $l \rightarrow \infty$ steps such that $l \cdot h = t_e$, the iterative solution $E_h^l(x) \rightarrow \Phi(x, t_e)$.

The compounded error bound at t_e becomes $l\frac{\ddot{x}(kh)}{2}h^2$. Though the error can be made arbitrarily small by reducing h, in turn l becomes arbitrarily large. A large l has two complications: (i) an arbitrarily large error from floating point computations is introduced and (ii) the computational performance becomes arbitrarily slow.

Multi-Stage Integration To mitigate the error and performance problems, at least to an extent, a multi-stage solver can be applied. For example, a trapezoidal integration scheme employs the average of the gradient at the beginning and end point of the integration step as

$$x((k+1)h) = x(kh) + \frac{h}{2}\left(\dot{x}((k+1)h) + \dot{x}(kh)\right). \quad (5)$$

This can be rewritten to

$$x((k+1)h) = x(kh) + h\dot{x}(kh) + \dots \\ \frac{h^2}{2}\left(\frac{\dot{x}((k+1)h) - \dot{x}(kh)}{h}\right) \quad (6)$$

and with a finite difference approximation $\ddot{x}(kh) = \frac{\dot{x}((k+1)h) - \dot{x}(kh)}{h} + O(h)$ this matches the Taylor series up to the third degree. The error term then becomes of order $O(h^3)$. As a result, the solution converges an order of magnitude quicker to 0, thereby reducing the size of l to get to t_e with the same error bound, while only requiring a linear increase in the number of computations.

The drawback is the use of $\dot{x}((k+1)h)$ on the right-hand side, which because of $k+1$ leads to an implicit integration scheme. This can be solved by computing a Forward Euler approximation $\dot{x}((k+1)h) = f(x(kh) + h\dot{x}(kh))$ to obtain an explicit scheme again.

It is important to note that the first-order and higher-order derivatives impose continuity constraints on the continuous-time behavior. Mixing discretized continuous-time with discrete-time behavior may invalidate the mathematical assumptions and corresponding error bounds.

3.2 Variable-Step Solver

Because of the inverse relation between the step size h and the rate of change in $f(x)$, the step size h can be varied over time as $f(x)$ changes, without compromising the local error bound. For x where $f(x)$ changes with a relatively high rate with respect to t, the system is said to be *stiff*. The solver can thus selectively choose small steps in stiff intervals, while larger steps can be taken elsewhere to improve efficiency.

Integration schemes with an adaptive step size are typically referred to as *variable-step solvers*. An estimate of the error term may be responsible for the change in step size during integration. In some approaches, the error term is approximated by evaluating the difference between the change of x as computed by two different numerical integration algorithms (e.g., the Dormand-Prince method). If this difference exceeds a given threshold, the step size chosen is reduced and the evaluation performed anew.

4. A FUNCTIONAL VARIABLE-STEP SOLVER

Previous work by Denckla and Mosterman (2008) developed a combined stream-based and state-based approach to defining the computational semantics of block diagrams. A functional semantics (i.e., *without* explicit state), BdFun, and a systems semantics (i.e., *with* explicit state), BdSys, were specified in the general *lambda calculus* (e.g., Peyton-Jones (1987)) framework of computation. The functional language Haskell (e.g., Jones (2003)) was chosen for the implementation which allows defining a stream as a potentially infinite (because of *lazy evaluation*) list of values. Embedding state-based semantics into a stream-based semantics was achieved by hierarchical decomposition. This, in turn, supported the implementation of a variable-step solver as a system with explicit state so as to let the variable-step solver manipulate the state freely.

The work presented here aims at eliminating the strict decomposition boundary around the variable-step solver by providing it as a stream-based representation. As such, reasoning about the continuous-time aspects embedded in a discrete-time model becomes completely transparent.

A variable-step solver is then represented as a pure function (i.e., without side effects), g, on an input stream, u, returning an output stream, y, as in

$$y = g(u). \quad (7)$$

To implement an explicit variable-step solver based on the Forward Euler and trapezoidal integration schemes of Section 3, a two-stage evaluation is required. The first stage computes the Euler approximation and the second stage employs this approximation to compute the average gradient over the integration step for the trapezoidal approximation. A functional implementation of the solver cannot rely on internal state to reinstate the values at the beginning of the integration step. Instead, the computed change in state is subtracted if the step size must be reduced. This results in the Euler integration scheme

$$y_{euler}(e) = \dots \begin{cases} \sum_{i=1}^{e} u(i)h(i) - u(i-2)h(i-2)p(i) & \mathit{if\ odd(e)} \\ y_{euler}(e-1) & \mathit{otherwise} \end{cases} \tag{8}$$

with e the evaluations as natural numbers larger than 0 and where p is 1 if the step size h must be reduced and 0 otherwise. The undefined initial values ($u(-1)$, $u(0)$, $h(-1)$, and $h(0)$) are taken to be 0. Note that the Euler integration is computed every other evaluation to allow the two-stage nature of the trapezoidal scheme.

The trapezoidal approximation adds the contribution at the beginning and end of the integration step based on the same integration step size. If necessary, the state is reinstated at the beginning of the integration step by subtracting the aggregate contribution computed for the previous step. This results in the following integration scheme

$$\begin{aligned} y_{trap}(e) = \sum_{i=1}^{e} &\frac{(u(i-1)+u(i))h(i-1)}{2} - \dots \\ &\frac{(u(i-3)+u(i-2))h(i-3)}{2} p(i-1) \end{aligned} \tag{9}$$

where undefined initial values can be taken to be 0.

The contributions of the Euler and of the trapezoidal approximations over the integration step are then compared based on the difference in the contribution to each of the states

$$\begin{aligned} d(e) = &(u(e-3) + u(e-2))\frac{h(e-3)}{2} - \dots \\ &u(e-2)h(e-2) \end{aligned}. \tag{10}$$

If the maximum of each of the absolute differences, $|d(e)|$, is less than a predefined tolerance, *tol*, the step is 'accepted' and time moves forward. Otherwise, the time step is reduced. The acceptance test is implemented by the variable p as

$$p(e) = \begin{cases} 0 & \mathit{if\ max}(|d(e)|) < \mathit{tol} \\ 1 & \mathit{otherwise} \end{cases}. \tag{11}$$

The step size is adapted based on bisection, starting from the maximum step size, h_{max}, as prescribed by the user

$$h(e) = h_{max}(1 - p(e)) + \frac{h(e-1)}{2} p(e). \tag{12}$$

The solver output, y, alternates between the Euler approximation (to enable the trapezoidal scheme) and the trapezoidal approximation, where the trapezoidal approximation is considered to be more accurate

$$\begin{aligned} y(2e+1) &= y_{euler}(2e+1) \\ y(2e+2) &= y_{trap}(2e+2) \end{aligned} \tag{13}$$

These equations are implemented in Simulink, using the *Memory* block as a 'pre' operator (a function $y(e) = g_{pre}(u(e))$ that produces $y(e) = u(e-1)$). The evaluations are performed iterating on a discrete evaluation step with nominal value, 1.

5. SIMULATING A STIFF AND HYBRID SYSTEM

The behavior of the solver developed in Section 4 is studied based on a controlled bouncing ball. The system combines stiff behavior when the ball is in contact with the floor with hybrid behavior because of continuous-time motion, discrete-time control, and discrete-event switching logic.

5.1 A Computational Model

Consider the model of a controlled bouncing ball in Fig. 2. A ball, a rigid body with mass m, is pulled off the floor by a hoist with a controlled force F_{pull}. In addition, there is a gravitational force, F_g, acting while the floor is modeled as a stiff spring, C, and damper, R, that combine to exert a reaction force, F_{floor}.

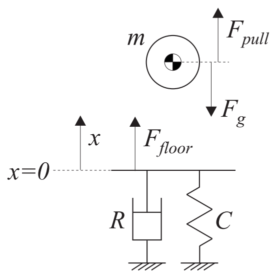


Fig. 2. A controlled bouncing ball

The behavior can be represented as an explicit ordinary differential equation on the position of the ball, x, and the velocity of the ball, v,

$$\begin{aligned} \dot{x}(t) &= v(t) \\ \dot{v}(t) &= \frac{F_{pull}(t) - F_g + F_{floor}(t)}{m} \end{aligned} \tag{14}$$

with the forces as a forcing term. The differential equations can be discretized by the solver of Section 4 based on the following mapping of variables

$$u = \begin{bmatrix} 1 \\ \dot{x} \\ \dot{v} \end{bmatrix}, \; y = \begin{bmatrix} t \\ x \\ v \end{bmatrix}, \; x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

where the first state variable, t, is included in order to obtain time as a function of evaluations e, $t(e)$. This allows a unifying framework where all dynamics are represented as functions of e.

For example, the pull force may be generated by a controller that operates at a given sample rate of 0.5 (s) and that can be modeled by a discrete-time representation with integer clock. In case an initial pull force is chosen as 20, then 10, and then 0 for the remainder of time, this leads to the following equations

$$F_{control}(k) = \begin{cases} 20 & \mathit{if}\ k = 0 \\ 10 & \mathit{if}\ k = 1 \\ 0 & \mathit{otherwise} \end{cases} \tag{16}$$

with k the natural numbers.

To combine (16) with the discretized differential equations, the behavior must be represented as a function of e. Because the control is only specified at isolated points in time, a rate transition from discrete time to continuous time is mandated. An often used zero-order hold (ZOH) with sample time T_s (here $T_s = 0.5$) gives

$$F_{pull}(t) = F_{control}(\left\lfloor \frac{t}{T_s} \right\rfloor) \tag{17}$$

with $\lfloor . \rfloor$ the *floor* function and t a function of the evaluations, e. Note that the ZOH rate transition introduces side effects such as additional high frequency components. In general, different rate transitions can be considered (e.g., zero padding) and it is important for the modeler to understand the implications to make an informed decision.

The stiff spring/damper system that models the floor compression is activated when the ball reaches the floor level. From a Computer Science perspective, hybrid systems are often modeled as hybrid state machines (e.g., Alur et al. (1994)) and sequential logic of finite state machines captures the activation. From a first principles perspective, modeling the physics as simultaneous constraints is a common approach (e.g., Otter et al. (2000)) and simultaneous inequalities capture the activation. To compare these two approaches, a detailed study follows.

Simultaneous Inequalities When represented by inequalities, the activation is evaluated simultaneously with the differential equations. The inequalities then hold on the continuous time domain as

$$F_{floor}(t) = \begin{cases} -\left(R \cdot v(t) + \frac{x(t)}{C}\right) & \textit{if } x(t) < 0 \\ 0 & \textit{otherwise} \end{cases}. \tag{18}$$

As a cautionary note, in a more accurate model, in addition to position, the switching condition of the reaction force by the floor is better expressed in terms of velocity and force (e.g., Mosterman and Biswas (1996); Pfeiffer and Glocker (1996)).

Finite State Machine An alternative approach models the discontinuities as sequential logic in the form of a finite state machine (e.g., Remelhe (2001)). For this case, a finite state machine is a tuple $\langle \Xi, \xi_0, \Sigma, \Theta, \delta \rangle$ with

$$\Xi = \{\xi_{free}, \xi_{contact}\} \tag{19}$$

the set of two states, where the ball is either *free*, ξ_{free}, or in *contact* with the floor, $\xi_{contact}$. The initial state is ξ_{free}

$$\xi_0 = \xi_{free}. \tag{20}$$

To detect a change in state, each time the differential equations are evaluated, the finite state machine is as well. Thus the event set consists of one event

$$\Sigma = \{\sigma_{evaluation}\} \tag{21}$$

and the event is generated for each evaluation (i being the natural numbers)

$$e = i \rightarrow \sigma_{evaluation}. \tag{22}$$

Transitions then occur based on conditions that guard the transition to free, θ_{free}, and to contact, $\theta_{contact}$,

$$\Theta = \{\theta_{free}, \theta_{contact}\}. \tag{23}$$

These conditions connect the discrete-event model part to the continuous-time model part by inequalities to determine their truth value

$$\begin{aligned} \theta_{contact} &= x < 0 \\ \theta_{free} &= x \not< 0 \end{aligned} \tag{24}$$

where $\not<$ logically negates the result of the $<$ expression. A transition function $\delta : \Xi \times \Theta \times \Sigma \rightarrow \Xi$ with the current state, the truth values of the conditions, and the event that is generated, formalizes the transition behavior

$$\delta : \begin{cases} \xi_{free} & \wedge\ \sigma_{evaluation} \wedge \theta_{contact} \rightarrow \xi_{contact} \\ \xi_{free} & \wedge\ \sigma_{evaluation} \wedge \theta_{free} \rightarrow \xi_{free} \\ \xi_{contact} & \wedge\ \sigma_{evaluation} \wedge \theta_{contact} \rightarrow \xi_{contact} \\ \xi_{contact} & \wedge\ \sigma_{evaluation} \wedge \theta_{free} \rightarrow \xi_{free} \end{cases}. \tag{25}$$

Finally, each of the discrete states corresponds to different continuous-time behavior of the forcing term in the differential equations as

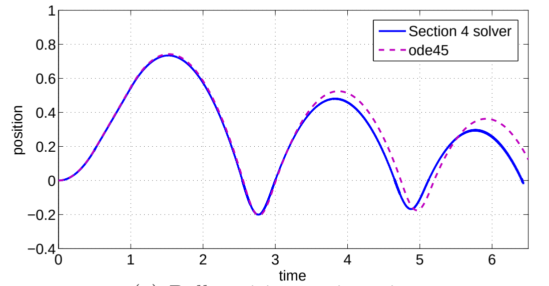
$$\begin{aligned} \xi_{contact} &: F_{floor}(t) = -R \cdot v(t) + \tfrac{x(t)}{C} \\ \xi_{free} &: F_{floor}(t) = 0 \end{aligned} \tag{26}$$

5.2 Simulation

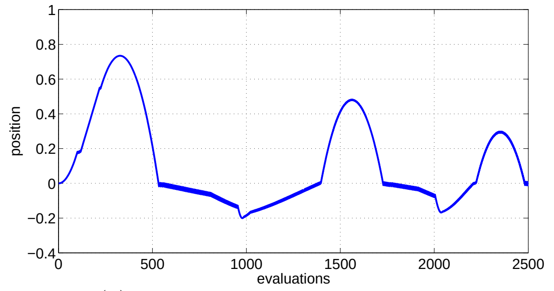
Simulation of the bouncing ball based on the specifications in Section 5.1 with the spring/damper modeled by simultaneous inequalities, is shown in Fig. 3 for model parameters $m = 7, R = 5, C = 2.5 \cdot 10^{-3}$ and solver parameters $h_{max} = 0.01$ and $tol = 7.5 \cdot 10^{-4}$. Figure 3(a) shows the position of the ball against time. After an initial upward motion, the ball returns to the floor and upon contact the floor is compressed till the velocity of the ball reverses and it starts moving up again. The solid line is the trajectory as computed by the solver developed in Section 4. The trajectory compares well with the dashed line, wich is the trajectory as computed by the Dormand-Prince integration method (`ode45`) in Simulink with the same relative tolerance and maximum step size. For longer simulation times the discrepancy increases as expected because the Dormand-Prince method is of higher order. The solver of Section 4 can now be studied in more detail.

Discretized Continuous Behavior Figure 3(b) shows the position of the ball as a function of evaluations. This illustrates that the number of evaluations per distance traveled increases significantly when the ball is in contact with the floor (position < 0). Figure 3(c) further clarifies this by showing time as a function of evaluations. The discretization in time becomes finer grained, because the variable-step solver reduces its step size to approximate the stiff behavior of the floor. Moreover, when reducing the step size, the solver moves time back, as Figure 3(d) shows in more detail in around 0.5 (s).

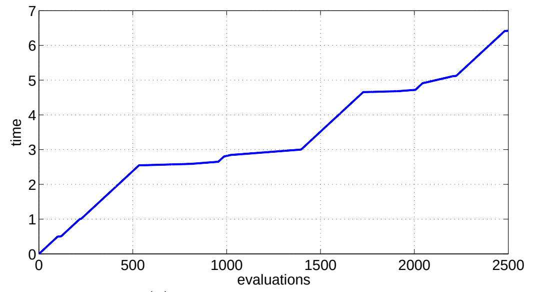
Figure 3(d) also reveals how the two-stage nature of the solver leads to time being held constant for two consecutive evaluations. After the pair of evaluations $e = 100$ (to complete the Euler approximation) and $e = 101$ (to complete the trapezoidal approximation), time is reversed as a consequence of the discontinuous change in control force at time 0.5 (s). When such a discontinuity occurs in an integration step, the Euler and the trapezoidal approximations result in significantly different values for the ball velocity. This is because the Euler approximation is based on the values at the beginning of the integration step only while the trapezoidal approximation is based on values at both the beginning and end of the integration step. Cellier (1979) observed this and employed a separate zero-crossing function to locate the discontinuity in time. During an integration step the discontinuity is not effected but an



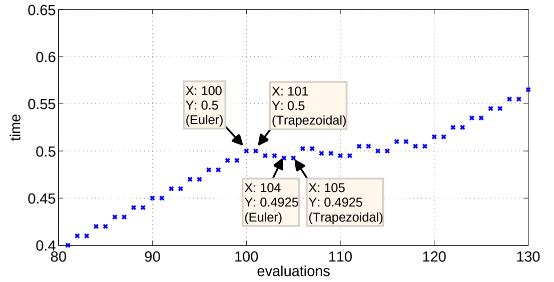
(a) Ball position against time



(b) Ball position against evaluations



(c) Progression of time



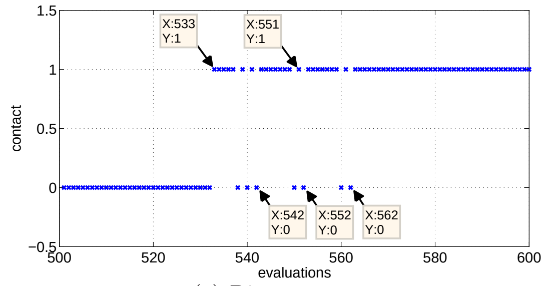
(d) Detailed view of time around 0.5 (s)

Fig. 3. Bouncing ball simulation.

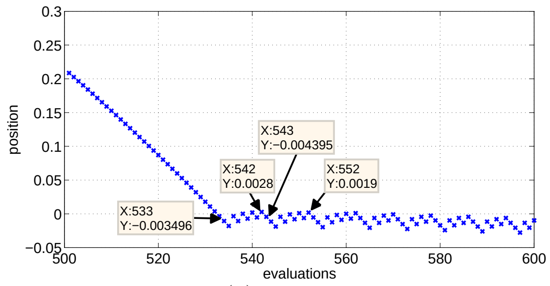
extrapolation is used for the numerical integration (the model is said to 'lie' to the solver). Note that such zero-crossing functionality significantly adds to the complexity of the solver (e.g., Park and Barton (1996)) thus making a mathematical analysis yet more complicated. Also note that the $\sigma_{evaluation}$ event then is only generated once a zero-crossing has been located within tolerance.

Contact Behavior in Detail With the stream-based functional representation available, unlike in previous work by Mosterman et al. (1998), all computations have become completely exposed and the different implementations of the contact behavior can be studied.

Figure 4 shows a simulation based on the simultaneous inequalities when the state changes from *free* to *contact*. Figure 4(a) shows how the system first attempts to switch to *contact* at $e = 533$ and fails. The underlying reason is clarified in the correlated position plot shown in Fig. 4(b).



(a) Discrete state



(b) Position

Fig. 4. Simultaneous inequality switching specification.

At $e = 533$, the trapezoidal stage first computes a position below $x = 0$. The reaction force of the floor is sufficiently low, though, for the velocity not to exceed the error bound as determined by (11), see Table 1. At $e = 534$ and $e = 535$ the Euler and trapezoidal stages both compute positions below $x = 0$ and the reaction force causes a difference between them that exceeds the tolerance ($tol = 7.5 \cdot 10^{-4}$), so the step size is reduced. Evaluation $e = 536$ and $e = 537$ reflect the computations at the new step size for the Euler and trapezoidal stages, respectively, and the corresponding error still exceeds the tolerance. The step size is reduced three more times, till for $e = 542$ and $e = 543$ the difference between the Euler and trapezoidal stage ($d = 4 \cdot 10^{-4}$) falls below the error tolerance and a new step of size h_{max} can be taken. At this point ($e = 542$) the Euler approximation of the position has retreated to a value above $x = 0$. A like scenario repeats around evaluation 550 and 560.

Table 1. Simultaneous inequalities switching

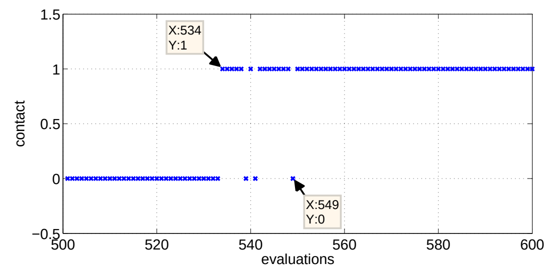
Eval	Time	Position	Velocity	F_{floor}	Error
532	2.5450	0.0037	-1.4381	0	
533	2.5450	-0.0035	-1.4381	8.5888	0
534	2.5550	-0.0107	-1.4398	11.4717	
535	2.5550	-0.0179	-1.4377	14.3430	0.0021
536	2.5500	-0.0035	-1.4348	8.5700	
537	2.5500	-0.0107	-1.4369	11.4555	0.0021
538	2.5475	0.0001	-1.4375	0	
539	2.5475	-0.0071	-1.4395	10.0333	0.0020
540	2.5462	0.0019	-1.4380	0	
541	2.5462	-0.0053	-1.4389	9.3125	0.0009
542	2.5456	0.0028	-1.4381	0	
543	2.5456	-0.0044	-1.4385	8.9508	0.0004

Figure 5 shows a simulation of the same scenario based on the finite state machine representation. In Fig. 5(a) the system is seen to attempt to switch to *contact* at

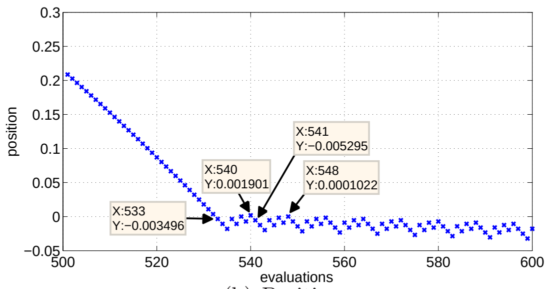
$e = 534$. Table 2 clarifies how the position at $e = 533$ is negative, which sets the $\theta_{contact}$ guard to *true* and enables the transition to $\xi_{contact}$ (abbreviated as ξ_{con}). Because the state change is not effected until the consecutive evaluation, *contact* is first achieved at $e = 534$. Another example of the lag between position computation and discrete state is the change out of the *contact* state at $e = 539$. This change is caused by the position computation that is slightly above $x = 0$ at $e = 538$. As a result, the reaction force of the floor is computed based on the previous position evaluation. If undesired, the discrete state may be kept unchanged instead till the point of moving into the *contact* state is located with sufficient accuracy (the model 'lies' about the discrete state).

Table 2. Finite state machine switching

Eval	Time	Position	Velocity	F_{floor}	Error	ξ_{con}
532	2.5450	0.0037	-1.4381	0		0
533	2.5450	-0.0035	-1.4381	0	0	0
534	2.5550	-0.0107	-1.4521	11.5331		1
535	2.5550	-0.0179	-1.4438	14.3980	0.0082	1
536	2.5500	-0.0035	-1.4348	8.5820		1
537	2.5500	-0.0107	-1.4369	11.4614	0.0021	1
538	2.5475	0.0001	-1.4375	7.1446		1
539	2.5475	-0.0071	-1.4382	0	0.0008	0
540	2.5462	0.0019	-1.4398	6.4386		1
541	2.5462	-0.0053	-1.4392	0	0.0006	0



(a) Discrete state



(b) Position

Fig. 5. Finite state machine switching specification.

The two different representations can now be compared directly. Table 3 shows the computations of the velocity contribution for each step of the Euler (even evaluations) and trapezoidal (odd evaluations) approximations in case of a model with the switching specified by simultaneous inequalities (labeled 'Ineq') or by a finite state machine (labeled 'State'). Each time the trapezoidal stage completes, the difference in contributions is computed to determine the error. In this scenario, the finite state machine specification converges to a value less than the tolerance ($tol = 7.5 \cdot 10^{-4}$) quicker than the simultaneous inequalities, thus resulting in the behavior in Fig. 4 and Fig. 5.

Table 3. Velocity contribution and error

Eval	Time	Contribution		Error	
		Ineq	State	Ineq	State
532	2.5450	-0.0140	-0.0140		
533	2.5450	-0.0140	-0.0140	0	0
534	2.5550	-0.0017	-0.0140		
535	2.5550	0.0003	-0.0058	0.0021	0.0082
536	2.5500	0.0032	0.0033		
537	2.5500	0.0012	0.0012	0.0021	0.0021
538	2.5475	0.0006	0.0006		
539	2.5475	-0.0015	-0.0002	0.0020	0.0008
540	2.5462	0.0000	-0.0018		
541	2.5462	-0.0009	-0.0012	0.0009	0.0006
542	2.5456	-0.0000			
543	2.5456	-0.0005		0.0004	

Note that as per (10), the effect of F_{floor} on the error evaluation is delayed. For example, for the case of simultaneous inequalities, the error to determine the step size that holds at $e = 536$ is computed at $e = 535$ as the difference in the previous Euler and trapezoidal contributions ($d = 0.0021$). Given the two-stage integration, the previous Euler contribution is computed based on F_{floor} at evaluation 533 (see Table 1) and displayed at 534 as -0.0017.

6. EVALUATION

This section briefly discusses specific previous work that is advanced and results in general. The related work is not exhaustive yet references throughout should have provided a good starting point for further investigation.

6.1 Advancing Previous Work

The presented approach supports continuous-time differential equation models that rely on pure functions on streams. In previous work on *synchronous languages* reported by Benveniste et al. (2003), it was shown how functions on streams can capture discrete-time and discrete-event behavior. Combined with the continuous-time support, a unifying computational structure results.

An important distinction over synchronous languages is that there is no implicit underlying discrete clock in the work presented here. Instead, a sequence of evaluations is defined which is related to the *tagged signal model* introduced by Lee and Sangiovanni-Vincentelli (1996). The total order on the model of time to describe physics, however, would have been too restrictive to capture the computational semantics of the variable-step solver in Section 4. Instead, time can increase and decrease with increasing evaluations. So, in this unifying computational framework, not only is time possibly constant, as was documented, for example, by Mosterman (2002, 2007)), it may recede as well.

Decreasing time is well established as 'roll back' in the *time warp* discrete-event simulation algorithm by Jefferson (1985). Though the algorithm has been utilized for rigid body simulation by Mirtich (2000), focus was on efficient simulation rather than a unifying semantics formulation.

6.2 Hybrid System Semantics Definition

In a general hybrid systems sense, the presented work holds value in three different dimensions. First, the long time behavior of numerical integration is relatively poorly understood. This lack of understanding has not been much of a practical impediment given that the parameters of continuous-time models are typically fit against measurements by using computational simulation. As a result, the solver characteristics become incorporated into the model and this renders computational *consistency* paramount, which necessitates an explicit computational semantics.

Second, as described by Jackson et al. (2009), the definition of a formalism such as a domain-specific one requires a precise specification of the semantics. This is important, for example, in order to be able to develop compilers and model transformations in general, but also to understand the expressiveness of a formalism. In addition, the semantics specification may provide a reference implementation that serves as an executable specification for more efficient execution engines. With a well-defined semantics, a model becomes truly defined by the semantics of the formalism, as opposed to the particulars of this underlying, often very sophisticated, execution engine. A precise and explicit semantics is especially important for hybrid systems with infinite sensitivity such as described by Nikoukhah (2007).

Finally, integrating different formalisms requires the study of their interaction semantics. The presented unifying framework introduces a common denominator that facilitates the systematic study of such interaction. Subtle complications become explicit as described by Denckla and Mosterman (2006). For example, the multi-rate character of a multi-stage numerical integration algorithm may present the need for a rate transition that can be easily overlooked otherwise.

7. CONCLUSIONS

Continuous-time behavior as represented by differential equations often does not have a closed form solution for the trajectories that the equations represent. While iterative application of numerical integration algorithms allows obtaining such trajectories, the behavior of the approximation error over repeated iterations is, in general, not well understood. To overcome this lack of a precise definition, a computational semantics of a variable-step solver has been presented.

In addition to better understand and capture the specifics of the solver behavior, a formalization as pure functions of streams fits the framework of other types of semantics such as discrete-time and discrete-event. It was illustrated how this allows a comprehensive study of interaction behavior in a unifying framework.

Future work intends to focus on identifying structure in the nonmonotonic time as exploited for defining the solver semantics. Also, the mapping of integers to a floating point representation is a subject of potential further study. The numerical effects can lead to dramatically different behavior, and a scheme to attempt to eliminate such sensitivity is important in combining continuous-time and discrete-time models.

REFERENCES

Adam, N., Altinakar, M., Thomas F. Corbet, J., Diaz, J., Kadtke, J., Krimgold, F., LeClaire, R., and Samsa, M. (2008). Workshop on future directions in critical infrastructure modeling and simulation—final report. Technical report, Department of Homeland Security.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1994). The algorithmic analysis of hybrid systems. In J. Bakkers, C. Huizing, W. de Roeres, and G. Rozenberg (eds.), *Proceedings of the 11th International Conference on Analysis and Optimization of Discrete Event Systems*, 331–351. Springer-Verlag. Lecture Notes in Control and Information Sciences 199.

Åström, K.J. and Wittenmark, B. (1984). *Computer Controlled Systems: Theory and Design.* Prentice-Hall, Englewood Cliffs, New Jersey.

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

Breedveld, P.C. (1984). *Physical Systems Theory in Terms of Bond Graphs.* PhD dissertation, University of Twente, Enschede, Netherlands.

Bujakiewicz, P. (1994). *Maximum weighted matching for high index differential algebraic equations.* PhD dissertation, TU Delft, Delft, Netherlands.

Caspi, P. and Pouzet, M. (1997). A Co-iterative Characterization of Synchronous Stream Functions. Technical Report 07, VERIMAG.

Cellier, F., Elmqvist, H., and Otter, M. (1996). Modelling from physical principles. In W. Levine (ed.), *The Control Handbook*, 99–107. CRC Press, Boca Raton, FL.

Cellier, F.E. (1979). *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools.* PhD dissertation, ETH, Zurich, Switzerland.

Denckla, B. and Mosterman, P.J. (2006). Block diagrams as a syntactic extension to haskell. In *Proceedings of the Workshop on Multi-Paradigm Modeling: Concepts and Tools.* Genoa, Italy.

Denckla, B. and Mosterman, P.J. (2008). Stream- and state-based semantics of hierarchy in block diagrams. In *Proceedings of the 17th IFAC World Congress.* Seoul, Korea.

Engstrom, E. and Krueger, J. (2000). A meta-modeler's job is never done: Building and evolving domain-specific tools with DOME. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, 83–88. Anchorage, Alaska.

Falk, G. and Ruppel, W. (1976). *Energie und Entropie: Eine Einführung in die Thermodynamik.* Springer-Verlag, Berlin, Heidelberg, New York.

Flatscher, R.G. (2002). Metamodeling in EIA/CDIF meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation*, 12(4).

Friedman, J. and Ghidella, J. (2006). Using model-based design for automotive systems engineering – requirements analysis of the power window example. In *Pro-*

ceedings of the SAE 2006 World Congress & Exhibition, CD–ROM: 2006–01–1217. Detroit, MI.

Guckenheimer, J. (2002). Numerical analysis of dynamical systems. In B. Fiedler (ed.), *Handbook of Dynamical Systems*, vol. 2, 345–390. Elsevier, Amsterdam, Netherlands.

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

Jackson, E., Thibodeaux, R., Porter, J., and Sztipanovits, J. (2009). Semantics of domain specific modeling languages. In G. Nicolescu and P.J. Mosterman (eds.), *Model-Based Design for Embedded Systems*. CRC Press, Boca Raton, FL.

Jefferson, D.R. (1985). Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3), 404–425.

Jones, H. (2005). Return on investment in Simulink® for electronic system design. Technical report, International Business Strategies.

Jones, S.P. (2003). *Haskell 98 Language and Libraries*. Cambridge University Press, Cambridge, UK.

Lee, E.A. and Sangiovanni-Vincentelli, A. (1996). 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, California.

Mens, T. and Gorp, P.V. (2006). A taxonomy of model transformation. In *Proceedings of the International Workshop on Graph and Model Transformation*, 125–142. Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam.

Mirtich, B. (2000). Timewarp rigid body simulation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 193–200. New York, NY.

Mosterman, P.J. (2002). HyBrSim—a modeling and simulation environment for hybrid bond graphs. *Journal of Systems and Control Engineering*, 216(1), 35–46.

Mosterman, P.J. (2007). On the normal component of centralized frictionless collision sequences. *ASME Journal of Applied Mechanics*, 74(5), 908–915.

Mosterman, P.J. and Biswas, G. (1996). Verification of dynamic physical system models. In *Proceedings of the ASME Congress '96*, 707–714. Atlanta, GA.

Mosterman, P.J., Bouldin, D., and Rucinski, A. (2008). A peer reviewed online computational modeling framework. In *Proceedings of the CDEN 2008 Conference*, CD–ROM. Halifax, Canada.

Mosterman, P.J., Otter, M., and Elmqvist, H. (1998). Modeling petri nets as local contraint equations for hybrid systems using modelica. In *SCS Summer Simulation Conference*, 314–319. Reno, Nevada.

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

Nicolescu, G. and Mosterman, P.J. (eds.) (2009). *Model-Based Design for Embedded Systems*. CRC Press, Boca Raton, FL.

Nielson, H.R. and Nielson, F. (1992). *Semantics with Applications: A Formal Introduction*. Wiley Professional Computing, Hoboken, NJ.

Nikoukhah, R. (2007). Hybrid dynamics in modelica: Should all events be considered synchronous. In *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT 2007)*, 37–48. Berlin, Germany.

Otter, M., Remelhe, M.A.P., Engell, S., and Mosterman, P.J. (2000). Hybrid models of physical systems and discrete controllers. *at - Automatisierungstechnik*.

Park, T. and Barton, P.I. (1996). State event location in differential-algebraic models. *ACM Transactions on Modeling and Computer Simulation*, 6(2), 137–165.

Paynter, H.M. (1961). *Analysis and Design of Engineering Systems*. The M.I.T. Press, Cambridge, Massachusetts.

Petzold, L.R. (1982). A description of DASSL: A differential/algebraic system solver. Technical Report SAND82-8637, Sandia National Laboratories, Livermore, CA.

Peyton-Jones, S. (1987). *The Implementation of Functional Programming Languages*. Prentice Hall, Englewood Cliffs, NJ.

Pfeiffer, F. and Glocker, C. (1996). *Multibody dynamics with unilateral contacts*. John Wiley & Sons, Inc., New York.

Potter, B. (2004). Use of the mathworks tool suite to develop do-178b certified code. In *ERAU / FAA Software Tools Forum*. Daytona Beach, Florida.

President's Information Technology Advisory Committee (2005). Computational science: Ensuring america's competitiveness. Technical report, Executive Office of the President of the United States.

Reekie, H.J. (1994). Modelling Asynchronous Streams in Haskell. Technical Report 94.3, Key Centre for Advanced Computing Sciences, University of Technology, Sydney.

Remelhe, M.A.P. (2001). Simulation and visualization support for user-defined formalisms using meta-modeling and hierarchical formalism transformation. In *Proceedings of the 2001 IEEE International Conference on Control Applications*, 750–755. Mexico City, Mexico.

Sanz-Serna, J.M. (1992). Symplectic integrators for hamiltonian problems: an overview. *Acta Numerica*, 1, 243–286.

SC-167, R. (1992). Do178b, software considerations in airborne systems and equipment certification.

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

Shampine, L.F., Reichelt, M.W., and Kierzenka, J.A. (1999). Solving index-1 daes in matlab and simulink. *SIAM Rev.*, 41(3), 538–552.

Simulink® (2008). *Using Simulink®*. The MathWorks™, Natick, MA.

Sullivan, M. (2005). 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.

Zhang, Y. and Xu, B. (2004). A survey of semantic description frameworks for programming languages. *ACM SIGPLAN Notices*, 39(3), 14–30.