The Discrete EVent System specification (DEVS) formalism

COMP 522 Fall Term 2002

Hans Vangheluwe

The DEVS formalism was conceived by Zeigler [Zei84a, Zei84b] to provide a rigourous common basis for discrete-event modelling and simulation. For the class of formalisms denoted as *discrete-event* [Nan81], system models are described at an abstraction level where the time base is continuous ($\mathbb{R}$), but during a bounded time-span, only a *finite number* of relevant *events* occur. These events can cause the state of the system to change. In between events, the state of the system does *not* change. This is unlike *continuous* models in which the state of the system may change continuously over time. As an extension of Finite State Automata, the DEVS (Discrete Event Systems) formalism captures *concepts* from Discrete Event simulation. As such it is a sound basis for meaningful model exchange in the Discrete Event realm.

1 The DEVS Formalism

The DEVS formalism fits the general structure of deterministic, causal systems in classical systems theory. DEVS allows for the description of system behaviour at two levels. At the lowest level, an *atomic DEVS* describes the autonomous behaviour of a discrete-event system as a sequence of deterministic transitions between sequential states as well as how it reacts to external input (events) and how it generates output (events). At the higher level, a *coupled DEVS* describes a system as a *network* of coupled components. The components can be atomic DEVS models or coupled DEVS in their own right. The connections denote how components influence each other. In particular, output events of one component can become, via a network connection, input events of another component. It is shown in [Zei84a] how the DEVS formalism is *closed under coupling*: for each coupled DEVS, a *resultant* atomic DEVS can be constructed. As such, any DEVS model, be it atomic or coupled, can be replaced by an equivalent atomic DEVS. The construction procedure of a resultant atomic DEVS is also the basis for the implementation of an *abstract simulator* or solver capable of simulating any DEVS model. As a coupled DEVS may have coupled DEVS components, *hierarchical* modelling is supported.

In the following, the different aspects of the DEVS formalism are explained in more detail.

1.1 The atomic DEVS Formalism

The atomic DEVS formalism is a structure describing the different aspects of the discrete-event behaviour of a system:

$$atomicDEVS \equiv \langle S, ta, \delta_{int}, X, \delta_{ext}, Y, \lambda \rangle.$$

The *time base* T is continuous and is not mentioned explicitly

$$T = \mathbb{R}.$$

The *state set* S is the set of admissible *sequential states*: the DEVS dynamics consists of an ordered sequence of states from S. Typically, S will be a *structured* set (a product set)

$$S = \times_{i=1}^{n} S_i.$$

This formalizes multiple (n) *concurrent* parts of a system. It is noted how a structured state set is often synthesized from the state sets of concurrent components in a coupled DEVS model.

The time the system *remains* in a sequential state before making a transition to the next sequential state is modelled by the *time advance function*

$$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}.$$

As time in the real world always advances, the image of ta must be non-negative numbers. $ta = 0$ allows for the representation of *instantaneous* transitions: no time elapses before transition to a new state. Obviously, this is an abstraction of reality which may lead to simulation *artifacts* such as infinite instantaneous loops which do not correspond to real physical behaviour. If the system is to stay in an end-state *s forever*, this is modelled by means of $ta(s) = +\infty$.

The internal transition function

$$\delta_{int} : S \to S$$

models the transition from one state to the next sequential state. δ_{int} describes the behaviour of a Finite State Automaton; ta adds the progression of time.

It is possible to *observe* the system output. The output set Y denotes the set of admissible *outputs*. Typically, Y will be a *structured* set (a product set)

$$Y = \times_{i=1}^{l} Y_i.$$

This formalizes multiple (l) output ports. Each port is identified by its unique index i. In a user-oriented modelling language, the indices would be derived from unique port *names*.

The output function

$$\lambda : S \to Y \cup \{\phi\}$$

maps the internal state onto the output set. Output events are *only* generated by a DEVS model at the time of an *internal* transition. At that time, the state *before* the transition is used as input to λ. At all other times, the non-event ϕ is output.

To describe the *total state* of the system at each point in time, the sequential state $s \in S$ is not sufficient. The *elapsed* time e since the system made a transition to the current state s needs also to be taken into account to construct the total state set

$$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$$

The elapsed time e takes on values ranging from 0 (transition just made) to $ta(s)$ (about to make transition to the next sequential state). Often, the *time left* σ in a state is used:

$$\sigma = ta(s) - e$$

Up to now, only an *autonomous* system was described: the system receives no external inputs. Hence, the *input set* X denoting all admissible input values is defined. Typically, X will be a *structured* set (a product set)

$$X = \times_{i=1}^{m} X_i$$

This formalizes multiple (m) input ports. Each port is identified by its unique index i. As with the output set, port indices may denote *names*.

The set Ω contains all admissible input segments ω

$$\omega : T \to X \cup \{\phi\}$$

In discrete-event system models, an input segment generates an input *event* different from the *non-event* ϕ only at a finite number of instants in a bounded time-interval. These *external events*, inputs x from X, cause the system to interrupt its autonomous behaviour and react in a way prescribed by the external transition function

$$\delta_{ext} : Q \times X \to S$$

The reaction of the system to an external event depends on the sequential state the system is in, the particular input *and* the elapsed time. Thus, δ_{ext} allows for the description of a large class of behaviours typically found in discrete-event models (including synchronization, preemption, suspension and re-activation).

When an input event x to an atomic model is not listed in the δ_{ext} specification, the event is *ignored*.

In Figure 1, an example state trajectory is given for an atomic DEVS model. In the figure, the system made an internal transition to state $s2$. In the absence of external input events, the system stays in state $s2$ for a duration $ta(s2)$. During

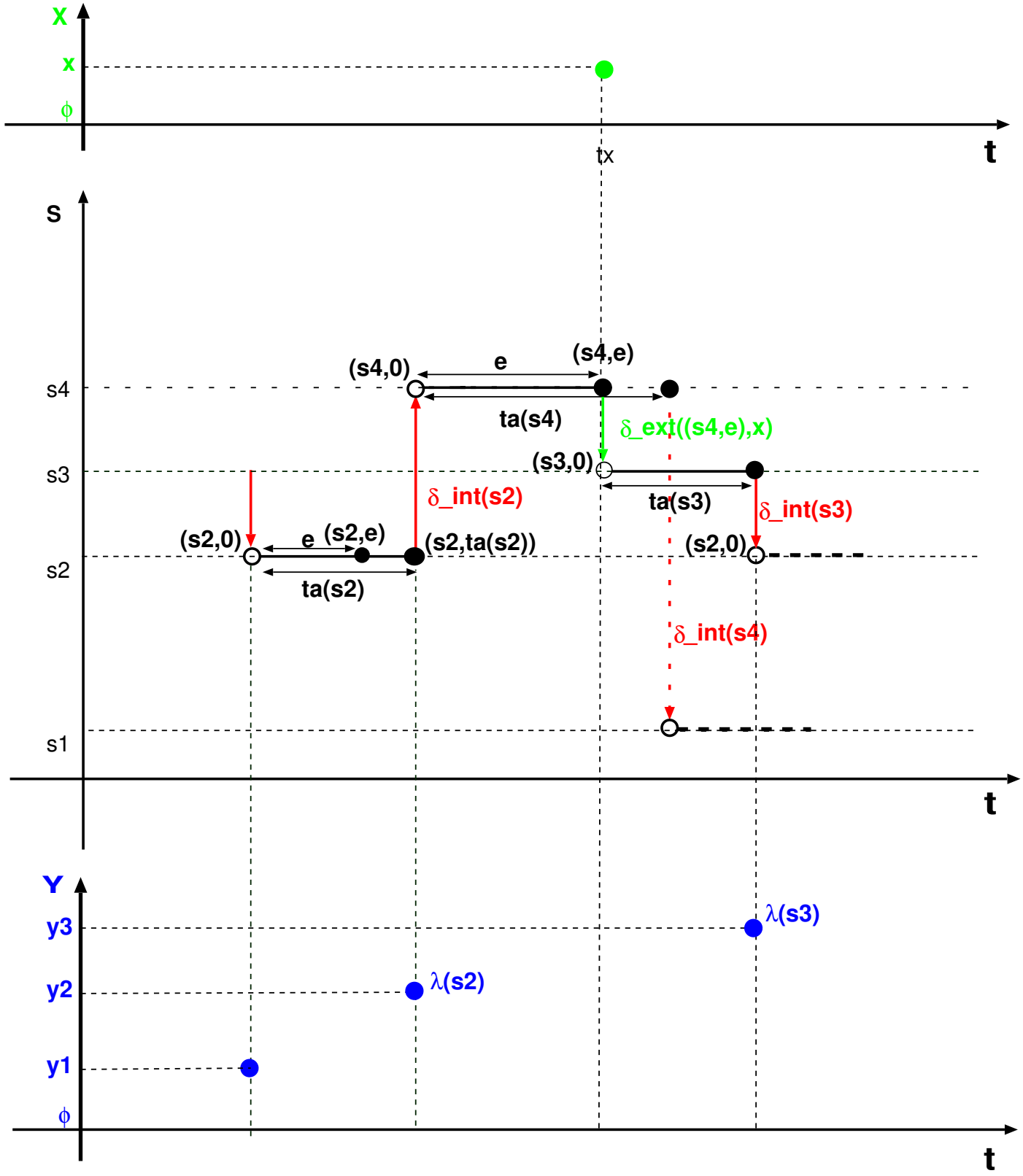

Figure 1: State Trajectory of a DEVS-specified Model

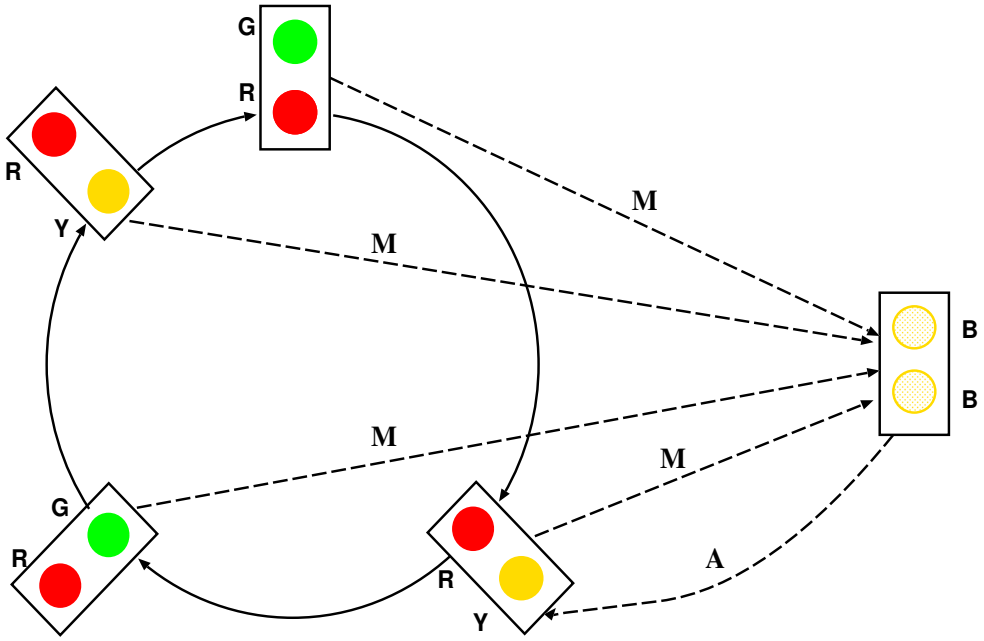

Figure 2: Traffic light example

this period, the elapsed time e increases from 0 to $ta(s2)$, with the total state $= (s2, e)$. When the elapsed time reaches $ta(s2)$, first an output is generated: $y2 = \lambda(s2)$, then the system transitions instantaneously to the new state $s4 = \delta_{int}(s2)$. In autonomous mode, the system would stay in state $s4$ for $ta(s4)$ and then transition (after generating output) to $s1 = \delta_{int}(s4)$. Before e reaches $ta(s4)$ however, an external input event x arrives. At that time, the system forgets about the scheduled internal transition and transitions to $s3 = \delta_{ext}((s4, e), x)$. Note how an external transition does not give rise to an output. Once in state $s3$, the system continues in autonomous mode.

As an example atomic DEVS, consider the model of two traffic lights depicted in Figure 2. In autonomous mode, the light transition in intuitive fashion. If the "switch to manual" (M) external event is received, lights in both directions blink yellow. If the "switch to automatic" event is received, the system switches back deterministically to state RY to resume autonomous mode. The atomic DEVS representation is given below.

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

$$
\begin{aligned}
&T = \mathbb{R} \\
&X = \{M, A\} \\
&\omega : T \to X \cup \{\phi\} \\
&S = \{RG, RY, GR, YR, BB\} \\
&\delta_{int}(RG) = RY;\ \delta_{int}(RY) = GR \\
&\delta_{int}(GR) = YR;\ \delta_{int}(YR) = RG \\
&ta(RG) = 60s;\ ta(RY) = 10s \\
&ta(GR) = 50s;\ ta(YR) = 10s \\
&ta(BB) = +\infty \\
&\delta_{ext}((RG, e), M) = BB \\
&\delta_{ext}((RY, e), M) = BB \\
&\delta_{ext}((GR, e), M) = BB \\
&\delta_{ext}((YR, e), M) = BB \\
&\delta_{ext}((BB, e), A) = RY \\
&Y = \{GREY, YELLOW, BLINK\} \\
&\lambda(RG) = \lambda(RY) = \lambda(GR) = GREY \\
&\lambda(YR) = YELLOW \\
&\lambda(BB) = BLINK
\end{aligned}
$$

1.2 The coupled DEVS Formalism

The coupled DEVS formalism describes a discrete-event system in terms of a network of coupled components.

$$coupledDEVS \equiv \langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select \rangle$$

The component $self$ denotes the coupled model itself. X_{self} is the (possibly structured) set of allowed external inputs to the coupled model. Y_{self} is the (possibly structured) set of allowed (external) outputs of the coupled model. D is a set of unique component references (names). The coupled model itself is referred to by means of $self$, a unique reference not in D.

The set of *components* is

$$\{M_i | i \in D\}.$$

Each of the components must be an atomic DEVS

$$M_i = \langle S_i, ta_i, \delta_{int,i}, X_i, \delta_{ext,i}, Y_i, \lambda_i \rangle, \forall i \in D.$$

The set of *influencees* of a component, the components influenced by $i \in D \cup \{self\}$, is I_i. The set of all influencees describes the coupling network structure

$$\{I_i | i \in D \cup \{self\}\}.$$

For modularity reasons, a component (including $self$) may not influence components outside its scope –the coupled model–, rather only other components of the coupled model, or the coupled model $self$:

$$\forall i \in D \cup \{self\} : I_i \subseteq D \cup \{self\}.$$

This is further restricted by the requirement that none of the components (including $self$) may influence themselves directly as this could cause an instantaneous dependency cycle (in case of a 0 time advance inside such a component) akin to an algebraic loop in continuous models:

$$\forall i \in D \cup \{self\} : i \notin I_i.$$

Note how one can always encode a self-loop ($i \in I_i$) in the internal transition function.

To translate an output event of one component (such as a departure of a customer) to a corresponding input event (such as the arrival of a customer) in influencees of that component, *output-to-input translation functions* $Z_{i,j}$ are defined:

$$\{Z_{i,j} | i \in D \cup \{self\}, j \in I_i\},$$

$$\begin{array}{rcll} Z_{self,j} & : & X_{self} \to X_j & , \forall j \in D, \\ Z_{i,self} & : & Y_i \to Y_{self} & , \forall i \in D, \\ Z_{i,j} & : & Y_i \to X_j & , \forall i, j \in D. \end{array}$$

Together, I_i and $Z_{i,j}$ completely specify the coupling (structure and behaviour).

As a result of coupling of concurrent components, multiple state transitions may occur at the same simulation time. This is an artifact of the discrete-event abstraction and may lead to behaviour not related to real-life phenomena. A logic-based foundation to study the *semantics* of these artifacts was introduced by Radiya and Sargent [RS94]. In sequential simulation systems, such transition *collisions* are resolved by means of some form of *selection* of which of the components' transitions should be handled first. This corresponds to the introduction of priorities in some simulation languages. The coupled DEVS formalism explicitly represents a *select* function for *tie-breaking* between simultaneous events:

$$select : 2^D \to D$$

select chooses a unique component from any non-empty subset E of D:

$$select(E) \in E.$$

The subset E corresponds to the set of all components having a state transition simultaneously.

1.3 Closure of DEVS under coupling

As mentioned before, it is possible to construct a *resultant* atomic DEVS model for each coupled DEVS. This *closure under coupling* of atomic DEVS models makes *any* coupled DEVS equivalent to an atomic DEVS. By induction, any *hierarchically* coupled DEVS can thus be flattened to an atomic DEVS. As a result, the requirement that each of the components of a coupled DEVS be an atomic DEVS can be relaxed to be atomic *or* coupled as the latter can always be replaced by an equivalent atomic DEVS.

The core of the closure procedure is the selection of the most *imminent* (*i.e.,* soonest to occur) event from all the components' scheduled events [Zei84a]. In case of simultaneous events, the *select* function is used. In the sequel, the resultant construction is described.

From the coupled DEVS

$$\langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select \rangle,$$

with all components M_i atomic DEVS models

$$M_i = \langle S_i, ta_i, \delta_{int,i}, X_i, \delta_{ext,i}, Y_i, \lambda_i \rangle, \forall i \in D$$

the atomic DEVS

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

is constructed.

The resultant set of sequential states is the product of the total state sets of all the components

$$S = \times_{i \in D} Q_i,$$

where

$$Q_i = \{(s_i, e_i) | s \in S_i, 0 \leq e_i \leq ta_i(s_i)\}, \forall i \in D.$$

The time advance function ta

$$ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$$

is constructed by selecting the *most imminent* event time, of all the components. This means, finding the smallest time *remaining* until internal transition, of all the components

$$ta(s) = min\{\sigma_i = ta_i(s_i) - e_i | i \in D\}.$$

A number of *imminent* components may be scheduled for a *simultaneous* internal transition. These components are collected in a set

$$IMM(s) = \{i \in D | \sigma_i = ta(s)\}.$$

From IMM, a set of elements of D, *one* component i^* is chosen by means of the *select tie-breaking* function of the coupled model

$$\begin{array}{rcccc} select & : & 2^D & \rightarrow & D \\ & & IMM(s) & \rightarrow & i^* \end{array}$$

Output of the selected component is generated *before* it makes its internal transition. Note also how, as in a Moore machine, input does not directly influence output. In DEVS models, *only* an internal transition produces output. An input can only influence/generate output via an internal transition similar to the presence of *memory* in the form of integrating elements in continuous models. Allowing an external transition to produce output could lead to infinite instantaneous loops. This is equivalent to algebraic loops in continuous systems. The output of the component is translated into coupled model output by means of the coupling information

$$\lambda(s) = Z_{i^*,self}(\lambda_{i^*}(s_{i^*})), \text{if } self \in I_{i^*}.$$

If the output of i^* is not connected to the output of the coupled model, the non-event ϕ can be generated as output of the coupled model. As ϕ literally stands for no event, the output can also be ignored without changing the meaning (but increasing performance of simulator implementations).

The internal transition function transforms the different parts of the total state as follows:

$$\delta_{int}(s) = (\ldots, (s'_j, e'_j), \ldots), \text{ where}$$

$$\begin{array}{rcll} (s'_j, e'_j) & = & (\delta_{int,j}(s_j), 0) & \text{, for } j = i^*, \\ & = & (\delta_{ext,j}(s_j, e_j + ta(s), Z_{i^*,j}(\lambda_{i^*}(s_{i^*}))), 0) & \text{, for } j \in I_{i^*}, \\ & = & (s_j, e_j + ta(s)) & \text{, otherwise.} \end{array}$$

The selected imminent component i^* makes an internal transition to sequential state $\delta_{int,i^*}(s_{i^*})$. Its elapsed time is reset to 0. All the influencees of i^* change their state due to an external transition prompted by an input which is the output-to-input translated output of i^*, with an elapsed time adjusted for the time advance $ta(s)$. The influencees' elapsed time is reset to 0. Note how i^* is not allowed to be an influencee of i^* in DEVS. The state of all other components is not affected and their elapsed time is merely adjusted for the time advance $ta(s)$.

The external transition function transforms the different parts of the total state as follows:

$$\delta_{ext}(s, e, x) = (\ldots, (s'_i, e'_i), \ldots), \text{ where}$$

$$\begin{array}{rcll} (s'_i, e'_i) & = & (\delta_{ext,i}(s_i, e_i + e, Z_{self,i}(x)), 0) & \text{, for } i \in I_{self}, \\ & = & (s_i, e_i + e) & \text{, otherwise.} \end{array}$$

An incoming external event is routed, with an adjustment for elapsed time, to each of the components connected to the coupled model input (after the appropriate input-to-input translation). For all those components, the elapsed time is reset to 0. All other components are not affected and only the elapsed time is adjusted.

Some limitations of DEVS are that

- a conflict due to simultaneous internal and external events is resolved by ignoring the internal event. It should be possible to explicitly specify behaviour in case of conflicts;
- there is limited potential for parallel implementation;
- the *select* function is an artificial legacy of the semantics of traditional sequential simulators based on an event list;
- it is not possible to describe variable structure.

Some of these are compensated for in parallel DEVS ([Cho96]).

1.4 Implementation of a DEVS Solver

The algorithm in Figure 3 is based on the closure under coupling construction and can be used as a specification of a –possibly parallel– implementation of a DEVS solver or "abstract simulator" [Zei84a, KSKP96]. In an atomic DEVS solver, the last event time t_L as well as the local state s are kept. In a coordinator, only the last event time t_L is kept. The next-event-time t_N is sent as output of either solver. It is possible to also keep t_N in the solvers. This requires consistent (recursive) initialization of the t_Ns. If kept, the t_N allows one to check whether the solvers are appropriately synchronized. The operation of an abstract simulator involves handling four types of messages. The $(x, from, t)$ message carries external input information. The $(y, from, t)$ message carries external output information. The $(*, from, t)$ and $(done, from, t_N)$ messages are used for scheduling (synchronizing) the abstract simulators. In these messages, t is the simulation time and t_N is the next-event-time. The $(*, from, t)$ message indicates an internal event $*$ is due.

When a coordinator receives a $(*, from, t)$ message, it selects an imminent component i^* by means of the tie-breaking function *select* specified for the coupled model and routes the message to i^*. Selection is necessary as there may be more than one imminent component (with minimum next remaining time).

When an atomic simulator receives a $(*, from, t)$ message, it generates an output message $(y, from, t)$ based on the old state s. It then computes the new state by means of the internal transition function. Note how in DEVS, output messages are only produced while executing internal events. When a simulator outputs a $(y, from, t)$ message, it is sent to its parent coordinator. The coordinator sends the output, after appropriate output-to-input translation, to each of the influencees of i^* (if any). If the coupled model itself is an influencee of i^*, the output, after appropriate output-to-output translation, is sent to the the coupled model's parent coordinator.

message m	**simulator**	**coordinator**
$(*, from, t)$	simulator correct only if $t = t_N$	

	$y \leftarrow \lambda(s)$
	if $y \neq \phi$:
	  send $(\lambda(s), self, t)$ to *parent*
	$s \leftarrow \delta_{int}(s)$
	$t_L \leftarrow t$
	$t_N \leftarrow t_L + ta(s)$
	send $(done, self, t_N)$ to *parent*
		send $(*, self, t)$ to i^*, where
		  $i^* = select(imm\_children)$
		  $imm\_children = \{i \in D \mid M_i.t_N = t\}$
		$active\_children \leftarrow active\_children \cup \{i^*\}$

$(x, from, t)$	simulator correct only if $t_L \leq t \leq t_N$ (ignore δ_{int} to resolve a $t = t_N$ *conflict*)	

	$e \leftarrow t - t_L$
	$s \leftarrow \delta_{ext}(s, e, x)$
	$t_L \leftarrow t$
	$t_N \leftarrow t_L + ta(s)$
	send $(done, self, t_N)$ to *parent*
		$\forall i \in I_{self}$:
		  send $(Z_{self,i}(x), self, t)$ to i
		  $active\_children \leftarrow active\_children \cup \{i\}$

$(y, from, t)$		$\forall i \in I_{from} \setminus \{self\}$:
		  send $(Z_{from,i}(y), from, t)$ to i
		  $active\_children \leftarrow active\_children \cup \{i\}$
		if $self \in I_{from}$:
		  send $(Z_{from,self}(y), self, t)$ to *parent*

$(done, from, t)$		$active\_children \leftarrow active\_children \setminus \{from\}$
		if $active\_children = \emptyset$:
		  $t_L \leftarrow t$
		  $t_N \leftarrow min\{M_i.t_N \mid i \in D\}$
		  send $(done, self, t_N)$ to *parent*

Figure 3: DEVS Simulation Procedure

$t \leftarrow t_N$ of topmost coordinator
repeat until $t \geq t_{end}$ (or some other termination condition)
  send $(*, main, t)$ to topmost coupled model top
  wait for $(done, top, t_N)$
  $t \leftarrow t_N$

Figure 4: DEVS Simulation Procedure Main Loop

When a coordinator receives an $(x, from, t)$ message from its parent coordinator, it routes the message, after appropriate input-to-input translation, to each of the affected components.
When an atomic simulator receives an $(x, from, t)$ message, it executes the external transition function of its associated atomic model.
After processing an $(x, from, t)$ or $(y, from, t)$ message, a simulator sends a $(done, from, t_N)$ message to its parent coordinator to prepare a new schedule. When a coordinator has received $(done, from, t_N)$ messages from all its components, it sets its next-event-time t_N to the minimum t_N of all its components and sends a $(done, from, t_N)$ message to its parent coordinator. This process is recursively applied until the top-level coordinator or *root coordinator* receives a $(done, from, t_N)$ message.

As the simulation procedure is synchronous, it does not support a-synchronously arriving (real-time) external input. Rather, the environment or Experimental Frame should also be modelled as a DEVS component.

To run a simulation experiment, the *initial conditions* t_L and s must first be set in *all* simulators of the hierarchy. If t_N is kept in the simulators, it must be recursively set too. Once the initial conditions are set, the main loop described in Figure 4 is executed.

References

[Cho96] A.C.-H. Chow. Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator. *Transactions of the Society for Computer Simulation International*, 13(2):55–68, June 1996.

[KSKP96] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed simulation of hierarchical DEVS models: Hierarchical scheduling locally and time warp globally. *Transactions of the Society for Computer Simulation International*, 13(3):135–154, September 1996.

[Nan81] Richard E. Nance. The time and state relationships in simulation modeling. *Communications of the ACM*, 24(4):173–179, April 1981.

[RS94] Ashvin Radiya and Robert G. Sargent. A logic-based foundation of discrete event modeling and simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(1):3–51, 1994.

[Zei84a] Bernard P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, London, 1984.

[Zei84b] Bernard P. Zeigler. *Theory of Modelling and Simulation*. Robert E. Krieger, Malabar, Florida, 1984.