Improved Cellular Models with Parallel Cell-DEVS

Gabriel A. Wainer

Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón I—Ciudad Universitaria, BuenosAires (1428), Argentina; E-mail: gabrielw@dc.uba.ar; URL: http://www.dc.uba.ar/people/proyinv/celldevs.

The Cell-DEVS paradigm allows the specification of executable cell spaces with timing delays. This approach allows easy definition of complex behavior in physical systems, which can be validated formally. The original definition of this formalism can lead to serialization and incorrect execution when the models are considered to execute in parallel. The extension presented here permits parallel specification of these models, and an associated simulation mechanism allows their execution. Cell-DEVS models include timing delay constructions, whose behavior was extended, and whose use is exemplified in detail. These new constructions improve the definition of complex timing behavior, reducing the complexity of the rules needed to represent it. In addition, neighborhood sizes can be reduced, cutting down the overhead involved, and allowing a higher number of quiescent cells in the model.

Keywords: Parallel DEVS models, DEVS paradigm, Cell-DEVS models, discrete event simulation, modelling methodologies

1. Introduction

This article presents an extension to the n-dimensional Cell-DEVS formalism [1], allowing the definition of parallel models. The formalism is derived from the binary timed Cell-DEVS [2], a combination of the DEVS paradigm [3, 4], and Cellular Automata [5].

Cellular Automata formalism is well suited to describe real systems that can be represented as cell spaces. A cellular automaton is an infinite regular n-dimensional lattice whose cells can take one finite value. The states in the lattice are updated according to a local rule in a simultaneous and synchronous way. The cell states change in discrete time steps as dictated by a local transition function using the present cell state and a finite set of nearby cells (called the neighborhood of the cell).

The DEVS (Discrete EVents Systems specification) formalism allows one to describe a real system in a modular fashion. It attacks the complexity using a hierarchical approach. A DEVS model can be described as composed of several submodels, each being behavioral (atomic) or structural (coupled). Tested models can be reused, enhancing reliability, reducing testing time and improving productivity.

Each model is described as a set consisting of a time base, inputs, states, outputs, and functions. A model uses input and output ports to communicate with the others. The internal and external events produce state changes, whose results are spread through the output ports. The influences of the ports determine if these values should be sent to other models.

Received: November 1999; Revised: January 2000; Accepted: February 2000

TRANSACTIONS of The Society for Computer Simulation International
ISSN 0740-6797/00

Volume 17, Number 2, pp. ##-##

When cellular automata are used to simulate complex systems, large amounts of computation time are required, and the use of a discrete time base poses restrictions in the precision of the model. The Timed Cell-DEVS formalism tries to solve these problems by using the DEVS paradigm to define a cell space where each cell is defined as a DEVS atomic model. The goal is to build discrete event cell spaces, improving their definition by making the timing specification more expressive.

Each cell can use one of two kinds of delay constructions [6]. The *transport* delays allow one to model a variable commute time for each cell with anticipatory semantics. Instead, *inertial* delays introduce preemptive semantics: some scheduled events are not executed due to a too-small interval between two input events [7]. The paradigm allows the inclusion of integer,

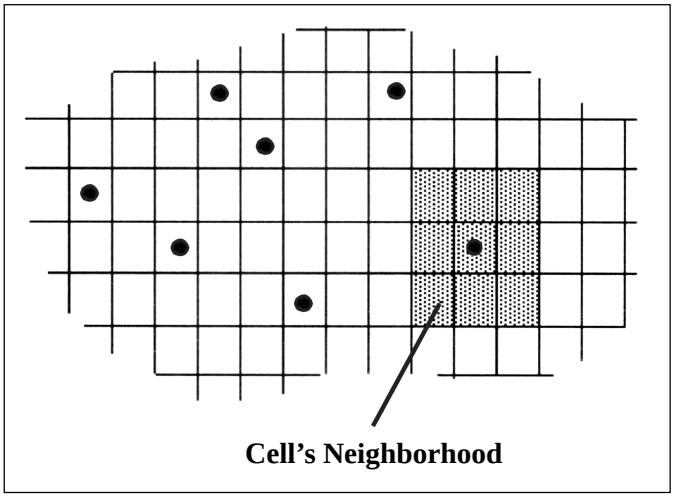

Figure 1. Sketch of a cellular automaton

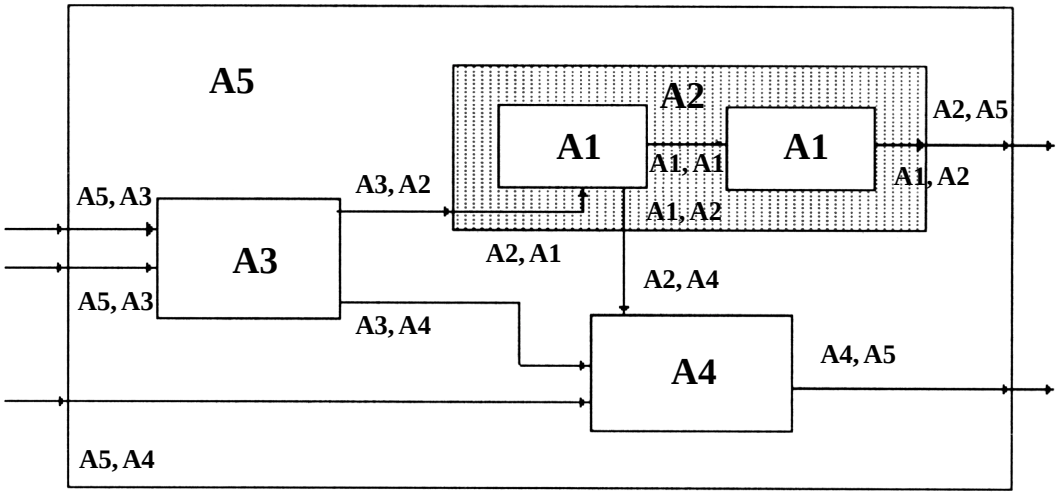

Figure 2. Coupling of DEVS models (A1, A3, A4: atomic models)

real, binary, or three-state values for each of the cells in the space, and the models can be n-dimensional. Simulation tools were built using the bidimensional paradigms [8, 9], and have been extended to n-dimensional models [10].

The simulation literature shows that the use of parallel simulation mechanisms is a promising approach to obtain results, because it allows speedups in the simulation process [11]. The provision of a meaningful sample of behavior by using sequential execution is a time consuming process. These assertions are valid for the simulation of Cell-DEVS, because they involve a high degree of computation time. Besides, cell spaces are inherently parallel, and their serial execution is too restrictive.

Unfortunately, the present definition of Cell-DEVS has several problems when the spaces are executed in parallel. The remaining sections of this work will be devoted to analyzing these problems, and to consider different solutions based on the Parallel DEVS formalism [12]. The work is organized as follows: the following section will present some serialization problems related to Cell-DEVS models. Then, a definition for parallel Cell-DEVS models is introduced. Finally, different definitions related to the timing delays used for each cell are depicted.

2. Serialization Problems in Cell-DEVS Models

Cell-DEVS has been formally specified to analyze several basic properties. In the following, a brief review of these specifications will be presented to allow a more detailed analysis in the following sections. A Cell-DEVS atomic model is defined by:

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

X is a set of external input events;
Y is a set of external output events;
I represents the model's modular interface;
S is the set of sequential states for the cell;
θ is the cell state definition;
N is the set of states for the input events;
d is the transport delay for the cell;
δ_{int} is the internal transition function;
δ_{ext} is the external transition function;
τ is the local computation function;
λ is the output function; and
D is the state's duration function.

Each cell uses a set of **N** input values to compute the future state. These values are received through a well-defined interface composed of a fixed number of ports. The cell computes a local function by using the cell's inputs and present state. A delay function can be associated with each cell, allowing one to defer the execution results of the local computing functions. To allow the deferral of the computations, a FIFO queue is used to keep track of the next events. Therefore, the outputs of a cell are not transmitted instantaneously, but after the consumption of the delay. The model advances through the activation of the internal, external, output, and state's duration functions, as in other DEVS models.

After the basic behavior for a cell is defined, the complete cell space will be constructed by building a coupled Cell-DEVS model:

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

Xlist is the input coupling list;
Ylist is the output 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;
$\{t_1,...,t_n\}$ 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.

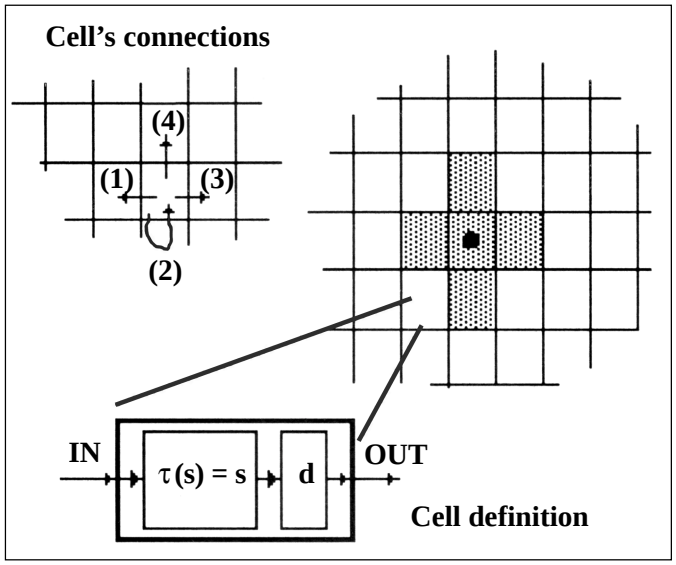

Figure 3. Informal definition of a Cell-DEVS model

The cell space defined by this specification is a coupled model composed of an array of atomic cells. Each of them is connected to the cells defined by the neighborhood. Because the cell space is finite, the borders should be provided with a different behavior than the rest of the space. Otherwise, the space is "wrapped," meaning that cells in one border are connected with those in the opposite one. Finally, the Z function allows one to define the internal and external coupling of cells in the model. This function translates the outputs of m^{th} output port in cell C_{ij} into values for the m^{th} input port of cell C_{kl}. Each output port will correspond to one neighbor and each input port will be associated with one cell in the inverse neighborhood.

As stated in [13], if we call **e** to the elapsed time since the occurrence of an event, a model can exist in the DEVS structure at e = 0 or e = D(s). In the case of coupled models, the modeler can use the **select** function to resolve the conflicts of simultaneous scheduled events. The case is different for basic models: once they are coupled, ambiguity arises when an event is received by a model scheduled for an internal transition. 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.

Chow [13] required that the following properties hold:

- **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 in the definition of Parallel DEVS [13]. 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 behavior of a model receiving external events at the time of its internal transition (e = D(s) or e = 0). A scheduled internal transition function is carried out. However, if there are colliding external and internal events, the confluent transition function is activated. The values of the events generated simultaneously before the execution of each internal function should be gathered 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 as in the Parallel DEVS paradigm, due to the definition of the cell's interfaces. They are defined such that only one input per port can be received at a time: each cell is connected with the others using a unique port, and they are not allowed to transmit two simultaneous events.

This assertion is based on the fact that each cell cannot use delays of zero time units. This assumption was made because the real systems under consideration never have delays or activation frequency of exactly zero time units. Moreover, zero-time delays can lead to non-deterministic behavior.

Lemma 1

The use of zero-time delays in Cell-DEVS models can lead to non-deterministic behavior.

Proof:

Let us suppose the proposition is false. That is, zero-time delays always lead to deterministic behavior. The following are counterexamples for this proposition. Figure 4(a) shows the original status for a subset of a cell space for the Life game [14]. The update rule for a cell in this model says that, if there are two or three living neighbors (denoted with a dot in the figure), the cell will remain alive. If fewer than two neighbors are active, the cell dies.

In this case, consider that a transport delay of zero time units is used for the cell (1,1) (being (0,0) the origin cell). Let us suppose now that the cells (0,0) and (1,1) should be activated simultaneously, and both execute in parallel. The cell (1,1) changes to 0 (it has only one living neighbor), and it sends a message to the cell (0,0), informing of the state change. When the cell (0,0) receives the message, it can treat its internal event prior to the external event. In this case, it will consider that, at

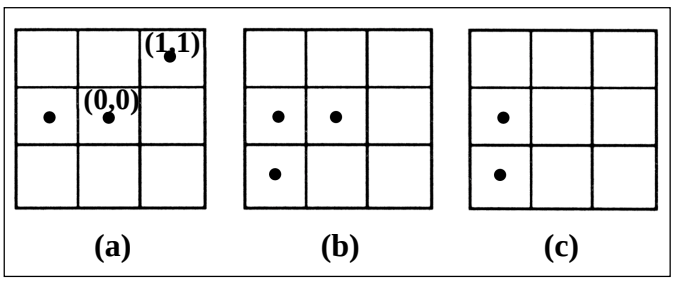

Figure 4. Execution evolution for the Life game; (a) original state; (b) results when (0,0) is activated first; (c) results when (1,1) is activated first

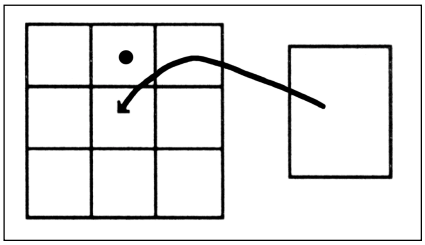

Figure 5. A Cell-DEVS model coupled with a traditional model

present, the cell has two living neighbors. Therefore, it will remain alive, and the result in Figure 4(b) will be obtained. Instead, if the external message is processed first, the cell will consider that there is only one living neighbor, and the result will be the one presented in Figure 4(c).

The case is similar when an external DEVS model produces two events with the same simulated time. The model in Figure 5 represents a section of urban traffic. The update rule says that if there is a new car arriving at a cell, and the north cell has a car, a collision occurs. Let us suppose that the external model sends the output value 1 (indicating a car's arrival). Then, another internal transition is executed in the same simulated time, and the value for the external model is 0. If the first external event is processed when it arrives, the collision status is raised for the cell. As a collision exists, the second event is ignored, and when the new external value (saying that there is no car in the external model) appears, the cell stays in collision status. Instead, if both events are treated together, the car coming from the north advances and no collision occurs.

As stated earlier, the definition for Cell-DEVS models considered that the delays should have non-zero values and that a DEVS connected as input should not activate simultaneous outputs. Nevertheless, if general models are needed, zero-time delays can lead to a non-uniform behavior. The following lemma will show that in those cases we must include an input bag for atomic Cell-DEVS.

Lemma 2

Cell-DEVS models should be built as Bag-DEVS when:

a. Cells with delays of zero-time units are used, or

b. A DEVS model connected to the cell is allowed to send two output events in the same simulated time.

Proof:

To show that the Lemma 2(a) is valid, it must be seen that only when the timing delays are zero, two different values can arrive simultaneously at a cell's port. The Cell-DEVS definition includes one input port for each connection with the other models, as defined in the model's interface, $\boldsymbol{I} = < \eta, \mu^x, \mu^y, P^x, P^y >$. Here, $\eta \in \boldsymbol{N}, \eta < \infty$ is the neighborhood's size, $\mu^x, \mu^y \in \boldsymbol{N}, \mu^x, \mu^y < \infty$ 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 $P_j^i = \{ (N_j^i, T_j^i) / \forall\ j \in [1, \eta+\mu^i], N_j^i \in [i_1, i_{\eta+\mu}]$ (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$\}$.

Figure 6 presents an example of the definition of this interface. The cell with a mark in Figure 6(a) is connected to the neighborhood and to other two DEVS models. The *A* model transmits integer values, and the *B* model uses real state variables. The internal coupling of this cellular model is defined as shown in the Figure 6(b) and 6(c). Because two neighbors are used, two input/output ports are included in each cell. Figure 6(b) shows the cells to which both output ports are connected. Figure 6(c) represents the ports giving input to the cell. Finally, because the marked cell is connected with other DEVS, two extra input/output ports are needed. Figure 6(d) shows the interface of this cell. In this case, $\eta = 2$, $\mu^x = \mu^y = 1$. Therefore, $P^x = \{ P_1^x, P_2^x, P_3^x \} = \{ (N_1^x$, binary), $(N_2^x$, binary), $(N_3^x$, integer)$\}$ and $P^y = \{ P_1^y, P_2^y, P_3^y \} = \{ (N_1$, binary), $(N_2$, binary), $(N_3$,real) $\}$.

A given input port can receive two different inputs with the same simulated time only if the influencer transmits more than one value in that given simulated time. Considering the semantics for a cell execution, this only occurs if two internal transitions are executed in the same instant (details of this semantics can be found in [1]). When the internal function is executed, the first event in the squeue is transmitted. This queue keeps all the delayed values; therefore, the simultaneous outputs in the influencer only can occur if zero-time delays are allowed. Therefore, the Lemma 2(a) is valid.

Lemma 2(b) considers that a cell space can be connected to other DEVS models, using the last μ input ports defined in the interface. The semantics for a DEVS model is that the output function transfers information between models, and they are activated prior to the internal transitions. Therefore, the execution of two simultaneous internal transitions can occur only if the

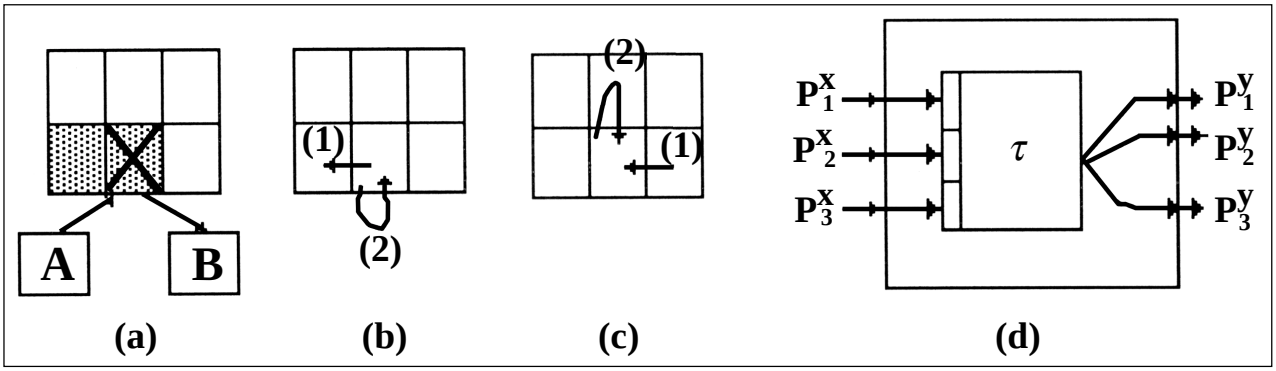

Figure 6. Cell-DEVS models coupling and interfaces

model schedules an internal transition with zero time, and Lemma 2(b) is valid.

A second problem for the defined Cell-DEVS is related to the desired behavior for **parallelism**. In this case, the following approach was used: the occurrence of external simultaneous events for a given cell is treated by the local computing function. Whenever an external event arrives, it is stored in the input set for the cell. Therefore, simultaneous models will transmit their generated values in parallel, which will be stored in the input set. If an event occurs simultaneously with an internal scheduled event, the internal transition function is activated prior to the external transition (as in the E-DEVS formalism [13]).

One final problem of the previous formalism is related to the **collision handling**. In most cases, a collision is controlled because the local computing functions use 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.

The formalism should be as general as possible, to allow the modelling of any kind of n-dimensional cell spaces. Hence, the prohibition of zero-time delays is too restrictive. Even when it was shown that Cell-DEVS behaves uniformly thanks to the interface definition, a bag-DEVS is needed for the cases of zero-time transitions. In addition, the Cell-DEVS models can be coupled with traditional DEVS submodels. All the factors considered in this section were taken into account, resulting in a redefinition of the Cell-DEVS formalism. The following section will present the main changes introduced to meet the stated goals.

3. Parallel Cell-DEVS

This section defines a new approach to using the concepts of confluent transition function explained earlier. Other extensions allow the management of complex delay behavior that was not previously possible. The first part of this section explains the definition of atomic models, and then coupled models are defined. The delay behavior will be used in the following sections to introduce complex timing for the cells.

3.1 Atomic Models

A parallel Cell-DEVS atomic model can be formally defined as:

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

The contents of a model are similar to those presented in Section 1. A detailed specification can be found in the Appendix, but most components have not changed. Two confluent functions have been added: δ_{con} and τ_{con}. In addition, the external transition and output functions have been changed to handle input/output bags (X^b and Y^b) for each cell. The external transition function activates the local computation, whose result is delayed using one of both kinds of constructions: transport or inertial delays. 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. Figure 7 shows a sketch of the contents of each cell.

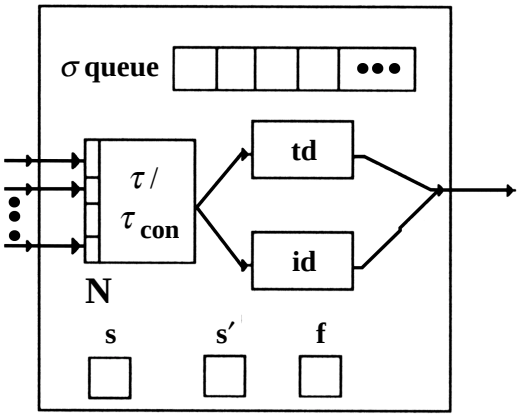

Figure 7. Cell's definition

The present definition changes the semantics of the delay functions. Originally, only one kind of delay of a given duration was related to each cell. Now, the local transition function will return the type and length of the delay, and the cell's outputs will be delayed accordingly. This redefinition allows to include complex timing behavior, as will be seen in Section 5. The confluent transition function δ_{con} is activated when there are collisions between internal and external events. It must activate the confluent local transition function τ_{con}, whose goal is to analyze the present values for the input bags, and to provide a unique set of input values for the cell. In this way, the cell will compute the next state by using the values chosen by the modeler. The semantics for the transition functions are defined in the Appendix.

In this case, the external transition function activates the local computation, whose result is delayed using one of both kinds of constructions. The output function, which executes prior to the internal transition function, is in charge to transmit the present values to other models. This is done after the execution of a delay function, carried out by the internal transition.

In case of a collision, the confluent transition function chooses members from the bag, and updates the inputs for the cell. Then it deletes the unnecessary members of the bag. Because $\sigma = 0$, 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.

3.2 Formal Specification of Generic Coupled Cell-DEVS Models

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

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

All the model's contents for this case have been defined in Section 1. The main change is that **C** is a 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 $I = \{ (i_1,...,i_n) / (i_k \in N \wedge i_k \in [1, t_k]) \ \forall k \in [1, n]\}$. That

is, each cell in the space is a parallel Cell-DEVS atomic cell using the δ_{con} and τ_{con} functions to avoid collisions. Hence, the **select** function has disappeared. A detailed definition can be found in [2].

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:

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

$\boldsymbol{I} = < P^x, P^y >$ represents the interface of the modular model. Here, $\forall\ j \in [1, \eta]$, $i \in \{X, Y\}$, $P_j{}^i$ is a definition of a port (input or output, respectively) where:

$$P_j{}^i = \{ (N_j{}^i, T_j{}^i)\ / \forall\ j \in [1, \mu], (\mu \in \boldsymbol{N}, \mu < \infty),$$
$$N_j{}^i \in [i_1, i_m] \text{ (port name), y } T_j{}^i = \text{port type}\};$$

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

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

$\mathbf{D} \in \boldsymbol{N}$ is an index for the components of the coupled model, and

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

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

is a General Coupled Cell-DEVS as those defined in Section 3.2, for cellular models, and:

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

otherwise.

$\mathbf{I_d}$ is the set of models influenced by the model d, and $\forall\ j \in I_d$, $d \in D$, $I_d \subseteq D \cup \{self\}$, $d \notin I_d$,

$\mathbf{Z_{dj}}$ is the translation function from d to j, where

$Z_{self\ j}$: $Y_{self} \rightarrow X_j$ if none of the implied models is Cell-DEVS, or

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

$Z_{d\ self}$: $Y_d \rightarrow X_{self}$ if none of the implied models is Cell-DEVS, or

$Z_{d\ self}$: $Y(c_1)_d \rightarrow X(c_2)_{self}$, 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}: $Y_d \rightarrow X_j$ 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.

In [15], 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 a coupled Cell-DEVS is a DEVS is equivalent to an atomic DEVS. Hence, the models can be integrated into a DEVS hierarchy. These results are summarized in the following propositions, which will not be proven here.

Lemma 3
Parallel Cell-DEVS models are equivalent to parallel DEVS models.

Lemma 4
Closure under coupling for parallel Cell-DEVS models: a coupled parallel Cell-DEVS model is equivalent to a basic parallel Cell-DEVS model.

4. Cell-DEVS Spaces Simulation

A Cell-DEVS model can be mapped onto an executable specification that can be simulated using an abstract mechanism. This is achieved by using a set of specialized *Processors* that drive the simulation process (as it was originally defined in Zeigler's works). The so called *Coordinators* are associated with the activities of the hierarchical coupled models. The others, called *Simulators*, are associated with the atomic model's activation.

In [9], an environment for Cell-DEVS modelling and simulation was built. It is defined as a class hierarchy of *models* and *processors*, and it has been redefined to include parallel models. This new strategy should allow the simulation of parallel Cell-DEVS; therefore, the original algorithms were changed, using the strategies defined in [12]. The main components of processors in this hierarchy are included in Figure 8.

In Cell-DEVS spaces, the coupled models are composed by several atomic cells, each associated with one simulator. The model's parameters (as specified in the previous section) are used to create the simulators and to define their names. The internal coupling is set up using the previous definitions, and a coordinator is associated with the coupled model.

The coordinators for the standard DEVS models follow the procedures defined in [12] based on the interchange of different messages. Instead, a new procedure has been defined for the simulators for cell spaces. The parallel processors include a synchronization mechanism based on the use of a specialized message, called the "*@-message*." When one of these arrives at a coordinator, all the models scheduled for the present simulated time (called the *imminent*) are activated. Their output values are collected into *y-message*s, and these are translated using the Z_{dj} function. New input messages (*x-message*s) are created, and their values are inserted into the input bags of the corresponding models. When the synchronization phase finishes, the resulting imminent models are executed by sending an **-message* to their simulators.

Figure 9 shows a sketch of the procedure executed by the simulators. (Note that the present mechanism only includes the basic behavior of the simulator.)

The imminent model will be a cell in the space. Therefore, the cell's coordinators should have a list of imminent children to detect it. Each simulator linked with an imminent cell activates

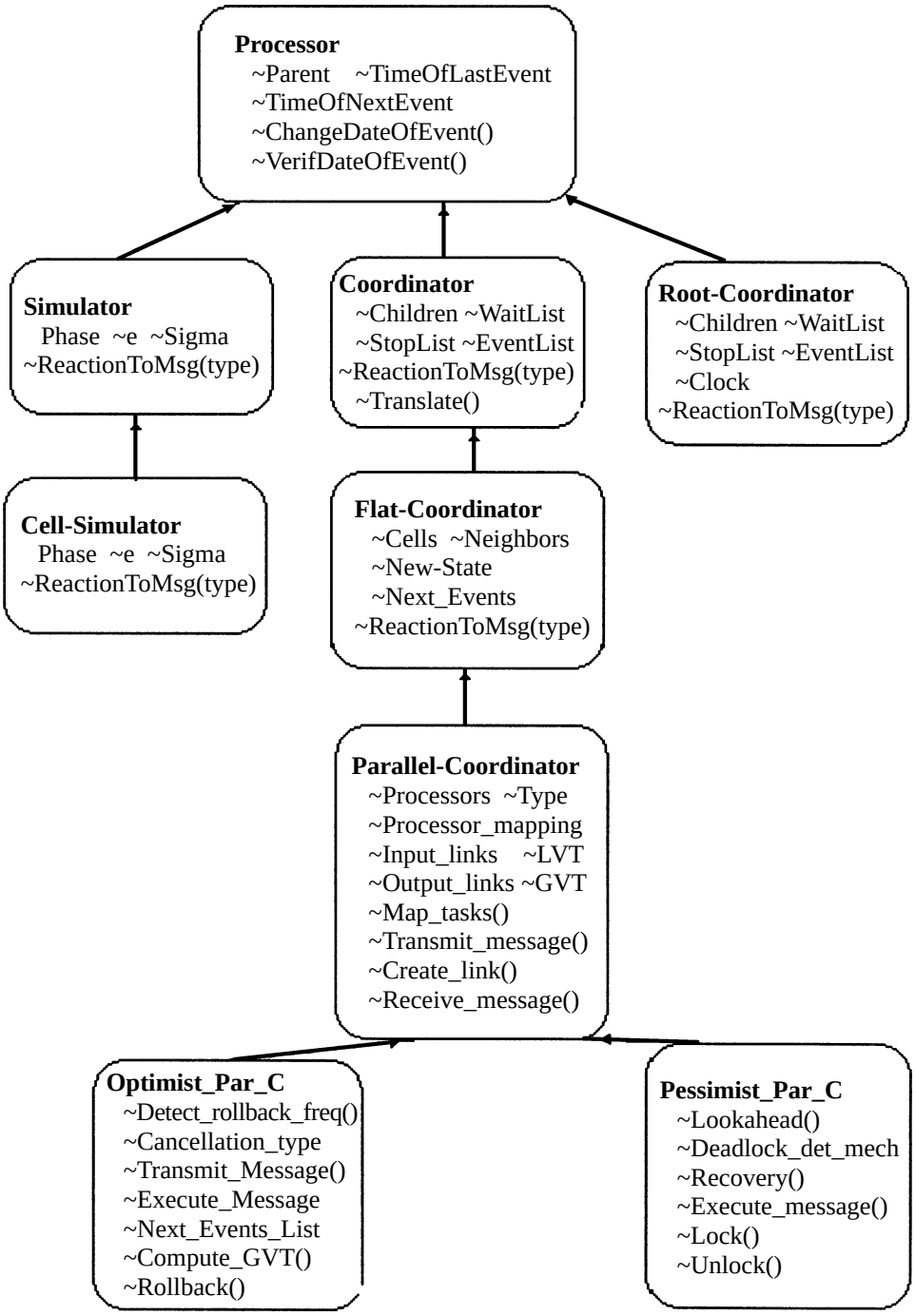

Figure 8. Class hierarchy for parallel Cell-DEVS processors

the model's output and internal transition functions, activating the local computation. Consequently, the *@-message* produces an output of the selected model, whose result is sent to the parent coordinator. The model's behavior upon arrival of an external message is simple: a *q-message* carrying the input value is queued into the bag corresponding to the input port receiving the message.

The main activity is driven by the **-message*. The simulator activates the external transition function if the external event arrives prior to the scheduled internal transition. Instead, if the message arrives simultaneously with the internal event, two different actions should be taken. If there are several events queued into the bag, the δ_{con} function should be activated to decide which messages will be used as inputs for the local transition. This decision depends on the behavior defined by the user by coding the τ_{con} function. Then, the internal transition function is executed with correct input values.

The simulators return *done-message*s and *y-message*s that will be translated to new *@-message*s, **-message*s and *q-message*s, respectively. The coordinators translate the messages by using the cell coupling previously defined. In this case, the arrival of the *@-message* produces a set of *@-message*s that are sent to the lower level processors (in this case, the cell's simulators).

The upper level coordinator is informed when all the imminent children have received the message. The behavior of the coordinators can be defined as seen in Figure 10.

The reaction for *y-message*s is different than the one obtained in other DEVS models. In this case, the output messages are sent to the neighbors, or to other models (in this case, using the Ylist). Here, the only task to execute when a new *y-message* is received is to queue it into the input bag of the cell. Finally, the **-message* is in charge of the transmission of the messages to the lower-level coordinators, and to synchronize the activity of the *@-message*s.

The standard coordinator should be modified, because now it is in charge of manipulating the mapping between the Xlist and Ylist. Therefore, the code for the simulator should use the new definition for the Z_{dj} function as seen in Figure 11.

As can be seen, the coordinators for cell spaces have a different behavior than that defined for standard DEVS models, because they have to execute the Z_{dj} function in a different way. The coordinators now behave differently when detecting the message destinations, as they are composed of a cell and a model

```
Cell_Simulator() {
Receive(message, port);

Case message of:
(@, t):  y = λ(s);
         send (y, t) to the parent coordinator;

 (q, t):  add q to the bag associated with the input port;

(*, t):  t ∈ [tl, tn]  ⇒  e = t – tl;  s = δext(s, e, bag);
         if (t = tn)
                  bag ≠ {∅} ⇒ N = δcon(s, N, d, bag); bag = {∅};
                  s = δint(s, N, d);
         tl = t;
         t = tn = tl + D(s);

send (done, t) to the parent coordinator;
}
```

Figure 9. Simulation mechanism for a cell

```
Cell_Coordinator() {

Receive(message, port);

Case message of:
(@, t):  ∀ imminent child (X1,...,Xn)
                  send (@, t) to (X1,...,Xn);
                  add (X1,...,Xn) in the SYNC set;
         wait (done, t) ∀ imminent child;
         send (done, t) to the parent coordinator;
         tl = t;

(y, t):  sender: cell (Y1,...,Yn)
         ∀ (X1,...,Xn) / (X1,...,Xn) belongs to the neighborhood of (Y1,...,Yn)
                  send (y, t) to the cell (X1,...,Xn);
                  add (X1,...,Xn) in SYNC;
         if ((Y1,...,Yn) ∈ Ylist of the coupled model) then send (y, t) to the parent coordinator;

(q, t):  Destination: Cell (X1,...,Xn)
         add q to the bag of cell (X1,...,Xn);

(*, t):  ∀ q ∈ bag of cell (X1,...,Xn)
                  send (q, t) to (X1,...,Xn);
                  add (X1,...,Xn) to the SYNC set;
         empty bag of the cell (X1,...,Xn);
         ∀ (Y1,...,Yn) ∈ SYNC send (*, t) to (Y1,...,Yn);
         wait all (done, tn);
         tl = t;
         tn = minimum of all the received tn's;
         empty SYNC;
         send (done, t) to the parent coordinator;
}
```

Figure 10. Cell spaces coordinators

```
influences j of child i
  if j is a Cell-DEVS model then
      ∀ (X1,...,Xn) ∈ Xlistj, (Y1,...,Yn) ∈ Ylisti
      y = value of cell (Y1,...,Yn)
      send (y, t) to the cell (X1,...,Xn) of the model j
```

Figure 11. Modification to the standard coordinator

name. The model name is used by the high-level coordinators and the cell position is employed by the cell space coordinator. The behavior for the root coordinator is the same that the defined for other Parallel-DEVS models, that is:

```
t = tn of the topmost coordinator
while t ≠ ∞
      send (@, t) to the topmost coordinator
      wait until (done, t) is received from it
      send (*, t) to the topmost coordinator
      wait until (done, tn) is received from it
```

In [1], a flat simulation mechanism was defined, allowing the reduction of intermodule interaction and the overhead produced by the simulation mechanism. A flat simulator is implemented as a bidimensional array of records associated with the cell space. Each record includes information of the state and delay for the cell and a neighborhood list to record the cell's influences for the cell space. Using this approach, the hierarchical message interaction is not needed, because the multiple processors are eliminated. This simulation mechanism has been redefined for parallel Cell-DEVS models, and its definition can be found in [15].

5. Cell's Delay Behavior

The behavior for the cell delays was presented in [1, 2]. Originally, each cell had a fixed kind and duration for the delay functions. Nevertheless, as shown in Section 3, this behavior was extended so each rule of the local function is allowed to activate the delay. This section analyzes the behavior of the delay functions under normal execution and under collisions.

5.1 Cell's Timing Basic Behavior

The behavior of transport delays allows reflecting the straightforward propagation of signals over lines of infinite bandwidth. They allow the modeling of variable commuting time for each cell with anticipatory semantics (every scheduled event is executed). Instead, inertial delays provide a preemptive semantics to represent that a state change needs an amount of energy to be provided to the system. In these cases, the scheduled events representing the system state changes cannot be executed due to a too-small interval between two inputs. These delays allow the analysis of the limit response frequency of the systems [7].

Let us consider a transport delay of five time units for a given cell. The input/output trajectories depicted in Figure 12(a) show the behavior of the delay function. It can be seen that the results are delayed for five time units, and the cell remains active while there are queued values waiting to be transmitted. In contrast, the behavior for atomic cells with inertial delays is presented in the input/output trajectories of Figure 12(b). In this case, an inertial delay function of five time units is used. The input values are delayed as in the previous case, but at the simulated time 19, the input value changed before the consumption of the delay, and the previous input is preempted. At the simulated time 15, a transition to the state zero has occurred. This external event schedules an internal event for simulated time 20. In that moment, the cell should send the state change produced as an output. Instead, a new external event arrives before the consumption of the delay, representing that the zero value was not kept during the delay. Therefore, this state change is preempted, and the previous state (1) is restored.

These constructions have been taken from the domain of circuit modeling. Nevertheless, they are useful for representing different phenomena. For instance, the transport delay can be used to represent the speed of a car in a traffic simulation (the inverse of the delay length). In addition, an inertial delay could represent cars arriving to a crossing. Let us suppose that a car is waiting to get into a crossing from the left. This cell checks if

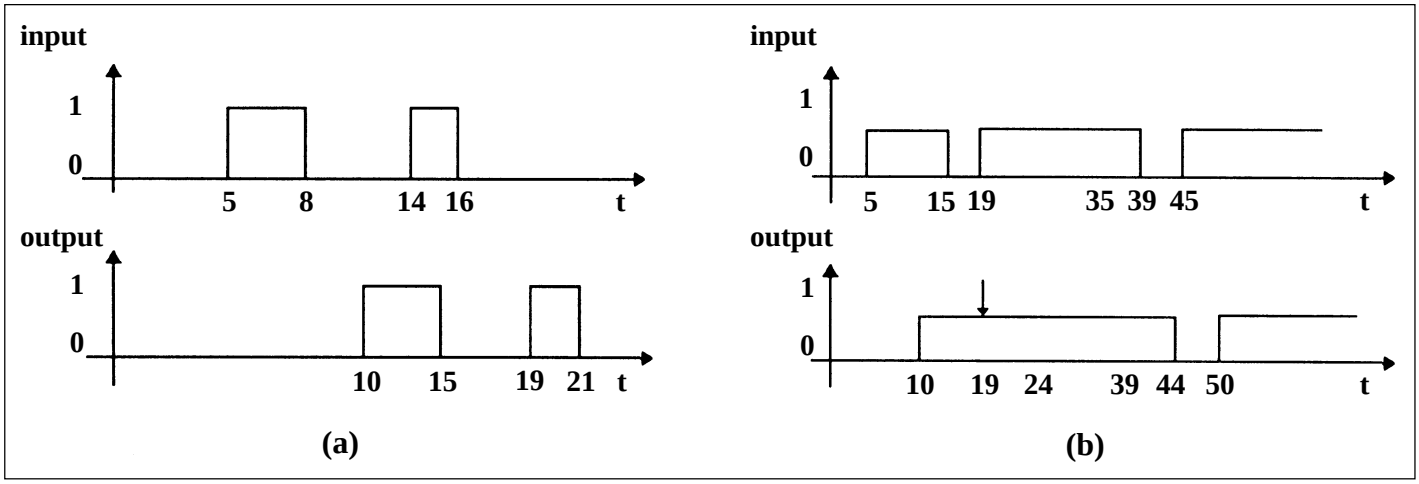

Figure 12. (a) Transport delay behavior; (b) inertial delay behavior

there is a car in the crossing or in the street to the right. If there is no car in either cell, a state change with inertial delay is scheduled. If the delay is consumed, the event is transmitted and the crossing receives the car. Instead, let us suppose that a high-speed car arrives from the right prior to the consumption of the delay. The neighborhood of the left cell has changed, and the local function must be computed. Since the previous state of the cell has changed, the scheduled event representing advance to the crossing is preempted.

5.2 Combined Delay Functions

The definitions presented in Section 3.1 allow the inclusion of a combination between transport and inertial delays. The behavior for each of them is the same as that defined originally. That is, if a transport delay is activated, the value is transmitted only after the delay (the σ queue is used to keep the value). Instead, an inertial delay is used to transfer the value only if it is kept during the whole length of the delay. Several combined behaviors will be included in the following paragraphs.

a. Transport/Inertial Delays

Figure 13 presents several behaviors that can be achieved by using different delay functions, and the results obtained under collisions. It shows an example of execution for an atomic binary cell with a combination between transport and inertial delays. The basic behavior is that, if an inertial delay is not accomplished and several values are waiting at the end of a transport delay, they are preempted.

It is supposed that the local computing function activates delays of different kinds and durations, depending on the simulated time of occurrence for each event. Initially, the cell uses a transport delay of 17 time units. Between 50 and 60, and from 90 to 100, an inertial delay of six time units is considered. The values obtained from 60 to 70 will be delayed using an inertial delay of nine time units. Finally, a transport delay of 25 time units is applied under collisions.

The input trajectories represent inputs to the delay function. This function reacts to the inputs by sending the outputs shown in the figure. The execution details of the model are presented in Table1. Each line of this table shows the state for the cell. 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. Each column represents a different state variable. First, we include the time advance (*t*), present state and computed state (s, s′). Then the model's phase (*A*ctive or *P*assive), next scheduled time (σ) and elapsed time for the model (*e*), are presented. Finally, the feasible future value for the cell (f), the kind of delay (*tr*ansport or *in*ertial), and the queue of scheduled events (σ*queue*) are defined. In several cases, we show the values of the state variables before and after the execution of the corresponding transition functions (*x*/*y*).

The delay function receives external events at instants 30 and 40, and they are delayed for 17 time units. These values are

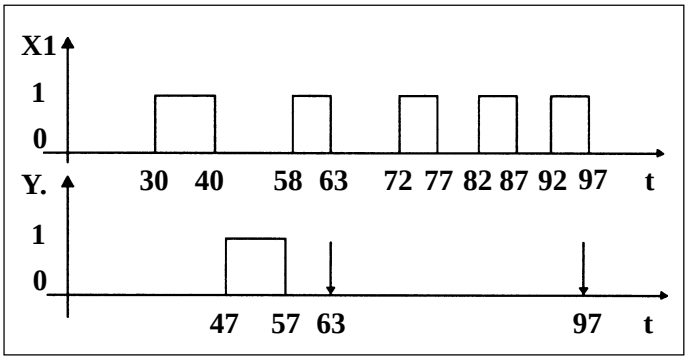

Figure 13. Input/output trajectories for combined transport/inertial delays

Table 1. Execution sequence of the previous trajectories

	t	s	s′	p	σ	e	f	d	σ queue
	…	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.	(1,5), (0,10), (1,15), (0,20), (1,25)
!	97	1/0	0	A	1/6	4	1/0	in.	
*	103	0/0	0	P	∞	7	0		

Table 2. Execution sequence of the previous trajectories

	t	s	s′	p	σ	e	f	d	σ queue
	…	0	0	P					
	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.	(1,5), (0,10), (1,15), (0,20), (1,25)
!	97	1/0	0	A	1/6	4	1/0	in.	(1,1), (0,6)
*	98	0/1	1	A	0/5	1/0	0	in.	(0,5)
*	103	1/0	0	P	0/∞	5/0	0		

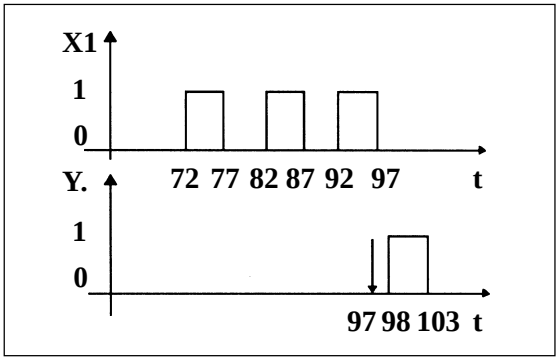

Figure 14. Input/output trajectories for selective preemption

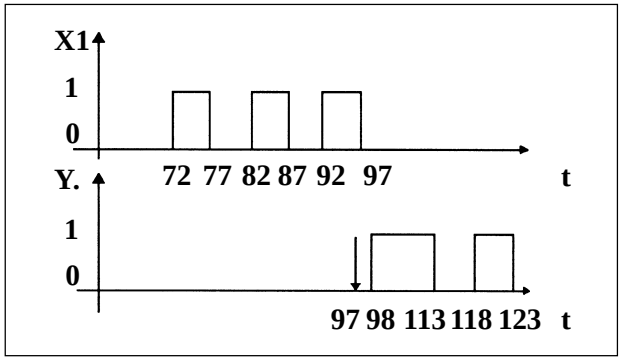

Figure 15. Input/output trajectories for selective preemption

Table 3. Execution sequence of the previous trajectories

	t	s	s′	p	σ	e	f	d	σ queue
	…	0	0	P					
	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.	(1,5), (0,10), (1,15), (0,20), (1,25)
!	97	1/0	0	A	1/6	4	1/0	in.	(1,1), (1,11), (0,16), (1,21)
*	98	0/1	1	A	0/10	1/0	1	in.	(1,10), (0,15), (1,20)
*	108	1/1	1	A	0/5	10/0	1	in.	(0,5), (1,10)
*	113	1/0	0	A	0/5	5/0	0	in.	(1,5)
*	118	0/1	1	A	0/5	5/0	0	in.	
*	123	1/0	1	P	0/∞	5/0	0		

stored in the σ queue, and they are transmitted when the transport delay is consumed. At simulated time 58, the input event is retarded using an inertial delay of six time units. As in instant 63, the input changes from 1 to 0, the value is not kept during the delay, and it is preempted. In addition, a collision is shown at instant 72. The transition that occurred at instant 63 was delayed using nine time units. As in the same simulated time an external transition is activated, the confluent transition function is executed, the local transition function is carried out, and its result is deferred using a transport delay of 25 time units. Finally, at simulated time 92, an external event occurs. The previous values were delayed using a transport delay of 25 time units. Here, the inertial delay is activated prior to the output of the previous transport delays. As the value is not kept during six time units, preemption occurs at instant 97, and the σ queue is emptied.

b. Inertial Delays with Selective Preemption

Two new constructions related to the execution of inertial delay functions were added. They are devoted to selectively preempt queued events for inertial delays. The *preempt-last(n)* and *preempt-first(n)* constructs will select the last/first *n* events in the queue, and they will be preempted when a new event arrives prior to the delay consumption.

Table 4. Execution sequence for the previous trajectories

	t	s	s′	p	σ	e	f	d	σ queue
	…	0	0	P					
	30	0/1	1	A	10	0	1	tr.	(1,10)
*	40	1	1	A	0/∞	10/0	1		
	42	1/0	0	A	5	0	0	tr.	(0,5)
*	47	0	0	A	0/∞	5/0	0		
	50	0/1	1	A	10	0	1	tr.	(1,10)
	56	1/0	0	A	4	6	0	tr.	(1,4), (0,5)
*	60	0	0	A	0/1	4	1		(0,1)
*	61	0	0	P	∞	1	0		

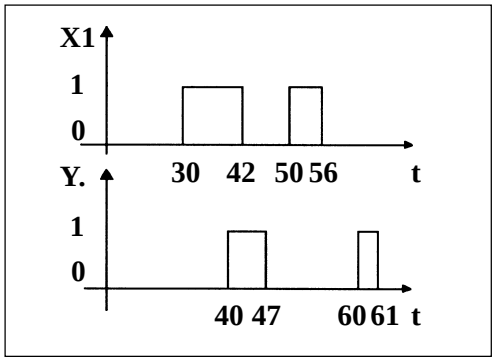

Figure 16. Input/output trajectories for rise/fall delays

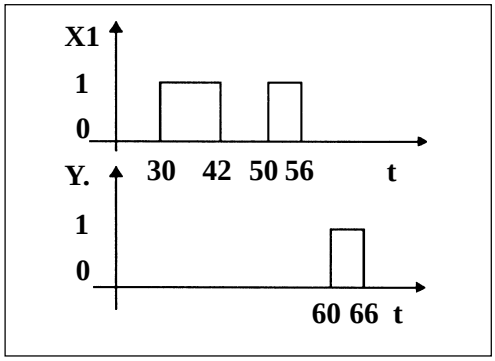

Figure 17. Input/output trajectories for direct preemption

Table 5. Execution sequence for the previous trajectories

	t	s	s′	p	σ	e	f	d	σ queue
	…	0	0	P					
	30	0/1	1	A	10	0	1	tr.	(1,10)
!	40	1/0	0	A	0/∞	10/0	1		
	42	1/0	0	A	5	0	0	tr.	(0,5)
!	47	0	0	A	0/∞	5/0	0		
	50	0/1	1	A	10	0	1	tr.	(1,10)
	56	1/0	0	A	4	6	0	tr.	(1,4), (0,5)
*	60	0/1	1	A	0/1	4	1		(0,6)
*	66	1/0	0	P	∞	6	0		

Figure 14 analyzes the behavior for such construction. The input/output trajectories of the previous example were considered, and in this case, at simulated time 97, an inertial delay using the *preempt-last(3)* function is executed.

In this example, the inertial delay is not accomplished, and the last three events are preempted, reflecting the transmission of the first ones.

The second construction introduced is the *preempt(start, end)* behavior. In this case, the modeler can choose the individual events to preempt when an event arrives. Figure 15 presents the execution flow for the previous example. In this case it is considered that when the preemption occurs, the *preempt(2, 2)*function is used.

In this case, as the value is not maintained during the six time units of the transport delay, it can be seen that the internal event scheduled for simulated time 103 is preempted. Therefore, the behavior is equivalent to having a longer sequence of the previous scheduled value.

c. Inertial/Transport Delays

As showed previously, if an inertial delay is activated after the reception of several events using transport delays, every queued event is preempted. The opposite combination does not introduce any change from the standard behavior. If the value produced by a transition function is delayed using an inertial delay function, and in the meantime a new external event arrives, it

can preempt the previous value. This will happen only if the new value is different from the original one, as shown in Section 5.1. The remaining events will be delayed using the transport delay.

d. Rise/Fall Delays

In [6], the Rise-Fall delay function is presented as a way to provide a different timing behavior according to the result obtained when a Boolean function is executed. In this case, a delay length is used when a rise occurs (0–1 transition), whereas a different one is chosen for a fall (1–0 transition). This basic behavior was extended, providing different delays associated with rules applied by a cell. In the following example, the cell uses a transport delay of 10 time units for a 0–1 transition, and a delay of five time units for 1/0 transitions, as is shown in Figure 16.

e. Direct Preemption

A final behavior that can be applied to a cell is to provide direct preemption to the inputs of the delay functions. In this case, we want to represent that a cell changes its present state, but the value is not sent to the neighboring cells. In the following example, the cell uses a transport delay of 10 time units, and the first inputs will be directly preempted. This construction is equivalent to an inertial delay of zero time.

5.3 Example: Representation of an Ecomodel

In this section, we will analyze the use of the new constructions using an example of an ecological system. It presents the combination between a field of ants and a fire diffusion model. The cell space represents the movements of ants in the ground, and it allows representing fire or rain in the cells. If the fire spreads to a cell with ants, the ants die. Instead, if it rains, the fire extinguishes. The model uses a 3×3 neighborhood. The execution results of the standard model are presented in Figure 18(a) and 18(b). We can see the movement of the ants (represented as light dots) in the ground. The ants move at random, using a transport delay of 500 ms. These movements can be seen in the lower part of Figure 18(a) and (b), which represents the model's advance after 500 ms. We have also introduced several cells with fire (dark dots), and we can see that the ants passing through them have died (this can be seen, for instance, in the lower left part of the figure).

The new constructions are useful to represent complex timing conditions that could not be represented in the previous specifications. For instance, Figure 18(c) and 18(d) show the application of the *rise/fall delay* construction. These figures also represent the model's advance after 500 ms. After 10 seconds, the fire cells introduced in Figure 18(a) and 18(b) have expanded, obtaining the present configuration. This behavior has been accomplished using a variable delay for each cell, according to its present state. The ant movement delay (500 ms) represents the speed of the ants in the ground. Instead, the fire spreads more slowly (a delay of five seconds). This is shown in Figure 18(d), where the fire remains fixed in several cells while some ants have moved. For instance, we can see that there are two cells in the middle-left containing ants that cannot move due to the fire. Nevertheless, the fire is not invading those cells due to the length of the delay used. In addition, we have introduced the influence of rain, showing some cells where the previous existing fires have disappeared.

These delay construction could be also used, for instance, in a heat diffusion model, where each cell will compute its future state by averaging the present values of the neighbors. The rise/fall delay construction can be used to represent that heat spreads faster than cold. Therefore, the hot cells can use a small delay, whilst the colder ones use a larger one. If this construction were not available, rules that are more complex should have been used to define the variable delay. For instance, when fire is detected in the ecomodel, several intermediate states representing the passage of fire should be used. Each intermediate state would represent the consumption of the delay associated with the ant movement (500 ms), and the fire status would rise after passing 10 intermediate states. As can be seen, this involves more complex rules and waste of computation time.

A similar situation occurs when the combination of *transport/inertial delays* is allowed. Let us suppose that we want to model the following case: three ants arrive at a cell, and then the cell changes to a "rain" state. The combined construction can be used as follows. A transport delay of 20 units is used for the arrival of each ant, and an inertial delay of 10 time units is used for the change to rain. In this way, if a fire condition occurs before the consumption of the delay, the arrival of the ants and the wet state are not transmitted to the neighboring cells. The ants that did not leave the cell die, and the rain state is not sent. Instead, if the inertial delay finishes before the fire, the ants and the rain will spread to the neighbors. This combination was not allowed in the original specifications, and this kind of situation could not be easily modeled. One way to represent it is

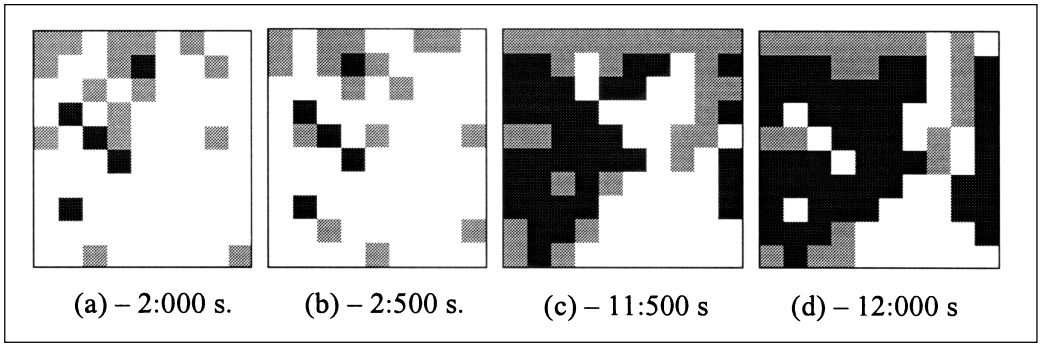

Figure 18. Execution of a simple Ecomodel

to include a wider neighborhood. In this case, we could detect the presence of fire in far cells, and in that case, activate an alternative rule turning on an intermediate state variable. This value reflects that fire is approaching, and another rule should avoid the diffusion of the wet state and the ants. Again, the introduction of several complex rules introduces a higher amount of computation time. Moreover, when the neighborhood size is increased, overhead is introduced due to the need of to keep a more complex data structure. In addition, a larger neighborhood produces activation of other cells, activating them even when they are quiescent.

Now, let us consider that a cell receives fire after the introduction of two ants. A transport delay of two time units is used for the ant arrival, and an inertial delay of five time units is used for the fire. We could include a rule such that, if at instant four of the inertial delay the rain arrives at the cell, the fire is preempted using *preempt_last(1)*. Therefore, the arrival of both ants is still recorded, and their existence in the present cell is transmitted to the neighbors. Instead, if the inertial delay finishes, the fire spreads and the arrival of the ants is not recorded. The original constructions would preempt all the previous events, loosing both ants. In this case, a larger neighborhood would have helped to detect the arrival of fire by looking to a greater distance, as in the previous case.

Finally, a cell could be provided with *direct preemption* of the inputs to represent, for instance, the transmission of the present content of a cell only when the number of ants in each cell is a multiple of five. Each input of an ant is directly preempted, except when a multiple is reached. This behavior could not be achieved in any way using the original constructions, because every time a state variable changes, the neighbors are informed after the delay consumption.

6. Conclusion

This work presented an extension of the Cell-DEVS paradigm, allowing the parallel execution of the models. To do so, the behavior under collisions was defined accurately. In this case, the user is in charge of defining the cell behavior under simultaneous events. A simulation mechanism related to this kind of model was presented, and a new extension to the flat coordination mechanism was developed.

The formalism enables the specification of complex cell-shaped models. 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. Parallel implementation of these models could not be achieved in the original definitions. Instead, this new approach introduces models than can be executed correctly in parallel environments. The use of a formal specification based on the DEVS formalism improves the validation of the specifications. An accurate semantics was defined, allowing one to ensure the validity of the models.

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

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. These new constructions allow the reduction of the sizes of the needed neighborhoods, and the complexity of the rules involved for each cell. They also introduce a simple definition of complex timing behavior that was not allowed in the original definitions, or was too difficult to develop. Consequently, performance gains and reductions in the development times can be achieved.

At present, a mapping between parallel Cell-DEVS models and the parallel simulation environment was defined, and is under implementation. In this way, a tool to run complex 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 using a parallel framework.

7. Acknowledgments

I would like to thank the anonymous referees and Dr. Bernard Zeigler for their comments on this article. This work was partially funded by ANPCYT Project 11-04460 and UBACYT Projects TX04 and JW10.

8. References

[1] Wainer, G. and Giambiasi, N. "Specification, Modeling and Simulation of Timed Cell-DEVS Spaces." Technical Report No. 98-007, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 1998.

[2] Wainer, G. "Discrete-Events Cellular Models with Explicit Delays." PhD Thesis, Université d'Aix-Marseille III, 1998.

[3] Zeigler, B. *Theory of Modeling and Simulation*, First Edition, Wiley, 1976.

[4] Zeigler, B. *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, 1984.

[5] Wolfram, S. *Theory and Applications of Cellular Automata. Vol.1, Advances Series on Complex Systems*, World Scientific, Singapore, 1986.

[6] Giambiasi, N. and Miara, A. "SILOG: A Practical Tool for Digital Logic Circuit Simulation." *Proceedings of the 16th D.A.C.*, San Diego, 1976.

[7] Ghosh, S. and Giambiasi, N. "On the Need for Consistency between the VHDL Language Constructions and the Underlying Hardware Design." *Proceedings of the 8th European Simulation Symposium, Vol. 1*, Genoa, Italy, pp 562-567, 1996.

[8] Barylko, A., Beyoglonian, J. and Wainer, G. "GAD: A General Application DEVS Environment." *Proceedings of IASTED Applied Modelling and Simulation '98*, Hawaii, 1998.

[9] Barylko, A., Beyoglonian, J. and Wainer, G. "CD++: A Tool to Develop Binary Cell-DEVS Models" (in Spanish). *Proceedings of the XXII Latin-American Conference on Informatics*, Quito, Ecuador, 1998.

[10] Rodriguez, D. and Wainer, G. "New Extensions to the CD++ Tool." In *Proceedings of the SCS Summer Computer Simulation Conference*, San Diego, 1999.

[11] Fujimoto, R. "Parallel Simulation of Discrete Events." *Communications of the ACM*, Vol. 33, No. 10, pp 30-53, 1990.

[12] Chow, A. and Zeigler, B. "Abstract Simulator for the Parallel DEVS Formalism." *Proceedings of the Winter Simulation Conference*, 1994.

[13] Chow, A. and Zeigler, B. "Revised DEVS: A Parallel, Hierarchical, Modular Modeling Formalism." Technical Report, University of Arizona, 1994.

[14] Gardner, M. "The Fantastic Combinations of John Conway's New Solitaire Game 'Life.'" *Scientific American*, Vol. 23, No. 4, pp120-123, April 1970.

[15] Wainer, G. "Definition of Parallel Cell-DEVS Spaces." Technical Report No. 98-021, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 1998.

Appendix

A parallel Cell-DEVS atomic model can be formally defined as:

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

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

$\boldsymbol{X} \subseteq T$;

$\boldsymbol{Y} \subseteq T$;

$\boldsymbol{I} = < \eta, \mu^x, \mu^y, P^x, P^y >$. Here, $\eta \in \boldsymbol{N}$, $\eta < \infty$ is the neighborhood's size; $\mu^x, \mu^y \in \boldsymbol{N}$, $\mu^x, \mu^y < \infty$ 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 $P_j{}^i = \{(N_j{}^i, T_j{}^i) / \forall\, j \in [1, \eta+\mu^i],\ N_j{}^i \in [i_1, i_{\eta+\mu}]$ (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 $i = Y\}$;

$\mathbf{S} \subseteq T$;

$\theta = \{(s, phase, \sigma queue, f, \sigma)$ /

$s \in S$ is the status value for the cell,

$s' \in S$ is an intermediate 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

$\sigma \in \boldsymbol{R_0}^+ \cup \infty\}$;

$\mathbf{N} \in S^{\eta+\mu}$;

$\mathbf{d} \in \boldsymbol{R_0}^+$, $d < \infty$;

$\delta_{\mathbf{int}}: \theta \rightarrow S$;

$\delta_{\mathbf{ext}}: Q x X^b \rightarrow \theta$, $Q = \{(s, e) / s \in \theta \times N \times d; e \in [0, D(s)]\}$;

$\delta_{\mathbf{con}}: q x X^b \rightarrow S$;

$\tau: N \rightarrow S \times \{inertial, transport\} \times d$;

$\tau_{\mathbf{con}}: X^b x N \rightarrow S \times \{inertial, transport\} \times d$;

$\lambda: S \rightarrow Y^b$; and

$\mathbf{D}: \theta \times N \times d \rightarrow \boldsymbol{R_0}^+ \cup \infty$.

The semantics definition for the transition functions is defined as follows (note that tail/head/add represent the methods used to manage the elements of a list):

δ_{int}:

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

$$\frac{\sigma = 0; \qquad \sigma_{queue} = \{\varnothing\}; \qquad phase = active}{\sigma = \infty \ \wedge\ phase = passive}$$

λ:

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

δ_{ext}:

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

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

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

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

δ_{con}:

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