Simulation Algorithms for DEVS Models

Viliam Solčány
FIIT STU Bratislava, Slovakia
solcany@fiit.stuba.sk

September 2008

Abstract: The Discrete Event System Specification (DEVS) is a modeling formalism with sound theoretical background, enabling to construct discrete event models in a hierarchical and modular way. Several variants of it have been developed during the years. This paper reviews two of them, in particular the original Classic DEVS, and the Parallel DEVS. We briefly summarize the modeling approach, and then provide an explanation of the simulation algorithms which is not easy to find in the literature. We also survey the parallel discrete event simulation (PDES) approaches to DEVS simulation.

1 Introduction

The Discrete Event System Specification (DEVS) formalism has been introduced in the early 1970s with the aim to enable constructing discrete event models in a hierarchical, modular manner. Beside this practical contribution, it is of importance also from the theoretical point of view. Its sound formal foundation allows to reason about various aspects of discrete event modeling and simulation, and about its relationships to other types of models. Due to its strengths, it is receiving increasing attention from different branches of the scientific community.

The basic DEVS formalism allows to describe the inputs, outputs and states of a model, and the relationships among them. Models expressed in this way are called *atomic*. They can be used as building elements to construct larger, *coupled* models. A coupled model describes its children *components* and their interconnections. DEVS strictly enforces modularity of the models, i.e. all model's interactions with the outside world occur through its input and output ports. No direct access to internal state is allowed. An important property of DEVS models is the *closure under coupling*. It ensures that all models (i.e. both atomic and coupled ones) use the same interface protocol. This allows to use a model as a component in another, larger model, leading to hierarchical model construction.

Since the time of the inception of the original DEVS formalism, several extensions of it have been developed. Some of them are meant as specialized formalisms for certain model classes (e.g. the Cell DEVS [10]), others are to resolve some shortcomings of the original. Beside the original formalism called *Classic DEVS* [12], the most notable general extension is the *Parallel DEVS* [1]. We describe these two kinds of DEVS in the following sections.

2 Classic DEVS

The atomic DEVS model is a tuple

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, s_0 \rangle$$

where

$X = \{(p, v)$ such that $p \in InPorts, v \in X_p\}$ is the set of input ports and values
S is the set of (sequential) states
$Y = \{(p, v)$ such that $p \in OutPorts, v \in Y_p\}$ is the set of output ports and values
$\delta_{int} : S \to S$ is the internal transition function
$\delta_{ext} : Q \times X \to S$ is the external transition function,
  where $Q = \{(s, e)$ such that $s \in S, 0 \leq e \leq ta(s)\}$,
  and e is the time elapsed since the last state transition
$\lambda : S \to Y$ is the output function
$ta : S \to R_0^+$ is the time advance function
$s_0 \in S$ is the initial state

The definition of the set X of inputs allows for multiple ports. The model has a set of input ports $InPorts$ where each port p has defined the set of possible values X_p which can appear in the port p. The situation is similar for the outputs.

The model transits among the states enumerated in the set S, using the transition functions δ_{ext} and δ_{int}. The external transition function δ_{ext} computes the new state in the case of external events, the internal transition function δ_{int} is similarly applied to the internal events[1]. The external events are triggered by external inputs to the model. The new state depends on the input $x \in X$, the current state $s \in S$, and the time e for which the model has been in this state. When no external event occurs, an internal event is triggered after time $ta(s)$ since the last state transition (either external or internal). The new state is given by the internal transition function applied to the old state. Right before any internal event, the model can generate an output $y \in Y$ given by the output function λ applied to the current (i.e. old) state. This is the only possibility how the model can generate outputs.

To illustrate the modeling approach, consider a DEVS model of a G/G/1 server with zero-length queue [14]. It has a single input port and a single output port, thus, for the sake of simplicity, we can omit ports altogether. Let all the jobs be identical. Then the input to the model can be any value, e.g. $''J''$ (standing for $''Job''$), and the set of inputs and outputs are $X = \{''J''\}$ and $Y = \{''J''\}$, respectively. The state information consists of two variables. One of them, *phase*, is a boolean status of the server, i.e. busy or idle. The other one, σ maintains the time remaining to the next internal state transition. Then, in mathematical terms, the set of states $S = \{''idle'', ''busy''\} \times R_0^+$. The σ variable allows to define the time advance function simply as $ta(phase, \sigma) = \sigma$. The external state transition function defines the behavior upon a job arrival:

$$\delta_{ext}(phase, \sigma, e, x) = \begin{cases} (''busy'', \text{SERV\_TIME}) & \text{if } phase = ''idle'' \\ (phase, \sigma - e) & \text{otherwise} \end{cases}$$

If the server is idle, it becomes busy. Otherwise the new job is ignored (lost). The SERV_TIME value is a parameter of the model. The job departure is accomplished through the internal state transition with the transition function defined as $\delta_{int}(phase, \sigma) = (''idle'', \infty)$. Right before the internal transition, an output is generated via the output function $\lambda(''busy'', \sigma) = ''J''$. The server is initially idle with no departure scheduled, thus the initial state $s_0 = (''idle'', \infty)$.

Numerous other example models can be found in the book [14].

[1] In the context of DEVS, events are often referred to as "state transitions".

DEVS models with ports can be connected together to form larger structures. The formal foundation for such modeling is provided by the coupled DEVS. The coupled DEVS model is a structure

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle$$

where

$X = \{(p, v)$ such that $p \in InPorts, v \in X_p\}$ is the set of input ports and values
$Y = \{(p, v)$ such that $p \in OutPorts, v \in Y_p\}$ is the set of output ports and values
D is the set of component names
M_d is the DEVS specification of model $d \in D$
I_d is the *influencer set* of model $d \in D \cup \{N\}$, where $I_d \subseteq D \cup \{N\} \setminus \{d\}$
$Z_{i,d}$ is the i to d output translation function, where $d \in D$, $i \in I_d$ and
$Z_{i,d} : X \rightarrow X_d$, if $i = N$
$Z_{i,d} : Y_i \rightarrow Y$, if $d = N$
$Z_{i,d} : Y_i \rightarrow X_d$, if $d \neq N$ and $i \neq N$
$Select : 2^D \setminus \emptyset \rightarrow D$ is the tie-breaking function to arbitrate occurrences of simultaneous events

The coupled model groups several DEVS models as components into a composite. The coupling between the components and between the components and the coupled model N itself is expressed by means of the influencer sets of each model. The influencer set of a model d is the set of models which can send values to d. Three kinds of coupling can be distinguished, corresponding to the three cases of the $Z_{i,d}$ function:

- External input coupling which connects external inputs to component inputs. This is the way how the inputs coming to the composite N can get to its components $d \in D$.
- External output coupling which connects component outputs to external outputs. This coupling enables to transmit output values from the components to the outside of the composite N.
- Internal coupling which connects component outputs to component inputs. Note in the definition of I_d that the component itself is excluded from its influencer set, meaning that no direct feedback loops are allowed in the internal coupling.

All the coupling information is part of the coupled model specification. As a consequence, data sent from one component to another via the internal coupling is actually routed through the coupled model N. Before delivering the data to the target component, it is translated using the $Z_{i,d}$ function. This allows to map the output specification of the source to the input specification of the target. Similar translation takes place in the external coupling.

As a result of coupling of concurrent components, there may be multiple components with simultaneous events. Thus, there may be multiple components which are candidates for the next *internal* state transition. Such components are called *imminents* in the DEVS terminology. The set of imminent components is some subset of the set D from which the *Select* function chooses a unique one to be executed next. This is analogous to priority-based ordering of simultaneous events in common discrete event simulators.

2.1 Simulation of Classic DEVS Models

In order to simulate DEVS models, a framework of *abstract simulators* has been proposed [14]. The framework consists of a hierarchy of *simulator objects* which mirrors the hierarchical structure of the DEVS model being simulated. There is a *DEVS simulator* corresponding to each atomic model, and a *DEVS coordinator* corresponding to each coupled model. At the top of the hierarchy, a *root coordinator* is in charge to control the progress of the simulation. The correspondence between the model and the simulator objects is illustrated by an example in Figure 1.

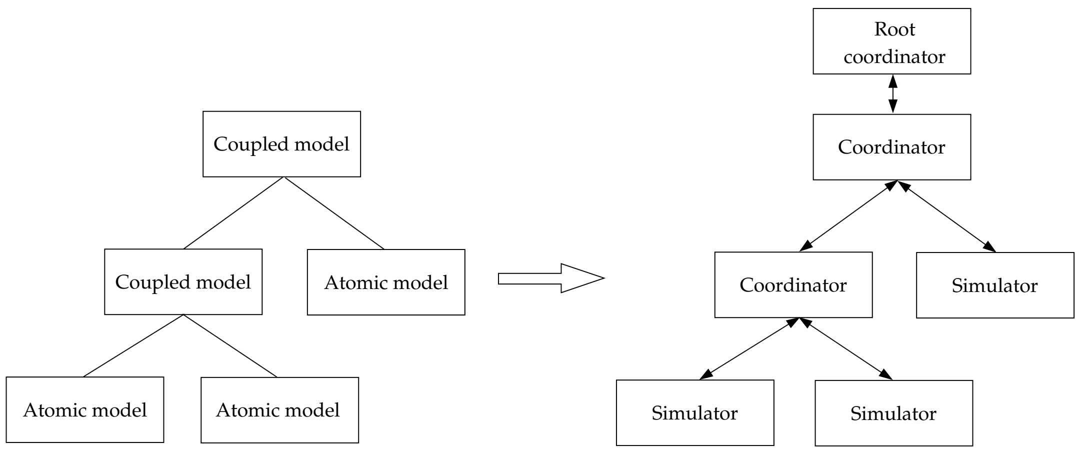

Fig. 1: Mapping a hierarchical DEVS model onto a hierarchical simulator

The DEVS simulator has no event calendar. Instead, it maintains two variables tl and tn. The former holds the simulation time when the last event occurred, and the latter holds the time of the next scheduled (internal) event. The simulator makes the tn variable available to its parent coordinator.

The coordinator maintains an event calendar which is a list of pairs (d, t) where d is the name of a subordinate component and t is the time of its next internal event. It is sorted by the event times t and, when the times are equal, according to the $Select$ function of the coupled model. Thus, the head of the calendar determines the selected imminent component which is the next to execute.

The root coordinator maintains a tn variable containing the time of the next scheduled event within the entire model.

During the initialization of the simulation, each atomic simulator initializes its state, calculates the tn value, and makes this value available to the parent coordinator. The coordinator, after receiving the tn values from all subordinates, sets up the event calendar, and makes the timestamp of the calendar head available to its parent. In this way, the initialization progresses bottom-up the tree until the root coordinator receives the minimum timestamp of the whole model.

After the initialization has been completed, the simulation proceeds in iterations. An iteration consists of the following steps:

1. The root coordinator initiates each iteration by issuing a message with its tn value as the timestamp. The message propagates down the tree. At each tree level, it is routed to the selected component, according to the associated coordinator's calendar. All of these selected components have the minimum timestamp equal to the root's tn value. Eventually, the message reaches the selected atomic component.
2. The selected atomic component invokes its output function λ, and computes its new

state using the internal transition function δ_{int}.

3. If there is an output message generated by the λ function immediately before the internal transition, this message propagates through the tree. In each coordinator which it passes on the way, it is translated by means of the $Z_{i,d}$ functions of the corresponding coupled model, as mentioned above. Eventually, the message affects some atomic model in the form of its external input. The corresponding external state transition function is then applied. In general, the component coupling may specify for a message to propagate to multiple routes, thus there may be multiple atomic components affected. All of these external state transitions occur at the same simulation time as the internal transition from which they originated.
4. All atomic models which underwent a state transition update their tl and tn times and the new values are propagated up the tree. The coordinators update their calendars using these values and eventually the root coordinator gets the new global minimum timestamp. This completes the simulation iteration and the simulator is ready to execute the next one.

The execution of Classic DEVS simulation is purely sequential. In each iteration, there is just one selected atomic component which executes its internal transition. This internal transition can cause zero or more external state transitions in other atomic components. All of them occur at the same simulation time, i.e. simulation time does not change during an iteration.

A more formal description of the simulation algorithm can be found in [14].

From the modeling perspective, the major drawback of Classic DEVS is how it handles the so called *conflicts*. A conflict arises in an atomic component when both an internal and an external transition occur at the same time instant. Consider two atomic components A and B and let both of them be imminent. Assume that when A executes its internal transition, it generates an output which arrives as input to component B. Then the component B experiences a conflict. To resolve the conflict, Classic DEVS allows to determine the order in which the two conflicting state transitions occur. The execution order of the conflicting transitions in B is given by the order in which the imminents A and B are scheduled to execute their respective internal state transitions. The latter can be user-specified by the *Select* functions. Since, in general, the resulting state of component B is different for one execution order than for the other, this may seem as a satisfactory solution. However, there are some flaws:

– The decision about the internal behavior of a component is made on the global level, outside the component itself. This contradicts to the principle of modular modeling.
– To achieve a desired execution order, the *Select* function of multiple coupled models may have to be "tuned". This is a cumbersome approach.
– The setting of the *Select* functions appropriate for solving collisions in one component may contradict to the desired solution for other components.
– For the case of a collision, the modeler may wish to define a different behavior than that which can be expressed by any execution order of the internal and external transition function.

From the simulation perspective, the deficiency of the *Select* function is that it serializes independent simultaneous events, which leads to less efficient simulation, especially for large scale models.

All of these problems are solved in the Parallel DEVS formalism which is described next.

3 Parallel DEVS

Parallel DEVS [1] differs from Classic DEVS in allowing all imminent components to be activated simultaneously and to send their output to other components. The receiver is responsible for examining this input and properly interpreting it. Because of multiple imminents, the inputs take the form of bags of elements. A bag X^b of elements in X is similar to a set except that multiple occurrences of an element are allowed. Similarly to sets, bags are unordered. Using bags as the message structures for collecting inputs sent to a component enables that inputs can arrive in any order and that more than one input with the same identity may arrive from one or more source components.

Another important difference is that Parallel DEVS introduces the confluent transition function. It gives the modeler complete control over the collision behavior when a component receives external input at the time of its internal transition. Rather than serializing model behavior at collision times through the *Select* function at the coupled model level, it resolves the collisions locally within the component. It can do so with or without serializing the internal and external state transitions.

The formal specification of the atomic Parallel DEVS is a structure

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta, s_0 \rangle$$

where

$X = \{(p, v)$ such that $p \in InPorts, v \in X_p\}$ is the set of input ports and values
S is the set of sequential states
$Y = \{(p, v)$ such that $p \in OutPorts, v \in Y_p\}$ is the set of output ports and values
$\delta_{int} : S \rightarrow S$ is the internal transition function
$\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function,
  where $Q = \{(s, e)$ such that $s \in S, 0 \leq e \leq ta(s)\}$,
  e is the time elapsed since the last state transition,
  and $\delta_{ext}(s, e, \emptyset) = (s, e)$
$\delta_{con} : S \times X^b \rightarrow S$ is the confluent transition function with $\delta_{con}(s, \emptyset) = \delta_{int}(s)$
$\lambda : S \rightarrow Y^b$ is the output function
$ta : S \rightarrow R_0^+$ is the time advance function
$s_0 \in S$ is the initial state

Parallel DEVS models can be composed similarly to the Classic DEVS. The formal modeling means for doing this is the coupled model specification. It is a structure

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\} \rangle$$

It is the same as in the Classic DEVS except that the *Select* function is omitted. Visually, this seems to be a tiny change. However it has substantial meaning for the semantics of the model in that all imminents are activated in parallel, as mentioned above.

3.1 Simulation of Parallel DEVS Models

As in the Classic DEVS, the abstract simulator for Parallel DEVS consists of simulator objects arranged in a tree structure mirroring the hierarchical structure of the model. The simulator objects use the same data structures, except that the ordering of simultaneous event records in coordinators' calendars is not defined, since there are no *Select* functions. Similarly to Classic DEVS, the simulation proceeds in iterations. However, in contrast to Classic DEVS, in a simulation iteration, all imminent components execute their internal state transitions simultaneously.

An important point in Parallel DEVS simulation is to decide for each atomic model component that is to make a state transition, which of the three transition functions to apply. The decision depends on the presence of internal and external events:

– If the component is imminent, and has no external events, the internal transition function applies.
– If the component just receives an external input, and no internal events are scheduled for it, the external transition function applies.
– If the component has simultaneous internal and external events, the confluent transition function is applied to determine the new state.

For the decision to be correct, all messages have to be delivered to their destinations prior to actually applying the transition functions. Then the steps of an iteration of the simulation algorithm are as follows:

1. The root coordinator initiates the iteration by issuing a message with its tn value as the timestamp. The message propagates down the tree. At each level, it is routed to all imminent components, until it eventually reaches all imminent atomic components.
2. All imminent atomic components invoke their output functions λ. The generated messages propagate through the tree structure until they reach their respective target atomic components.
3. All imminent atomic components and also those which have received one or more messages in the previous step execute their state transitions. They apply the three rules listed above to select the appropriate state transition function.
4. All atomic models which underwent a state transition update their tl and tn times and the new values are propagated up the tree. The coordinators update their calendars using these values and eventually the root coordinator gets the new global minimum timestamp. This completes the simulation iteration and the simulator is ready to execute the next one.

The Parallel DEVS solves the modeling flaws of the Classic DEVS approach, and, at the same time, opens the possibility to employ multiple processors in its simulation. However, the parallel execution relies on the central role of the root coordinator and is strictly synchronous. Just the state transitions occurring at the same time can be executed in parallel. In spite of that, there can be significant performance improvement in comparison to the Classic DEVS, especially in large scale models. This can be further improved if some small temporal inaccuracy of the simulation can be tolerated. Then the set of imminent components includes not just those which have the next event scheduled exactly at time tn, but also those which are within a small interval ahead of tn. This markedly increases the degree of concurrency [14]. Still, the root coordinator dictates the synchronous advance of the simulation and thus is a principal bottleneck.

4 PDES Approaches to DEVS Simulation

In order to extend the time and/or memory limits of DEVS simulations, the techniques developed in the area of *Parallel Discrete Event Simulation* (PDES) [2] can be applied[2]. There are studies involving both *conservative* and *optimistic* methods known from the literature. However, just few of them belong to the conservative class. In [14], a simulation framework is described for DEVS models with no hierarchy using conservative synchronization. For each atomic component, there is a DEVS simulator, as in the sequential case. Each DEVS simulator has a conservative simulator associated with it which controls the DEVS simulator. Such simulators are aggregated into submodels, each of which has a component responsible for message distribution to other submodels. Each conservative simulator maintains the estimates for the *earliest input time* (EIT), and *earliest output time* (EOT). The component never receives an input with time less than its EIT. Based on this estimate and the local state, it can determine the *lookahead* which serves to calculate the EOT value. The EIT/EOT values are propagated among the components by means of *null messages*. Whenever the EOT value of a component changes, it sends null messages to all of its influencees. Real messages carrying inputs originating in other components are not processed immediately but stored in an event calendar, so that the timestamp order of their processing is ensured.

The overhead of this approach due to the null messages can be large in comparison to event execution. It can be reduced using an approach known from the conservative methods, such as null messages sent with a delay, or on demand, etc. Even with such improvements, the performance is low when the real message traffic is low. Further, as with conservative methods in general, good lookahead is needed for reasonable performance.

Another conservatively synchronized DEVS simulator has been reported in [3], and [8]. It allows for hierarchical models, but its performance is rather low too. Since DEVS is a general modeling formalism, and the aim is to construct a high performance general purpose simulator for it, conservative methods are not likely to be the right choice.

Optimistic parallel execution of DEVS models is possible through combining *Time Warp* with hierarchical scheduling of the conventional DEVS simulator [14]. The model is partitioned so that each submodel is a single coupled model. Then the hierarchical scheduling is used locally within a simulation processor as in conventional abstract simulator, with extensions supporting state saving and rollback. Additionally, each submodel has a root coordinator which realizes the Time Warp mechanism.

The extension of the DEVS simulator includes a queue to store the state information together with the tl and tn times. The simulator processes conventional messages in usual way. Additionally, it reacts to a rollback message by searching the queue for the appropriate state and tl, tn values, and restoring them. The extended coordinator processes a rollback message by simply forwarding it to all children with tl value greater than the timestamp of the rollback message.

The Time Warp root coordinator performs the optimistic synchronization itself. It optimistically initiates hierarchical simulation cycles. It stores input and output messages from and to other submodels and takes care of *anti-messages*. When it receives an output

[2] Beside the performance-focused parallel simulation of DEVS models, this modeling formalism can serve as a basis also in the interoperability-focused distributed simulation [13, 15]. Probably the most notable effort in this area was the development of the DEVS/HLA simulation framework [6]. It is based on a more general concept for interoperability called DEVS Bus [7]. Recently, an approach to interoperability of models written in multiple programming languages has been proposed [11].

message from the child, the message is stored in the output queue, marked as transient, and sent to the influenced submodels. At the receiving submodel, the message is stored in the input queue, and its transient mark is cleared. Messages from the input queue are processed in timestamp order. If the incoming message is a straggler, a rollback message is sent to the child and anti-messages are sent for all outputs in the output queue with later time. When an anti-message requests cancellation of an input event, the corresponding input message is deleted from the input queue and a rollback is performed. In addition to this local responsibilities, the root coordinator is involved in global GVT computation, based on which *fossil collection* is performed.

Beside the sketched solution, other variants of the optimistic DEVS simulation are possible. One of them is a *risk free* optimistic simulator, where a globally derived information is used to decide when to transmit a message [14]. Other known solution eliminates the hierarchy from the run-time structure of submodels. This leads to reduction of message overhead, and subsequently to better performance [8, 3]. An optimistic PDES approach to discrete event simulation of continuous systems based on their DEVS representation has been designed in [9]. According to the cited literature, performance of the optimistic DEVS simulation is significantly better in comparison to the conservative one.

Any PDES simulation with Classic DEVS as the modeling approach runs into trouble when the processing order of simultaneous events of the sequential counterpart has to be preserved. The *local causality constraint* is obeyed with any processing order of such events. If they are to be processed in a particular order, special care needs to be taken [4, 5]. Parallel DEVS is not subject to this problem.

5 Conclusions and Future Work

We have presented a high level view of the simulation algorithms for DEVS models, which we believe may be helpful to better understand the algorithms themselves, and in turn, the modeling approach (the modeling interface) too. The brief survey of the PDES approaches to DEVS simulation assumes some degree of familiarity with the area of PDES.

In the future, this work may be continued by a detailed, formal description of the algorithms. However, the latter should not be understood as a replacement of the former. The two formulations of the algorithms complement each other. Although the formal description contains every detail, the high level description can help to understand the details in the overall algorithm context.

Bibliography

1. Alex C. Chow and Bernard P. Zeigler. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th Winter Simulation Conference*, pages 716–722, San Diego, CA, USA, 1994. Society for Computer Simulation International.
2. Richard M. Fujimoto. *Parallel and Distributed Simulation Systems.* John Wiley & Sons, Inc., 1999.
3. Ezequiel Glinsky and Gabriel Wainer. New parallel simulation techniques of DEVS and Cell-DEVS in CD++. In *Proceedings of the 39th Annual Simulation Symposium*, pages 244–251, Washington, DC, USA, 2006. IEEE Computer Society.
4. Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed optimistic simulation of hierarchical DEVS models. In *Proceedings of the 1995 Summer Computer Simulation Conference*, pages 32–37, July 1995.

5. Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Ordering of simultaneous events in distributed DEVS simulation. *Simulation Practice and Theory*, 5:253–268, 1997.
6. Y. J. Kim, J. H. Cho, and T. G. Kim. DEVS/HLA: Heterogeneous simulation framework using DEVS Bus implemented on RTI. In *Proceedings of the 1999 Summer Computer Simulation Conference*, Chicago, IL., 1999.
7. Y. J. Kim, J. H. Kim, and T. G. Kim. Heterogeneous simulation framework using DEVS BUS. *Simulation*, 79(1):3–18, Jan 2003.
8. Qi Liu and Gabriel Wainer. Parallel environment for DEVS and Cell-DEVS models. *Simulation*, 83(6):449–471, 2007.
9. James Joseph Nutaro. *Parallel Discrete Event Simulation with Application to Continuous Systems*. PhD thesis, Department of Electrical and Computer Engineering, University of Arizona, 2003.
10. Gabriel A. Wainer and Norbert Giambiasi. N-dimensional Cell-DEVS Models. *Discrete Event Dynamic Systems*, 12(2):135–157, 2002.
11. Thomas Wutzler and Hessam S. Sarjoughian. Interoperability among Parallel DEVS simulators and models implemented in multiple programming languages. *Simulation*, 83(6):473–490, 2007.
12. Bernard P. Zeigler. *Theory of Modeling and Simulation, First Edition*. Wiley Interscience, 1976.
13. Bernard P. Zeigler, Doohwan Kim, and Herbert Praehofer. DEVS formalism as a framework for advanced distributed simulation. In *Proceedings of the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications (DIS-RT '97), Eilat, Israel*, pages 15–21, 1997.
14. Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation, Second Edition*. Academic Press, Inc., Orlando, FL, USA, 2000.
15. Bernard P. Zeigler and Hessam S. Sarjoughian. Creating distributed simulation using DEVS M&S environments. In *Proceedings of the 32th Winter Simulation Conference*, pages 158–160, 2000.