AVOIDING SERIALIZATION IN TIMED CELL-DEVS

Gabriel A. Wainer * §
gabrielw@dc.uba.ar

Norbert Giambiasi §
Norbert.Giambiasi@iuspim.u-3mrs.fr

§ DIAM-IUSPIM
Université d'Aix-Marseille III
Av. Escadrille Normandie Niemen
13397 Marseilles Cédex 20 – FRANCE

** Departamento de Computación*
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Pabellón I – Ciudad Universitaria
Buenos Aires (1428) – ARGENTINA

Keywords: Modeling methodology: DEVS models, cellular automata, Cell-DEVS models, Simulation methods: Discrete-event simulation.

ABSTRACT

The Timed Cell-DEVS paradigm allows the specification of executable cell spaces with timing delays. The use of this formalism can lead to serialization and incorrect execution when models are considered to be executing in parallel. This work is devoted to present an extension that allows the specification of these models in parallel. Furthermore, a specification for timing delays used in these models is presented, and the behavior for those delays is exemplified with detail. This approach allows the easy definition of complex behavior in physical systems, which can be validated formally.

INTRODUCTION

In (Wainer and Giambiasi 1998), the Timed Cell-DEVS paradigm was defined as an extension to the DEVS (Zeigler 1976) and Cellular Automata (Wolfram 1986) formalisms. The paradigm allows to define a cell space where each cell is defined as a DEVS atomic model, and a procedure to couple cells is depicted. The goal is to improve precision and to reduce the computation time using a continuous time base.

Each cell also can include a delay construction (Giambiasi and Miara 1976), allowing the definition of complex behavior, thus improving the definition for each of the submodels. Two kinds of delays can be defined: transport and inertial.

The paradigm included binary or three-state values for each of the cells in the space, and only allowed two-dimensional models. This work is devoted to present an extension to a more general formalism that allows to model n-dimensional spaces with generic state sets. A motivation for parallel definition is included, and models that can run in parallel environments is formally presented.

The work is organized as follows: the following section will provide some background on the DEVS and Cell-DEVS formalisms. After, serialization problems related with Cell-DEVS models are presented. Then, a definition for parallel Cell-DEVS models is introduced. Finally, different behavior related with the delays for each cell is depicted.

BACKGROUND

Zeigler's paradigm (Zeigler 1976) formally specify discrete events systems using a modular description. The use of such a hierarchical approach attacks the quantitative complexity of the problems. A model is seen as composed by behavioral (atomic) submodels than can be combined into structural (coupled) models. As the formalism is closed under closure, coupled models can be seen as new base models that can integrated hierarchically. This strategy allows the reuse of tested models, improving the safety of the simulations and allowing the reduction of the development times. Recalling Zeigler's definitions, a DEVS atomic model can be formally described as:

$$M = < I, X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

where

$\boldsymbol{I} = < \mu, P^X, P^Y >$ is the model's modular interface. Here, $\mu \in \boldsymbol{N}$, $\mu < \infty$ is the number of input/output ports, and $\forall\, j \in [1, \eta]$, $i \in \{X, Y\}$, P_j^i is the definition of a port (input or output respectively), defined as $P_j^i = \{ (N_j^i, T_j^i) \,/\, \forall\, j \in [1, \eta+\mu],\ N_j^i \in [i_1, i_{\eta+\mu}]$ (port name), and $T_j^i \in I_i$ (port type)$\}$, where $I_i = \{x \,/ x \in \boldsymbol{X}$ if $i = X \}$ or $I_i = \{x \,/\, x \in \boldsymbol{Y}$ if $i = Y \}$;

$\boldsymbol{X}$ is the input events set;

S is the state set;

$\boldsymbol{Y}$ is the output events set;

$\boldsymbol{\delta_{int}}$: $S \rightarrow S$, is the internal transition function;

$\boldsymbol{\delta_{ext}}$: $Q \times X \rightarrow S$, is the external transition function; where

$$Q = \{ (s, e) \,/\, s \in S, \text{ and } e \in [0, D(s)]\};$$

λ: $S \rightarrow Y$, is the output function; and

D: $S \rightarrow \boldsymbol{R}_0^+ \cup \infty$, is the elapsed time function.

An atomic model can be integrated with other DEVS models to

This work was partially supported by the UBACYT research project TX-004, “Concurrency in distributed systems”.

build a structural model. These models are called coupled, and are composed by base models, e.g., atomic or coupled ones. DEVS coupled models are formally defined as:

$$CM = < I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select >$$

where

I = < μ, P^X, P^Y > is the model's modular interface defined like in the atomic models;

X is the set of input events;

Y is the set of output events;

D $\in$ N, D $<\infty$ is an index for the components of the coupled model, and $\forall$ i $\in$ D, $\mathbf{M_i}$ is a basic DEVS model, where

$$M_i = < I_i, X_i, S_i, Y_i, \delta_{inti}, \delta_{exti}, ta_i >$$

$\mathbf{I_i}$ is the set of influencees of model i, and $\forall$ j $\in$ I_i;

$\mathbf{Z_{ij}}$: $Y_i \rightarrow X_j$ is the i to j translation function; and

select is the tie-breaking selector.

Cellular Automata are discrete-time discrete models described as cells organized in n-dimensional infinite lattices. Each cell in the automaton has a discrete value that is updated by a local computation function. This function uses the present value for the cell and a finite set of neighbors to compute the new state. The use of discrete time poses constraints in the precision and execution performance of these complex models. To achieve the desired accuracy, smaller time slots must be used, producing higher needs of processing time.

To avoid these problems, the Timed Cell-DEVS paradigm was introduced (Wainer and Giambiasi 1998; Wainer 1998). This formalism allows to define a cell space where each cell is defined as a DEVS atomic model, and a procedure to couple cells is depicted.

Recalling the definitions seen in (Wainer 1998), Cell-DEVS atomic models can be formally specified as:

$$TDC = < X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

X is the set of external input events;
Y is the set of external output events;
I represents the model's modular interface;
S is the set of sequential states for the cell;
$\boldsymbol{\theta}$ is the cell state definition;
N is the set of states for the input events;
d is the transport delay for the cell;
$\boldsymbol{\delta_{int}}$ is the internal transition function;
$\boldsymbol{\delta_{ext}}$ is the external transition function;
$\boldsymbol{\tau}$ is the local computation function;
$\boldsymbol{\lambda}$ is the output function; and
D is the state's duration function.

A Cell-DEVS coupled model is defined by:

$$GCC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z, select >$$

Ylist is the output coupling list;
Xlist is the input coupling list;
I represents the definition of the interface for the modular model;
X is the set of external input events;
Y is the set of external output events;
n is the dimension of the cell space;
$\{\mathbf{t_1,...,t_n}\}$ is the number of cells in each of the dimensions;
N is the neighborhood set;
C is the cell space;
B is the set of border cells;
Z is the translation function; and
select is the tie-breaking function for simultaneous events.

The definition for DEVS coupled models was extended to include Cell-DEVS submodels. Therefore, a DEVS coupled model can be defined as:

$$CM = < I, X, Y, D, \{M_d\}, \{I_d\}, \{Z_{dj}\}, select >$$

***I*, *X*, *Y*, D, $\mathbf{I_d}$** and **select** are defined as in the previous section, and:

$\mathbf{M_d}$ is a DEVS basic model $\forall$ d $\in$ D, where

$$M_d = GCC_d = < Xlist_d, Ylist_d, I_d, X_d, Y_d, n_d, \{t_1,...,t_n\}_d, N_d, C_d, B_d, Z_d, select_d >$$

if the coupled model is Cell-DEVS, and

$$M_d = < I_d, X_d, Y_d, S_d, \delta_{int}d, \delta_{ext}d, D_d >$$

otherwise.

$\mathbf{Z_{dj}}$ is the translation function from d to j, where
Z_{dj}: $Y_d \rightarrow X_j$ if none of the implied models is Cell-DEVS, or

Z_{dj}: $Y(c_1)_d \rightarrow X(c_2)_j$, with $(c_1) \in Ylist_d$, and $(c_2) \in Xlist_j$ if any of the models d or j is a GCC.

The use of multiple processors is useful for the simulation of Cell-DEVS models, due to the high degree of compute time involved. If the models execute sequentially, to obtain a meaningful sample to study the desired problem is a time consuming process. Besides, inherently parallel models should execute serially.

The most natural solution would be to execute each cell in a different processor. Unfortunately, this approach is not feasible due to practical reasons. Even with massively parallel processors, the granularity of the problem makes impossible the implementation. Instead, the cell space can be divided into several subspaces executing in different processors. These definitions have several problems when the cells in the space are considered as executing in parallel, as will be presented in the following section.

SERIALIZATION PROBLEMS IN CELL-DEVS

As stated in (Chow and Zeigler 1994a), if we call **e** to the elapsed time since the occurrence of an event, a model can exist in the DEVS structure at either e=0 or e=D(s). In the case of coupled models, the modeler can use the *select* function to solve the conflicts of simultaneous scheduled events. The case is different for basic models: once they are coupled, ambiguity arises when a model scheduled for an internal transition receives an event. The problem here is how to determine which of both elapsed times should be used. The *select* function solves the ambiguity by choosing only one of the imminent models. This is a source of potential errors, because the serialization may not reflect the simultaneous occurrence of events. Moreover, the serialization reduces the possible exploitation of parallelism among concurrent events.

It was discussed that the models should accomplish the following properties:

- **Collision handling**: the behavior of a collision must be controllable by the modeler.
- **Parallelism**: the formalism must not use any serialization function that prohibits possible concurrencies.
- **Uniformity**: the hierarchical construction must have uniform behavior: different hierarchical constructs of the same model must display the same behavior.

These properties resulted into the definition of Parallel DEVS (Chow and Zeigler 1994b). In this approach, the *select* function was eliminated, and a new transition function was created to manage the collisions. This function, called (δ_{con}, the confluent transition function), should be defined by the modeler. Its goal is to define the model's behavior when a model receives external events at the time of its internal transition (e = D(s) or e = 0).

If no external transition occurs, a scheduled internal transition function is carried out. However, if both collide, the confluent transition function is activated. The values for simultaneous events generated before each internal function should be kept together. Therefore, the inputs for each model are collected into a bag (multiset).

In the timed Cell-DEVS formalism, the desired **uniformity** was addressed in a different way. In this case, there was no need to include a bag construction, due to the definition of the interfaces of the models. This interface is defined such that, in each model, one event per port is allowed at a time. Each model is connected with the others using a unique port, therefore, it cannot ever occur to have more than one input per port. This affirmation is valid only if the cell's delays have non-zero values. These assumptions were done considering that the modeled real systems never have delays or activation of exactly zero time units.

As a second problem for the defined Cell-DEVS models, the desired behavior for **parallelism** was not achieved. The *select* function was used to choose between different models in the case of external simultaneous events. Besides, if an event occurs simultaneously with an internal scheduled event, the behavior of the E-DEVS formalism (Wang and Zeigler 1993) was applied: the internal transition function is activated prior to the external transition.

The main problem with the previous formalism is related with the **collision handling**. In most cases, the collision is controlled because the local computing function uses the values obtained through each input port, and it has been shown that they cannot have more than one value. Instead, if zero-time delays or simultaneous events are considered, the user cannot manage the model behavior.

In addition, the Cell-DEVS models can be coupled with traditional DEVS submodels. Even these models have been redefined to have a uniform interface, a bag is needed for the cases of zero-time transitions. Considering these factors, the Cell-DEVS models were redefined to include parallel behavior.

PARALLEL ATOMIC CELL-DEVS MODELS

This section defines a new approach extending the concepts of confluent transition function and introducing a bag for each model. Other extensions that have also been included allowing the management of complex delay behavior. First, the definition of atomic models has been included. After, coupled models are defined.

The behavior for the cell delays presented in (Wainer and Giambiasi 1998) has been extended to allow mixed behavior and general cell states. Therefore, in the present definition, the semantics of these constructors have been modified. Originally, each cell had a fixed delay, but in this case, it is considered that each rule of the local transition function can activate an inertial or transport delay. Therefore, the local transition function should return the kind of delay that is wanted, and the outputs for the cell should be delayed accordingly. An informal description for the cell can be seen in the following figure.

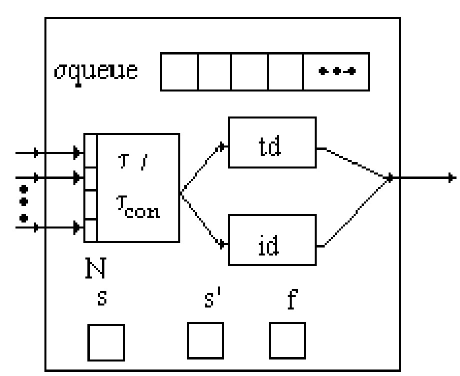

Figure 1. Cell's definition.

Each cell receives a set of input values that are used to compute the local transition function. The results of the computations are delayed (using transport or inertial delays). This cell can be formally specified as:

$$TDC = < X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \delta_{con}, \tau, \tau_{con}, \lambda, D >$$

In this case, $\#T < \infty \ \wedge \ T \in \{\boldsymbol{N}, \boldsymbol{Z}, \boldsymbol{R}, \boldsymbol{\{0,1\}}\} \cup \{\phi\}$;

$\boldsymbol{X} \subseteq T$;
$\boldsymbol{Y} \subseteq T$;
$\boldsymbol{I} = < \eta, \mu, P^X, P^Y >$. Here,
$\eta \in \boldsymbol{N}, \eta < \infty$ is the neighborhood's size,
$\mu \in \boldsymbol{N}, \mu < \infty$ is the number of other input/output ports, and
$\forall j \in [1, \eta], i \in \{X, Y\}, P_j^i$ is a definition of a port (input or output respectively), with $P_j^i = \{ (N_j^i, T_j^i) / \ \forall j \in [1, \eta+\mu]$, $N_j^i \in [i_1, i_{\eta+\mu}]$ (port name), y $T_j^i \in I_i$ (port type)}, where
$I_i = \{ x / x \in \boldsymbol{X}$ if X } or $I_i = \{ x / x \in \boldsymbol{Y}$ if i = Y } ;

$\mathbf{S} \subseteq T$;

$\boldsymbol{\theta} = \{$ (s, phase, σqueue, f, σ) /
$s \in S$ is the status value for the cell,
phase $\in$ {active, passive},
$\sigma queue = \{ ((v_1,\sigma_1),...,(v_m,\sigma_m)) / m \in \boldsymbol{N} \wedge m < \infty) \wedge \forall (i \in \boldsymbol{N}, i \in [1,m]), v_i \in S \wedge \sigma_i \in \boldsymbol{R}_0^+ \cup \infty \}$;
$f \in T$; and
$\sigma \in \boldsymbol{R}_0^+ \cup \infty$
} ;
$\mathbf{N} \in S^{\eta+\mu}$;
$\mathbf{d} \in \boldsymbol{R}_0^+, d < \infty$;
$\boldsymbol{\delta_{int}}: \theta \to S$;
$\boldsymbol{\delta_{ext}}: QxX^b \to \theta, Q = \{ (s, e) / s \in \theta \text{ x N x d}; e \in [0, D(s)]\}$;
$\boldsymbol{\delta_{con}}: \theta x X^b \to S$;
$\boldsymbol{\tau}: N \to S$ x {inertial, transport} x d;
$\boldsymbol{\tau_{con}}: X^b xN \to S$ x {inertial, transport} x d;
$\boldsymbol{\lambda}: S \to Y^b$; and
$\mathbf{D}: \theta \text{ x N x d} \to \boldsymbol{R}_0^+ \cup \infty$.

Each of the components in the definition has the same meaning than the presented in the Background section. Two new functions have been added: $\boldsymbol{\delta_{con}}$ and $\boldsymbol{\tau_{con}}$. In addition, the external transition and output functions have been changed to work with input/output bags (X^b and Y^b). The confluent transition function is activated when there are collisions between internal and external events. Its main task is to activate the confluent local transition function, which analyzes the present values for the input bags, and updates the present input values for the cell. In this way, the cell will compute the next state using the values chosen by the modeler.

A parallel Cell-DEVS coupled model can be represented as:

$$GCC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z >$$

Here,

Ylist is the output coupling list;
Xlist is the input coupling list;
I represents the definition of the interface for the modular model;
X is the set of external input events;
Y is the set of external output events;
n is the dimension of the cell space;
$\{\mathbf{t_1,...,t_n}\}$ is the number of cells in each of the dimensions;
N is the neighborhood set;
C is the cell space, with $\mathbf{C} = \{ C_c / c \in \mathbf{I} \wedge C_c = < I_c, X_c, Y_c, S_c, N_c, d_c, \delta_{int_c}, \delta_{ext_c}, \delta_{con_c}, \tau_c, \tau_{con_c} \lambda_c, D_c > \}$,where C_c is a parallel Cell-DEVS atomic model, and $I = \{ (i_1,...,i_n) / (i_k \in N \wedge i_k \in [1, t_k]) \ \forall k \in [1, n]\}$;
B is the set of border cells; and
Z is the translation function.

The detailed definitions for these sets can be seen in (Wainer 1999), and the meaning of the components is the same. The main difference is that each cell in the space is a parallel Cell-DEVS atomic cell that uses the δ_{con} and τ_{con} functions to avoid collisions. Therefore, the select function appearing in the original definitions has been eliminated.

DEVS coupled models have been redefined to include base models that can be seen as cell spaces. Therefore, a coupled DEVS model will be defined as:

$$CM = < I, X, Y, D, \{M_d\}, \{I_d\}, \{Z_{dj}\} >$$

Here, ***I*, *X*, *Y*, D** and $\mathbf{I_d}$ are defined as in the Background section. Instead, the **select** function has been eliminated, and:

$\mathbf{M_d}$ is a DEVS basic model $\forall d \in D \cup \{self\}$, where

$M_d = GCC_d = < I_d, X_d, Y_d, Xlist_d, Ylist_d, n_d, \{t_1,...,t_n\}_d, N_d, C_d, B_d, Z_d >$

if the coupled model is Cell-DEVS, and

$M_d = < I_d, X_d, Y_d, S_d, \delta_{int_d}, \delta_{ext_d}, \delta_{con_d}, D_d >$

otherwise.

$\mathbf{Z_{dj}}$ is the translation function from d to j, where
$Z_{self\,j}: Y_{self} \to X_j$ if none of the implied models is Cell-DEVS, or
$Z_{self\,j}: Y(c_1)_{self} \to X(c_2)_j$, with $(c_1) \in Ylist_d$, and $(c_2) \in Xlist_j$ if any of the models self or j is a GCC;

$Z_{d\,self}: Y_d \to X_{self}$ if none of the implied models is Cell-DEVS, or
$Z_{d\,self}: Y(c_1)_d \to X(c_2)_{self}$, with $(c_1) \in Ylist_d$, and $(c_2) \in Xlist_{self}$ if any of the models d or self is a GCC;

$Z_{dj}: Y_d \to X_j$ if none of the implied models is Cell-DEVS, or
$Z_{dj}: Y(c_1)_d \to X(c_2)_j$, with $(c_1) \in Ylist_d$, and $(c_2) \in Xlist_j$ if any of the models d or j is a GCC.

In (Wainer 1999) it has been shown the equivalence between the parallel Cell-DEVS models and parallel DEVS models. In addition, the closure under coupling has been proved, showing that the models can be integrated into a DEVS hierarchy. A simulation mechanism related with the parallel models has also been included. Finally, a mapping between a Parallel Cell-DEVS simulation environment and a parallel synchronization mechanism was defined, providing the basis for the construction of simulation tools using this paradigm.

CELL'S DELAY BEHAVIOR

The behavior for the cell delays was presented in (Wainer and Giambiasi 1998 and Wainer 1998). In the previous section, the new formal specification for parallel atomic cell has been shown. The semantics for these constructors has been modified. Originally, each cell had a fixed delay, but in this case, it is considered that each rule of the local transition function can activate an inertial or transport delay. The semantics for the transition functions can be defined as follows:

δ_{int}:

$$\frac{\sigma = 0; \quad \sigma queue \neq \{\emptyset\}; \quad phase = active}{\forall\, i \in [1, m],\ a_i \in \sigma queue,\ a_i.\sigma = a_i.\sigma - head(\sigma queue.\sigma);\ \ \sigma queue = tail(\sigma queue);\ \ s = head(\sigma queue.v);\ \ \sigma = head(\sigma queue.\sigma);}$$

$$\frac{\sigma = 0; \quad \sigma queue = \{\emptyset\}; \quad phase = active}{\sigma = \infty \quad \wedge \quad phase = passive}$$

λ:

$$\frac{\sigma = 0;}{out = s;}$$

δ_{ext}:

$$\frac{(s', transport) = \tau(N_c); \sigma \neq 0; e = D(\theta \times N \times d);\ phase = active;}{s \neq s' \Rightarrow (s = s' \ \wedge\ \forall\, i \in [1,m]\ a_i \in \sigma queue,\ a_i.\sigma = a_i.\sigma - e \ \wedge\ \sigma = \sigma - e;\ add(\sigma queue, <s', d>) \wedge f = s\ )}$$

$$\frac{(s', transport) = \tau(N_c);\ \sigma \neq 0;\ e = D(\theta \times N \times d);\ phase = passive;}{s \neq s' \Rightarrow (\ s = s' \ \wedge\ \sigma = d \ \wedge\ phase = active \ \wedge\ add(\sigma queue, <s', d>) \ \wedge f = s\ )}$$

$$\frac{(s', inertial) = \tau(N_c);\ \sigma \neq 0;\ e = D(\theta \times N \times d);\ phase = passive;}{s \neq s' \Rightarrow (\ s = s' \ \wedge\ phase = active \ \wedge\ \sigma = d \ \wedge\ f = s\ )}$$

$$\frac{(s', inertial) = \tau(N_c);\ \sigma \neq 0;\ e = D(\theta \times N \times d);\ phase = active;}{s \neq s' \Rightarrow s = s' \ \wedge\ (f \neq s' \Rightarrow \sigma queue = \{\emptyset\} \wedge \sigma = d \ \wedge\ f = s)}$$

δ_{con}:

$$\frac{N_c; \quad X^b; \quad e = 0 \ \vee\ e = D(\theta \times N \times d);}{N_c = \tau_{con}(X^b); \quad \sigma = 0; \quad X^b = X^b - \{X / e = 0\};}$$

In this case, tail/head/add represent the methods used to manage the elements of a list. As it can be seen, the external transition function activates the local computation, whose result is delayed using one of both kinds of constructions. The output function executes prior to the internal transition function, transmitting the present values to other models. The δ_{int} function is in charge of keeping the values for a transport delay.

The confluent transition function chooses members from the bag, and updates the inputs for the cell, deleting the unnecessary members in the bag. As $\sigma = 0$, an internal transition function is scheduled immediately. The modeler should define the behavior for the τ_{con} function in each cell, thus allowing the definition for this behavior under collisions. Each cell can use an inertial, transport or combined delays, whose behavior will be exemplified following.

The behavior of transport delays allows reflecting the straightforward propagation of signals over lines of infinite bandwidth (anticipatory semantics). They allow the modeling of variable commutation time for each cell with anticipatory semantics (every scheduled event will be executed).

Let us consider a transport delay of 17 time units for a given cell. The input trajectories depicted in the Figure 2 are used to show the behavior for the transport delays, considering the formal definition of the semantics. The cell's behavior for those input trajectories using the semantics defined in the previous section is analyzed in Table 1. This table shows each transition, its activation time and the cell's state values. The * sign identifies the execution of internal transition functions, while the remaining lines represent the execution of external transitions (the fields containing two values separated by a slash represent the variable values before and after execution).

	t	S	s'	phase	σ	e	σqueue
	...	0	0	Passive			
	30	0/1	1	Active	17	0	(1,17)
	40	1/0	0	Active	7	10	(1,7),(0,17)
*	47	0	0	Active	0 / 10	17 / 0	(0,10)
	55	0/1	1	Active	2	8	(0,2), (1,17)
*	57	1	1	Active	0 / 15	10 / 0	(1,15)
	60	1/0	1/0	Active	12	3	(1,12), (0,17)
*	72	0	0	Active	0 / 5	15	(0,5)
*	77	0	0	Passive	∞	5	

Table 1. Execution sequence of a transport delay cellular model.

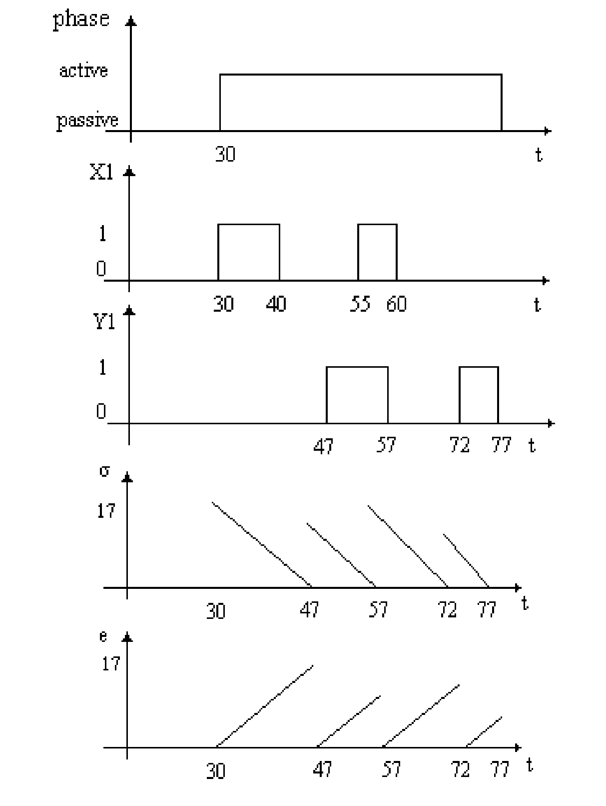

Figure 2. Input and output trajectories with transport delay.

It can be seen that the results are delayed for 17 time units, and the cell remains active while there are values queued waiting to be output.

Inertial delays use a preemptive semantics to represent that to change a state some quantity of energy should be provided to the system. Therefore, some scheduled events are not executed due to a too small interval between two input events. This kind of delay allows the analysis of the frequency limit response of systems. The inertial delay construction is useful to represent certain physical phenomena in which certain components change their internal state depending on the input tension values and their duration. Hence, a simulation paradigm providing the ability to represent such kind of behavior could help to specify such models easily.

Apart from the circuit modeling domain, the construction can be useful to model other spatial physical models. Let us consider a model to study fire in a wood. In this case, fire can be represented as a value for a cell. The value must be transmitted to the neighboring cells, but it will influence them only if the value is kept during the inertial delay. Instead, if it does not sustain for a certain time, it should not expand (allowing representing influences of the wind, rain, or firefighters).

The behavior for atomic cells with inertial delays can be studied in the following example. The input and output trajectories for the delay presented in Figure 3 have an inertial delay of 5 time units.

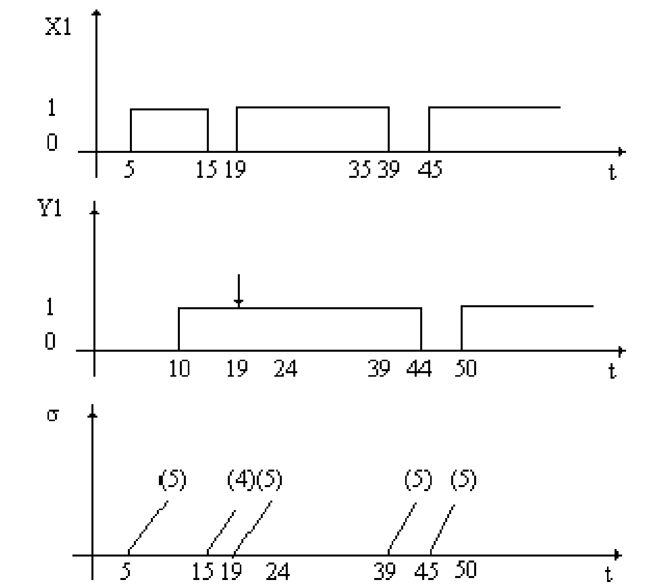

Figure 3. Input and output trajectories with inertial delays.

In the following table, the execution flow of the transition functions can be easily analyzed. There, the representation is the same that was previously used in Table 1. The lines marked with the "!" signs (arrows in the figure) represent the behavior of the model under preemption.

	T	S	s'	Phase	σ	e	x	f
	...	0	0	Passive	∞			
	5	0/1	1	Active	5	0	1	1
*	10	1	1	Passive	0/∞	5		
	15	1/0	0	Active	5	0	0	0
!	19	0/1	1	Active	1/5	4/0	1	0/1
*	24	1	1	Passive	0/∞			
	39	1/0	0	Active	5	0	0	1/0
*	44	0	0	Passive	0/∞			
	45	0	0	Active	5	0	1	0/1
*	50	1	1	Passive	0/∞			

Table 2 - Execution sequence of Figure 3.

The definitions presented in the previous section allow the inclusion of a combination between transport and inertial delays. The behavior for each of them is the same that the defined originally. That is, if a transport delay is activated, the value is output after the delay (the σqueue is used to keep the value). Instead, an inertial delay is used to output the value if it is kept during the delay. If an inertial delay is activated when several values are waiting for the transport delay, they are preempted.

The following figure shows an example of execution for an atomic cell with different delays. The local transition function uses different delays depending the moment when the transition function is executed. In this case, it will be supposed that the result will be delayed using a transport delay of 17 time units. Between the times 50 and 60, and from 90 to 100, an inertial delay of 6 time units will be used. The values obtained from 60 to 70 will be delayed using an inertial delay of 9 time units. Finally, under collisions, the identity function is executed and after a transport delay of 25 time units is applied. In the following figure it can be seen the behavior for the cell under different inputs.

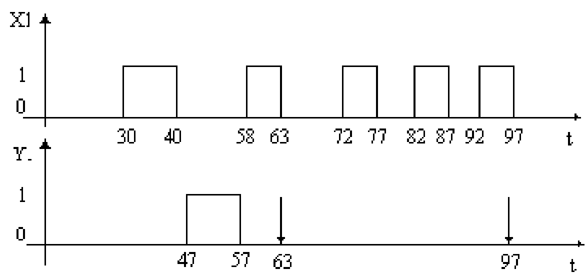

Figure 4. Execution flow for the previous example.

The execution details are presented in the following Table 3. Each line shows the state for the cell (defining present and future values in most cases). There, the lines marked with a "*" symbol represent the execution of the internal transition functions. The lines marked with a "!" symbol (arrows in the figure) represent preemption.

	T	s	s'	p	σ	e	f	d	σqueue
	...	0	0	P					
	30	0/1	1	A	17	0	1	tr.	(1,17)
	40	1/0	0	A	7	10	0	tr.	(1,7), (0,17)
*	47	0	0	A	0/10	17/0	0		(0,10)
*	57	1	1	P	0/∞	10/0	0		
	58	0/1	1	A	6	0	1	in.	
!	63	1/0	0	A	1/9	5	1/0	in.	
*	72	0	0	P	0/∞	0	0		
	72	0/1	1	A	25	0	1	tr.	(1,25)
	77	1/0	0	A	20	5	0	tr.	(1,20), (0,25)
	82	0/1	1	A	15	5	1	tr.	(1,15),(0,20), (1,25)
	87	1/0	0	A	10	5	0	tr.	(1,10),(0,15), (1,20),(0,25)
	92	0/1	1	A	5/6	5/0	1	in.	
!	97	1/0	0	A	1/6	4	1/0	in.	
*	103	0/0	0	P	∞	7	0		

Table 3. Execution sequence of a transport delay cellular model.

The cell receives an external transition in the instants 30 and 40, that are delayed for 17 time units. The values of σqueue are consumed, and the value is transmitted when the transport delay has been consumed. In the instant 58, the output is stopped during an inertial delay of 6 time units. As the value is not kept during the delay, it is preempted. In addition, it can be seen a collision in the instant 72. In this case, the confluent transition function is executed, activating the local transition function and establishing a transport delay of 25 time units.

Finally, in the simulated time 92, the inertial delay is activated prior the output of the previous transport delays. Therefore, the σqueue is emptied. Besides, as the value is not kept during 6 time units, preemption occurs in the instant 97.

CONCLUSION

This work was devoted to present an extension of the Cell-DEVS paradigm, allowing the parallel execution of the models. To do so, the behavior under collisions must be defined accurately. In this case, the user will be in charge to define this behavior.

The formal specification of the delays for Cell-DEVS models was presented, in such a way that the modeler could model complex behavior using simple constructions. These constructions are useful in different domains, including the digital circuit design, prediction of the behavior in ecological systems, analysis of traffic in urban populations, etc.

The formalism entitles the definition of complex cell-shaped models using a high-level specification language. In this way, the construction of the simulators is improved, enhancing their safety and development costs. Besides, the parallel execution allows performance improvements without adding extra costs in development or maintenance.

The original definition for Cell-DEVS was extended to improve the modeling. Parallel DEVS models were considered and combined with the Cell-DEVS definition. Combined delay behavior was allowed, depending on the rules executed by each cell.

The combination of both behaviors can improve the definition of these complex models. The use of a formal specification based on the DEVS formalism improves the validation of the specifications. An accurate semantics was defined, allowing to ensure the validity of the models.

At present, a tool implementing non-parallel n-dimensional Cell-DEVS models has been built (Rodríguez and Wainer 1999). In addition, an environment for parallel asynchronous simulation has been modified, including several synchronization techniques. In this way, the simulation can obtain the maximum speed, according with the model to execute. At present, a mapping between parallel Cell-DEVS models and the parallel environment was defined, and is being under implementation.

In this way, a complex tool to run n-dimensional Cell-DEVS models with timing delays will be available. This tool will reduce development costs of the application (as was proven for the two-dimensional binary case), and efficient execution will be achieved through the use of parallelism.

REFERENCES

Chow, A. And Zeigler, B. 1994. "Revised DEVS: a parallel, hierarchical, modular modeling formalism". Technical Report. AIS-ECE. University of Arizona.

Chow, A. and Zeigler, B. 1994."Abstract Simulator for the parallel DEVS formalism". In *Proceedings of the SCS Winter Simulation Conference*.

Giambiasi, N. and Miara, A. 1976. "SILOG: A practical tool for digital logic circuit simulation". In *Proceedings of the 16th. D.A.C.*, San Diego, U.S.A.

Rodríguez, D. and Wainer, G. 1999. "New extensions to the CD++ tool". To be published in *Proceedings of the SCS Winter Simulation Conference*. Chicago, U.S.A.

Wainer, G. 1998. "Cellular models with explicit timing delays". Ph. D. Thesis. DIAM/IUSPIM. Université d'Aix-Marseille III. France.

Wainer, G. 1999. "Definition of parallel Cell-DEVS spaces". Technical Report 99-004. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

Wainer, G. and Giambiasi, N. 1998. "Specification, modeling and simulation of timed Cell-DEVS spaces". Technical Report 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Submitted for publication.

Wang, Y. and Zeigler, B. 1993. “Extending the DEVS formalism for massively parallel simulation”. *Discrete Event Dynamic Systems: Theory and Applications.* No. 3:193-218.

Wolfram, S. 1986. *Theory and applications of cellular automata*. Vol. 1, Advances Series on Complex Systems. World Scientific, Singapore.

Zeigler, B. 1976. *Theory of Modeling and Simulation*. Wiley, N.Y.

AUTHOR'S BIOGRAPHY

Gabriel A. Wainer received the Licentiate degree from the Universidad de Buenos Aires, Argentina (1993) and the Ph.D. degree (with honors) from the DIAM/IUSPIM, Université d'Aix-Marseille III, France (1998).

He is Adjunct Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina. He has been assistant teacher in the same department for 11 years. He has published more than 40 articles in the field of operating systems, real-time systems and Discrete-Events simulation. He is author of a book on real-time systems and co-author of a book on Operating Systems.

Norbert Giambiasi has received the DEUG in Physics and the Maitrise d'électronique-electrotechnique-automatique at the Université des Sciences et Techniques du Languedoc, Montpellier, France (1969). He has received the DEA degree (1972), Ph.D. (with honors, 1974) and Doctorat d'état (1980) at the same University.

At present, he is Full Professor at the DIAM/IUSPIM, Université d'Aix-Marseille III, France.

He has participated in the foundation of the EERIE in Nimes, France, and created the LERI (Research Laboratory in Informatics).

He has been scientific head of many research (ESPRIT, EUREKA, CNRS, MRE) and industrial (Bull, Dassault, Siemens, Phillips, Sollac) contracts.

He has published more than 200 articles in the simulation and automatics area, and he is co-author of a book on CIM.