A Combined Continuous-Time/Discrete-Event Computation Model for Heterogeneous Simulation Systems

Andreas Franck[1] and Volker Zerbe[2]

[1] Mission Level Design GmbH Ilmenau,
Ehrenbergstrasse 11,
98693 Ilmenau, Germany
[2] Technische Universität Ilmenau,
Department of Automatic Control and System Engineering,
P.O.Box 100565,
98684 Ilmenau, Germany

Abstract. Complex electronic systems contain components that have to be described by many different model types. An efficient design process requires to validate these models through all phases of development, [7]. It is therefore required to have multi-domain tools that can analyze these complex systems in an integrated way. MLDesigner a design tool of the latest generation is in the process of developing, [9].
In this paper, we describe a model of computation that combines continuous-time and discrete-event elements. We show that the developed formalism is well suited for frameworks like MLDesigner supporting heterogeneous modeling and simulation.

1 Introduction

1.1 Heterogeneous Modeling and Simulation

Heterogeneous modeling and simulation permits analysis and design of complex systems.

Instead of modeling a system as a whole, using a complex description language, in heterogeneous modeling every components is modeled by a suitable representation.

These submodels are combined and executed together to analyze the overall system behaviour.

An important research project in the field of heterogeneous modeling and design is the Ptolemy Project, at the University of California, Berkeley. Research focusses on modeling, simulation, and design of concurrent, real-time, embedded systems. In this project, two application programs have been developed, Ptolemy 0.x [1], now called Ptolemy classic, and Ptolemy II [8], which is the current research environment. Throughout this paper, we refer to the heterogeneous modeling and simulation concepts of the Ptolemy project and furthermore MLDesigner.

Essential to the concept of heterogeneous simulation is the notion of a *model of computation (MoC).* A model of computation defines the semantics of the models formulated in this MoC. Many models of computations have been developed and are in use, although in limited fields of application. Examples of computational models are Finite State Machines, Petri Nets or Dataflow graphs.

A model of computation defines the following characteristics of a model:

- The way how behaviour of the model is described.
- How these model descriptions are executed.
- For the case that models are described by sets of components, the model of computation defines how model components interact.

In Ptolemy, a model of computation is encapsulated in a *domain.* A domain is a set of C++ classes in which all aspects of simulation specific to the computational model are implemented.

Heterogeneous simulation frameworks often allow arbitrary couplings of domains. This requires thall all domains share a common interface which permits interaction between heterogeneous model parts. This interface has to define:

- How information is exchanged between different components.
- How to coordinate the execution of interacting components.

In Ptolemy classic, this interface is called *EventHorizon* [2].

1.2 Continuous-Time Computation Models

Computation Model The characteristic property of continuous-time formalisms is that the trajectories of input-, output- and state-values are continuous in time.

Mathematically, this class of systems is modeled by systems of ordinary differential equations *(ODE).* The change of a state variable is described by its time derivative $\dot{x}(t)$.

$$\dot{x}(t) = f(x, u, t) \tag{1}$$

$$y(t) = g(x, u, t) \tag{2}$$

A system model is described by its differential equation (1) and an *output equation* (2), where $u(t)$ is the input, $x(t)$ is the state and $y(t)$ is the systems output.

Signal Form. At every instant of the simulation interval, inputs, outputs and states have a distinct value. Therefore, signals in continuous-time models are continuous functions. The resulting signal form is shown in Figure 1.

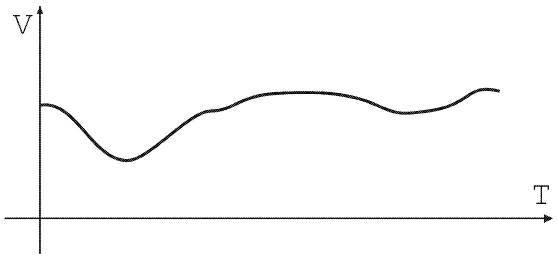


Fig. 1. Signal Form of a Continuous-Time Wave

Modeling. Besides mathematical representation like ODE and simulation languages as CSSL *(Continuous System Simulation Language)*, graphical oriented modeling of continuous-time systems is becoming predominant. In applications like SIMULINK from Mathworks, models are created and manipulated as interconnected blocks, defining the dynamics of the system. Besides the easy-to-use interface, the visual representation often reflects the structure of the modeled system.

Furthermore, most visual interfaces are hierarchical, permitting the combination of submodels into modules and using these modules as blocks. This allows hierarchical structuring of complex models and reusability of existing components. These features are essential to modern system development.

Simulation. Simulation of continuous-time models uses numerical methods for solving systems of ordinary equations, more precisely initial value problems. These algorithms are often called numerical integration methods.

Starting with an initial value x_0, the trajectories of the state and output values is approximated at a finite number of time points in the integration interval.

Numerical methods for solving ODE systems is an area of extensive research. A large number of different algorithms is available, with different accuracy, computational effort and suitability for distinct classes of problems. Therefore, continuous simulation applications must include different integration algorithms and offer capabilities to configure these algorithms according to the system to be simulated.

Limitations. These numerical methods generally require the differential equation $f(x(t), u(t), t)$ and input trajectory $u(t)$ to be smooth. More precisely, these functions must be, dependending on the used ODE solver, sufficiently differentiable.

For real systems, these requirements are often not met. One cause is discontinuity in the state transition function. Systems with friction or hysteresis effects show such behaviour.

Often, input functions change their values discontinuously. Escpecially in systems where continuous components interact with digital devices.

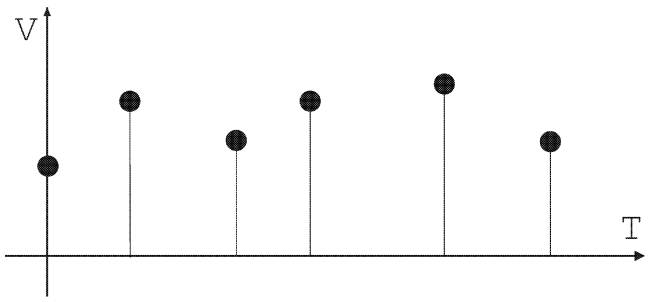


Fig. 2. Signals in Discrete-Event Simulations

Most numerical algorithms fail or decrease or have significantly reduced accuracy when stepping over discontinuity points. Therefore, practical simulation software must offer facilities to manage discontinuity points and handle them appropriately. This feature is usually called *breakpoint handling.*

1.3 The Discrete-Event-Formalism

In Discrete-Event (DE) models, evolution of the system state is modeled as a finite set of discrete state changes. These changes can occur at arbitrary distinct points of the underlying, usually real-valued, time base and are called *events.* A formal definition of this computational model can be found in [3], where it is introduced as DEVS *(Discrete Event Specified System)* formalism.

Modeling. In modern design-tools, Discrete-Event models are usually designed at the coupled model level and represented a graphical form. Examples of such tools are BONeS, a commercial network simulator from Cadence Design Systems, or the DE domain in MLDesigner.

In such coupled models, informations between different blocks are transmitted in form of messages associated with a usually real-valued timestamp. These messages are also called discrete events. Therefore, signals in Discrete-Event-models are sequences of events, as depicted in Figure 2.

Simulation. The heart of a Discrete-Event-simulator, often called *scheduler*, is a priority queue structure usually named *event queue* or *event calendar.* Model components schedule events to be processed by inserting them into this queue. The scheduler executes these events sequentially, according to the order of the time stamps.

Discrete-Event and Heterogeneous Simulation. Among different models of computation, the Discrete-Event-formalism has a special role. It has been stated that DE is suited to represent the input-output-behaviour of models formulated in other models of computation. Zeigler [3]:

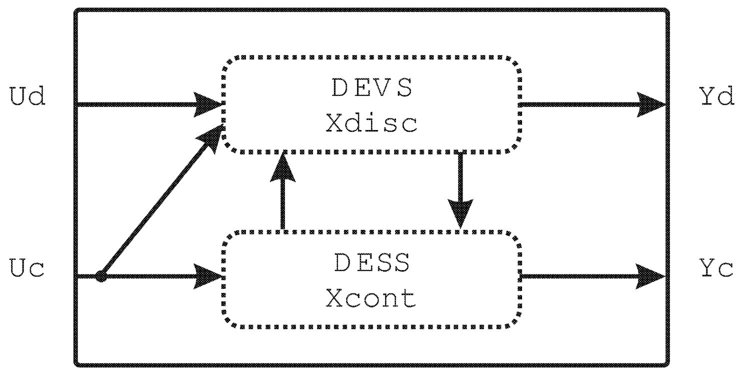


Fig. 3. Basic DEV&DESS-Model (according to [3])

> "... DEVS is capable of expressing models formulated in the other major formalisms ..., so that these can be integrated into an all-DEVS simulation environment supporting multiformalism modeling."

Therefore, the discrete-event-formalism is well suited to define the interface for communication between model components in a heterogeneous simulation environment.

2 Combined Continuous / Discrete-Event Models of Computation

In section (1.2), we described limitations of pure continuous-time formalisms. For real-world applications, this model of computation has to be extended.

Combined continuous/Discrete-Event models of computation [4] form a well-defined and especially expressive extension of pure continuous-time formalisms.

In [5], this formalism is introduced as DEV&DESS *(discrete event and differential equation defined system specification)* and formally defined. Here, we will give only a short, non-formal description.

2.1 The Computational Model

As depicted in Figure (3), a combined model consists of a continuous and a Discrete-Event component.

Both model parts may contain inputs, outputs and states of their respective types. Characteristic of the combined model of computation is the way these components influence each other. These interactions are also called events and can occur as:

Time events, related to events in pure Discrete-Event models. They are initiated in the discrete part, but possibly change the state in the continuous part, too.

External input Events, input messages that occur at the input ports of the Discrete-Event-partition. They are similiar to time events and are handled in the same way.

State events, initiated by the continuous model part. A state event is triggered when a condition that depends on continuous state or input values, is satisfied. In general, this condition can be expressed by a *state event equation*:

$$C(u(t), x(t), t) = 0 \tag{3}$$

Only after an integration step algorithm can it be determined if a state event occured within this time period. This is because the event condition depends on values that change continuously.

2.2 The CTDE-Domain

To investigate this computational model presented here, we developed a new domain called CTDE-domain *(Continuous-Time/Discrete-Event)*. We developed representation forms for combined models, and simulation algorithms.

2.3 Model Structure

To represent combined continuous/discrete-event models, we chose a graphical form. Models are represented as graphs of interconnected blocks with input and output ports. We define the base element of the modeling language, the *general hybrid block*. This block is closely related to the atomic DEV&DESS model introduced in section (2.1). This element may contain both continuous and discrete-event elements, and have an arbitrary combination of discrete and continuous input and output ports.

As an example of a general hybrid block, the general integrator block is shown in Figure (4). Beside the continuous input and state output of a pure continuous integrator, it contains a discrete input for resetting the state value and a saturation output, which signals reaching the upper or lower limit as a discrete event.

Pure continuous or discrete blocks are special cases of the general hybrid and can be modeled using this atomic element. This allows a uniform representation of all blocks in a combined model.

Specific to the CTDE-domain is the coexistence of two distinct signal forms. Couplings between continuous ports are continuous wavefroms and have value-semantics. Over connections between discrete ports messages are transmitted as discrete events.

Direct connections between discrete and continuous ports are not allowed in CTDE, but must be modeled by the distinct insertion of conversion Blocks. This allows static checks of the model structure before simulation startup and helps in avoiding modeling errors in the design.

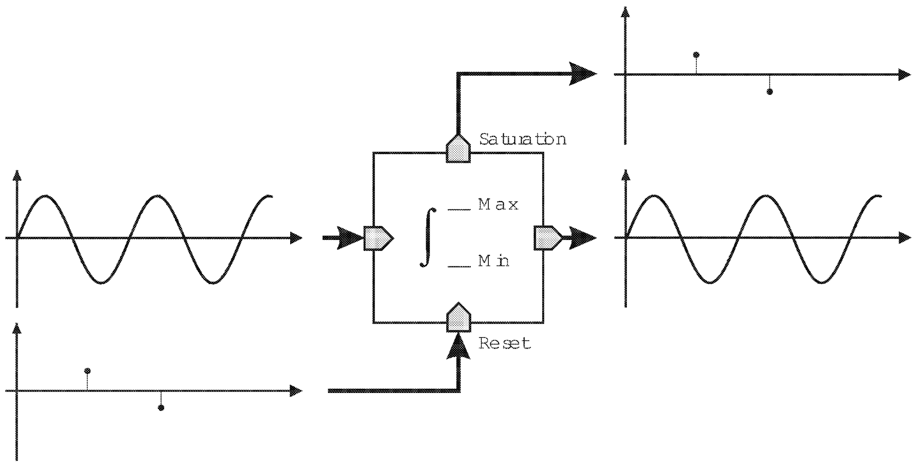


Fig. 4. Hybrid Block Example: General Integrator

2.4 Simulation Algorithm

Much like the model structure, the simulator is divided into a Discrete-Event and a Continuous-Time component. These parts interact as follows:

At a given time in the simulation interval, the Discrete-Event scheduler processes all events with the current timestamp. After that, the simulator changes to the continuous part. In this mode, the trajectory of the continuous states is approximated by a numerical ODE solver. At the time of the next scheduled discrete event, execution changes back to the Discrete-Event scheduler.

Detection and processing of state events requires special treatment. In the graphical model representation used in the CTDE domain, state event conditions are modeled as Blocks, called *state event generators*. Since state event conditions depend on continuous variables, they must be handled in the continuous part of the simulator. After each step of the ODE solver, the state event generator blocks are queried if a state event occured during the current step. In this case, this integration step has to be rejected. Integration will be repeated with adjusted step sizes until the event condition is met with sufficient accuracy. For this case, the simulator immediately changes into the Discrete-Event mode and executes the action triggered by the state event as an ordinary discrete event.

Simulation algorithms for combined continuous/discrete-event simulators are treated in more detail in [4] and [3].

2.5 Example: *Bouncing Ball-Model*

Here, we show an example commonly used for demonstrating the handling of state-events. A ball bounces repeatedly on a surface. The impact is modeled by a state-event-generator, the block `ZeroCrossingDetector`, which ensures that the time of the hit is determined accurately.

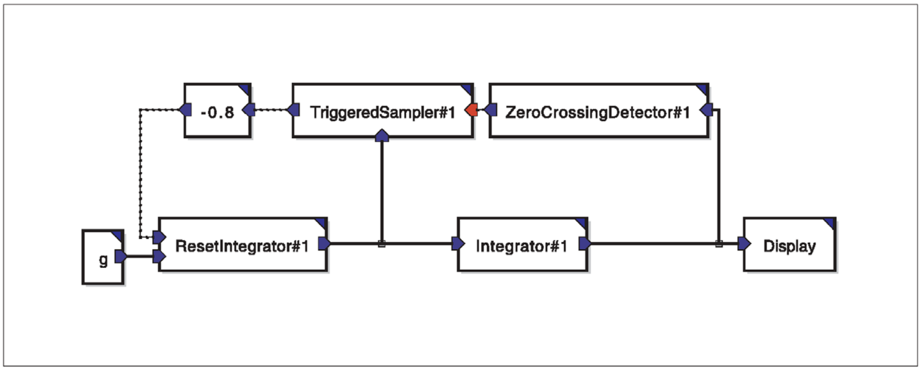


Fig. 5. Model of the bouncing-ball system

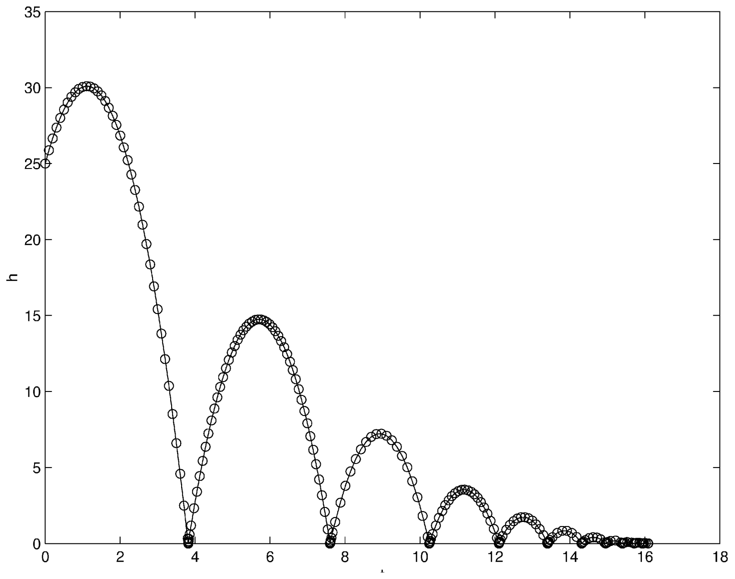


Fig. 6. Bouncing-Ball-Demo: Position of the ball

This model formulated in the CTDE domain shows the combination of continuous and discrete dynamics clearly. Whereas the movement of the ball is modeled with continuous elements and signals (solid lines), the signaling of the impact, calculation of the reflected velocity and inverting the movement is modeled using discrete logic (dotted lines).

3 Combined Formalisms in Heterogeneous Simulations

At the first glance, a combined model of computation contradicts the concept of heterogeneous modeling and simulation. Instead of modeling each subsystem in an appropriate formalism and to combine these models for simulation, a larger and more complex formalism was created.

In our opinion, the combined continuous/discrete models of computation shows several advantages that make them especially suited for integration into heterogeneous simulation environments.

Therefore, we compare the CTDE domain with an implementation of a pure continuous-time formalism. As an example, we chose the CT-domain *(Continuous Time)* from Ptolemy II [6]. This domain implements a model of computation based on ODE's represented in a graphical form. Additionally, CT offers facilities to interact with Discrete-Event-models. This kind of model interaction is called *Mixed-Signal-Simulation.*

3.1 Signals

Because of their signal form, continuous signals cannot be represented by a finite amount of data. Therefore, continuous signals are not suitable for communication between model components. In section 1.3 it was stated that the Discrete-Event formalism, and the event sequence as the associated signal form, are suitable semantics for interaction between components in a heterogeneous simulation framework. Therefore, a continuous-time domain mixed-signal-simulation, or more generally, heterogeneous simulation, must implement facilities to handle discrete-event signals.

In Ptolemy II, two additional classes of blocks, *EventInterpreters* and *EventGenerators* are introduced to perform the conversion between continuous and discrete waveforms. These elements have to be inserted at the boundaries of continuous models.

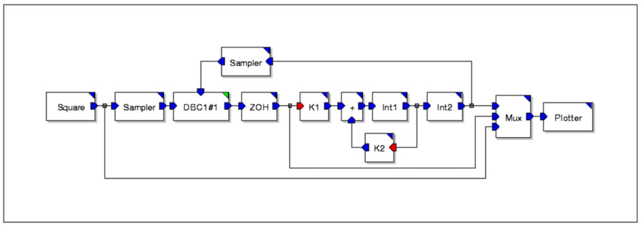
In CTDE, discrete-event signals are an integral part of the domain. Neither the comptutional model nor the hierarchy of the modeling elements has to be extended to realize interaction with heterogeneous components.

3.2 Modeling

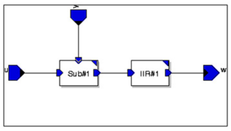
The choice of the general hybrid block as the base element of modeling increases the expressiveness of the modeling language. There are elements that cannot, or not effectively, be expressed as combinations of discrete, continuous and signal-conversion blocks. The `ResetIntegrator` used in the Bouncing-Ball demo is an example for such an element.

Another difference in the modeling methodology is the structure of a mixed-signal model. In a pure continuous-time computation model such as the Ptolemy II CT-Domain, every model part with discrete dynamics has to be encapsulated in a separate submodel. In CTDE of MLDesigner, the model structure can be chosen according to the logical partitioning of the system. Again, the Bouncing-Ball demo serves as an example where integration of continuous and discrete dynamics into a single model makes sense and is more concise.

In modeling environments like Ptolemy, composition of heterogeneous components is done by means of hierarchical embedding. A submodel formulated in



(a) Top-Level-Model with continuous plant



(b) Controller model as dataflow graph

Fig. 7. Example (3.4): Continuous-time plant with Discrete-Time Controller

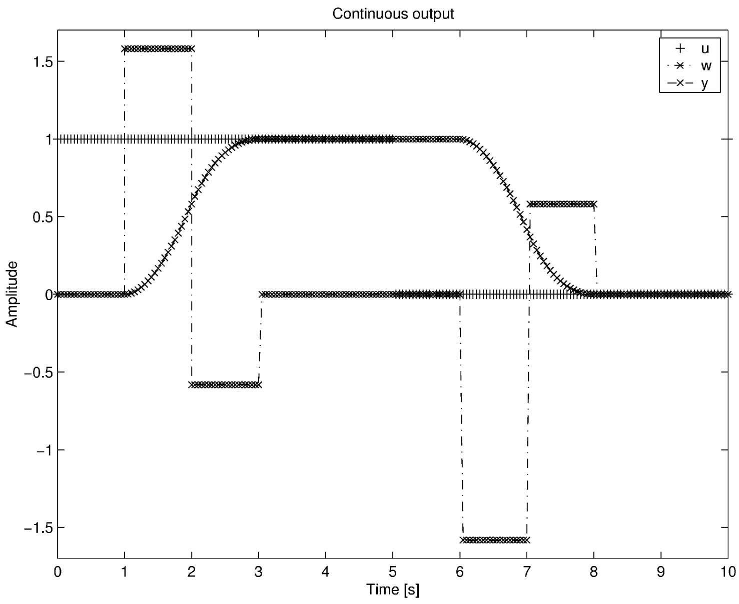


Fig. 8. Example (3.4): Continuous values

a different formalism is inserted as a block into the embedding model. Since heterogeneous components communicate with the CTDE domain through Discrete-Event semantics, these submodels are handled as pure discrete blocks. Since discrete elements form a special case of the general hybrid block, no special treatment is necessary. In pure Continuous-Time models of computation, these submodels form a separate class of modeling elements.

3.3 Simulation

As shown above, in the CT domain in Ptolemy II, the model representation was extended by three classes of blocks (EventInterpreters, EventGenerators and bloccks representing submodels) to permit mixed-signal simulation. Since these element classes need to be handled by the simulator separately, this makes the simulation algorithms more complicate and cumbersome.

In contrast, the CTDE has need to extend the model of computation to allow interaction with heterogeneous model components. This results in a better structured simulator design and (presumably) better efficiency.

3.4 Example: Discrete Controller

As an example of a Mixed-Signal-model formulated in the CTDE domain, we present closed-loop control consisting of a continuous plant and a discrete-time controller (Figure 7). The discrete controller, more precisely a deadbeat-algorithm, is modeled as a dataflow graph in the SDF domain (Figure 7(b)).

The coupling between the model components is modeled by dedicated elements for signal conversion. These elements enable the designer to accurately model the behaviour of systems combining digital and continuous components.

The results of a simulation run are depicted in Figure (8).

4 Conclusion

In this article, we present a model of computation that combines dicrete-event and continuous dynamics in a single model description.

This formalism forms a distinct model of computation and is especially suited for integration in heterogeneous modeling and simulation environments, implemented in MLDesigner. This extended computational model permits to simulate complex continuous and mixed-signal models accurately.

The use of well-defined and better structured simulation algorithms simplifies simulation software design.

References

1. Buck, J.; Ha, S.; Lee, E.; Messerschmitt, D.; Ptolemy: A framework for simulating and prototyping heterogeneous systems. Int. Journal of Computer Simulation, special issue on Simulation Software Development 4/1994, p. 155–182
2. Chang, W. T.; Ha, S.; Lee, E. A.; Heterogeneous simulation - mixing discrete event models with dataflow. Journal of VLSI Signal Processing 15/1997, p. 127–144
3. Zeigler, B. P.; Praehofer, H.; Kim, T. G.; Theory of Modeling and Simulation. second. ed. Academic Press, 2000
4. Cellier, F. E.; Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools. PhD thesis, ETH Zurich, 1979
5. Praehofer, H.; Auring, F.; Reisinger, G.; An environment for DEVS-based multiformalism simulation in common Lisp/CLOS. Discrete Event Dynamic Systems: Theory and Applications, 3/1993, p. 119–149
6. Liu, J. Continuous time and mixed-signal simulation in ptolemy ii. Ms report, University of California, Berkeley, 12/1998

7. Zerbe, V.; Mission Level Design of Control Systems. In: Proceedings of SCI/ISAS 5th International Conference on Information Systems, 1999, p. 237–243
8. II., J. D.; Goel, M.; Hylands, C.; Kienhus, B.; Lee, E. A.; Ptolemy II: Heterogeneous Concurrent Modelling and Design in Java. 1.0 ed. Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2001
9. MLDesigner: http://www.mldesigner.com