Embedding DEV&DESS in DEVS

Bernard P. Zeigler
Arizona Center of Integrative Modeling and Simulation (ACIMS)
ECE Department, University of Arizona
Tucson, AZ 85721
zeigler @ece.arizona.edu

Abstract

DEV&DESS was defined to represent combined continuous and discrete model event simulation models and was shown to have the properties expected from a universal representation of such models in "Theory of Modeling and Simulation." Also in that theory, it was shown that DEVS can provide accurate simulations of DESS models. In this paper, we offer a proof that DEVS on its own is capable of accurately implementing the combination of DEVS and DESS formalisms as formulated in DEV&DESS, thus filling in the missing pieces to complete the picture. We elucidate the characteristic primitives needed to express hybrid or combined continuous/discrete event models and provide a rigorous proof of the ability to express these within a purely discrete event computational framework. We discuss the implications of these results for multi-formalism modeling and the implication of the results for a related, but alternative, formalization of hybrid systems called Heterogeneous Flow System Specification.

1 Introduction

Although DEVS (Discrete Event System Specification) has been employed in a number of studies in which both discrete and continuous elements appear [1,2,3, 4], it has not received a formal proof of its capability to express such combined or hybrid models as they are often called [5,6,7] . In this paper, we offer such a proof by supplying some of the missing pieces that complete the picture presented in Theory of Modeling and Simulation [8]. Praehofer [9] defined the combined formalism, DEV&DESS to represent combined continuous discrete models that are simulatable in simulation languages such as SLAM and ARENA [10]. As defined, DEV&DESS describes a subclass of dynamic systems that includes the subclasses specified by DEVS and DESS (Differential Equation System Specification), respectively. As such it provides a rigorous framework for considering properties of hybrid models but it does not provide a computational framework for simulating such models. Barros [11,12] has also provided a DEVS-based extension to include continuous system modeling with implementation on a SmallTalk platform [13], to which we shall return later. On the other hand, the DEVS formalism has been implemented in numerous platforms [14-19] thus affording an advantageous computational basis for simulation of combined models. Therefore, an official proof of the embedding of DEV&DESS into DEVS would provide a formal justification for the

use of the latter for combined or hybrid models. Furthermore, we show how multiformalism modeling [20] – the integrated use of many formalisms in the same framework – can be explicitly supported in such a framework.

The possibilities for expression of a formalism relative to DEVS are illustrated in Figure 1.

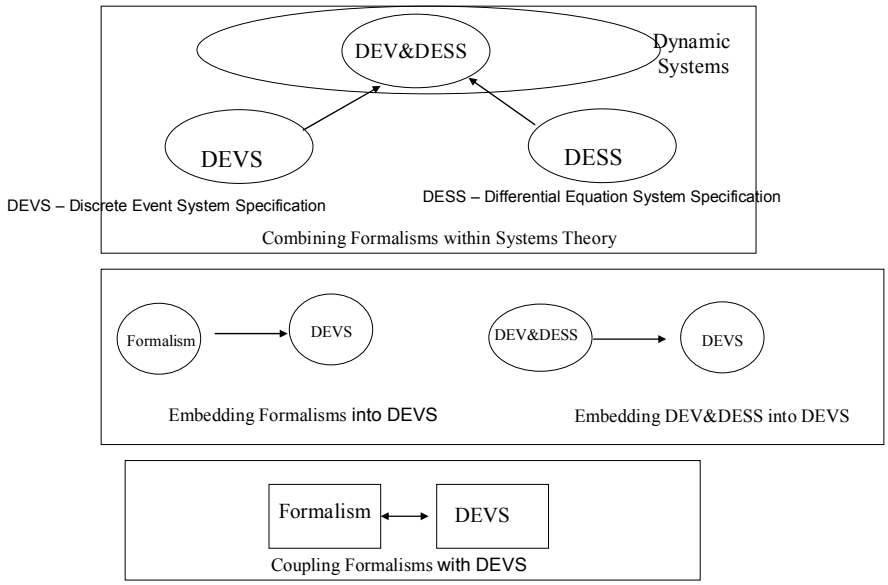

Figure 1. DEVS-based Multi-formalism Embedding and Compositional Approaches

Three approaches are depicted:

1. *Extending DEVS into a larger subclass of dynamic systems*: As mentioned earlier, the extension to DEV&DESS was performed by Praehofer [9] in his doctoral dissertation. He proved closure under coupling, thereby establishing that the structures and couplings included in the basic definitions are sufficiently expressive to represent arbitrary hierarchical compositions of discrete and continuous components. DEV&DESS is therefore a good candidate to consider as basis for hybrid and multi-formalism modeling theory investigations.

2. *Embedding a formalism into DEVS:* TMS2000 [8] also proved that DEVS is universal for discrete event systems, i.e., any system within the sub-class of dynamic systems having discrete event input/output behavior (when reduced to its canonical form) is isomorphic to a DEVS. This invites the possibility of exploiting the properties of special formalisms for discrete event systems by

expression them as subclasses of DEVS. Studies of this nature has been done for Petri Nets [21, 22], state charts [23] and timed automata [24].

3. *Interoperating a non-dynamic systems formalism with DEVS:* Sarjoughnian et al. [25] have shown the benefit of this approach, where linear and non-linear programs are coupled with discrete event simulation to optimize supply chains. The approach employs an interface specification to mediate between the two forms of knowledge (called a knowledge interchange broker) that provides a generic means of coupling a variety of simulation models with optimization strategy implementations.

Within this framework of discussion, we focus on embedding formalisms into DEVS. In particular, we show that the DEV&DESS formalism, the extension of DEVS with DESS, can be embedded within DEVS itself – within an accuracy specification that relates to the quantization-based representation of DESS within DEVS. As suggested above, the proof provides a justification for employing DEVS-based simulation environments as tools for hybrid and multi-formalism modeling – the latter taking into account the capabilities of DEVS to embed more restricted formalisms and to be interoperated with non-dynamic systems formalisms.

2 Characteristic Behaviors of the DEV&DESS Formalism

As background for the embedding of DEV&DESS into DEVS we first briefly review the DEV&DESS formalism presented in [8] and then go on to elucidate the characteristic behaviors need to achieve full DEV&DESS expressive capability.

2.1 Review of DEV&DESS

The DEV&DESS formalism [8] envisions a composition of discrete event and continuous parts as illustrated in Figure 2.

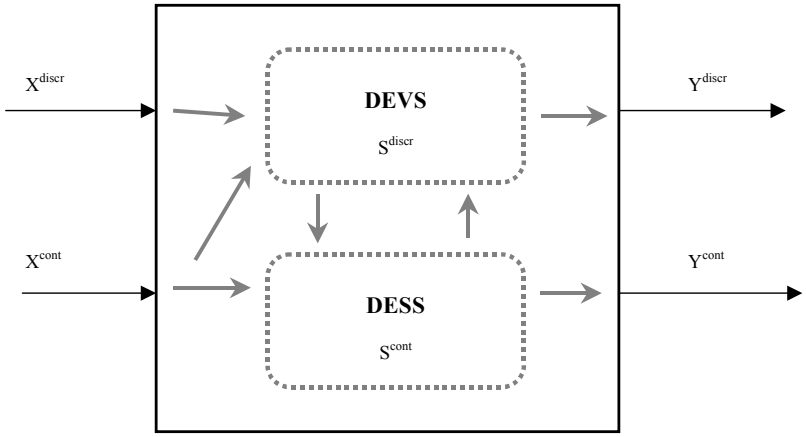

Figure 2. DEVS and DESS combined model

The DEV&DESS formalism prescribes the components and their interfaces. It is presented as follows:

$$\text{DEV\&DESS} = < X^{discr}, X^{cont}, Y^{discr}, Y^{cont}, S^{discr}, S^{cont}, \delta_{ext}, C_{\text{int}}, \delta_{\text{int}}, \lambda^{discr}, f, \lambda^{cont} >$$

X^{discr}, Y^{discr} are sets of discrete event inputs and outputs, resp.

X^{cont}, Y^{cont} are sets of continuous inputs and outputs, resp.

S^{discr}, S^{cont} are sets of discrete and continuous states, resp.

$\delta_{ext} : Q \times X^{discr} \times X^{cont} \to S$ is the external transition function

where $S = S^{discr} \times S^{cont}$ is the sequential state set, and

where $Q = \{(s,e) \mid s \in S, e \in \square_0^+\}$ is the total state set

$\delta_{\text{int}} : Q \times X^{cont} \to S$ is the discrete event internal transition function

$\lambda^{discr} : Q \times X^{cont} \to Y^{discr}$ is the discrete event output function

$\lambda^{cont} : Q \times X^{cont} \to Y^{cont}$ is the continuous output function

$f : S \times X^{cont} \to S^{cont}$ is the derivative function

$C : Q \times X^{cont} \to Bool$ is the event detection condition predicate

The semantics of the DEV&DESS formalism are given in terms of the subclass of dynamic systems that it defines [8]. A *state event* is considered to be the occurrence of a change in the value of the event condition predicate from false to true. The concept of state event derives from Pritsker's [26,27] original formulation of combined continuous and discrete simulation in which it was defined as a threshold crossing of the continuous state. In the DEV&DESS formulation, the concept of state event is generalized so that it can occur due to a change in any of its arguments, of which the continuous state is one component.

We note that as just formulated, the DEV&DESS formalism does not include a time advance function. Instead, triggering of the internal transition function is performed by the output of the event detection condition. We will show later that the time advance trigger can be captured within the formalism.

2.2 Capabilities Exhibited in the DEV&DESS Formalism

DEV&DESS defines certain behavior primitives that are manifest in the formalism that are not found in either of its components separately. As illustrated in Figure 3, these are: a) state event detection, b) derivative function change, c) integrator state reset, and d) event detector threshold change.

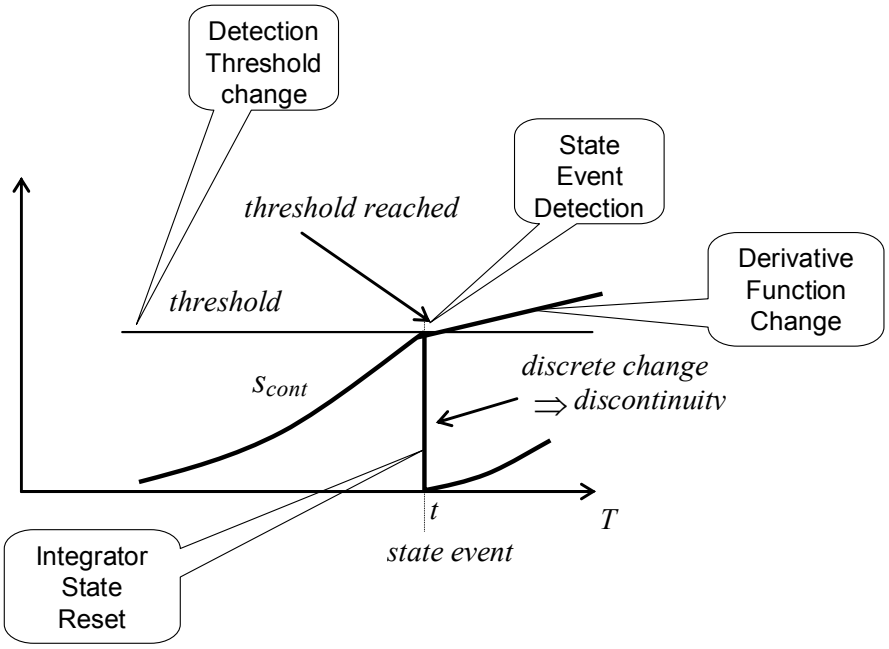

Figure 3.Behaviors in DEV&DESS

Let's see how the formalism makes use of these basic primitives to obtain behaviors characteristic of hybrid systems.

Behavior 1. *Discrete Events, both state and external, can control the derivative function, thus causing an instantaneous change in the derivative of the continuous state trajectory.*

By expressing the sequential state set as a cross product of the discrete and continuous state sets, the derivative function takes the form:

$$f : S^{discr} \times S^{cont} \times X^{co\,nt} \rightarrow S^{cont}$$

Since the discrete state changes only through discrete events, we can consider f to be a family of functions indexed by S^{discr}, with the interpretation that the differential equation

$$\frac{ds^{cont}}{dt} = f_{s^{discr}}(s^{cont}, x^{co\,nt})$$

holds for the interval during which s^{discr} is the state of the discrete event part as depicted in Figure 4.

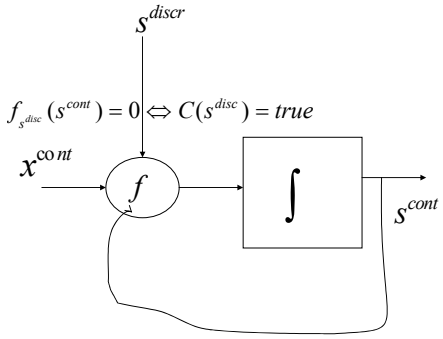

Figure 4. Illustration of discrete event control of derivative function

External events, through the invocation of the external transition function can change the discrete state and hence control the derivative function. Similarly, a state event, through the invocation of the internal transition function, can exhibit the same effect. An example given in TMS2000 concerns the filling of a container until full. This requires setting the derivative of the filling function to zero to shut off a flow, either externally, or when a level is exceeded (a state event).

Behavior 2. *Discrete Events, both state and external, can control the event detection condition, thus instantaneously changing the thresholds governing event detection.*

In the same manner as done for the derivative function, we can consider the event detection condition to be indexed by the discrete state set:

$$C_{s^{discr}} : S^{cont} \times X^{cont} \rightarrow Bool$$

where a particular predicate $C_{s^{discr}}$ holds during the interval in which the discrete state s^{discr} prevails (Figure 5).

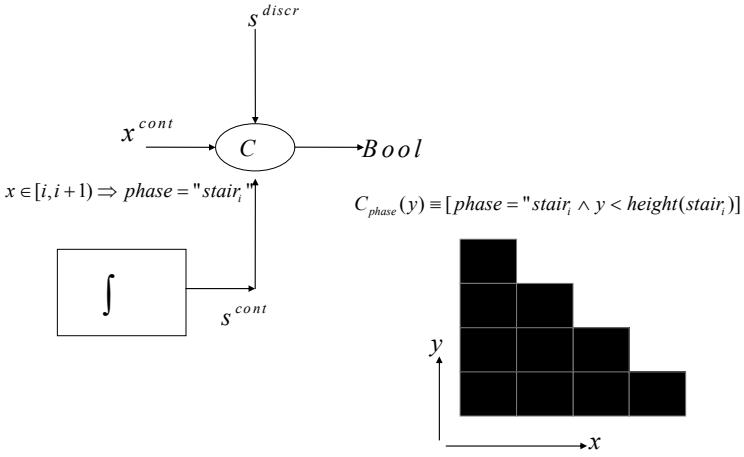

Figure 4. Illustration of discrete event control of event detection function

As an example, in a model first developed by Kofman [19], a ball bounces down a staircase. In our reformulation, the stair currently underneath the ball can be tracked by a phase variable in the discrete event part. The event detection condition controlled by this phase is made up of two parts corresponding to the vertical and horizontal coordinates of the ball. In the vertical component, the detection condition monitors the ball's vertical position for crossing the level of the current stair. In the horizontal component, the detection condition monitors the ball's horizontal position for the transition to the next lower stair. Thus the horizontal position of the ball determines the stair phase, e.g. $x \in [i, i+1) \Rightarrow phase = "stair_i"$. This in turn determines the height of the stair currently below the ball and thereby the level to be crossed in order to change the derivative function to represent the bouncing region, e.g. $C_{phase}(y) \equiv [phase = "stair_i \wedge y < height(stair_i)]$

Similarly, the continuous input or and/an external event can change the event detection function. Furthermore, in the latter case, the elapsed time can be taken into account. When an external event arrives after an elapsed time e since the last event, it can trigger

the external transition function which can change the discrete state based on the current state and the elapsed time e.

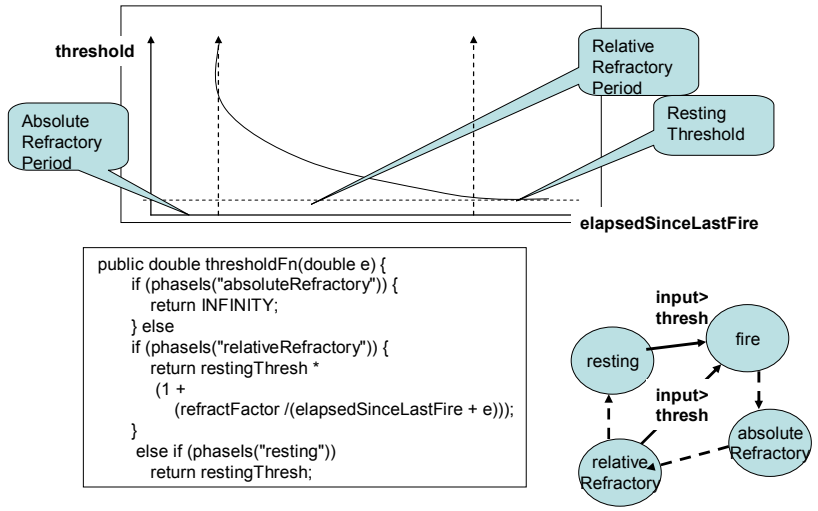

Figure 5. Firing threshold of a leaky integrator neuron

An example of dependence on external events is given by a model of the firing threshold of a leaky integrator neuron [28] as modeled in DEV&DESS. As depicted in Figure 5, the model has a discrete event phase variable such that the firing threshold depends both on the phase as well as the elapsed time since it was last fired. In the "resting" phase the threshold equals the resting threshold value. When the input exceeds the threshold, the phase changes to "absoluteRefractory" in which the neuron cannot fire (the threshold effectively takes on an infinite value). The passage from "absoluteRefractory" to "relativeRefractory" is an event that is triggered by the passage of a fixed refractory time. In the "relativeRefractory" phase, the threshold is given as a function of time since last fired, which itself is updated through the elapsed time, e.

Behavior 3. *Discrete Events, both state and external, can instantaneously change the continuous state (causing discontinuities from the point of view of differential equation systems)*

This is true because the continuous state is one of the components in the range of the external and internal transition functions. For example, we have

$$\delta_{ext} : Q \times X^{discr} \times X^{cont} \rightarrow S^{discr} \times S^{cont}$$

Since the occurrence of an external event triggers this function, the external event can directly change the continuous state, which is the composite state of the integrators underlying the differential equation of the continuous part.

The barrel filling model provides an example of a state event triggering an integrator reset. When the barrel becomes full (state event),e.g. $C(s^{cont}, s^{disc}) = [s^{cont} > fullLevel]$ the contents of the barrel (hence the integrator representing the barrel) is reset back to zero. An example of an external event triggering a reset is provided by a model of the synapse of a leaky integrator neuron [28]. Spike arrivals at the synapse (external events) cause an immediate jump in the soma integrator state (which then proceeds to decay to zero).

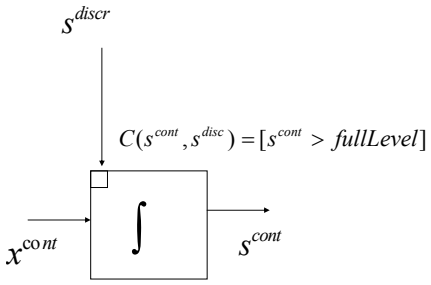

Figure 6. Illustration of discrete event control of integrator state

SIMULINK can reset integrator states during simulation but only to the state that was used in initializing the model – hence for a return to the initial state. This meets the requirements of the barrel filler example. However. it does work not for spikes arriving to a synapse where the reset is to a potentially different state on each occasion. In an inverted pendulum model [29], the angle of the pendulum must be set to 0 when it reaches 2π . This is not the same as the initial angle of the pendulum which is π).

2.2 Regaining the DEVS time advance function trigger of the internal transition function

In DEV&DESS as presented in [8], there is no explicit time advance and hence no triggering of the internal transition function due to time events. This is a simplification of the definition of DEV&DESS as first presented by Praehofer [9]. It is important to show that the simplified definition does not reduce the class of systems specified by the original definition. The time advance triggering functionality is regained by the approach illustrated in Figure 7. Incidentally, the approach also demonstrates the necessity of the three characteristic behaviors. We start by recognizing that time can be represented by the output of an integrator with constant unit input. To get the effect of a scheduled time

advance, we need a condition predicate that detects when the output crosses the given time advance threshold. The output triggers the internal transition function that resets the time integrator back to zero as well as sending the new time advance value to the condition predicate. Thus all three DEV&DESS characteristic behaviors are needed to capture the effect of the time advance trigger in DEVS.

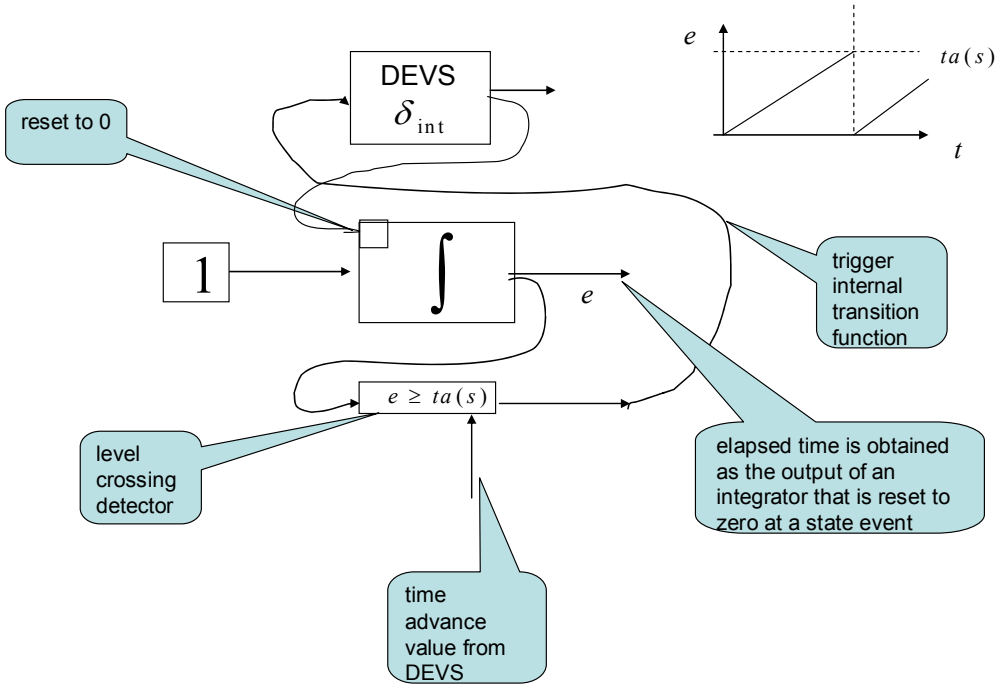

Figure 7. Expressing the time advance triggering of the internal transition function

3 DEVS implementation of DEV&DESS

3.1 Embedding DEV&DESS into DEVS

The DEVS implementation of DEV&DESS will be shown by the embedding of DEV&DESS into DEVS. This is done by showing how an arbitrary DEV&DESS model is realized by a DEVS model. The realization employs the same component-wise simulation concepts that were employed in [8] to provide the embedding of DESS into DEVS. The embedding in Figure 8 implements the individual DEVS and DESS parts of a DEV&DESS model as modular DEVS components coupled within the DEVS formalism. The DEVS part is implemented by a corresponding DEVS denoted by DEVS'. The DESS part is implemented using the quantized integrator approach [8,19], with accuracy determined by the quantum size. The instantaneous derivative functions in this representation are augmented with input ports that allow output events from the DEVS'

component to control the derivative functions. Likewise, the DEVS integrators have additional ports to allow output events from the DEVS' component to instantaneously change the integrator state. The event detection condition is implemented as a DEVS component with input ports receiving outputs of the integrators. Finally, the output ports of the event detector are coupled to the input ports of the DEVS' providing the event detection information to it. This realization which is, in part, an exercise in converting from non-modular to modular form, requires that zero time advances are employed to transmit state and control information that is immediately available in the non-modular form. In this manner, the three behaviors characteristic of DEV&DESS are implemented with DEVS itself.

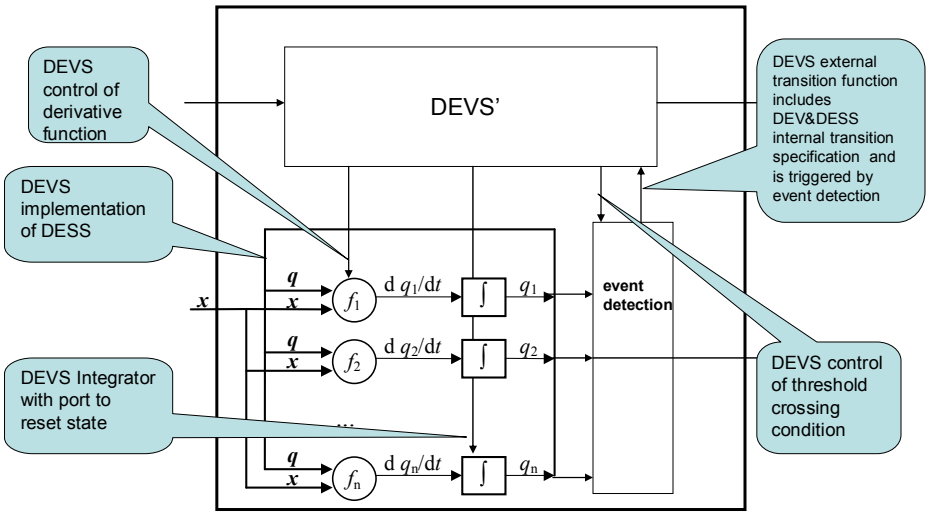

Figure 8. DEVS implementation of DEV&DESS

3.2 Well-defined DEV&DESS: Potential Zero-Time Loop in Integrator Resetting

We recall that the criterion for well-definition of any system specification requires that it specifies a unique system in the class of mathematical input/output systems [8]. The well-definition for DEV&DESS was formulated as a combination and extension of the separate well-definition criteria for DEVS and DESS separately. The DESS requirement is that solutions satisfying the underlying differential equation exist and define unique state trajectories and are guaranteed the Lipshitz condition as well as the absence of algebraic loops, (see page 164 of [8]). The essence of DEVS well-definition is the *legitimacy criterion* which requires that time always moves forward through the accumulation of time advances. An equivalent formulation was given in terms of events, namely, there are only a finite number of external and internal events in any finite time

interval of a simulation run. The extension of this criterion to DEV&DESS is called *state-event legitimacy* and requires that the number of state events be finite for every admissible input segment (page 213 of [8]). We recall that a loop of components all having zero time advances violates this legitimacy criterion. However, this is not the only source means by which illegitimacy arises. In general the convergence of infinite series of time advances must be avoided (see [8] page 142), akin to Zeno's paradox (see [3] for recent perspective.)

Let's now consider how this applies to the testing a DEV&DESS model for legitimacy by employing its embedding in DEVS as in Section 3.1. Theorem 5 of Chapter 15 in [8] is easily amended to state that that the absence of algebraic loops in DESS is equivalent to the absence of zero-time advance loops in its DEVS realization. Assuming Lipshitz conditions hold, this fact reduces the test of well-definition of the DESS part to that of legitimacy of its DEVS representation. However, there remains a source of state-illegitimacy that needs to be considered. It relates to the resetting integrator states by the discrete events.

Figure 10 a) extends Figure 6 by explicitly showing the feedback loop from an integrator through the condition function that decides when and how to reset the integrator. This is a zero-time loop if the integrator's output is instantaneously sent to the decision condition which instantaneously resets the integrator. The example in Figure 10 b) illustrates a feedback coupling between an integrator, *x* and a level crossing detector, *xUp.* The integrator broadcasts its state with zero time advance upon any resetting of its state. The level crossing detector has zero time advance to send the value to the integrator reset port.

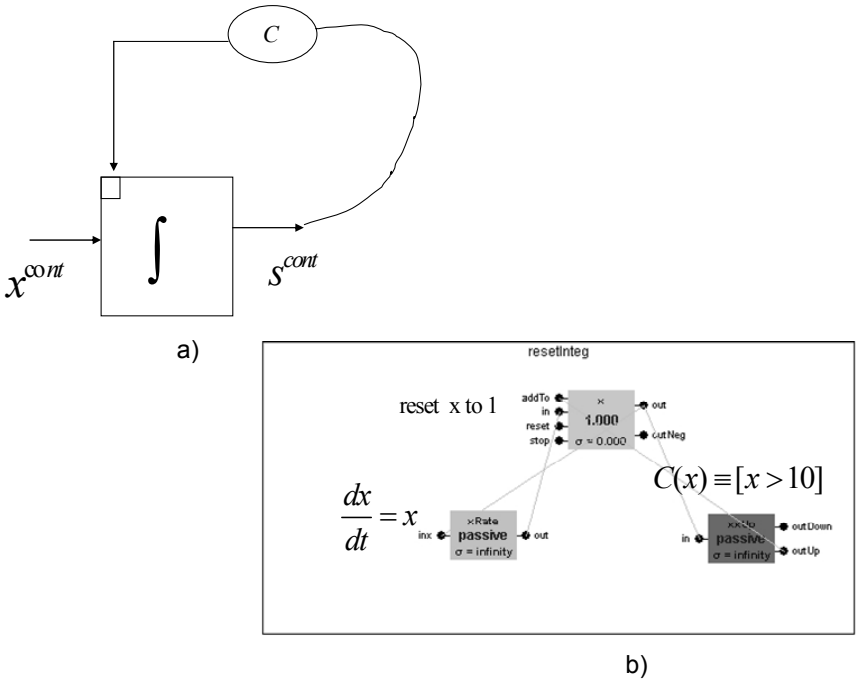

Figure 10. Potential Zero-time Loop in Integrator Reset

Under certain circumstances, this will form a loop of zero time advances. However, this cycle need not be manifest. For example, if the state to which the integrator is reset no longer satisfies the event detector's output criterion then the latter passivates, thus breaking the zero time advance loop. In Figure 10 b), when the level 10 is crossed from below, the integrator is reset back to 1, thus passivating the detector. Note that passivation (setting the time advance to infinity) is available only in a discrete event framework.[1]

We concluded that the well-definition of a DEV&DESS model can be tested in its DEVS embedding by testing for Lipshitz conditions on the DESS derivative functions and state-event legitimacy of the DEVS. A necessary condition for the latter is absence of zero time-advance loops in the DESS part as well as in the interaction between the DEVS and the integrator resets. Although it deals with a major source of illegitimacy, the condition is not sufficient since the latter may still arise through converge in the manner of Zeno's paradox as mentioned above.

4 The Need for Multiple Formalisms in Information Technology-based Systems

The need for multi-formalism modeling in developing modern information technology-based systems is increasingly recognized [2,20]. The ability of DEVS to support such modeling follows from the results just presented. Since DEVS implements DEV&DESS and leads to computationally effective and efficient environments, it offers the basis for combined discrete and continuous (hybrid) modeling. Moreover, as mentioned, since DEVS is universal for discrete event systems, special formalisms such as Petri Nets can be included when appropriate. Finally, DEVS-based simulation components can be interoperated with optimization and other non-dynamic software components. We now briefly discuss robot collaboration to show how this works to address new information-technology challenges.

[1] The same situation shows up in a continuous simulation language that allows integrator resetting as well. However, in that context, it may be treated as an algebraic loop that is detected by the syntactic analyzer. To resolve this situation, SIMULINK adds a state output port to the integrator which provides the last known state to the condition function. Syntactically, this breaks the algebraic loop but may not resolve the true problem. The formulation in Figure 10 within the realization of DEV&DESS in DEVS provides a useful context for examining the problem.

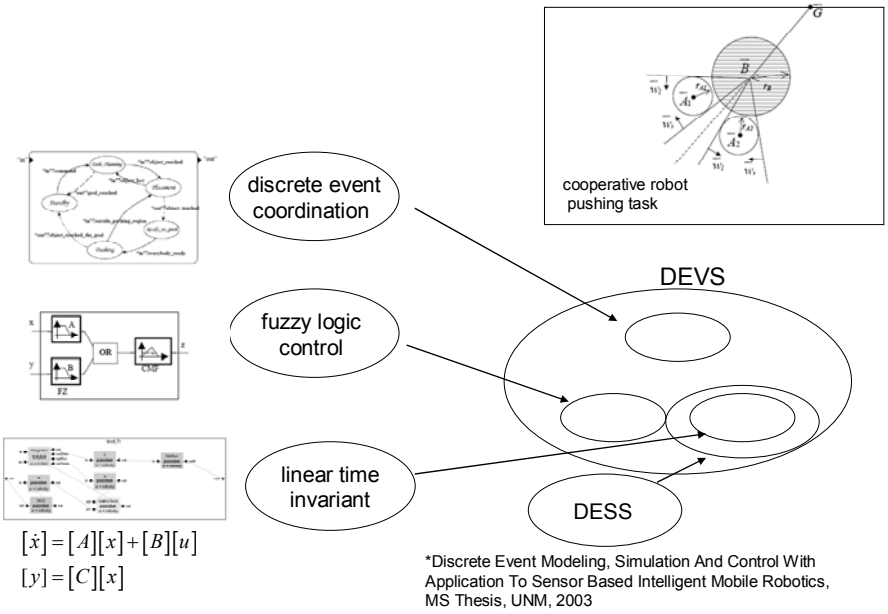

Figure 9.Mulltiformalism Modeling within DEVS

Figure 9 illustrates how several formalisms were employed in the development of cooperative robotics. To design robotic controls so that two or more robots can push an object to a given location, the following formalisms were implemented as packages in DEVSJAVA [4]:

- *Linear time invariant (LTI) systems*– implemented as coupled models that represent differential equations specified in the form:

  $$[\dot{x}] = [A][x] + [B][u]$$
  $$[y] = [C][x]$$

  where the lower case elements are real-valued vectors of inputs, states, and outputs, and the upper case elements are matrices that define the derivative and output functions. LTI is a subclass of DESS that forms the basis of classical control theory. Although not sufficient for robot dynamics, LTI simplifies specification, has a legacy of analytical support, and can be employed within sub-components of non-linear systems, for example, in representing robot and object motion with Newton's laws.

- *Fuzzy Logic Control (FLC) systems* – implemented as coupled models with components for fuzzification, fuzzy rule processing, and defuzzification. FLC is a sub-class of instantaneous functions that has been used in a variety of intelligent control applications [4,18]. FLC provides robust control logic for guiding robots toward goal locations, avoiding obstacles, and cooperatively pushing objects.

- Discrete Event Coordination (DEC) systems – directly implemented within DEVS to provide overall coordination and supervision of robot task executors. DEC provides control phases, analogous to the states of Statecharts, to direct and synchronize individual robot activities toward a global objective.

5. Conclusions: Comparison with Alternative Formalisms

The *Heterogeneous Flow System Specification* (HFSS) [11,12,13] formalism provides an alternative formulation of combined discrete and continuous systems. The formalism is an extension of DEVS that builds in an input sampling mechanism as well as variable structure capability. The approach allows multirate numerical methods for solving differential equations to be represented and supports event detection. Barros [11] claims that the HFSS formalism provides a unified framework for representing digital control, signal processing, numerical integration and hybrid systems. For example, it is possible to describe numerical integrators as part of the formalism and not as an external construct that needs to be incorporated into a pure modeling formalism. Barros [11] also claims that HFSS offers precise semantics to enable the interoperability of hybrid systems in a similar manner existing for discrete systems. However, there has been no discussion of the conditions under which models in the formalism are well-defined nor whether closure under coupling holds for the formalism. The discussion of these issues herein should carry over in large measure to HFSS. A precursor of HFSS, Continuous Flow Specified System (CFSS) was mapped into DEVS as a sub-formalism [30]. Essentially, the mapping shows how to simulate the sampling behavior of CFSS explicitly as a request and wait protocol in DEVS. Since HFSS is an extension of CFSS that preserves this sampling construct, the relationship of HFSS to DEVS an open question for further investigation. Other representations for combined systems within DEVS need attention as well [31,32].

This paper has explicitly pointed out the behavioral features in the DEV&DESS formalism that need to be supported in DEVS for the latter to be able to implement the former. In a broader perspective, the results elucidate the characteristic primitives needed for hybrid or combined continuous/discrete event models and provide a rigorous proof of the ability to express these within a purely discrete event computational framework. These results lay the foundation for a user-friendly modeling environment that puts the primitives directly at the user fingertips as templates for DEVS atomic and coupled models that can be easily instantiated and composed.

References

1. Enrique V. Kortright Modeling and Simulation, Article in Encyclopedia of Computer Science and Technology: Volume 40 -, Allen Kent, James G Williams Taylor and Francis, CRC Press. 1999, *pp* 177-190

2. R.W. Sierenberg, Combined discrete/continuous modeling, in: L. M. Tijskens (Editor), M. Hertog (Editor), B. Nicolai (Editor) Food Process Modelling, 2001

3. Le Goc, M. and C. Frydman, "SACHEM, a Real Time Intelligent Diagnosis System based on the Discrete Event Paradigm", *Transaction of Society for Modeling and Simulation International* (SCS), 2003.

4. Shahab, S., "Discrete Event Modeling, Simulation And Control With Application To Sensor Based Intelligent Mobile Robotics," MS Thesis, UNM, 2003

5. ALUR, R., GROSU, R. LEE, I., AND SOKOLSKY, O. 2001. Compositional refinement for hierarchical hybrid systems. In *Hybrid Systems: Computation and Control. Proceedings of the 4th International Conference (HSCC'01)*. Lecture Notes in Computer Science, vol. 2034. Springer-Verlag, New York, 33-48.

6. R. Alur and D.L. Dill, A theory of timed automata, Theoretical Computer Science, V.126, pp 183-235, 1994

7. ANTSAKLIS, P. J. 2000. A brief introduction to the theory and application of hybrid systems. *Proc. IEEE 88*, 7, 879–887.ACM Transactions on Modeling and Computer Simulation, Vol. 13, No. 3, July 2003.

8. Zeigler, B. P., T.G.Kim and H. Praehofer "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, second edition Academic Press, Boston", 2000. 510 pages

9. Praehofer, H., System Theoretic Formalisms for Combined Discrete-Continuous System Simulation. Int. J. Gen. Sys., 1991. 19(3): p. 219-240.

10. R.C. Huntsinger, Simulation Languages and Applications;, in: George A. Bekey, Boris Y. Kogan (Editors), Modeling and Simulation : Theory and Practice A Memorial Volume for Professor Walter J. Karplus, Kluwer, 2000.

11. F.J. Barros. "Towards a Theory of Continuous Flow Models," *International Journal of General Systems*, Vol. 31, No. 1, 29-39, 2002.

12. F.J. Barros. "Modeling and Simulation of Dynamic Structure Heterogeneous Flow Systems." *SIMULATION: Transactions of The Society for Modeling and Simulation International*, Vol. 78, No. 1, 18-27, 2002.

13. F.J. Barros, "Dynamic Structure Multi-Paradigm Modeling and Simulation," A*CM Transactions on Modeling and Computer Simulation*, Vol. 13, No. 3, pp. 259-275, 2003.

14. Q. Liu, G. Wainer "Simulating Market Dynamics with CD++".. In *Proceedings of the International Conference on Computational Science*. Lecture Notes in Computer Science. Atlanta, GA. 2005

15. Jong-keun Lee, Min-Woo Lee, Sung-Do Chi, “DEVS/HLA-Based Modeling and Simulation for Intelligent Transportation Systems”, SIMULATION, Vol. 79, No. 8, 423-439 (2003).

16. Bernard P.Zeigler, Doohwan Kim, Stephen J. Buckley, “Distributed supply chain simulation in a DEVS/CORBA execution environment”, December 1999 Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 2.

17. Bernard P.Zeigler, Yoonkeon Moon, Doohwan Kim, Jeong Geun Kim, “DEVS-C++: A High Performance Modeling and Simulation Environment”, January 1996, Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96) Volume 1: Software Technology and Architecture.

18. Bernard P.Zeigler, Hessam S. Sarjoughian, Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models, http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/DevsJavaUserGuidev1.1.zip

19. Kofman, E. (2003). "Quantization-Based Simulation of Differential Algebraic Equation Systems". Simulation (Journal of The Society for Computer Simulation International). 79(7). pp 363-376.

20. Hans Vangelieuve and P. Mosterman, “Computer Automated Multi-Paradigm Modeling,” *TOMACS. 12*, 4, 1–7.

21. C. Jacques, G. Wainer Using the CD++ DEVS toolkit to develop Petri Nets. In Proceedings of the 2002 Summer Computer Simulation Conference. San Diego, CA. USA. 2002.

22. Carmen-Veronica Bobeanu, Eugene J. H. Kerckhoffs,Hendrik Van Landeghem,”Modeling of Discrete Event Systems: A Holistic and Incremental Approach Using Petri Nets,” ACM Transactions on Modeling and Computer Simulation, Vol. 14, No. 4, October 2004.

23. Spencer Borland and Hans Vangheluwe. Transforming Statecharts to DEVS. In A. Bruzzone and Mhamed Itmi, editors, *Summer Computer Simulation Conference. Student Workshop*, pages S154 - S159. Society for Computer Simulation International (SCS), July 2003. Montréal, Canada.

24. H. Dacharry, N. Giambiasi ``Formal Verification with Timed Automata and DEVS Models: a case study", *in: ASSE 2005 Simposio Argentino de Ingeniería de Software - 34 JAAIO Jornadas Argentinas de Informatica e Investigacion Operativa*, *pp. 251-265*, *Rosario, Argentine*, 29 août- 2 septembre 2005

25. H. Sarjoughian, “Hybrid Discrete Event Simulation With Model Predictive Control For Semiconductor Supply-chain Manufacturing,” *Proceedings of the 2005 Winter Simulation Conference*, Orlando, FL, 2005.

26.F.E. Cellier, 1979 *Combined Discrete/Continuous System Simulation by Use of Digital Computers*. Techniques and Tools. PhD Thesis, Swiss Federal Institute of Technology,.

27. Pritsker, A.A.B., The GASP IV Simulation Language, John Wiley & Sons, New York, N.Y., 1974

28. Rachel, Olivier, “Une approche evenementielle pour la modelisation et la simulation de reseaux de neurones impulsionnels,” Doctoral Dissertation, Henri Pontcarre University, 2004.

29. K. J. Åström†and K. Furuta‡ Swinging Up A Pendulum By Energy Control, Paper presented at IFAC 13th World Congress, San Francisco, California, 1996

30. F.J. Barros, B.P. Zeigler, Model Interoperability in the Discrete Event Paradigm: Representation of Continuous Models, in: George A. Bekey, Boris Y. Kogan (Editors), Modeling and Simulation : Theory and Practice A Memorial Volume for Professor Walter J. Karplus, Kluwer, 2000.

31. M. D'Abreu, G. Wainer. M/CD++: modeling continuous systems using Modelica and CD++, MASCOTS 2005. Atlanta, GA. 2005

32. N. Giambiasi, G. Wainer ``Using G-DEVS and Cell-DEVS to model complex continuous systems", *in: Simulation: Transactions of the Society for Modeling and Simulation International*, *vol. 81*, *n° 2*, *pp. 137-151*, February 2005.

33. A. Cotaldo, E.L. Lee, X. Liu, E. Matsikouda and H. Zheng, “Discrete-Event Systems: Generalizing Metric Spaces and Fixed-Point Semantics”, J. DEDS,