Discrete Event Solution of Gas Dynamics within the DEVS Framework

J. Nutaro, B. P. Zeigler, R. Jammalamadaka, and S. Akerkar

Arizona Center for Integrative Modeling and Simulation
University of Arizona
Tucson, AZ
{nutaro,zeigler,rajani,salila}@ece.arizona.edu

Abstract The DEVS (Discrete Event Systems Specification) formalism has been applied to continuous and discrete phenomena. The use of discrete events, rather than time steps, as a basis for simulation has been shown to reduce computation time by orders of magnitude in many applications. However, the application of DEVS to partial differential equation (pde) simulation has only recently been investigated. Here, in an application to a shockwave problem, we show that the time to solution is significantly reduced when a discrete event integration scheme is employed compared to a representative conventional approach. Recent theory suggests that speed advantages are to be expected for pdes that are characterized by heterogeneity in their time and space behavior. The implications for use of DEVS as a basis for adaptive control of large scale distributed simulations are discussed.

1 Introduction

The Discrete Event System Specification (DEVS) formalism provides a means of specifying a mathematical object called a system [7]. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs. Discrete event systems represent certain constellations of such parameters just as continuous systems do. For example, the inputs in discrete event systems occur at arbitrarily spaced moments, while those in continuous systems are piecewise continuous functions of time. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. Having this abstraction, it is possible to design new simulation languages with sound semantics that easier are to understand. The DEVJAVA environment [3,8] is an implementation of the DEVS formalism in Java that enables the modeler to specify models directly in its terms. In this paper, we employ the adevs implementation in C++ [1]. A brief review of the DEVS concepts and formalism is provided in the Appendix.

The DEVS formalism has been applied to a number of continuous as well as discrete phenomena (e.g., see [2], [5]). The use of discrete events, rather than time steps, as a basis for simulation has been shown to reduce computation time by orders of magnitude in many applications. However, the application of DEVS to partial

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 319–328, 2003.

differential equation (pde) simulation has only recently been investigated. In pdes, there is an interaction between time and space that is more intimate (through for example, the Courant condition) than in lumped parameter continuous models. Our research has been investigating DEVS solutions to relatively simple pdes while at the same time seeking indicators and supporting theory to predict where significant advantages may be gained.

2 Formulation of the DEVS Model

This paper describes a discrete event solution to a one dimensional gas dynamics problem. The goal is to solve a conservation law with state variable **u** in one spatial dimension *x* whose form is

$$\mathbf{u}_t + [f(\mathbf{u})]_x = 0.$$

A solution can be approximated by dividing the space *x* into discrete cells (grid points) and determining the flux of each state variable across a cell boundary (through a grid point). The motion of the solution is then computed by integrating over the net flux through a cell. To find the flux requires solving the Riemann problem with a particular solution being dependent on the physics of the model. To set up the discrete event approach we started with a conventional discrete time solution to the conservation law and the associated exact, iterative Riemann solver provided by Andrei A. Chernousov (source code available from http://www.geocities.com/andrei_chernousov). As in the latter approach, our discrete event approach employs Chernousov's Riemann solver. However, the essential difference between the two lies in the encompassing integration technique. Figure 1 shows the structure of the space-discretized form of the conservation law.

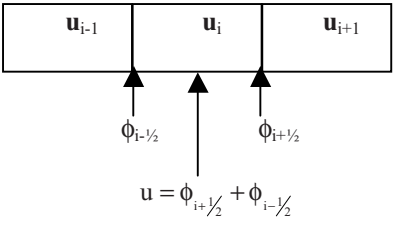

Fig. 1. Spatial discretization of the conservation law.

This system can be modeled by a coupled DEVS [7] that consists of three basic models. The first basic model computes the integral at each cell. It is based on the quantized integrator presented in [4], extended to integrate an N dimensional vector using a volume in N-space as the quantum description. The second basic model receives the outputs of the neighboring quantized integrators and computes the flux at a cell boundary using the exact, iterative Riemann solver. The third is a simple adder that computes the flux across a cell.

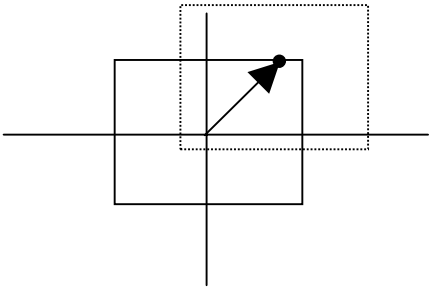

Fig. 2. A quantized integrator with rectangular quantum regions.

The quantized integrator works by drawing a box around the initial point **u**(0) whose dimensions are $D_1 \times D_2 \times \ldots D_N$, where D_i is the quantum size for the i^{th} element of **u**. Figure 2 depicts such an integrator operating in a two-dimensional state space. An output is produced *only* when **u**(t) reaches a boundary of the box. When this occurs, a new box is drawn around the point **u**(t) and the time advance is set to the time required to reach the boundary of this new box. The time advance is used to schedule the boundary crossing. A boundary crossing is represented by an internal event. It is important to note that no computation internal to cell occurs until the internal event is executed – unless an external event is received. An external event occurs when the output of a neighboring cell is received. The time advance after an external event takes into account the updated state as well as the remaining quantum to be consumed.

The discrete event model for this integrator can be formally described with the DEVS formalism as follows:

The state variables for the DEVS model are

q , the current value of the state variables,

$\mathbf{q}_l$, the value of the state variables at the last boundary crossing,

$\dot{\mathbf{q}}$, the first derivatives of the elements of **q**,

σ, the time until the next boundary is crossed.

The model is parameterized by the size of the state vector, denoted by N, and the vector **D** whose elements are the dimensions of the box. The input and output for the integrator are vectors of length N. Before describing the dynamics of the integrator, we need the function T(x,y,z,d) that computes the time to the next internal event in any one of the state dimensions. It is defined by:

$T(x,y,z,d) = \infty$ if $z = 0$

$T(x,y,z,d) = (d - |x - y|) / |z|$ if $z \neq 0$.

When x = y = 0, the situation prevailing after a just completed internal event, T returns the predicted time to reach the next box limit using derivative z. Following an external event, the arguments x and y represent the updated current state and the last

boundary crossing, respectively. Thus, d – |x – y| is the quantum remaining to be consumed, and T returns the predicted time to consume this remaining quantity.

We also define a function $\Gamma(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{d})$ which returns the smallest time to the next event over all state dimensions. $\Gamma(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{d}) = \min \{ T(x_i , y_i, z_i, d_i) \}$ where **x**, **y**, **z**, and **d** are vectors with N elements and the index i ranges over N. With these definitions we have:

The time advance function is given by

$$ta(\mathbf{q}, \mathbf{q}_l, \dot{\mathbf{q}}, \sigma) = \sigma.$$

The state transition function is described by

$$\delta_{int}(\mathbf{q}, \mathbf{q}_l, \dot{\mathbf{q}}, \sigma) = (\mathbf{q} + \sigma\dot{\mathbf{q}}, \mathbf{q} + \sigma\dot{\mathbf{q}}, \dot{\mathbf{q}}, \Gamma(\mathbf{0},\mathbf{0},\dot{\mathbf{q}},\mathbf{D})),$$

$$\delta_{ext}((\mathbf{q}, \mathbf{q}_l, \dot{\mathbf{q}}, \sigma),e,\mathbf{x}) = (\mathbf{q} + e\dot{\mathbf{q}}, \mathbf{q}_l, \mathbf{x}, \Gamma(\mathbf{q} + e\dot{\mathbf{q}},\mathbf{q}_l,\mathbf{x},\mathbf{D})), \text{ and}$$

$$\delta_{con}((\mathbf{q}, \mathbf{q}_l, \dot{\mathbf{q}}, \sigma), \mathbf{x}) = (\mathbf{q} + \sigma\dot{\mathbf{q}}, \mathbf{q} + \sigma\dot{\mathbf{q}}, \mathbf{x}, \Gamma(\mathbf{0},\mathbf{0},\mathbf{x},\mathbf{D})).$$

Finally, the output function is given by

$$\lambda(\mathbf{q}, \mathbf{q}_l, \dot{\mathbf{q}}, \sigma) = \mathbf{q} + \sigma\dot{\mathbf{q}}.$$

The boundary flux model has two state variables $\mathbf{q}_{left}$ and $\mathbf{q}_{right}$ that are the estimates of the state variables in the left and right cells, relative to the boundary. When the boundary flux model receives an input, it updates these values, computes the flux across the boundary, and then immediately outputs the new flux. The summers operate similarly by remembering two values (in this case the left and right fluxes) and responding to input by updating the two state variables and outputting the sum.

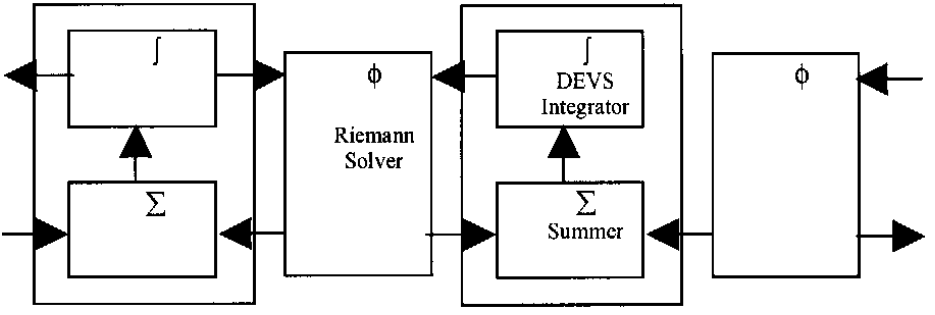

Fig. 3. A DEVS coupled model that approximates the conservation law.

Figure 3 depicts the coupled model that realizes the discrete event approximation based on the spatial discretization shown in Figure 1. The symbol ‡denotes the quantized integrator, ƒis the summer, and ϕ is the Riemann boundary flux solver.

3 Implementation and Experimentation

The discrete event model and corresponding discrete time model were implemented in C++ and executed on a desktop computer with a 500 MHz AMD processor and 198M RAM running the Linux operating system. The discrete time model uses explicit Euler to integrate through time (see [4] for a comparison of explicit Euler and the first order explicit quantized integrator). As indicated, the same iterative Riemann solver was used for both implementations.

An initial value problem was solved using both techniques with identical discretizations in space and a fixed quantum size for the discrete event problem. The step size for the discrete time problem was taken to be the smallest value of the time advance function computed as a result of an internal or confluent event. This gives similar error bounds for both solution techniques. Figure 4 shows the initial conditions for the test problem.

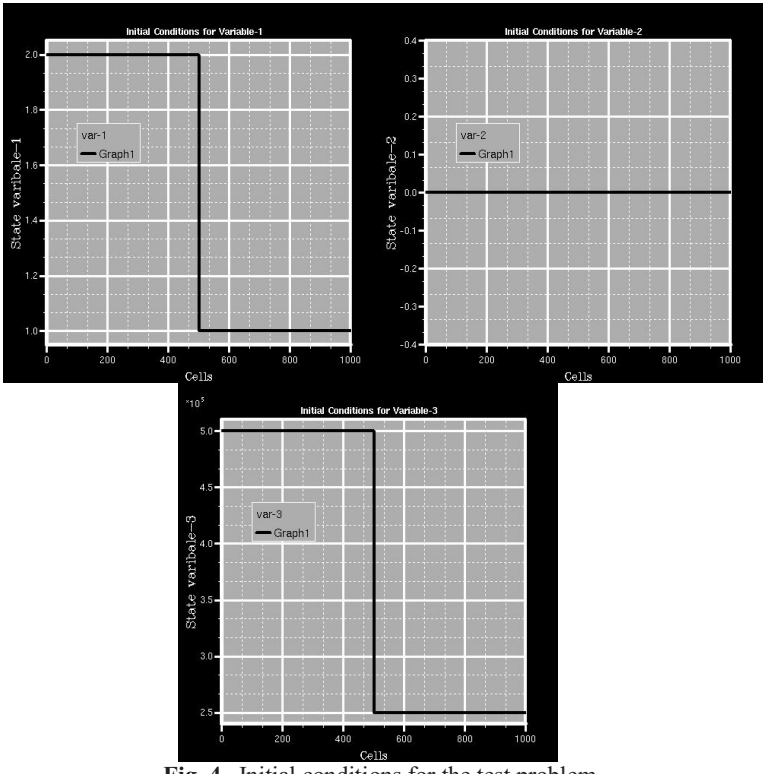

Fig. 4. Initial conditions for the test problem.

The simulation was run for 0.2 units of time. The solution generated by both techniques for a discretization in space using 1000 cells is shown in figure 5.

The allocation of computational effort for the discrete time and discrete event solution for the same 1000 cell run is shown in figure 6. It can be seen that the discrete event solution dynamically focuses computational effort on the highly active

portions of the solution as the shockwave moves outward (left figure). When all computations are accumulated, each active cell receives similar attention but this number is orders of magnitude less than it receives in the discrete time approach (right figure). Note that at the edges are cells to which activity has not propagated and which have therefore not been updated at all.

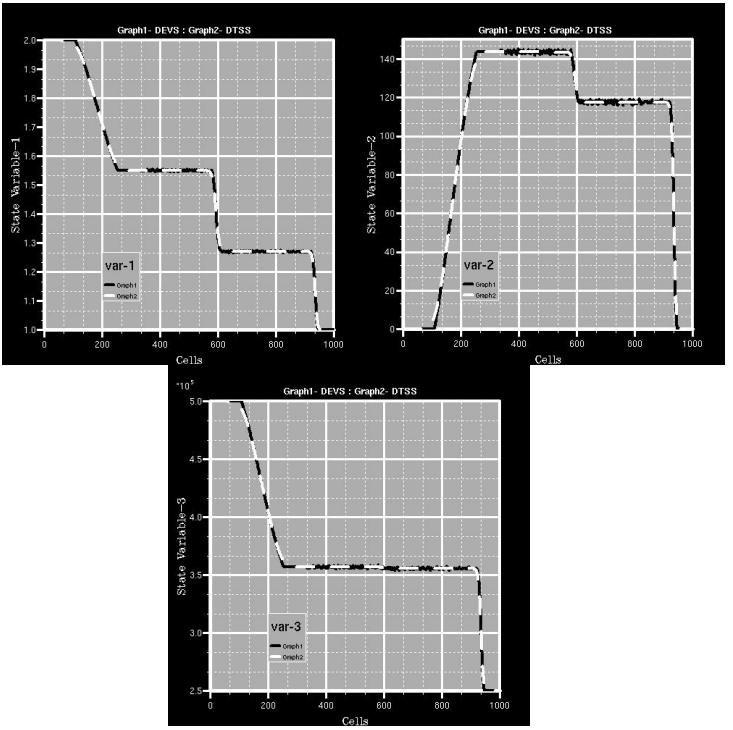

Fig. 5. Solution to the gas dynamics problem at t = 0.2.

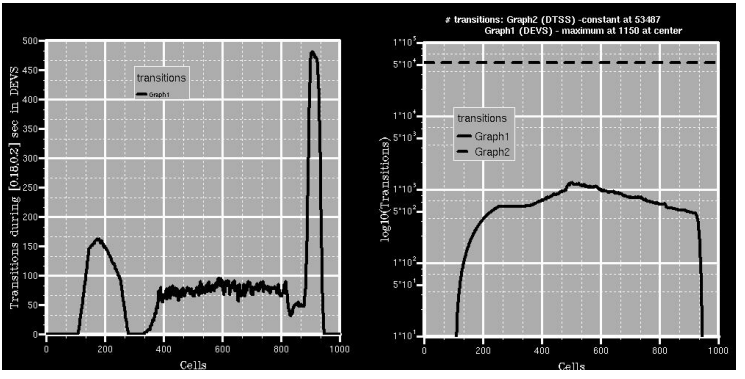

Fig. 6. The number of state changes computed by the DEVS simulator during the time interval [0.18,0.2] (left) and [0.0,0.2] (right).

Table 1 shows the parameters used for each choice of discretization in space and the resulting time to solution. It can be seen that the speed advantage of the DEVS solution, relative to the discrete time solution, increases as the number of cells grows from 200 to 1600. At 1600 cells, the speedup appears to level off at a value of about 34. The combination of focused computational effort and significantly reduced number of computations per cell has produced more than an order of magnitude improvement in the time to solution.

Table 1. Solution parameters and speedup for DEVS and discrete time (DTSS) techniques.

Cells	Quantum size (U1,U2,U3)	Time step	DEVS soln. time (seconds)	DTSS soln. time (seconds)	Relative speedup of DEVS
200	(0.1,1.0,1000.0)	1.87E-005	2.23	19.01	8.52
1000	"	3.74E-006	13.37	380.76	28.5
1600	"	2.34E-006	26.95	941.73	34.9
2000	"	1.87E-006	41.19	1429.42	34.7
4000	"	9.35E-007	165.73	5482	32.8
6000	"	6.23E-007	339.55	10534.5	31.0
8000	'	4.67E-007	622.94	21585	34.7

4 Conclusions

We are developing a theory that suggests that the potential speed advantages with DEVS are to be expected for pdes that are characterized by heterogeneity in their time and space behavior. In such cases, as exemplified by the example discussed above, discrete events are a natural way to focus attention on the portions of the solution that are exhibiting high activity levels at the moment. In fact, theory suggests a way to characterize the activity of solutions over time and space independently of the solution technique that might be employed. This activity measure, when divided by a quantum size, predicts the number of boundary crossings (computations) required by the DEVS simulator for the accuracy afforded by that quantum size. Where significant heterogeneity of activity exists, the number of discrete event computations may be orders of magnitude lower than that required by a uniform allocation of computational resources across both space and time. To realize these potential gains, the choice of data structures used to implement the discrete event simulation engine is of critical importance. A key feature of the discrete event simulator is the use of data structures whose relevant operations have a time complexity of at most $O(\log_2 N)$, where N is the total number of models currently in the simuation (e.g., cells and Riemann boundary flux solvers in the above example).

The work reported here, and research in progress, suggests that DEVS can offer significant performance advantages for simulation of continuous phenomena

characterized by spatiotemporal heterogeneity. Since the DEVS hierarchical, modular framework accommodates coupled models containing both discrete and continuous components, it offers a scalable, efficient framework for very large scale distributed simulation. An important avenue to explore is the incorporation of spatial disaggregation techniques (such as adaptive mesh refinement) within the variable structure capabilities of DEVS modeling and simulation.

Acknowledgement.
This research has been supported in part by NSF Grant No. DMI-0122227, "Discrete Event System Specification (DEVS) as a Formal Modeling and Simulation Framework for Scaleable Enterprise Design" and in part by the Scientific Discovery through Advanced Computing (SciDAC) program of the DOE, grant number DE-FC02-01ER41184.

References

1. Adevs software, http://www.ece.arizona.edu/~nutaro
2. J. Ameghino, A. Tróccoli, G. Wainer. "Models of Complex Physical Systems using Cell-DEVS", Proceedings of the Annual Simulation Symposium, Seattle, Washington, 2001.
3. DEVSJAVA software, http://www.acims.arizona.edu
4. Kofman, E.. "Quantization Based Simulation of Differential Algebraic Equation Systems", Technical Report LSD0203, LSD, Universidad Nacional de Rosario, 2002.
5. Alexandre Muzy, Eric Innocenti, Antoine Aiello, Jean-Francois Santucci, and Gabriel Wainer. "Cell-DEVS Quantization Techniques in a Fire Spreading Application", Winter Simulation Conference, San Diego, California, 2002.
6. Zeigler, B.P., (2002). "The brain-machine disanalogy revisited", *BioSystems,* Vol. 64, pp. 127-140.
7. Zeigler, B.P., T.G. Kim, et al. Theory of Modeling and Simulation. New York, NY, Academic Press, 2000.
8. Zeigler, B.P., H.S. Sarjoughian. "Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations", http://www.acims.arizona.edu, 2001.

Appendix: A Brief Review of DEVS Concepts

The structure of a model may be expressed in a mathematical language called a *formalism.* The formalism defines how to generate new values for variables and the times the new values should take effect. The discrete event formalism focuses on the changes of variable values and generates time segments that are piecewise constant. Thus an event is a change in a variable value that occurs instantaneously. An important aspect of the DEVS formalism is that the time intervals between event occurrences are variable (in contrast to discrete time where the time step is generally a constant number).

To specify modular discrete event models requires that we adopt a different view than that fostered by traditional simulation languages. As with modular specification

in general, we must view a model as possessing input and output ports through which all interaction with the environment is mediated. In the discrete event case, events determine the values appearing on such ports. More specifically, when external events, arising outside the model, are received on its input ports, the model description must determine how it responds to them. Also, internal events, arising within the model, change its state, as well as manifesting themselves as events on the output ports, which in turn are to be transmitted to other model components.

A *basic model* contains the following information:

- the set of input ports through which external events are received,
- the set of output ports through which external events are sent,
- the set of state variables and parameters: two state variables are usually present, "phase" and "sigma" (in the absence of external events the system stays in the current "phase" for the time given by "sigma"),
- the time advance function which controls the timing of internal transitions – when the "sigma" state variable is present, this function just returns the value of "sigma",
- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed,
- the external transition function which specifies how the system changes state when an input is received – the effect is to place the system in a new "phase" and "sigma" thus scheduling it for a next internal transition; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state,
- the confluent transition function which is applied when an input is received at the same time that an internal transition is to occur, and
- the output function which generates an external output just before an internal transition takes place.

A *Discrete Event System Specification* (*DEVS*) is a structure

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

where

X is the set of input values.
S is a set of states.
Y is the set of output values.
δ_{int}: $S \to S$ is the *internal transition* function.
δ_{ext}: $Q \times X^b \to S$ is the *external transition* function, where
$Q \in \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$ is the *total state* set,
e is the *time elapsed* since last transition,
X^b denotes the collection of bags over X (sets in which some elements may occur more than once).
δ_{con}: $S \times X^b \to S$ is the *confluent transition* function.
λ: $S \to Y^b$ is the output function.
ta: $S \to \mathbf{R}^+_{0,\infty}$ is the *time advance* function.

The interpretation of these elements is illustrated in figure 7. At any time the system is in some state, *s*. If no external event occurs the system will stay in state *s* for time *ta*(*s*). Notice that *ta*(*s*) could be a real number and it can also take on the values 0 and ∞. In the first case, the stay in state *s* is so short that no external events can intervene – we say that *s* is a *transitory* state. In the second case, the system will stay in *s* forever unless an external event interrupts its slumber. We say that *s* is *a passive* state in this case. When the resting time expires, i.e., when the elapsed time, $e = ta(s)$, the system outputs the value, $\lambda(s)$, and changes to state $\delta_{int}(s)$. Note that output is only possible just before internal transitions.

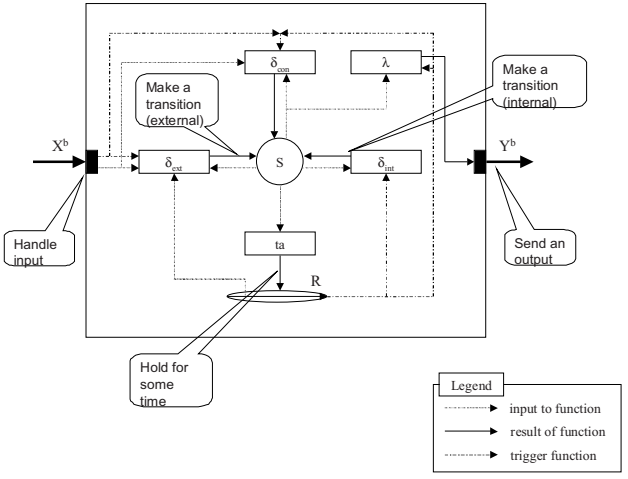

Fig. 7. Interpretation of the DEVS structure.

If an external event $x \in X^b$ occurs before this expiration time, i.e., when the system is in total state (*s*, *e*) with $e \leq ta(s)$, the system changes to state $\delta_{ext}(s,e,x)$. Thus the internal transition function dictates the system's new state when no events have occurred since the last transition. While the external transition function dictates the system's new state when an external event occurs – this state is determined by the input, *x*, the current state, *s*, and how long the system has been in this state, *e*, when the external event occurred. In both cases, the system is then is some new state s' with some new resting time, $ta(s')$ and the same story continues.

Note that an external event $x \in X^b$ is a bag of elements of X. This means that one or more elements can appear on input ports at the same time. This capability is needed since DEVS allows many components to generate output and send these to input ports all at the same instant of time.

Basic models may be coupled to form a *coupled model*. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction.