Departamento de Computación

*Facultad de Ciencias Exactas y Naturales*

Universidad de Buenos Aires

**INFORME TÉCNICO**

# Specification, modelling and simulation of timed Cell-DEVS spaces

Gabriel A. Wainer, Norbert Giambiasi

Report n.: 97-006

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

http://www.dc.uba.ar

Title: Specification, modeling and simulation of timed Cell-DEVS spaces

Authors: Gabriel A. Wainer; Norbert Giambiasi.

E-mail: gabrielwl@dc.uba.ar

Report n.: 97-006

Key-words: DEVS modelling and simulation, Cellular Automata, Cell spaces.

Abstract: This work is devoted to present several formal descriptions for cell spaces' modeling and simulation. The formalisms are based on the DEVS and Asynchronous Cellular Automata paradigms. The concepts of transport or inertial delays used in the circuit modeling domain have been combined, allowing simple specification of accurate timed models. The specifications have been defined for binary or three-states systems. The formalisms allow automatic definition of the spaces, and eases verification of the models, allowing efficient and cost-effective development of simulators.

---

# Specification, modeling and simulation of timed Cell-DEVS spaces


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



**Abstract**

This work is devoted to present several formal descriptions for cell spaces' modeling and simulation. The formalisms are based on the DEVS and Asynchronous Cellular Automata paradigms. The concepts of transport or inertial delays used in the circuit modeling domain have been combined, allowing simple specification of accurate timed models. The specifications have been defined for binary or three-states systems. The formalisms allow automatic definition of the spaces, and eases verification of the models, allowing efficient and cost-effective development of simulators.


## INTRODUCTION

Computer simulation tools are widely used to study complex systems. For these complex systems, a formal specification of the simulation model is necessary before its implementation. The conceptual simulation model must be validated before its implementation in a simulation environment.

Several modeling paradigms have been developed to specify dynamic systems formally; each one is better suited to model different kind of system. Some of these paradigms have been developed to model Discrete Events Dynamic Systems (DEDS). A DEDS is a system with a discrete state space and piecewise constant input-output trajectories. Most of the DEDS paradigms ignore the occurrence times of the events, leading to the so-called *logical DEDS models.* These models are used to solve problems in which only the orders of the events are to be taken into account.

In some applications (for instance, the simulation of systems in the digital circuit modeling domain), the timing information is crucial. In these cases it is necessary to use paradigms for the specification of *timed models*, that allows to represent the occurrence time of the events. B. Zeigler [Zei76] proposed the DEVS formalism for this kind of specifications. In this formalism, the delays are described as a lifetime of the model states.

In the digital circuit domain, several useful constructions are employed to model the delays as basic components that can be associated with functional components (ideal boolean functions). Those delays are useful not only in hardware design applications, but also in modeling other physical phenomena that can be described as cell spaces. Two different delays are usually employed: transport and inertial delays. The behavior of transport delays allows to reflect the straightforward propagation of signals over lines of infinite bandwidth (anticipatory semantics). Instead, inertial delays use a preemptive semantics to represent that to change a state some quantity of energy should be provided to the system.

The present work introduces several formalisms that allow the description of cellular models, using the basic ideas presented in this section. The new paradigm is based on the DEVS and Asynchronous Cellular Automata formalisms; it will be called timed Cell-DEVS.

An original part of the work consists in adapting the modeling of delay ideas developed in the domain of digital circuit modeling [GM76]. These concepts allow to represent different kinds of delays, and their specification can be done in a simple fashion besides the quantitative complexity of the manipulated models.

The article is organized as follows. First, a formal description of binary Cell-DEVS models is depicted, considering specifications for one cell, closed and general Cell-DEVS coupled models. In section 0, a three-state Cell-DEVS formalism with transport delays is defined. After that, an extension to deal with inertial delays is presented. Section 0 is devoted to examine the implementation of the formalism, considering the user point of view and the simulation of the cell spaces. In this section a method to flatten the hierarchical description of the cell spaces is given. After that, an introduction to cell models' parallel execution is presented. Finally, a detailed example will be presented to show the interaction of the simulation mechanisms and the specification.

# BACKGROUND

The use of modeling formalisms eases the development of the executable models because the behavior of a conceptual model can be validated against the real system. Also, the response of the executable model can be verified against the conceptual specification. A formal paradigm also must capture the system dynamics, allowing to model the dynamic comportment of the system. It also should provide a decomposition mechanism to reflect the hierarchical nature of most physical phenomena. A formalism also should allow the translation of a specification into an executable model.

This section is devoted to recall the basic aspects of two important modeling paradigms that meeting these requirements. They will be subsequently used as a basis for the present work: the DEVS formalism and the Cellular Automata paradigm.

## DEVS formalism

The DEVS (Discrete events systems specification) paradigm is a discrete events formalism proposed by Zeigler [Zei76]. This approach allows modular description of the models and attacks the complexity using a hierarchical approach. Object Oriented programming concepts have been incorporated to the formalism [Zei90, Zei95].

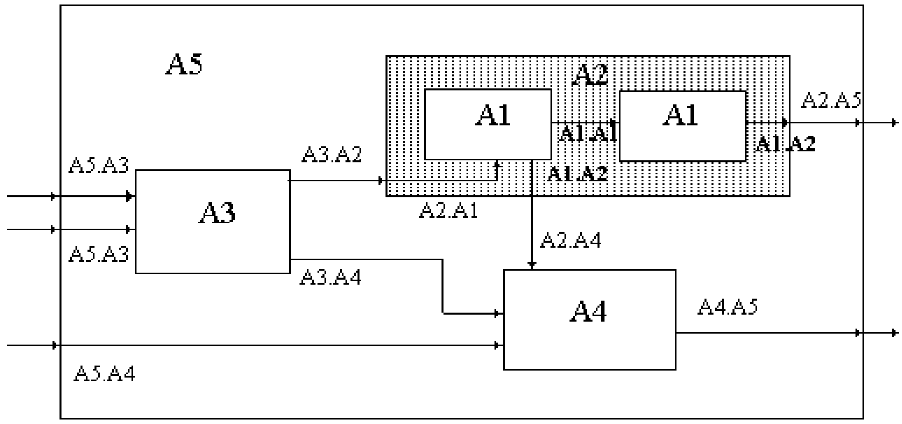



***Figure 1. Example of coupling of DEVS models (A1, A3, A4: atomic models).***

A real system modeled with this paradigm can be described as several submodels. Each model can be a behavioral (atomic) model or a structural (coupled) model. Each model consists of a time base, inputs, states, outputs and functions to compute the next states and outputs.

As the formalism is closed under closure, coupled models can be integrated into a model hierarchy. The use of this hierarchical modeling strategy allows the reuse of created and tested models, enhancing the security of the simulations, reducing the testing time and improving productivity.

## Cellular Automata.

Other formalism well suited to describe certain complex systems with different description levels are the synchronous Cellular Automata [Wol86]. A model built using this formalism uses discrete variables for time, space and system states. A conceptual cellular automaton is an infinite regular n-dimensional lattice where each cell in the lattice can take a value of a finite set. The states in the lattice are updated according with a local rule in a simultaneous and synchronous way. The cell states change in discrete time steps according with a local computing function. The function considers the present cell state and a finite set of nearby cells (called the cell's neighborhood).

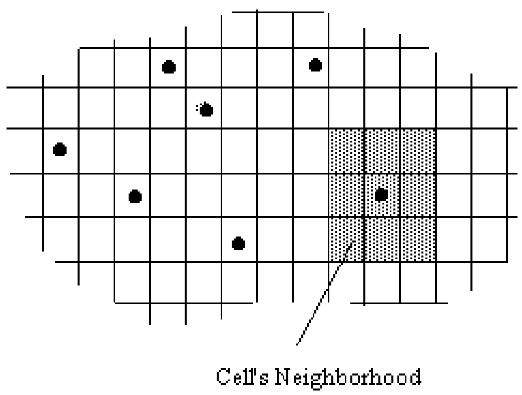



***Figure 2. Sketch of a Cellular Automaton***

Formally, a cellular automaton is defined as:

$$CA = < A, S, C, N, T, \tau, c.Z_0^{+} >$$

where

$A \subseteq \boldsymbol{Z} \wedge \#A < \infty$ represents the alphabet of cell's states;

$S \in A$ is the set of possible states for each cell;

$C = \{ c_{ij} / i,j \in \boldsymbol{Z} \wedge c_{ij} \in S\}$ is the cell's space;

N is the neighborhood set, defined as

$$N = \{ (i_p,j_p) / i_p, j_p \in \boldsymbol{Z} \wedge i_p, j_p \in [-1, 1] \ \forall p \in \boldsymbol{N}, p \in [1,\eta] \},$$

where $\eta \in \boldsymbol{Z}$, is the neighborhood's size;

$T: C \rightarrow C$ is the global transition function;

$\tau: N \rightarrow N$ is the local computing function; and

$c.\boldsymbol{Z}_0^+$ is the cellular automaton's discrete time base.

As we can see in the definition, the cell space is composed of individual cells, evolving in discrete time steps. Each cell in the space can take a value of a finite alphabet set.

The evolution of the automaton is defined by the execution of a global transition function that changes the global state. The behavior of this transition function depends on the results of local computing functions executing locally in each cell's neighborhood. Conceptually, the computation of these local functions is done in a synchronous and parallel way in every cell of the cell space. For two-dimensional cellular automata, the local computing function can be defined as:

$$s_{ij}\,[t+1] = \tau(s_{i0,j0}[t],..., s_{i\eta,\, j\eta}[t])$$

Here, $s_{ij}[t]$ is the state for the cell (i,j) in simulated time t and $\tau$ denotes the local computing function for the automaton. The parameters (i0,j0),..., ($i\eta$,$j\eta$) are defined by the cell's neighborhood. Though two-dimensional cellular have been defined, n-dimensional extensions can be defined.

The present formalism uses a discrete time base, that, as stated earlier, poses restrictions in the precision and efficiency of the simulated models. In the particular case of complex cellular automata, large amounts of compute time will be necessary to obtain the desired precision degree.

Nevertheless, in several cases most cells of the automaton do not need actualization on each time step. The presence of quiescent states allowed to define a new modeling paradigm (called asynchronous cellular automata) avoiding the restrictions of the synchronous approach. The main aspect of asynchronous automata is their evolution in continuous time. Instead of using a discrete time base, instantaneous events that can occur asynchronously at unpredictable times are considered.

As stated earlier, the use of a continuous time base lead to higher precision and skips periods of inactivity in the simulation, thus improving the use of the computer resources. However, this approach requires explicit synchronization of the cells to activate, thus increasing the simulation overhead. In some cases, this overhead can overrule the performance improvements of the asynchronous approach.

## FORMAL DESCRIPTION OF DEVS CELLS MODELS

This section presents a specification paradigm to describe cellular models as DEVS cells with transport delays. The definition is based in one previously defined in [GG96], a formal specification for ideal transport delay models. First, a modular definition of atomic models is included, and after a definition for the coupled models is presented. Finally, a formal specification of Cell-DEVS models that can be coupled with non-cell models is introduced.

## Formal specification of Transport Delay modular Cell-DEVS atomic models.

This section introduces modular Cell-DEVS models defined as a space composed of individual cells with transport delays. The cells are defined as atomic models that can be lately coupled to form a complete cell space.

Figure 3 depicts informally the basic contents for an atomic cell. Upon the occurrence of an external event, the boolean function is executed, consuming the input value for the event. After, the atomic cell input is delayed for **d** time units and an internal event is scheduled at the conceptual level. A queue for input events is used to hold the events' values and scheduled times because several external events can arrive during the delay. When the time for the internal event arrives, the cell's value is output.
As the influencees must be activated when the influencing cell changes, the result of the local computing function will be output only when the state changes. To allow this distinction, the present and future values are kept for the cell. The description for the atomic cell also keeps a local copy for the neighbors' values because the atomic model behavior must be independent of that of other models.

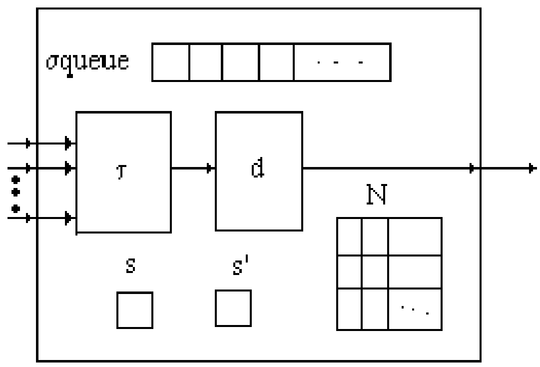



***Figure 3. Informal description of an atomic cell with transport delays***

Such a modular atomic binary cell model with transport delays can be formally described as:

$$TDC = < I, X, Y, S, N, d, \delta int, \delta ext, \tau, \lambda, D >$$

where

**I** = $<\eta, P^X, P^Y>$ represents the definition of the modular model interface. Here,

$\eta \in \boldsymbol{N}$ is the neighborhood size, and,

for $i = X \mid Y$, $P^i$ is a port definition (input or output respectively), where

$$P^i = \{ (N_j^i, T_j^i) \ / \ \forall \ j \in [1, \eta],\ N_j^i \in [i_1, i_\eta] \text{ (port name), and } T_j^i = \text{binary (port type)}\};$$

$\boldsymbol{X}$ = {0, 1} is the set of input external events;

$\boldsymbol{Y}$ = {0, 1} is the set of output external events;

**S** is the state set, where

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

$s \in \{0,1\}$,

$phase \in \{active, passive\}$,

$\sigma queue = \{ ((v_1,\sigma_1),...,(v_m,\sigma_m))$ / $m \in \boldsymbol{N} \wedge \forall\ i \in [1,m], i \in \boldsymbol{N}, \sigma_i \in \boldsymbol{R}_0^{+} \cup \infty$,

$vi \in \{0, 1\} \}$, and

$\sigma \in \boldsymbol{R}_0^{+} \cup \infty$

$\}$ ;

**N** $\in \{0, 1\}^{\eta}$, is the set of the neighborhood's binary states;

**d** $\in \boldsymbol{R}_0^{+}$ is the transport delay for the cell;

**δint**: $S \rightarrow S$ is the internal transition function;

**δext**: $QxX \rightarrow S$ is the external transition function, where Q is the state set defined by

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

**τ**: $N \rightarrow N$ is the local computing function;

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

**D**: $S \rightarrow \boldsymbol{R}_0^{+} \cup \infty$, is the state's lifetime function.

The present definition is independent of the simulation technique used. Therefore, it allows to specify the system behavior independently of the implementation details.

The specification allows to define a cell as a modular atomic model. Each cell has a well-defined interface whose definition is based on the modular specification presented in [GE97]. The interface is composed of a fixed number of ports, each one connected with a neighbor, and the number of input and output ports of each cell is equal to the neighborhood size.

Each port in the interface has a name and a type. Their names are composed of an identifier (**X** for input; **Y** for output) and a natural number (port number). As the cells being defined are binary, the type for each cell port is also binary. Input and output sets for each cell are also binary.

The cell's state is defined as a set composed of:

- The cell's present value (a binary variable);
- The **σ** variable, representing the simulated time for the next scheduled internal event;
- A FIFO queue (**σqueue**), used to keep track of the next event times and their associated input values; and
- The model's phase, that can be active or passive.

The **N** set represents the binary states of neighbor's cells, represented as a k-uple $(s_1,..., s_k)$ with $s_i \in \{0,1\}$. This set records the present values used to compute the future state value through the boolean function **τ**. The atomic model's behavior should be independent of that of other cells, but the local computing functions need to use their present values. Therefore, neighbors' state values are included in the cell's specification. These values should be kept by the simulation mechanism.

A transport delay can be associated with each cell, allowing to defer the execution of the internal transition functions.

The state's lifetime function **D** is used to keep track of the elapsed time for a cell state. Here, **D**(s, phase, σqueue, σ) = t, represents the time the model stays in the state **s** if there are no external events. The state's lifetime function (and the delay) has domain in the real numbers as the time base selected is continuous.

Finally, transition (**δint**, **δext**) and output (λ) functions are included. The functionality of these transition functions is similar to those of other DEVS models. The internal transition function is used to define state changes due to internal events, and the external transition function expresses the occurrence of external events. Instead, the semantic of these functions is different.

For DEVS models, a state **s** where **D**(**s**) = ∞ is said passive (a non passive state is said active). The internal transition function **δint** specifies a state change in the model after the time given by **D**. If an external event occurs in (**t+e**), the external transition function **δext** executes, and the model changes immediately to the state **s'= δext**(**s**,**e**,**x**). The variable **e** is the elapsed time in the current state, and is reset to zero at each state change.

Instead, the transport delays allow to defer the state change upon the occurrence of an external event. Only when the transport delay is consumed, the internal transition function is executed and the system state changes.

A cell will be active while external events are received or internal events are scheduled. The cell passivates only when there are no further scheduled events, that is, when the **σqueue** is empty. The value of the **σ** variable is equal to the first value of this queue.

The semantic for the external transition function is that if an event arrives, a neighbor has changed and the cell must compute the local computing function. If the transport delay for the cell is 0, the result of the computation is output immediately. Instead, if a delay exists, the external transition function must schedule the internal transition function. To do so, it uses the transport delay value, storing the delay and input values in the **σqueue**. If the cell is in passive state, it must be activated. If the cell is active, the values of σ stored in the queue must be updated to reflect the elapsed time e.

```
δext((s, phase, σqueue, σ), e, x) = (s', phase, σqueue, σ), where
{
    Update the local copy of the neighborhood with the x value;
    s' = τ(N);          /* Compute the local computing function */

    if (s' != s) then {        /* Output must be issued only if the state changes */

        s = s';
        if (phase = passive) then     /* Cell is passive. It must be activated */
                    phase = active;
                    σ = d;                      /* Transport Delay */
        else
              for (ai ∈ σqueue)  ai.σ = ai.σ - e;      /* Cell is active. The elapsed time
              σ = σ - e;                                 must update future values of σqueue */
        endif
        σqueue = insert(σqueue, <s, d>);
    endif
}


δint(s, phase, σqueue, σ) = (s', phase, σqueue, σ), where
{       /* σqueue times must be updated */
        for (ai ∈ σqueue)  ai.σ = ai.σ - first(σqueue.σ);
        σqueue = tail (σqueue);

        if empty(σqueue) then    /* No more scheduled events */
              phase = passive;
              σ = ∞;
        else    /* schedule next event */
              phase = active;
              σ = first(σqueue.t);
        endif
}

λ(s) {
  return s;
}
```

***Figure 4. Definition of δint, δext and λ for TDC models.***

The output and internal transition functions are activated when σ=0. As the transport delay has expired, therefore the local computing function must be executed, and the cell's state updated if necessary. The internal transition function removes the first member of the queue (whose value is the same that of σ).
The output function, λ, is executed before the **δint** function as in other DEVS specifications. The output function should only be executed if the cell's state has changed. This occurs because a change in a cell can produce state changes in the influencees (i.e., the neighborhood). Conversely, if the state of a cell does not change, its neighbors cannot change either. Hence, the cell remains in a quiescent state.

An example of the behavior of both transition functions can be seen in Figure
***Figure 4***. This cell (previously presented in [WFG97a] as an extension of the work done by [GG96]) represents an ideal boolean function.

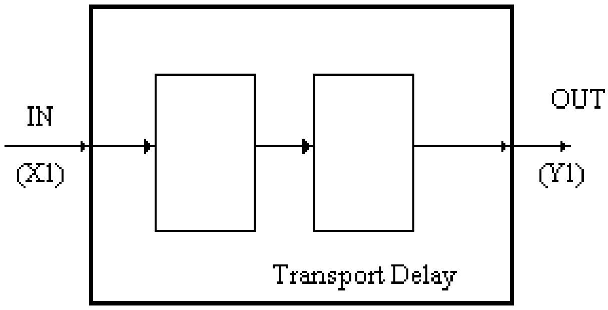



***Figure 5. Boolean function with transport delay***

Let us consider a transport delay of 17 time units. In this case we have:

$I = <0, P^x, P^y>$,
where $\eta = 0$, $P^x = \{X1\}$, $P^y = \{Y1\}$;

$X = Y = \{0,1\}$;
$d = 17$;
and $\tau(s) = s$.

Let us suppose the following input/output trajectories:

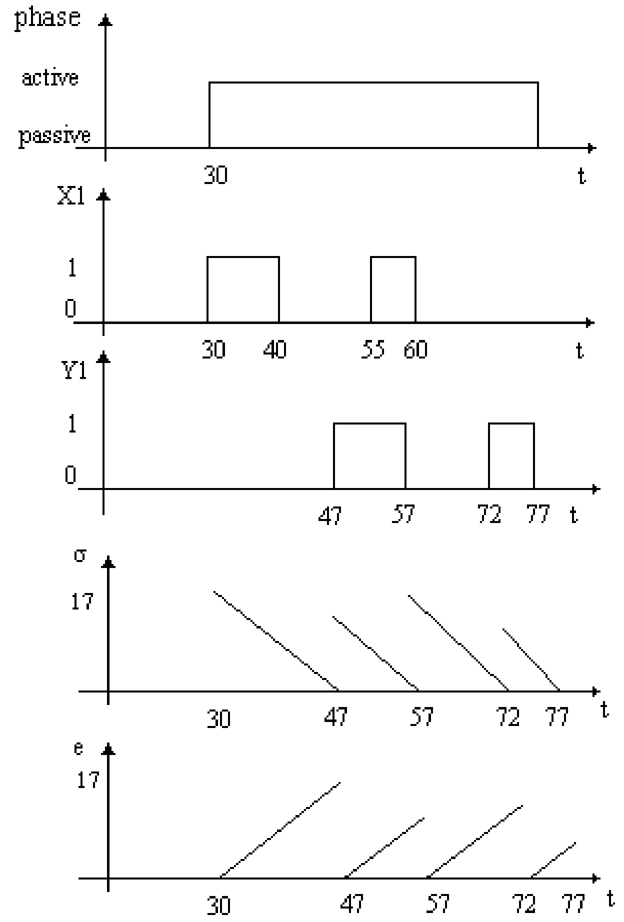



***Figure 6. Input and output trajectories with transport delay.***

The cell's behavior for these trajectories 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.

| | 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.*

The present and future states are included in the columns tagged s and s' respectively. The fields containing two values separated by a slash represent the variable values before and after execution. The output values will be the state of the **s** variable in the lines corresponding to internal transition functions.

## Formal specification of closed coupled Cell-DEVS models with transport delays.

The atomic transport delay cell models presented in section 0 can be coupled with other cell models, forming a multicomponent or coupled model. In this section, a formal description of a Cell-DEVS coupled model with transport delays is specified. These models will be defined as a space consisting of DEVS cells connected by a neighborhood relationship. The model is closed (i.e., it cannot be coupled with other base models). As no inputs or outputs will be used, there is no need to include an interface.

The main goal of this section is to define the coupling of atomic models such as those presented in section 0, allowing the ulterior definition of more general models. A conceptual executable model of coupled DEVS cells with transport delays can be specified as:

$$CTD = < \eta, N, \{m, n\}, C, B, Z, select >$$

where

$\eta \in \mathbf{N}$ is the neighborhood size;

**N** is the neighborhood set, defined as

$$N = \{ (i_p, j_p) \,/\, \forall\, p \in \mathbf{N},\, p \in [1,\eta] \Rightarrow i_p, j_p \in \mathbf{Z} \wedge i_p, j_p \in [-1, 1] \};$$

**{m, n}** $\in \mathbf{N}$ is the size of the cell space;

**C** is the cell space set, defined as $C = \{C_{ij} \,/\, i \in [1,m], j \in [1,n]\}$, where

$$C_{ij} = < I_{ij}, X_{ij}, Y_{ij}, S_{ij}, N_{ij}, d_{ij}, \delta int_{ij}, \delta ext_{ij}, \tau_{ij}, \lambda_{ij}, ta_{ij} >$$

is a Cell-DEVS component such as those defined in section 0;

**B** is the border cells set, where

- $B = \{\varnothing\}$ if the cell space is wrapped; or
- $B = \{C_{ij} \,/\, \forall\, (i = 1 \vee i = m \vee j = 1 \vee j = n) \wedge C_{ij} \in C\}$, where

$$C_{ij} = < I_{ij}, X_{ij}, Y_{ij}, S_{ij}, N_{ij}, d_{ij}, \delta int_{ij}, \delta ext_{ij}, \tau_{ij}, \lambda_{ij}, ta_{ij} >$$

is a Cell-DEVS component, such as those defined in section 0 if the atomic border cells have different behavior than the rest of the cell space.

**Z** is the translation function, defined by:

$$Z: P_{ij}^{Y_q} \rightarrow P_{kl}^{X_q}, \text{ where } P_{ij}^{Y_q} \in I_{ij}, P_{kl}^{X_q} \in I_{kl}, q \in [0,\eta] \text{ and } \forall (f,g) \in N, k = (i+f) \bmod m; l=(j+g) \bmod n;$$

$$P_{kl}^{Y_q} \rightarrow P_{ij}^{X_q}, \text{ where } P_{kl}^{Y_q} \in I_{kl}, P_{ij}^{X_q} \in I_{ij}, q \in [0,\eta] \text{ and } \forall (f,g) \in N,$$
$$k = (i-f) \bmod m; l = (j-g) \bmod n;$$

and

**select** is the tie-breaking selector function, with the restriction that select $\subseteq mxn \rightarrow mxn \,/\, \forall\; E \neq \{\varnothing\}$, select(E) $\in$ E.

Here, the cell space C is a coupled model defined as a fixed size (m x n) array of atomic cell models. The definition characterizes two dimensional cell spaces, but it can be extended to n-dimensions. In conceptual cellular automata (such as those defined in section 1), m and n parameters were not needed as cell spaces were supposed infinite. In this case, the size must be specified because the definition of an executable conceptual model is being considered.

Each cell has a set of $\eta$ neighbor cells, defined by the neighborhood set (N). The set is represented as a list of pairs defining the relative position between the neighbor and the origin cell. This definition only allows regular neighborhoods and neighbors in adjacent cells. If other kinds of neighborhoods are to be defined (including different neighborhoods for each cell), the definition must be extended to allow them. In this case, N is defined like an array of neighborhood's lists that can be formally specified as:

$$N = \{N_{kl} \mathbin{/} k \in [1,m], l \in [1,n]\}$$

where

$$N_{kl} = \{(i_p, j_p) \mathbin{/} \forall p \in \boldsymbol{N},\ p \in [1,\eta_{kl}],\ i_p, j_p \in \boldsymbol{Z};\ |k - i_p| \geq 0 \wedge |l - j_p| \geq 0 \wedge \eta_{kl} \in \boldsymbol{N}\}$$

In this case, each cell will have a neighbor list of $\eta_{kl}$ elements. This list is composed of pairs of indexes relative to the origin cell (that cannot exceed the cell space limit).

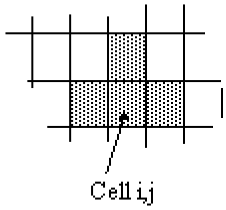
The B set defines the cell's space border, and it can be defined in two ways. If B = {∅}, every cell in the space has the same behavior. Cells in one border are connected with those in the opposite one using the neighborhood relationship. Otherwise, the border cells will have different behavior than those of the rest of the model. They can, for instance, self-generate their state, or consume the state of their neighbors.

Finally, the Z function allows to define the coupling between cells in the model. This function translates the outputs of m-eth output port in cell $C_{ij}$ into values for the m-eth 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 [Zei76].

The ports' names are generated using the following notation: $P_{ij}{}^{X_q}$ refers to the q-eth input port of cell $C_{ij}$, and $P_{ij}{}^{Y_q}$ to the q-eth output port. These ports correspond with the port names denoted as $X_q$ or $Y_q$ for each cell, as mentioned in section 0. The number of the cell to be coupled with, will be generated by adding the numbers in the neighbor's list to the present cell number. The first output port of a cell will be connected the first input port of the neighbor, according with the order of the neighbor list.

A sketch of this procedure can be seen in Figure 7. Figure 7(a) shows the neighborhood of cell (i,j) and its representation using the neighbor's list. Figure 7(b) shows how the first output port of cell (i,j) is connected with the first input port of the first neighbor in the list; the second port with the second neighbor, etc. Instead, for the input ports, the connection is done through the inverse neighborhood list. For each pair (i,j) in the neighborhood, the pair (-i, -j) must be included in this list.

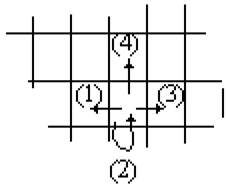
This procedure allows to define complete cell spaces in a parametric fashion. In following sections, this framework will be used to define the coupling other kind of cell spaces.




Neighborhood list:
{ (0, -1), (0,0), (0,1), (-1,0)}

**Note:** -1: left, up;
1: right, down

(a)



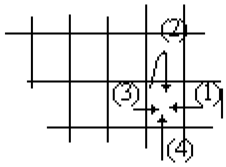

$P_{ij}\ Y_1 \rightarrow P_{i,j-1}\ X_1$ (1)

$P_{ij}\ Y_2 \rightarrow P_{ij}\ X_2$ (2)

$P_{ij}\ Y_3 \rightarrow P_{i,j+1}\ X_3$ (3)

$P_{ij}\ Y_4 \rightarrow P_{i-1,j}\ X_4$ (4)

(b)




Inverse Neighborhood list:
{ (0, 1), (0,0), (0, -1), (1, 0)}

$P_{ij}\ X_1 \leftarrow P_{i,j+1}\ Y_1$ (1)

$P_{ij}\ X_2 \leftarrow P_{ij}\ Y_2$ (2)

$P_{ij}\ X_3 \leftarrow P_{i,j-1}\ Y_3$ (3)

$P_{ij}\ X_4 \leftarrow P_{i+1,j}\ Y_4$ (4)

(c)

***Figure 7. (a) A cell, its neighborhood and the neighbor's list; (b) connection of the output ports of cell i,j (using the neighborhood list); (c) connection of the input ports of cell i,j (using the inverse neighborhood list).***

## Formal specification of general coupled Cell-DEVS models with transport delays.

In this section, the definition for coupled models is extended to permit the specification of Cell-DEVS spaces that can be mixed with other basic models in a DEVS hierarchy. For instance, a Cell-DEVS model could be coupled to other with different neighborhood or cell behavior, or to non-cell DEVS models. This definition allows to build complex models consisting of several submodels with different behavior using different paradigms or abstraction levels.

To permit multiple definition of submodels and their coupling, the formal definition of the previous section must be extended. A Cell-DEVS model can be represented as:

$$GCTD = < I, X, Y, Xlist, Ylist, \eta, N, \{m, n\}, C, B, Z, select >$$

Here, the definition for the sets previously defined in section 0 remains unchanged, and the following sets have been added or modified:

**I** = < $P^X$, $P^Y$ > represents the definition of the modular model interface. Here,

for $i$ = X | Y, $P^i$ is a port definition (input or output respectively), where

$P^i = \{ (N(f,g)^i, T(f,g)^i) \ / \ \forall (f,g) \in$ Xlist, $N(f,g)^i = i(f,g)_k$ (port name), and $T(f,g)^i$ = binary (port type)};

***X*** = {0, 1} is the set of input external events;

***Y*** = {0, 1} is the set of output external events;

**Ylist** = { (k,l) / k ∈ [0,m], l ∈ [0,n]} is the list of output coupling;

**Xlist** = { (k,l) / k ∈ [0,m], l ∈ [0,n]} is the list of input coupling; and

**select** = { (k,l) / (k,l) ∈ N } is the tie-breaking selector function,.

The present specification allows automatic definition of a complete DEVS model consisting of with different submodels. This specification is completely independent of the simulators, and can be used to validate the model's correctness. Different phenomena can be studied with several approaches. Each submodel can be built by using different modeling techniques; each better adapted to analyze different behaviors.

Several differences can be found with the closed Cell-DEVS specification previously presented. First, as in any coupled DEVS model admitting inputs and outputs, sets *X* and *Y* are included. As these models can be coupled with other DEVS models, the interface **I** has also been defined. The interface here presented only allows binary inputs and outputs, as the Cell-DEVS automata being considered are binary. The specification of the interface should be redefined to allow other discrete states.

The **Z** function defined in the previous section is still used to carry out the internal coupling of the cell space. Finally, two new sets were included. **Xlist** is a list of cell's positions where the model's external events are received. **Ylist** is a list of cell's positions whose outputs will be collected to be sent to other models in the hierarchy.

The **select** function is defined as a list of positions in the neighborhood. The list is ordered according the selection criteria to be used when more than one cell is active simultaneously.

In DEVS coupled models, the occurrence of internal events in a model, is preceded by the execution of the output function λ. This function outputs the value to other models through the ports in the interface. In Cell-DEVS models, the behavior will be different. First, if an event occurs in a cell, its neighbors are automatically influenced through the execution of the Z function. Besides, certain cells in the space are chosen as input and output cells, and will be included in the Xlist and the Ylist respectively. The values of these cells will be considered the inputs and outputs of the complete cell space.

The definition for DEVS coupled models has been changed to allow the definition of cell spaces. If the model considered is a cell space, the coupling uses the internal and external definition of input and output cells. Therefore, DEVS coupled models can be defined as:

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

where

***X*** is the set of input events;
***Y*** is the set of output events;

**D** is an index for the components of the coupled model, and

$\forall i \in D$, $\mathbf{M}_i$ is a basic DEVS model, where

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

and

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

$\mathbf{Z}_{ij}$ is the i to j translation function, where

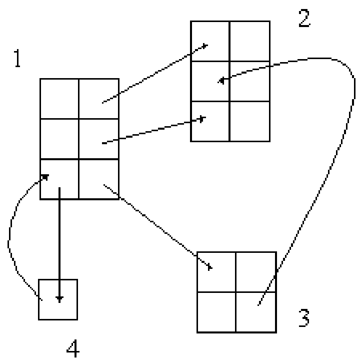
$Z_{ij}$: $Y_i \rightarrow X_j$ if none of the models involved are Cell-DEVS, or
$Z_{ij}$: $Y(f,g)_i \rightarrow X(k,l)_j$, with $(f,g) \in Ylist_i$, and $(k,l) \in Xlist_j$

if either of the models i or j is a general Cell-DEVS model.

Finally, **select** is the tie-break selector.

When a Cell-DEVS model is executed, the $Z_{ij}$ function translates the outputs into inputs between one cell and the models external to the cell space by using the two previously defined lists.

To exemplify the external coupling definition, let us suppose that the models seen in Figure 8 must be connected. The coupling will be done following the $Z_{ij}$ function definition uses the contents of Xlist and Ylist, as it can be seen in the figure.




$Xlist_1 = \{ (3,1) \}$
$Ylist_1 = \{ (1,2), (2,2), (3,2) \}$
$Xlist_2 = \{ (1,1), (2,1), (3,1) \}$
$Ylist_2 = \{\varnothing\}$
$Xlist_3 = \{(1,1)\}$
$Ylist_3 = \{(2,2)\}$

$Y(1,2)_1 \rightarrow X(1,1)_2$
$Y(2,2)_1 \rightarrow X(3,1)_2$
$Y(3,2)_1 \rightarrow X(1,1)_3$
$Y(3,1)_1 \rightarrow X_4$
$Y_4 \rightarrow X(3,1)_1$
$Y(2,2)_3 \rightarrow X(2,1)_2$

(a) (b) (c)

***Figure 8 - Example of connection using Zij function for Cell-DEVS spaces (a) Basic models; (b) Xi and Yi lists for each model; (c) Zij coupling.***

The names of the input and output ports are also defined by using the contents of the Xlist and Ylist. The port names will be automatically generated by using an identifier (X for input, Y for output) and a cell position.

As we can see, the present definition for coupled models only allows binary states. The definition can be extended by considering *X* and *Y* (and the corresponding I/O ports) as sets in ***Z*** or ***R***. The transition functions should compute their results in any of these domains.

The definition of the Cell-DEVS models is equivalent to those of other DEVS coupled models. In this case, D = m x n, $\{M_i\}$ is defined by m, n, C and B, Ii is defined by N, and $Z_{ij}$ is defined by Z and $Z_{ij}$..

## CELLULAR MODELS WITH INERTIAL DELAYS

In several cases, the introduction of the Inertial Delays [GM76] can help when modeling complex systems. The construction allows to represent a behavior with preemptive semantic for a cell. The semantic for this construction is that an input must be discarded if its value is not kept for a certain period. Instead, if the input flow is steady during that time (called the inertial delay for the cell) the state must change. Figure 9 depicts such behavior for an atomic cell.

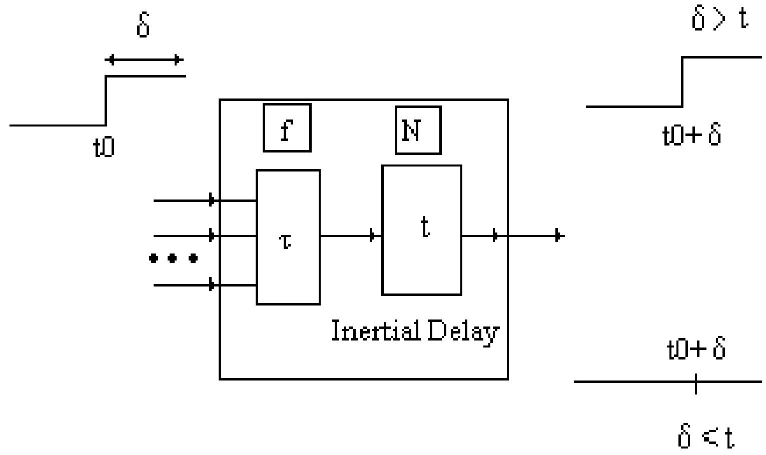



***Figure 9. Behavior of an atomic cell model with inertial delay.***

This construction is useful to represent certain physical phenomenam 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.

Besides 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 output to the neighbor cells, but will influence them only if the value remains for the inertial delay. Instead, if it does not sustain for a certain time, it should not expand (allowing to represent, influences of the wind, rain, or firefighters).

The formal specification of a cell with inertial delays is the same that the one presented in section 0 for transport delays. The main change is a different semantic for the variable **d** that is now the value for the inertial delay. Another change is the representation for a state value. In this case,

S = { (s, phase, f, $\sigma$) /
$s \in \{0,1\}$,
phase $\in$ {active, passive},
$f \in \{0,1\}$, and
$\sigma \in \boldsymbol{R}_0^+ \cup \infty$
} ;

where s, phase and $\sigma$ have the same meaning previously defined, and f states for the feasible future state for the cell. If the input value for the cell is kept during the inertial delay, the future state will be f.

The main change for the definition is the transition functions' behavior. When modeling inertial delays, the functions must be redefined as follows:

```
δext((s, phase, σqueue, σ), e, x) = (s', phase, σqueue, σ)
{
        s' = τ(f, N);
        Update the local copy of the neighborhood with the x value;

        if (s != s') then {
          s= s';

          if (phase = passive) then
                phase = active;
                σ = d;    /* Inertial Delay */
          else
                σ = σ - e;
                if (σ > 0 .AND. f != x)  σ = d;  /* Preemption */
          endif

        f = x;
}
δint(s, phase, σqueue, σ) = (s', phase, σqueue, σ)   {
        if (σ = 0) then
        phase = passive;
        σ = ∞;
}

λ(s)  {
        return s;
}
```

***Figure 10. External and internal transition function for inertial delays models.***

As we can see, the external transition function records the present value for the cell and detects if the input is conserved during the inertial delay. If not, the previous input value is preempted.

The behavior for atomic cells with inertial delays can be studied in the following example. The input and output trajectories presented in Figure 11 (based on the same atomic cell used in section 0) are considered using an inertial delay of 5 time units.

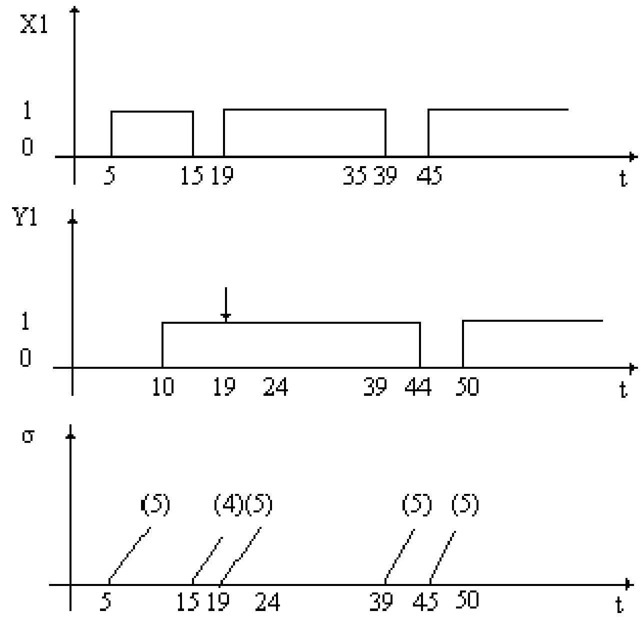



***Figure 11. Input and output trajectories with inertial delays.***

In Table 2, 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 preemptions.

| | 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 11.*

The last arrived future event can be preempted if there is a new input before the consumption of the inertial delay. This happens only if the new external value is different of that one previously stored. If both values are the same, the new external event occurred has the same value than the previous one.

This phenomenon is similar to have a unique external event with longer inertial delay. An example can be seen in simulated time 24 of the previous table. In instant 19, a new external event with value 1 arrived. The system state at that moment was 1 (as the previous external event was preempted), therefore, the input is not explicitly output.

## THREE-STATES TRANSPORT DELAY MODELS

In previous sections, the interface and behavior of binary Cell-DEVS models with different delays have been formally described. In several cases, the introduction of a third state reflecting undefined behavior for the cell is useful. This state will be denoted as $\phi$, and it will serve to define the behavior of those cells whose binary state is unknown. This fact occurs because the delay values are not known with precision, but only a range of delay values can be determined with precision.

The present section defines the behavior of three-state Cell-DEVS spaces with different kind of delays. These models can be formally described as:

$$TDCD = < I, X, Y, delay, S, d, \delta int, \delta ext, \tau, \lambda, D >$$

where

**I** = $<\eta, P^X, P^Y>$ represents the definition of the modular model interface. Here,

$\eta \in \boldsymbol{N}$ is the neighborhood size, and,

for $i$ = X | Y, $P^i$ is a port definition (input or output respectively), where

$$P^i = \{ (N_j^i, T_j^i) \ / \ \forall j \in [1, \eta],\ N_j^i \in [i_1, i_\eta] \text{ (port name), and } T_j^i = \text{binary (port type)}\};$$

$\boldsymbol{X}$ = {0, 1, $\phi$} is the set of input external events;

$\boldsymbol{Y}$ = {0, 1, $\phi$} is the set of output external events;

**delay** $\in$ {transport, inertial, none}.

**S** is the state set, where

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

s $\in$ {0,1, $\phi$},

phase $\in$ {active, passive},

$\sigma$queue = { ( $(v_1,\sigma_1)$,..., $(v_m,\sigma_m)$ ) / $m \in \boldsymbol{N} \wedge \forall i \in [1,m], i \in \boldsymbol{N}$,

$a_i \in \boldsymbol{R}_0^+ \cup \infty$, vi $\in$ {0, 1} }, and

$\sigma \in \boldsymbol{R}_0^+ \cup \infty$

} if delay = transport, or

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

s $\in$ {0,1},

phase $\in$ {active, passive},

f $\in$ {0,1}, and

$\sigma \in \boldsymbol{R}_0^+ \cup \infty$

} if delay = inertial;

**N** $\in \{0, 1, \phi\}^\eta$, is the set of the neighborhood's binary states;

**d** $\in \boldsymbol{R}_0^+$ is the transport delay for the cell;

**$\delta$int**: S$\rightarrow$S is the internal transition function;

**$\delta$ext**: QxX $\rightarrow$ S is the external transition function, where Q is the state set defined by

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

**$\tau$**: N $\rightarrow$ N is the local computing function;

$\lambda$: S$\rightarrow$*Y* is the output function; and

**D**: Q $\rightarrow \boldsymbol{R}_0^+ \cup \infty$, is the state's lifetime function.

As we can see, the definition is similar to that of section 0 but including the third state. The description for coupled models with three states will not be included because the coupling can be done using the specifications presented in sections 0. and 0. These formalisms also allow to describe multicomponent models with three states. The main changes to the description are in the state, input and output sets, where the unknown state is included. The cell space can also include a kind of delay unique for all the cells in the space. According with the delay chosen, the transition functions of section 0 or section 0 will be activated.

Let us consider now the behavior of the ideal gate with transport delay presented in Figure 5. Let us suppose a transport delay of 15 time units. The execution of the transition functions and the behavior of the S set can be seen in the following figure:

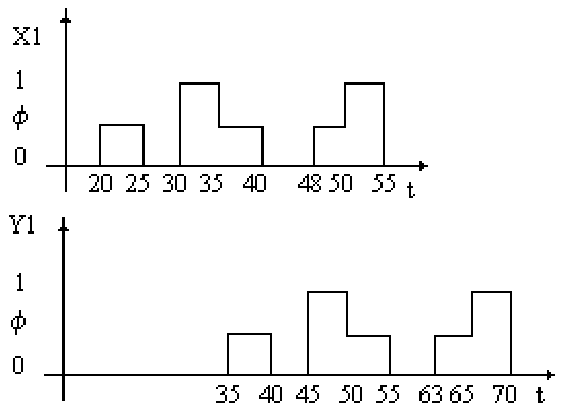



***Figure 12. Input and output trajectories of three-state cells with transport delays***

| | t | s | s' | phase | σ | e | σqueue |
|---|---|---|---|---|---|---|---|
| | ... | 0 | 0 | passive | | | |
| | 20 | 0/ϕ | ϕ | active | 15 | 0 | (15,ϕ) |
| | 25 | ϕ/0 | 0 | active | 10 | 5 | (10,ϕ), (15,0) |
| | 30 | 0/1 | 1 | active | 5 | 10 | (5,ϕ), (10,0), (15, 1) |
| * | 35 | 1 | 1 | active | 0/5 | 15/0 | (5,0), (10, 1) |
| | 35 | 1/ϕ | ϕ | active | 5 | 0 | (5,0), (10, 1), (15,ϕ) |
| * | 40 | ϕ | ϕ | active | 0/5 | 5/0 | (5, 1), (10,ϕ) |
| | 40 | ϕ/0 | 0 | active | 5 | 0 | (5, ϕ), (10,ϕ), (15,0) |
| * | 45 | 0 | 0 | active | 0/5 | 5/0 | (5,ϕ), (10,0) |
| | 48 | 0/ϕ | ϕ | active | 2 | 3 | (2,ϕ), (7,0), (15,ϕ) |
| * | 50 | ϕ | ϕ | active | 0/5 | 5/0 | (5,0), (13,ϕ) |
| | 50 | ϕ/1 | 1 | active | 5 | 0 | (5,0), (13,ϕ), (15,1) |
| * | 55 | 1 | 1 | active | 0/5 | 5/0 | (8, ϕ), (10,1) |
| | 55 | 1/0 | 0 | active | 8 | 0 | (8, ϕ), (10, 1), (15, 0) |
| * | 63 | 0 | 0 | active | 0/5 | 8/0 | (2, 1), (7, 0) |
| * | 65 | 0 | 0 | active | 0/5 | 5/0 | (5,0) |
| * | 70 | 0 | 0 | passive | ∞ | | |

*Table 3. Execution sequence of the three-state transport delay cellular model.*

# IMPLEMENTATION MODEL FOR DEVS-CELLS

In previous sections, formal specifications for conceptual models based on Cell-DEVS models have been presented. This section is devoted to analyze several issues related with the development of implementation models following the conceptual framework recently introduced.

Two points of view were considered. First, the user point of view has been taken into account to produce safe simulations. A Cell-DEVS simulation environment should provide facilities to execute implementation models specified accordingly with the formalisms previously defined. The user should be able to specify a conceptual model by using any of the previous definitions, and that specification should be executable. In this way, if the specification is sound, the correctness of the executable model is automatically verified.

This approach allows to specify the system model formally, and to validate it against the real system behavior. As the execution model follows the formal specification (if the simulation environment is correctly built), the simulators will execute correctly. The main issues related with the user's point of view will be explained in section 0.

Besides, the point of view of the designer of the modeling environment has been considered. The environment should execute Cell-DEVS models specified by the formalisms, and to do so, previous existing definitions for DEVS model simulation have been considered. Section 0 will be devoted to explain several implementation issues considering the designer point of view. Zeigler's class hierarchy for object-oriented modeling environments [Zei90] was modified, allowing to execute the implementation models for Cell-DEVS spaces efficiently.

## Definition of a Cell-DEVS implementation model.

The user of a simulation environment for Cell-DEVS modeling must be able to represent three basic aspects of the cell space: dimension, influencees and behavior. The dimension and influencees' issues are related with the definition for the cell space. Instead, the behavior aspect is related with the definition for each atomic cell. This section is devoted to present a tool that allows the definition of those aspects, that was defined in [WFG97] and it is being currently implemented [BBW97b]. The environment was built to follow the conceptual definitions previously studied in sections 0 and 0.

The tool lets the user to specify a cell space by defining the three mentioned basic aspects. Regarding the **dimension** facts, the user should define the size for the cell space (m and n parameters of the formal specification).

Afterward, the **influencees** of a cell must be specified by defining the cell's neighborhood. Input and output ports for each cell will be automatically created by following the procedures stated in the formal definition. The coupling of cells will be automatically specified based on the neighborhood definition by using the procedure presented in section 0. When the cell space can be coupled with other DEVS models, the external connections also should be specified. Therefore, the contents of the $X_i$list and the $Y_i$list sets should be fulfilled. Using these parameters, a complete space of cells can be generated, and its coupling defined.

The remaining parameters must be completed to define the behavior of the cell. This includes the kind of delay (transport, inertial, none), its duration, and the behavior of the local computing function.

To represent the **behavior** of the local computing function for each cell, a simple specification language has been defined (whose complete BNF has been included in the Appendix). A compiler will translate the specification to the internal behavior representation of the Cell-DEVS models using a procedure described in section 0.

The specification for one cell will be defined as a list of sentences with the following format:

$$\{0 \mid 1 \mid \phi\} \text{ (Expression)}$$

Each sentence is a logical expression relating several boolean or integer expressions. The language only allows binary or three-states. The expressions can include relational operations and calls to predefined functions. The results for each cell are computed by analyzing each sentence, and considering the present state for the cell and its neighborhood. The main operators and functions that can be used are presented in the following table:

| **Operators** | **Behavior** |
|---|---|
| <, <=, >, >= | Inequality operators |
| =, != | Equality operators |
| AND, OR, NOT | Boolean operators |
| ( ) | Precedence operator |
| +, -, *, /, ^ | Arithmetic operators |
| **Functions** | |
| one_count(); | Returns the number of neighbors with their state in one |
| zero_count(); | Returns the number of neighbors with their state in zero |
| fi_count(); | Returns the number of neighbors with their state undefined |
| **Delays** | |
| tr_delay(d); | Delays the cell execution in d units using a transport delay |
| in_delay(d); | Delays the cell execution in d units using an inertial delay |

*Table 4. Main operators and functions to define cell behavior*

The dimension and influencees' parameters for the cell space are also used when defining the cell's behavior. In this way, the user of the environment can characterize a complete implementation model without programming.

As the environment will follow the formal definitions, the procedures described in previous sections serve as a basis to complete the cell space specification. First, the interfaces are defined as presented in sections 0 and 0. The state's lifetime function **D** will be controlled by the simulation environment, as it will be the actualization of the local information for the neighbor's states. The **δint** and **δext** functions will have the behavior previously defined when each of the paradigms were presented.

The model specification will be completed by defining the behavior of the border cells. The cell space can be wrapped or the specification language can be used to provide different behavior for the border.

Finally, the priorities to treat simultaneous events must be specified, allowing to classify the imminent cells in the cell space. The priorities can be used to define the *Select* function of the DEVS specification (or the **δcon** function if the R-DEVS [ChZ94] formalism is used).

The mentioned parameters are used to create an array of atomic cell objects, and to define the influencees for each cell. When the parameters have been completed for cell space, the simulation can start. To do so, initial conditions for each cell should be defined. Also, the conditions to stop the simulation cycle (as a maximum evolution time for the simulation) should be included.

## Cell-DEVS spaces Simulation.

Once the cell space has been defined, the environment should simulate the coupled model. To do so, the approaches defined in [Zei84, Zei90] have been used, respecting the basic definition of classes to simulate DEVS-based models. Several previously defined tools used to simulate DEVS models have been used as a basis for the development [Zei90, Zei97, Cho95, GFE95, BBW97a]. The class hierarchies defined in those frameworks have been modified to allow the definition of a complete cellular model based on the previous formalisms. The goal is to define the model properties in a parametric fashion.

To permit the definition and simulation of DEVS cell spaces, the class hierarchy used to simulate DEVS models has been redefined. The changes of this definition are presented in Figure 13.

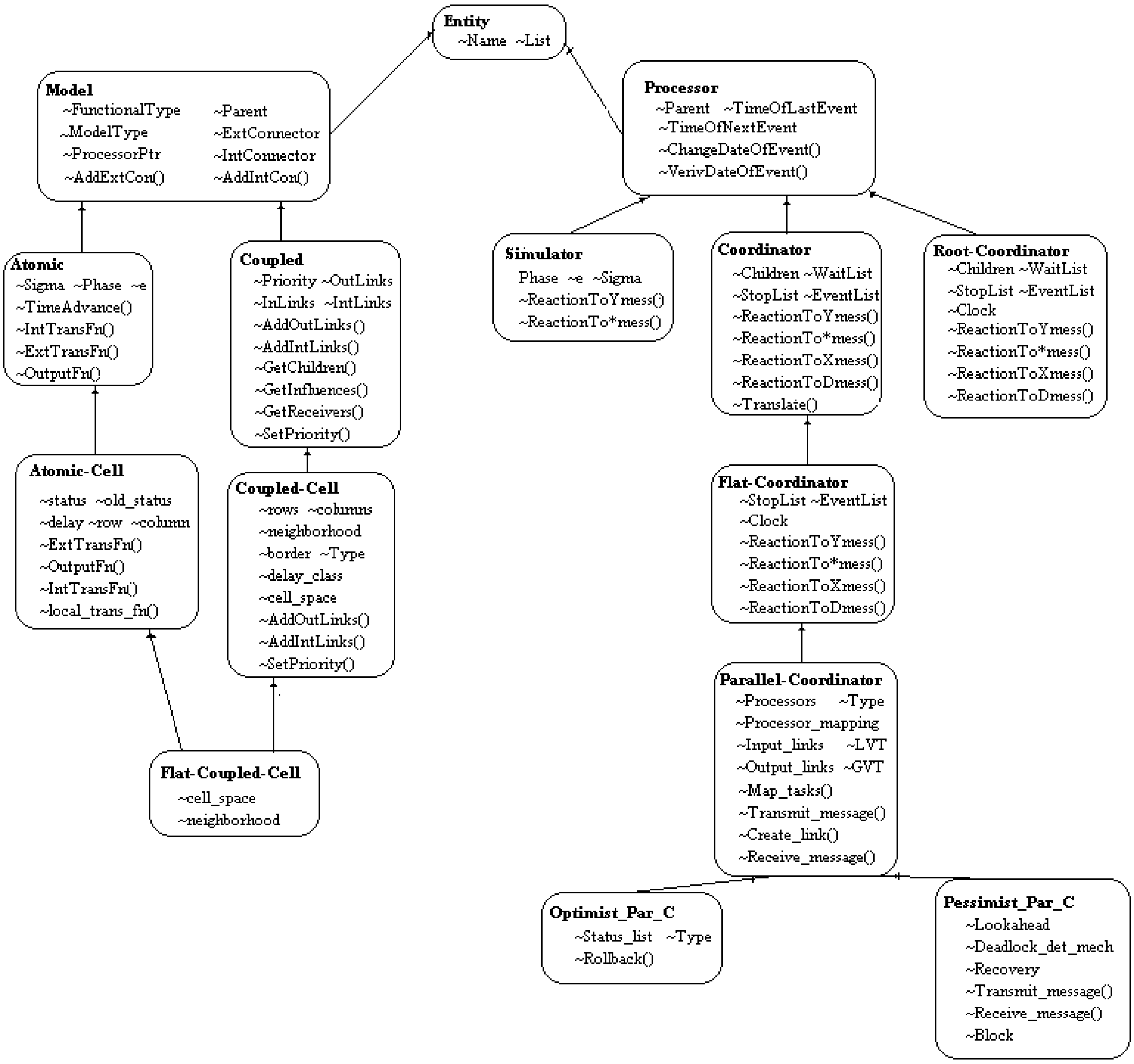



***Figure 13. Cell-DEVS class hierarchy.***

Recalling the functionality of the classes in the hierarchy, we can see two main classes derived of one base Entity class: Models and Processors. Model class allow to define the behavior for the models in the system. Each model can be Atomic or Coupled. The Atomic-Cell and Coupled-Cell classes act as interface between the user and the simulator environment. The instantiation of these classes allows to define the model specification presented in section 0.

Processor sub-classes, devoted to drive the simulation process also have been redefined. Several coordinators and simulators with different functionality were defined to simulate the behavior of cell spaces. Coordinators and simulators are associated with hierarchical coupled and atomic models, as in other DEVS environments. This is done in the way originally defined in [Zei90].

When a coupled cell model is defined, the set of processors associated with the cell space is automatically created. As the coupled model consists of several atomic ones (the cells), one simulator is associated with each cell. Considering the model's parameters, the simulators are created and their names defined. Also the internal coupling is automatically set up as presented in section 0, and a coordinator is associated with the coupled model.

The children for this coordinator are the simulators created using the present procedure. Each of the simulators has also a local copy of the cell's neighborhood. The initialization routines set up the contents of these local copies.

Simulation advances like in other DEVS-based simulation tools. When the imminent model is selected, a *-message is sent to its simulator, passing through the middle level coordinators. The coordinators now behave differently when detecting the message destinations, as they are composed of a cell and a model name. The model name is used by the high level coordinators and the cell position is employed by the cell space coordinator.

For Cell-DEVS spaces, the imminent model will be a cell in the space. Each of the coordinators is provided with a list of imminent children with this purpose. The simulator linked with that cell activates the model's output and internal transition functions, that will activate the local computing function.

When an external message arrives, a X-message is consumed and the external transition function executed. The simulators return done-messages and Y-messages that are converted to new *-messages and X-messages, respectively. The messages are translated using the cell coupling previously defined.

To allow the integration between flat models and other DEVS models, the Coordinator class should include the model coupling mechanism presented in section 0. The output ports in the interface of a modular model should be connected with the input ports in the other. To do so, the following sequence should be executed:

for (each (p,q) in Ylist)
  create and connect one Y port of cell $C_{pq}$ of model $M_i$ with a binary input port in model $M_j$;

for (each (p,q) in Xlist)
  create and connect a binary output port of model $M_i$ with one X port of cell $C_{pq}$ in model $M_j$.

To execute the cell model, the specification of the cell's behavior must be translated to an executable specification. To do so, a table-based approach has been taken, allowing the execution of different computing functions according with the cell space. Since the local computing function is constructed with a fixed behavior, it must not be reprogrammed for different models. Instead, a new tabular specification is created for each case.

Each expression in the specification is composed of one or several functions related by logical connectors. The translator will scan each expression, parse it, and place the results in a table with the following format:

| Result | Variables | | | | | | Operators | | | | | | Value | And |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $a_{11}$ | ... | $a_{33}$ | $\Sigma 0$ | $\Sigma 1$ | $\Sigma \phi$ | < | <= | > | >= | = | != | | |
| 0 | * | | | | | | | | * | | | | 7 | |
| | | | * | | | | * | | | | | | 5 | |
| 1 | | | | * | | | | | | | * | | 2 | * |
| | | | | | | * | | * | | | | | 5 | |
| ... | | | | | | | | | | | | | | |

*Table 5. Format of local computing function tables.*

Each column on the left represents one of the possible variables involved in the local computation. Columns tagged $a_{11}$ to $a_{33}$ are used to represent the content values for each neighbor. The other columns represent the result obtained when the predefined functions are executed. The variables on the left are related with the value on the right by using one of the existing operators. If the result of the boolean operation is true, the local computing function returns the result of the corresponding row.

Each expression will be inserted in a separate line expressing the desired operation, its operands, and the result to be obtained. Logical expressions connected by ORs will be splitted in two lines. Therefore, different sentences will be considered as connected by OR relational operators, and placed in separated lines. Expressions connected by ANDs will be copied in two following lines, but distinguished with a special mark.

When several subexpressions are connected by parenthesis, the corresponding result is placed in the line corresponding to the first expression.

The local computing function will scan the table one line at a time, verifying the logical expressions included. In this way, the new state value for the cell is obtained and updated.

Several errors in the specification can be found at runtime. If there are several results for a cell with the same original state and neighbors, the specification is ambiguous. Instead, if no result exists for a certain state, the cell specification is incomplete. If different expressions produce the same result, the simulation can continue, but the user must be notified. Instead, if different results are found, the simulation should stop because the future state of the cell cannot be determined.

If none of these situations occurs, the model is correct. The checking allows the detection of inconsistencies in the model definition. These can be reported to the user, that can change the specification to ensure its correctness. In this way, the validity of the model can be verified.

## Flat simulation of the cell spaces.

The DEVS paradigm allows hierarchical module definition, and the simulation mechanism also has hierarchical interaction. Special simulation entities (coordinators and simulators) drive the simulation, sharing information through message passing. This intermodule interaction produces a high degree of overhead. This overhead can reduce the execution performance of Cell-DEVS simulations, because these models are usually composed of a high number of cells.

One way to reduce the overhead and to avoid this interaction is by flattening the model hierarchy. This is the goal of the Flat_Coordinator class included in Figure 13.

A flat simulator is implemented as a bidimensional array of records, associated with the cell space. Each record will include information of the state and delay for the cell. A neighborhood list is used to record the cell's influencees for the cell space.

A neighbor's list stores the neighborhood shape as a list of cell positions relative to the origin cell. A Next-Event list (ordered by sigma value) is kept to record the cells scheduled to execute their transition functions. The cells' array is used to record the present cell's space state and to detect changes in the model. When a change is detected, the Next-Event list must be updated.

A New-States list is also necessary, as the states for every cell with the same time for internal or external transitions should be computed together. The values for a certain simulated time must be collected, and the results for the computations should be stored in the New-States list. The values recorded are used to update the cells array when all the computations for the same simulated time have finished. In this way, the flow of the global transition function for the cell space will be computed correctly.

Using this approach, the hierarchical message interaction is not needed, because the multiple processors are eliminated. The differences between both approaches can be seen in Figure 14.

The coordinator for flatten models will work as follows. It will start the simulation by detecting quiescent states for the initial state of the cell space. Non-quiescent cells (i.e., those whose state can change) will be added in the Next-Events list. The values will be inserted using the next event time.

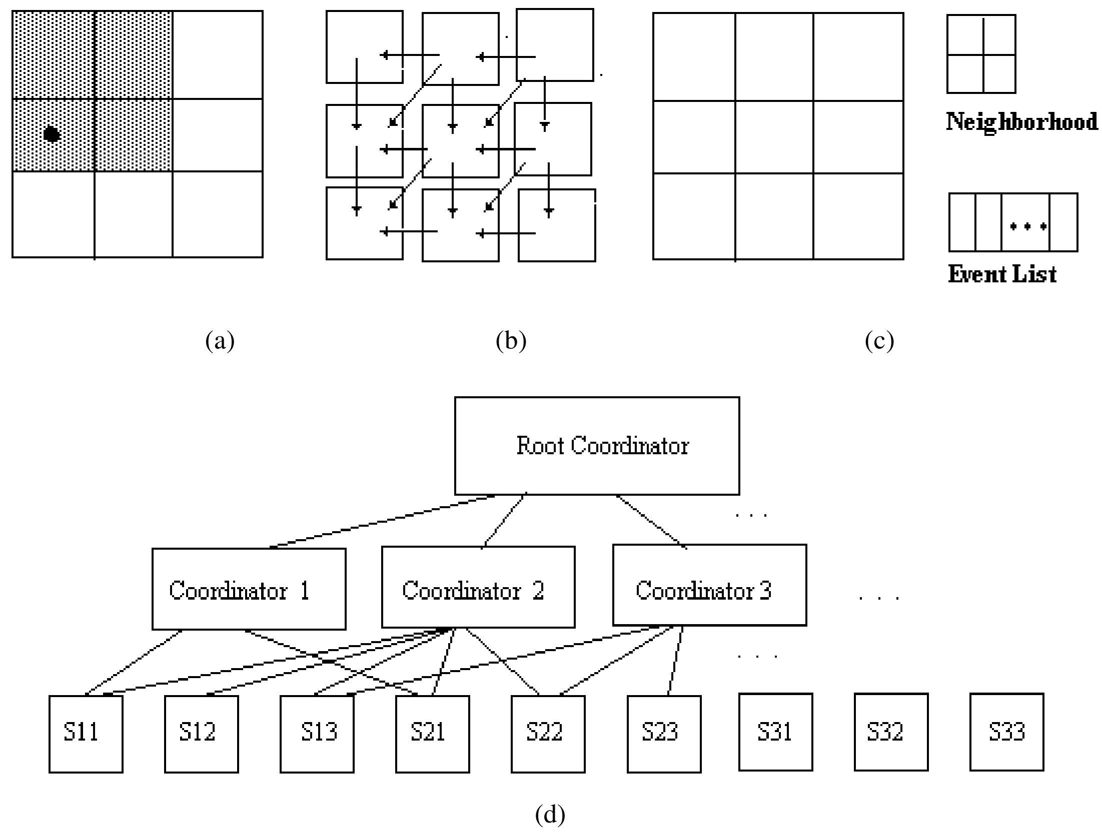
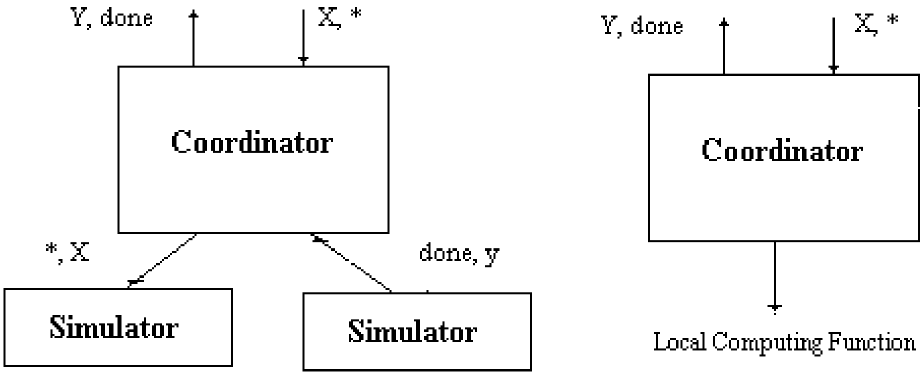



***Figure 14. Cell space structures. (a) Basic cell space and its neighborhood. (b) Coupling using hierarchical coordinators. (c) Definition using flat models. (d) Processors hierarchy for (b).***

After that, the coordinator will remove the first element of the Next-Events list. If the event considered is an internal event, the local computing function for the cell is activated. If the cell state has changed, its value is returned and the next event time is determined for the cell. The coordinator will create a Y-message containing the cell state value, and will transmit it to the upper level coordinator. Finally, a new message is created and added in the next-event list for each cell in the neighborhood.




(a) (b)

***Figure 15. Coordinator reaction to messages (a) Hierarchical (b) Flat.***

A flat coordinator is activated when a *-message is received, and activates the corresponding cell simulator. X-messages from other external models are simply sent to the simulator to be inserted in the Next-Event list (as any other external event). Another task of the simulator is to convert the input messages into the cell format. To do so, it uses the external coupling function $Z_{ij}$. Y-messages received from the flat simulator are also translated (in an inverse fashion), and transmitted to the upper level coordinators (see Figure 15). If the simulation finishes, a done-message is created.

The detailed behavior of the flat simulator for binary Cell-DEVS spaces (with transport or inertial delays), can be represented by the following algorithm:

```
Data structures:
Cells = { binary state; float delay; } [m x n];
Neighbors = { (int i, int j), i = -1..1; j = -1..1};        /* Relative cell's positions */
Next-Event list = { Cell_position (int i, int j), float time};
New-States list = { Cell_position(int i, int j), binary Future_state };
float tn, tl, e; /* time of next event , last event and elapsed time */

Begin
Load the initial state and delays for each cell in the array;
Detect non-quiescent cells;
For each non-quiescent cell do
        Add {Cell_position(i,j), delay} to the Next-Event list;
endfor

While not end of simulation do
        Take the first item of the Next-Event list;
        tl = tn;
        tn = time of the first item of the Next-Event list;
        e = 0;

        /* Take imminent cells. Execute their internal transition functions */
        For every message in the Next-Event list with time = tn do
          New-State = τ(s, N);      /* Compute the local computing function */
          If (New-State != State) then
            save {Cell_position(i,j), Cells[Cell_position].state} in New-States list;
          endif

          If (cell is in Ylist) then            /* External coupling */
                create a Y-message (including the Next-Event time)
                        and send it to the upper level coordinator;
          endif

        If (INERTIAL-DELAY) then
                If (there is an element for that cell with Sigma
                                                greater than the new one) then
                        discard that element from the list;  /* Preemption */
                endif
        endif
                /* Common for inertial and transport delays */
                For each pair (k,l) in the Neighbors list do
                        Next_event_time = tn + Cells(i+k, j+l).delay;
                        /* Do the internal coupling */
                        Add { (i+k, j+l), Next_event_time} in the Next-Event list;
                        If a cell position does exists for the event time then
                                Insert it using the Select or δcon function;
                        endif
                  endfor
                e = tn - tl;
```

```
        endfor

        /* One step finished: update the cell space */
        For each element in the Next-States list do
                update cell's state (if necessary);
        endfor

        Empty the Next_States list
        Send a done-message to the upper level coordinator;
        Wait to coordinate with the other models: if an X-message arrives,
                insert it in the Next-Events queue. If an *-message arrives, continue.
endwhile
end.
```

***Figure 16. Flat simulation for Cell-DEVS spaces.***

## Parallel simulation of a Cell-DEVS system.

When complex systems such as those based on Cell spaces are simulated, a high number of computations are usually performed. If the models are executed sequentially, the provision of statistically significant samples attaining to reliable results is a time consuming process. Another problem for sequential solutions is that inherently parallel models must be executed serially.

There are many solutions to improve the performance of the simulations by using parallel and distributed approaches (see, for instance, [Fuj90]). These approaches allow to exploit the parallelism existing in the physical system to be simulated by associating more than one processor to the simulation. It has been shown that asynchronous parallel techniques provide a high degree of speedup of the simulators.

At present there are two main groups of techniques for asynchronous parallel simulation. Pessimist approaches [ChM79] try to define message paths in the simulation avoiding causality errors (where the future influences the past due to message reception order). Optimist mechanisms allow higher degree of parallelism by ignoring possible causality errors [Jef85]. It is supposed that the messages will arrive in the correct temporal order. If a causality error occurs, the process will make rollback and recover its previous state. Each of the techniques has its advantages and problems and are better adapted to different problems [Lip90].

Most existing parallel simulation approaches are based on the Logical Process (LP) paradigm. An LP is a set of basic information allowing to simulate each physical process of the real system. In pessimist approaches, the LP processes the events in a safe way by reading the messages in increasing arrival order. An LP with at least one non processed message in each input link updates its Local Virtual Time (LVT) to the lower timestamp of the arriving messages. Then all the messages with that timestamp are processed and an output set is produced. This procedure can be repeated while there is a message in an input link. If the link with lower LVT is empty, the logical process is blocked (because it can receive a message with a lower timestamp than the other links). This approach can produce deadlock if a cycle of blocked logical processes is created. There are different approaches to avoid or recover from deadlocks.

Lookahead properties are related with the ability to predict what will happen in the simulated future. The presence of lookahead is an important property to be considered in pessimist approaches. If all the future events can be predicted, the simulation can easily avoid deadlocks and better performance can be achieved. It has been shown that in pessimist approaches the performance depends on the lookahead knowledge provided by the programmer. In this case, a very conservative estimation can lead to poor performance, and optimism can produce causality errors.

Optimist approaches avoid this kind of concerns, exploiting parallelism where it is possible. These protocols invoke recovery procedures when causality errors are detected. When an event with lower timestamp than the local virtual time is received, a rollback process is executed. Consequently, the state before the timestamp must be recovered and the simulation process should be repeated.

To cancel the effects of wrong messages, every state from the last correct simulated time should be stored. The Global Virtual Time (minimum of all the Local Virtual Times) can be used with this purpose: there is no chance to receive messages with a lower timestamp.

If infinite memory exists, optimist algorithms cannot produce deadlock. Approaches such as lazy cancellation [Gaf88] also allows to exploit lookahead properties without explicit programming. Therefore, in Cell-DEVS models optimist approaches are better suited for the user point of view.

One disadvantage of this approach is that it is memory consuming. The computation of the GVT is also a problem: if it is computed frequently, the overhead will be high. If not, fossil collection is activated with low rates and memory is wasted.

As stated earlier, Cell-DEVS applications are inherently parallel. The Cell-DEVS formalisms presented in the previous sections rely on asynchronous approaches; therefore, the application of parallel algorithms is straightforward.

To improve execution times, the idea is to implement the coordinators in a parallel or distributed fashion. One flat coordinator will be associated to each processor, and will synchronize through optimist or pessimist approaches. Each coordinator will be coded as a logical process including three events lists: a local event list, input and output links. The simulation should be divided according with the power of each processor to allow balanced load distribution.

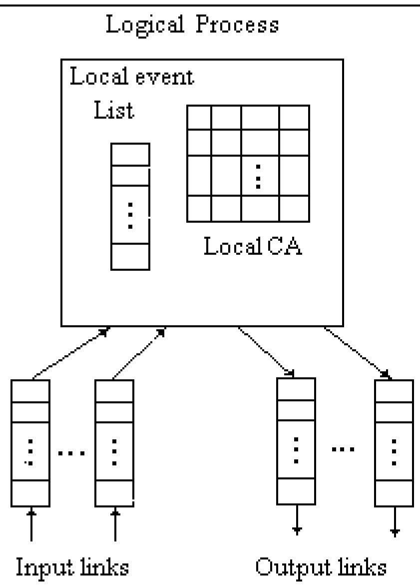



***Figure 17. Structure of the logical process for cell spaces.***

The coordinator locally selects the imminent cells to simulate and produces an internal transition. Also X-messages can arrive to each processor, locally or remotely. As a result, Y and done-messages are generated. If the new event belongs to the local processor, it is added in the next-event list. If destination cell does not belong to the local processor, the message is added to the output queue, and forwarded to other processor. There, will be added as new messages in the input queue.

As it can be seen in the new class hierarchy, both optimist and pessimist approaches will be considered. Both will use the recently presented strategy to execute models in parallel. In this way, the chosen approach can be fitted to the application, obtaining improved results for each case.

## AN APPLICATION EXAMPLE

In this section a detailed example for application of the formalisms previously presented will be introduced.

Figure 18 shows a model with the goal to simulate traffic in a section of urban population. The model is composed of different submodels; each built using a different approach. The behavior of each model has been simplified by adding several constraints (although the rules can be extended to represent more complex phenomena).

In this case, A is a Cell-DEVS model representing a residential neighborhood. The model is built to study pollution values by using a cellular automaton representing particles of smoke on the air. Influences of the traffic in the highway and smoke of trucks in the factory are considered to generate pollution flow. The local computing function also will model the influence of the wind on the smog. If a particle stays in a cell for certain time, it is diffused to the neighbors. If not, the wind removes it.

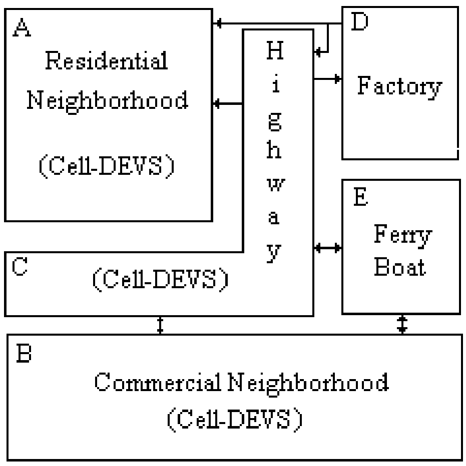



***Figure 18. Coupling of Cell-DEVS and other DEVS models.***

Model B represents the traffic movement in a commercial neighborhood. The streets are one-way (Northbound or Westbound) and no semaphores are modeled. This Cell-DEVS model could be used to study the traffic flow to install semaphores.

The Cell-DEVS model C represents a one-way highway passing across neighbors A and B. Traffic flow in the highway will be represented using a cellular model of the traffic flow in one-way routes. The model serves to study the traffic flow on the highway and its influence on the rest of the city.

Atomic model D represents a factory, with trucks arriving from the highway and other trucks getting into the highway. The simulation results could be used to schedule the input and output of trucks to the factory, improving the flow of products.

Finally, atomic model E represents the entrance to a ferry-boat connecting the city with an island. This model could be used to study the traffic flow to the ferry boats (to determine the optimal number of boats to be used depending on the hour).

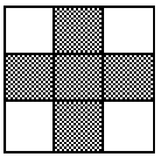
To simplify the example, it will be considered that a car or truck generates a degree of smog equivalent to the quantity of particles to activate a cell in model A. If the relationship between cars and pollution should be different, an intermediate translator model could be used.

The following section analyzes the definition of the cell spaces of this example by using the specification language. After, the formal specification for each of the models is considered. A brief explanation related with the simulation mechanisms (hierarchical and flatten) is presented afterward. Finally, an explanation of the interaction between the models and the simulation mechanisms will be included.

## Cells behavior specification

Following, the behavior for the cell spaces of Figure 18 will be depicted. To do so, the local computing functions will be described by using the specification language presented in section 0. In the specification, the $a_{ij}$ variables define a 9x9 matrix representing the cell's neighborhood. These values are taken as precondition for the local computing function. The result of the computation is the new state for each cell.

## Model A (Smog in the Residential Neighborhood)



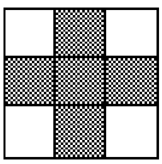
Neighborhood: { ($a_{12}$, $a_{21}$, $a_{22}$, $a_{23}$, $a_{32}$) };
Select: { ($a_{12}$, $a_{23}$, $a_{22}$, $a_{21,}$ $a_{32}$) };
wind = 3;
m = 9; n = 10; in_delay(wind).

| Result | Neighborhood state |
|---|---|
| 1 | ($a_{22}$ = 1 AND NOT (ALL = 0) ) OR<br>( (input = 1) AND ($a_{22}$ = 0) AND ($a_{12}$ = 1 OR $a_{21}$ = 1OR $a_{23}$ = 1 OR $a_{32}$ = 1) ) |
| 0 | (ALL = 0) AND ($a_{22}$ = 1) |

***Figure 19. Specification for model A***

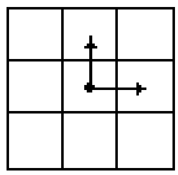
In this case, the smog diffusion is considered. The inertial delay has been used to model the pollution diffusion so that if the wind removes the smog, pollution does not expand. If there is an input with an inertial delay of 3 time units, the particles remain in the cell. If there is a particle of smog in a neighbor cell, and the particle sustains for 3 time units, the particle expands to the present cell. Otherwise, the influence of the wind is considered, and the neighbors' influence is preempted, leaving the cell unchanged.

## Model B (Traffic in the Commercial Neighborhood)



Neighborhood: { ($a_{12}$, $a_{21}$, $a_{22}$ ,$a_{23}$, $a_{32}$) };
Select: { ($a_{32,}$ $a_{21,}$ $a_{22}$, $a_{12}$, $a_{23}$) };
speed = 5;
m = 50; n = 10;
tr_delay(speed).

(a)



(b)

| Result | Neighborhood state |
|---|---|
| 1 | (input = 1) AND<br>( ($a_{22}$ = 0 AND $a_{32}$ = 1) /* Northbound */<br>OR ($a_{22}$ = 0 AND $a_{21}$ = 1) ) /* Westbound */<br>OR ($a_{22}$ = 1 AND $a_{23}$ = 1)<br>OR ($a_{22}$ = 1 AND $a_{32}$ = 1) |
| 0 | ($a_{22}$ = 1 AND $a_{12}$ = 0) /* Northbound */<br>OR ($a_{22}$ = 1 AND $a_{23}$ = 0) /* Westbound */<br>OR ($a_{22}$ = 0 AND $a_{32}$ = 0 AND $a_{21}$ = 0) |

*Figure 20. Specification for model B. (a) Cell space parameters. (b) Car movement description.*

The specification of model B represents the movement of cars in one-way streets of the commercial neighborhood. The first rule of the specification represents the input of a new car into an empty cell, or a car that cannot move on that cell. The second rule represents the car's displacement (departure of a car from the origin cell), as it can be seen in Figure 20(b). A cell also remains without cars if there is a space surrounded with empty cells.

It can be seen that the displacement of a car is not explicitly registered in one rule. Instead, the state change will be recorded in two different movements. The simulation mechanism will expand it to the influenced cells, that will record the change. The state change expands to all the neighbors, but it will only be registered in those where the movement is valid.

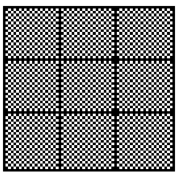
The first movement is related with the origin cell. If there is a car into that cell, the state will change. As the car will be removed, the state change will be detected by the simulation mechanism and the imminent cells will be activated.

When the corresponding cell is activated, it will receive the input value. If the movement is valid, the cell will change its state to record the arrival of the new car. If the state changes from 1 to 0, the influencees are activated and the local function computed, leading to new car movement.

If there is a crossing, the car on the right passes because northbound flow is evaluated first. Also, the collision of states in the simulation is avoided using the select function.

The transport delays allow to model the acceleration delay of the cars. The car movement is delayed prior to the next movement to the following cell, allowing to model different speeds. The delay could be represented as a random number to model the different speeds of each car.

## Model C (Highway)



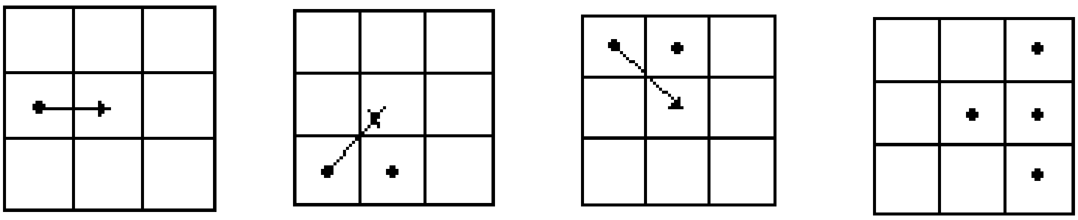
Neighborhood: { ($a_{11}$, $a_{12}$, ..., $a_{33}$) };
Select = { $a_{23}$, $a_{13}$, $a_{33}$, $a_{22}$, $a_{12}$, $a_{32}$, $a_{11}$, $a_{21}$, $a_{31}$ };
speedH = 2;
m = 15; n = 4;
tr_delay(speed).

| Result | Neighborhood state |
|---|---|
| 1 | ( (input = 1) AND<br>/* Normal flow */<br>( ($a_{22}$ = 0 AND $a_{21}$ = 1) OR<br>/* Surpassing through the left */<br>($a_{22}$ = 0 AND $a_{21}$ = 0 AND $a_{31}$ = 1 AND $a_{32}$ = 1) OR<br>/* Surpassing through the right */<br>($a_{22}$ = 0 AND $a_{21}$ = 0 AND $a_{11}$ = 1 AND $a_{12}$ = 1) ) )<br>/* Bottleneck */<br>OR ($a_{22}$ = 1 AND $\text{Column}_3$ = 1) |

| | |
|---|---|
| 0 | /* Normal flow */<br>($a_{22} = 1$ AND $a_{23} = 0$)<br>/* Surpassing through the left */<br>OR ($a_{22} = 1$ AND $a_{23} = 1$ AND $a_{13} = 0$ AND $a_{12} = 0$)<br>/* Surpassing through the right */<br>OR ($a_{22} = 1$ AND $a_{23} = 1$ AND $a_{33} = 0$ AND $a_{13} = 1$ AND $a_{32} = 0$)<br>/* Empty cell not considered in rule 1 */<br>OR ($a_{22} = 0$) |

***Figure 21. Specification for model C.***

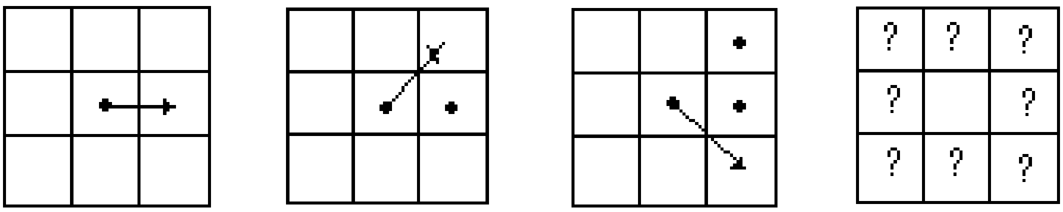
This specification is used to represent a 4-lane one-way highway. In this case, the car movement is also divided in two different movements. The first rule included represents the arrival of a new car to the present cell, or a car delayed into a cell. Each of the subexpressions of this rule represents either of the following movements:



(I) (II) (III) (IV)

***Figure 22. Valid movements for rule 1 (different expressions).***

The second rule reflects the following movements:



(I) (II) (III) (IV)

***Figure 23. Valid movements for rule 2 (different expressions).***

## Formal specification of the models

In this section, each of the models presented in the previous section will be formally specified.

Initially, the two traditional DEVS submodels will be considered. In both cases, the models are constructed as queuing servers simulating the arrival and departure of cars. The values could be generated, for instance, by using random numbers based on a Poisson distribution.

The model D, representing the flow of trucks at the factory, can be specified as:

$$M_D = \langle I_D, X_D, S_D, Y_D, \delta int_D, \delta ext_D, ta_D \rangle$$

$\mathbf{I_D}$ = { ($IN_D$, binary), ($OUT_D$, binary)};
$\mathbf{X_D} = \mathbf{Y_D}$ = {0, 1};

**$S_D$**:

Descriptive variables:

Truck_q (positive integer): Number of trucks waiting in the factory;

Parameters:

Load_time (positive integer): time devoted to fill up a truck. Average time: 100 minutes.

```
δext_D(s,e,x) {

when x is received in port IN_D
  case phase
        passive:
                Load_time = random(100);
                phase = active;
                hold_in(Load_time);       /* Activate the model. Simulate the load */
        active:
                Truck_q = Truck_q + 1;   /* Increase number of trucks in the factory */
                continue;                 /* and continue with the time advancement */
endcase
}

δint_D(s,e) {
 case phase
        active:
                Truck_q = Truck_q - 1;   /* The truck leaves */

                if (Truck_q = 0)
                        passivate;
                else
                        Load_time = random(100);
                        hold_in(Load_time);
                endif
        passive:
                /* Never occurs */
endcase
}

λ (s) {
        send Truck_q to OUT_D
}
```

***Figure* 24. *Model D definition.***

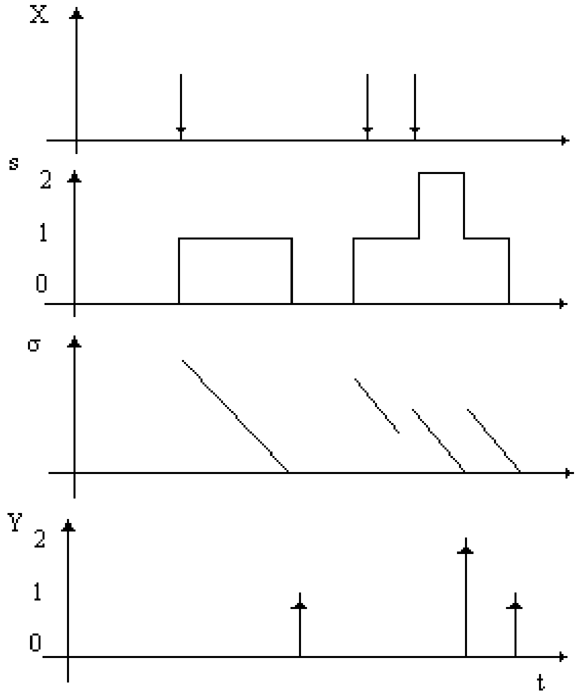



***Figure 25. Model D example execution.***

The description of the ferry boat ($M_E$) is similar to that of the Factory and will be not detailed. In this case, the random variables could be generated with an average time of 30 minutes to be served.

Let us now consider the complete definition of the cell space A. A detailed sketch of this model can be seen in Figure 26.

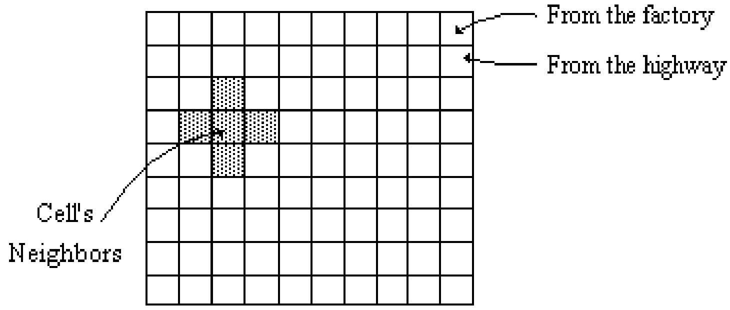



***Figure 26. Cell space A; input/output cells of the model.***

The formal definition for this cell space is:

$$M_A = < I_A, X_A, Y_A, \text{Xlist}_A, \text{Ylist}_A, \eta_A, N_A, \{m_A, n_A\}, C_A, B_A, Z_A, \text{select}_A >$$

According with the previous definition for the interface and port names,

**Xlist**$_A$ = { (1,10); (2,10) };
**Ylist**$_A$ = { ∅ }.

$\eta_A = 5$;

$\mathbf{I}_A$ = < $P^X$, $P^Y$>,  where

$P^X$ = { <X(1,10), binary>, <X(2,10), binary> };
$P^Y$ = { $\varnothing$ };

$\mathbf{N}_A$ = { (0,0), (-1,0), (1, 0), (0,1), (0,-1) };

$\boldsymbol{X}_A = \boldsymbol{Y}_A = \{0, 1\}$;

$\mathbf{m}_A = 9$; $\mathbf{n}_A = 10$;

$\mathbf{B}_A = \{\varnothing\}$;

$\mathbf{C}_A$ is the cell space set, defined as $C_A = \{C_{Aij} \,/\, i \in [1,9],\ j \in [1,10]\}$, where each $C_{Aij}$ is a Cell-DEVS component such as those defined in section 0. The specification for each cell will be described later.

$\mathbf{Z}_A$ is defined like in section 0. In this case, if we consider the Neighborhood list N and the inverse neighborhood, the port interconnection for the $Z_A$ function will be defined as:

| | |
|---|---|
| $P_{ij}^{Y_1} \rightarrow P_{i,j-1}^{X_1}$ | $P_{i,j+1}^{Y_1} \rightarrow P_{ij}^{X_1}$ |
| $P_{ij}^{Y_2} \rightarrow P_{i+1,j}^{X_2}$ | $P_{i-1,j}^{Y_2} \rightarrow P_{ij}^{X_2}$ |
| $P_{ij}^{Y_3} \rightarrow P_{i,j+1}^{X_3}$ | $P_{i,j-1}^{Y_3} \rightarrow P_{ij}^{X_3}$ |
| $P_{ij}^{Y_4} \rightarrow P_{i-1,j}^{X_4}$ | $P_{i+1,j}^{Y_4} \rightarrow P_{ij}^{X_4}$ |
| $P_{ij}^{Y_5} \rightarrow P_{ij}^{X_5}$ | $P_{ij}^{Y_5} \rightarrow P_{ij}^{X_5}$ |

**Figure 27. Port interconnection of $Z_A$ function.**

$\mathbf{select}_A$ = { (-1,0), (0,1), (0,0), (0,-1), (1, 0) };

Each cell of the space can be formally described as:

$$C_{Aij} = < I_A, X_A, S_A, Y_A, N_A, \delta int_A, \delta ext_A, d_A, \tau_A, \lambda_A, ta_A >$$

Here, the description for $X_A$, $Y_A$ $\eta_A$, $m_A$, $n_A$ and $N_A$ are the same than those presented for the cell space.  Besides,

$\mathbf{I}_A$ = < 5, $P^X$, $P^Y$>,  where

$P^X$ = { ($X_1$, binary), ($X_2$, binary), ($X_3$, binary), ($X_4$, binary), ($X_5$, binary) };
$P^Y$ = { ($Y_1$, binary), ($Y_2$, binary), ($Y_3$, binary), ($Y_4$, binary), ($Y_5$, binary) }.

$\mathbf{S}_A$:

<u>Descriptive variables:</u>

$$s = \begin{cases} 1 \text{ if there is smog in the cell;} \\ 0 \text{ otherwise} \end{cases}$$

<u>Parameters:</u>

wind: delay related with the wind influence.

$\mathbf{d}_A$ = wind;

$\lambda_A$, $\boldsymbol{\delta}\mathbf{int}_A$ and $\boldsymbol{\delta}\mathbf{ext}_A$ behaves as the functions defined in section 0.

$\tau_A$: $S_A$ x $N_A \rightarrow S_A$ has been previously defined using the specification language.

Now, let us suppose the higway cellular automata has the following shape:

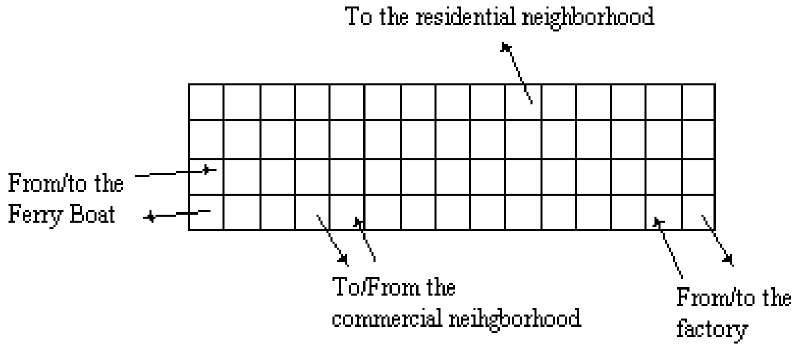



***Figure 28. Cell space C; input/output cells of the model.***

The formal definition for the cell space is:

$$C = < I_C, X_C, Y_C, \text{Xlist}_C, \text{Ylist}_C, \eta_C, N_C, \{m_C, n_C\}, C_C, B_C, Z_C, \text{select}_C >$$

In this case,
$\mathbf{Xlist}_c$ = { (4,14), (4,5), (3, 4) };
$\mathbf{Ylist}_c$ = { (1,10), (4,4), (4,15), (4,1) }.

$\mathbf{I}_C$ = < $P^X$, $P^Y$>, where

$P^X$ = {<X(1,3), binary>,<X(4,14), binary>,<X(4,5), binary>,<X(4,3), binary> };
$P^Y$ = {<Y(1,10), binary>,<Y(4,4), binary>,<Y(4,15), binary>,<Y(4,1), binary> }.

$\eta_C = 9$;

$\mathbf{N}_C$ = { (-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1, 0), (1, 1) };

$\boldsymbol{X}_C = \boldsymbol{Y}_C = \{0, 1\}$;

$\mathbf{m}_C = 4$; $\mathbf{n}_C = 15$;

$\mathbf{B}_C = \{\varnothing\}$;

$\mathbf{C}_C$ is the cell space set, defined as in the previous example. In this case, $\mathbf{S}_C$ is defined as:

<u>Descriptive variables:</u>

$$s = \begin{cases} 1 \text{ if there is a car in the cell;} \\ 0 \text{ otherwise} \end{cases}$$

Initial value: s = 0.

<u>Parameters:</u>

speedH: delay related with the cars speed.

$\mathbf{Z_C}$ is also defined like in previous example.

$\mathbf{select_C}$ = { (0,1), (-1,1), (1, 1), (0,0), (-1,0), (1, 0), (-1,-1), (0,-1), (1,-1) };

The definition for model B is similar of both previous cases. The definition of the model's interface will be included with the goal of analyzing the coupling between the different models.

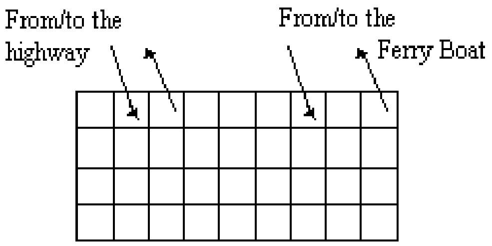



***Figure 29. Cell space B; input/output cells of the model.***

In this case,

$$B = < I_B, X_B, Y_B, Xlist_B, Ylist_B, \eta_B, N_B, \{m_B, n_B\}, C_B, B_B, Z_B, select_B >$$

with

$\mathbf{Xlist_B}$ = { (1,2), (1, 7) };
$\mathbf{Ylist_B}$ = { (1,3), (1, 9) }.

$\mathbf{I_B} = < P^X, P^Y >$, where

$P^X$ = { <X(1,2), binary>, <X(1,7), binary>};
$P^Y$ = { <Y(1,3), binary>, <Y(1, 9), binary> }.

$\eta_B = 9$;

$\mathbf{N_B}$ = { (0,0), (-1,0), (1, 0), (0,1), (0,-1) };

$\mathbf{m_B} = 4$; $\mathbf{n_B} = 9$;

$\mathbf{select_B}$ = { (1, 0), (0,-1), (0,0), (-1,0), (0,1) };

Finally, let us consider the formal specification for the complete model. In this case,

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

where

$\boldsymbol{X} = \boldsymbol{Y} = \{\varnothing\}$;

**D** = { A, B, C, D, E }, and $\forall i \in D$, $M_i$ is one of the basic DEVS models previously defined.

$\mathbf{I_i}$ is the set of influencees of model i. In this case,
$I_A = \{\varnothing\}$;
$I_B = \{ C, E \}$;
$I_C = \{ A, B, D, E \}$;
$I_D = \{ A, C \}$; and
$I_E = \{ B, C \}$.

$\forall j \in I_i$, $\mathbf{Z_{ij}}$: $Y_i \rightarrow X_j$ if i and j are DEVS models, or
$\mathbf{Z_{ij}}$: $Y(f,g)_i \rightarrow X(k,l)_j$, with $(f,g) \in Ylist_i$, and $(k,l) \in Xlist_j$ if either model is Cell-DEVS.

In this case, the $\mathbf{Z_{ij}}$ function is defined as:

$Z_{BC}$: $Y(1,3)_B \rightarrow X(4,5)_C$
$Z_{BE}$: $Y(1,9)_B \rightarrow IN_f$
$Z_{CA}$: $Y(1,10)_C \rightarrow X(2,10)_A$
$Z_{CB}$: $Y(4,4)C \rightarrow X(1,2)_B$
$Z_{CD}$: $Y(4,15)_C \rightarrow IN_f$
$Z_{CE}$: $Y(4,1)_C \rightarrow IN_b$
$Z_{DA}$: $OUT_f \rightarrow X(1,10)_A$
$Z_{DC}$: $OUT_f \rightarrow X(4,14)_C$
$Z_{EB}$: $OUT_b \rightarrow X(1,7)_B$
$Z_{EC}$: $OUT_b \rightarrow X(3,4)_C$

Finally, **select** = { C, A, B, D, E}.

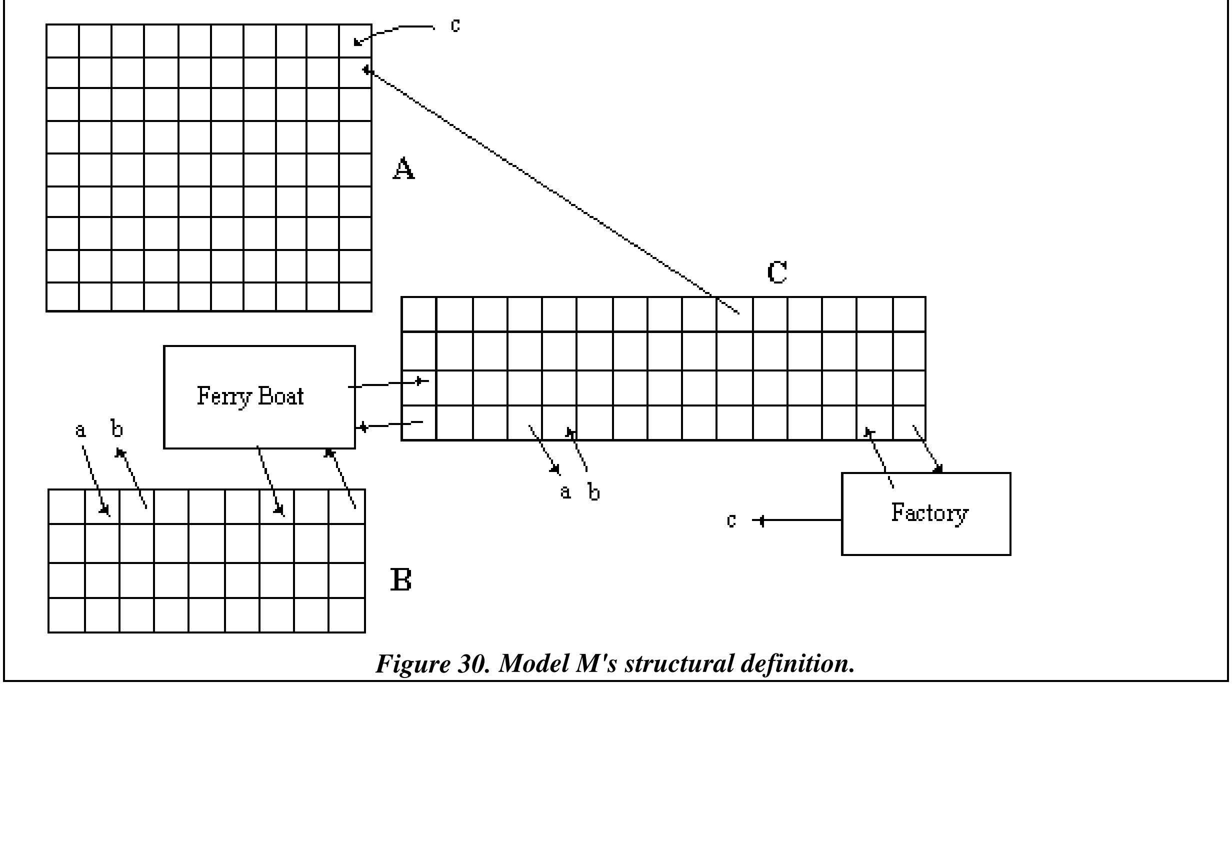



***Figure 30. Model M's structural definition.***

## Simulation mechanism

This section is devoted to analyze the behavior of the mechanisms employed in cell spaces simulation by using the example defined in the previous sections. First, the hierarchical simulation mechanisms are exemplified. After, the example attacks the flatten simulation procedures are considered.

## Hierarchical simulation of cell spaces

Figure 31 depicts the hierarchy of simulation processors for the present example. As stated in section 0, the simulators for the cell-space models A, B and C are automatically created when the cell space is defined. To do so, the parameters presented in the previous sections should be defined by the modeler.

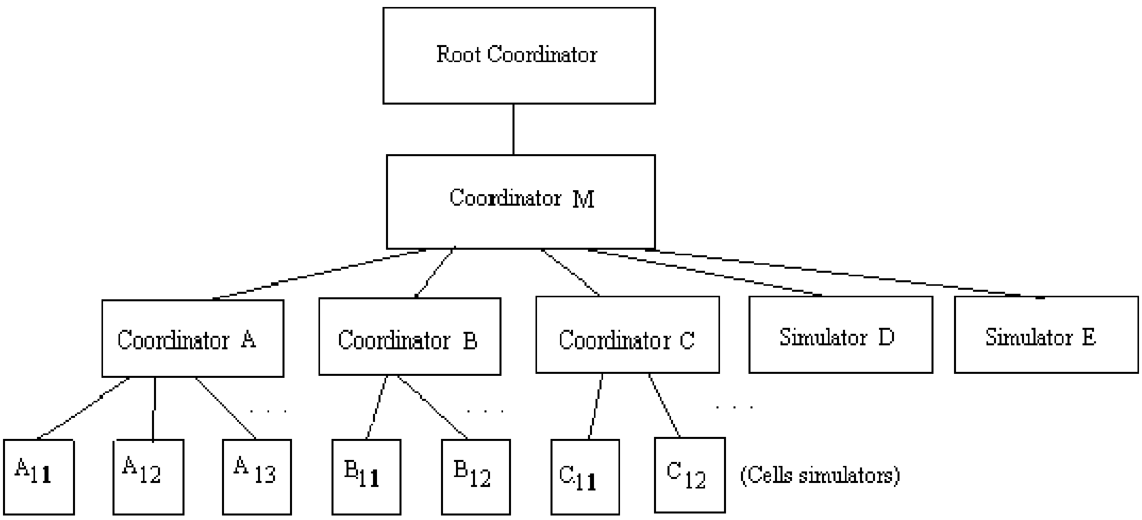



*Figure 31. Hierarchy of simulation processors for model M.*

In this section the message flow between the different processors shown in the figure is studied. Let us suppose first that in simulated time 10, the model's contents are those of Figure 32. There is only one active cell in the model: cell (1,9) of model C.

**Notation:** from now, X(i,j) refers to ij-eth cell of cell-DEVS model X or its corresponding simulator.

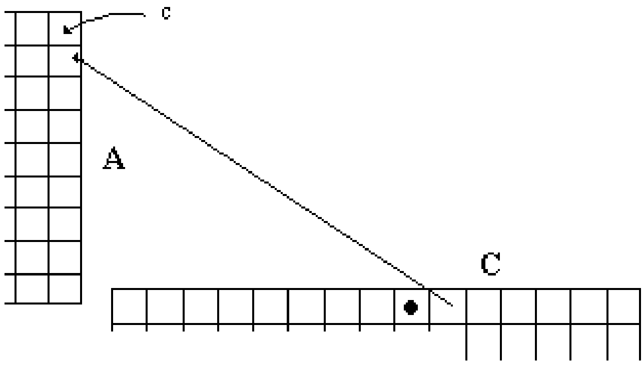



*Figure 32. Model M's initial state.*

At present there is only one message waiting to be processed in the Event-List of the root coordinator. Therefore, the contents of the data structures of the simulators and coordinators are the following:

| | |
|---|---|
| Root Coordinator:<br><br>**Clock** = 10;<br>**Associated coordinator**: Coordinator M. | Coordinator M<br><br>**Father**: Root Coordinator<br>**Children**: { A, B, C, D, E }<br>**Associated coupled model:** {M}<br>**Waiting list**: { }<br>**Imminent children:** { C }<br>**tn:** 10; **tl:** 0. |
| Coordinator A<br><br>**Father**: Coordinator M<br>**Children**: { $A_{11}$, $A_{12}$, ... }<br>**Associated coupled model:** {A}<br>**Waiting list**: { }<br>**Imminent children:** { }<br>**tn:** 0; **tl:** 0. | Coordinator B<br><br>**Father**: Coordinator M<br>**Children**: { $B_{11}$, $B_{12}$, ... }<br>**Associated coupled model:** {B}<br>**Waiting list**: { }<br>**Imminent children:** { }<br>**tn:** 0; **tl:** 0. |
| Coordinator C<br><br>**Father**: Coordinator M<br>**Children**: { $C_{11}$, $C_{12}$, ... }<br>**Associated coupled model:** {C}<br>**Waiting list**: { }<br>**Imminent children:** {(1,9) }<br>**tn:** 0; **tl:** 0. | Coordinators D, E<br><br>**Father**: Coordinator M<br>**Children**: { }<br>**Associated coupled model:** {D} ( {E } )<br>**Waiting list**: { }<br>**Imminent children:** {}<br>**tn:** 0; **tl:** 0. |

***Figure 33. Initial contents of simulators/coordinators' data structures.***

The message contents are the following:

< Type, Source | Destination, Date, [Value] >

The simulation begins when the root coordinator creates the following message:

< *, Root, 10 >

This message is sent to the M coordinator, that queries its list of imminent children. There, the coordinator selects model C (the first of its imminent list), and sends the message <*, M, 10> to its coordinator. Coordinator C receives this message, and selects its imminent child (in this case, cell [1,9]) by consulting its imminent children list. It also verifies that the time of next-event is equal to the simulated time included in the message.

The message <*, C, 10> is sent, and arrives to the simulator $C_{19}$, that will execute the internal transition function. As a first step, the time of next event **tn** is verified. As it is the same of that of the *-message, the arrived message is correct and the output function is executed, activating the local computation function.

The second rule of the second expression in the function's specification is true. Therefore, the result of the local computation is s' = 0. As s = 1, the cell's state has changed and its present value should be output.

A Y-message is generated with the following values:

< Y, C(1,9), 10, 0 >

and it is transmitted to the coordinator C. After, the event times corresponding to the coordinator are updated:

tl = tn = 10;
tn = tl + D(s) = 10 + ∞ = ∞.

Finally, the simulator updates the local values of the neighborhood set by making $a_{22}$ = 0 in the neighbor's values of cell $C_{19}$.

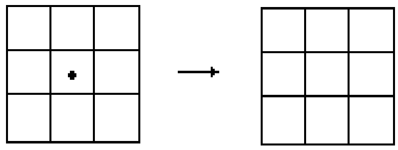


***Figure 34. Update of local neighborhood values.***

The Y-message is received by the coordinator C, that queries its coupling scheme (defined in section 0). As this message is local to the C model, it should be translated into an X-message, and sent to the following cells: (1,10), (1,8), (2,8), (2,9), (2,10), (4,8), (4,9), (4,10). The *-message <*, C(1,9), 10 > is sent to each of the cell's simulators and the cells numbers are added into the coordinator's waiting list.

In each simulator, the message time is checked to see if it is between the last event and next event times. As this is correct, the elapsed time and the event times are updated.

e = t - tl = 10 - 0 = 10, where t is the present time;
$\sigma$ = tn - e = 10 - 10 = 0;
tl = tn = 10; and
tn = tl + D(s) = 10 + 2 = 12.

Following, the input functions are executed. In this case, the input values and its corresponding $\sigma$ are queued in the $\sigma$queue, as shown in sections 0and 0. In this case, D(s) includes the value of the transport or inertial delays.

Then, each simulator creates a message

< done, C(i,j), 12 >

because the transport delay is of 2 time units. When these messages are received, the coordinator eliminates the children from the wait-list and adds the message in its imminent list. In this case, all the messages have the same time, and the select function is used to choose the imminent (messages are queued by using this criterion).

**Imminent children:** { (1,10), (1,8), (2,8), (2,9), (2,10), (4,8), (4,9), (4,10) }.

When the waiting list is empty, all the influencees have finished with the execution of the external transition function. Therefore, the smallest imminent children is ready for execution.

The first element of the imminent children list is chosen, and the message

< done, C(1,10), 12 >

is sent to the coordinator M. In this way, if more than one imminent model exists in the hierarchy, one of them can be chosen to be simulated.

To do so, M adds the model C in its queue of imminent children. As this is the only message in the queue, it will be the imminent child. Therefore, the next and last events times can be updated. A done message is sent to the root coordinator because the coordinator is not waiting other events (the waiting list is empty).

The root coordinator updates the clock to 12, and creates a new message:

$$< *, \text{Root}, 12 >$$

When the M coordinator receives this message, it verifies the time, takes the imminent child C from its imminent child queue, and sends it the message

$$< *, \text{M}, 12 >$$

to its coordinator. The coordinator selects the imminent children (1,10) and activates its simulator.

These procedures are repeated for each of the imminent children of coordinator C.

When the simulators are activated in turn, the local computing functions are executed. Every cell of those activated by the event in (1,9) remain in a quiescent state, except for cell (1,10). Recalling the specification of Figure 21, the only valid rule for these cells is the fourth of the second sentence. Therefore, the sate value will remain in 0 and the new external events are not generated.

Then, a message

$$< \text{done}, C(i,j), \infty >$$

will be generated for each of these cells and transmitted to the C coordinator.

Instead, for the cell (1,10) the first rule of the first sentence is valid. The cell's state changes, $\lambda$ is executed and a message

$$< Y, C(1,10), 12, 1 >$$

is generated and sent to the coordinator C.

This message is expanded to the neighbor cells by repeating the procedures presented recently. This will cause actualization in the local copies of the neighborhoods for each simulator.

In this case, the Y-message also influences the links that are external to the C model. Therefore, it should be retransmitted to the parent coordinator M to execute the $Z_{ij}$ function, allowing to send the change to the other models.

When the message is received by the M coordinator, it queries the external coupling function, and it determines that the coupling

$$Y(1,10)_C \rightarrow X(2,10)_A$$

should be used. Therefore, it generates the following message:

$$< X, A(2,10), 12, 1 >$$

The message is sent to the A coordinator, that expands it up to its arrival to the cell A(2,10), where the external transition function is executed and the message is queued, generating a message

$$<\text{done}, A(2,10), 15>$$

## Flatten simulation of cell spaces.

In this section, the execution of the examples of Figure 32 will be repeated, but considering the flat coordinator. In this case, message interaction is completely avoided within the cell spaces, allowing the improvement of execution times.

When the procedure of Figure 18 is considered to execute the present example, we can see that the Next-Events lists of all the cellular models are empty, except for model C. For this coordinator,

Next-Events= {(1,9), 10}.

The coordinator executes, taking the data for the first event of the list. The simulation times are updated: tl = 0, tn = 10 and e = 0.

In this case, when the local computing function is executed, the result obtained is New-state = 0. As Cells[1,9].state = 1, the new state should be stored in the New-states list.

Therefore,

New-states = { (1,9), 0}

After, as the cell is not in Ylist, nor inertial delays are used, the following sentences are executed, making

Next-event time = 10+2;
Next-events = { <(1,10), 12>, <(1,10), 12>, <(1,8), 12>, <(2,8), 12>, <(2,9), 12>, <(2,10), 12>, <(4,8), 12>, <(4,9), 12>, <(4,10), 12> }.

The insertion in the Next-events list is done by using the select function, because all the events have the same time.

After, as the Next-Events list does not have other events with time = 10, one simulation step has finished. Therefore, the cell space is updated by doing:

c[1,9] = 0;
New-states = { }.

When the next events have been updated, a message

< done, 10, C >

is sent to the upper level coordinator. The coordinator detects that C is the imminent child, therefore, after the Root Coordinator updates the global clock, a *-message is sent to the C coordinator, that continues with the simulation.

The simulator cycles, and it makes tn = 12, tl = 10, and e = 0. Cell (1,10) is chosen from the Next-Events list. In this case,

New-state = 1; and
Cells[1,9].state = 0, therefore

New-states = { <(1,10), 1> }.

The cell is in the Ylist, therefore a Y-message is created and sent to the coordinator M. The coordinator will react in the same way explained in the previous section. When the message

< Y, C(1,10), 1, 12 >

arrives to the M coordinator, it is translated into the message

$$< X, A(2,10), 12, 1 >$$

that will be transmitted to the flat coordinator A. This coordinator will insert it into the Next-Events queue, to be treated in the future.

## CONCLUSION

In the present work, several new formal descriptions for cell spaces modeling and simulation are presented. The formalisms are based on the DEVS and Asynchronous Cellular Automata formalisms, using transport or inertial delays. These concepts have been taken from the digital modeling domain, allowing to specify complex behavior in a simple fashion, independently of the quantitative complexity of the models.

A formal description of binary Cell-DEVS models was depicted, considering specifications for one cell, closed and general Cell-DEVS coupled models. After, a three-state Cell-DEVS formalism with transport delays was defined. After that, an extension to deal with inertial delays is presented.

The methods presented allow automatic definition of cell spaces using the DEVS formalism. Integration of multiple views for each submodel is possible, letting to integrate different modeling paradigms in an efficient fashion.

Possible implementations of the paradigms were introduced, allowing efficient and cost-effective development of simulators. The user and the developer point of view were considered. Two different descriptions for the simulation mechanisms were included. The first one considered hierarchical simulation. After, a method to flatten the hierarchical description of the cell spaces is given.

Finally, an introduction to parallel execution of the cell models was presented. The formalisms allow to improve the security and cost in the development of the simulations.

At present, a centralized and a parallel implementation using the formalisms are being developed. To test the usefulness of the formalisms, experimental analysis will be carried out for both approaches.

## REFERENCES

[BBW97a] BARYLKO, A.; BEYOGLONIAN, J.; WAINER, G. "GAD: a General Application DEVS environment". Technical Report No. 98-001. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1997.

[BBW97b] BARYLKO, A.; BEYOGLONIAN, J.; WAINER, G. "Implementation of a Cell-DEVS modeling environment". Technical Report No. 98-002. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1997.

[ChM79] CHANDY, K; MISRA, J. "Distributed simulation: a case study in design and verification of distributed programs", IEEE Transactions on Software Engineering, September 1979.

[ChZ94] CHOW, A.; ZEIGLER, B. "Revised DEVS: a parallel, hierarchical, modular modeling formalism". Proceedings of the SCS Winter Simulation Conference. 1994.

[Cho95] CHO, Y. "Parallel Implementation of container using parallel virtual machine". M.Sc. Thesis. Department of Electrical and Computer Engineering. The University of Arizona. 1995.

[Fuj90] FUJIMOTO, R. "Parallel Simulation of Discrete Events". Communications of the ACM. Vol. 33. No 10. 30-53. 1990.

[Gaf88] GAFNI, A. "Rollback mechanisms for optimistic distributed simulation systems". Proceedings of the SCS Multiconference on Distributed Simulation. pp. 61-67. 1988.

[GG96] GHOSH, S.; GIAMBIASI, N. "On the need for consistency between the VHDL language constructions and the underlying hardware design". Proceedings of the 8th. European Simulation Simposium. Genoa, Italy. Vol. I. pp. 562-567. 1996.

[GFE95] GIAMBIASI, N.; FRYDMAN, C.; ESCUDE, B. "Hierarchical/Multi-view modeling and simulation". Proceedings of the 7th. SCS European Conference on Computer Simulation. 1995.

[GE97] GIAMBIASI, N.; ESCUDE, B. "Discrete Events Scpecification of hybrid coupled models". Proceedings of the 9th. European Simulation Symposium and Exhibition. Passau, Germany. 1997.

[GM76] GIAMBIASI, N.; MIARA, A. "SILOG: A practical tool for digital logic circuit simulation. Proceedings of the 16th. D.A.C., San Diego, U.S.A. 1976.

[Jef85] JEFFERSON, D. "Virtual Time". ACM TOPLS, 7(3): 404-425. July 1985.

[LM90] LIPTON, B; MIZELL, D. "Time Warp vs. Chandy-Misra: a worst-case comparison". Proceedings of the SCS Multiconference on Distributed Simulation. 1990.

[WFG97a] WAINER,G.; FRYDMAN, C.; GIAMBIASI, N. "An environment to simulate cellular DEVS models". Proceedings of the SCS European Multiconference on Simulation. Istanbul, Turkey. 1997.

[WFG97b] WAINER,G.; GIAMBIASI, N.; FRYDMAN, C. "Cell-DEVS models with transport and inertial delays". Proceedings of the 9th. European Simulation Symposium and Exhibition. Passau, Germany. 1997.

[Wol86] WOLFRAM, S. “Theory and applications of cellular automata”. Vol. 1, Advances Series on Complex Systems. World Scientific, Singapore, 1986.

[Zei76] ZEIGLER, B. "Theory of modeling and simulation". Wiley, 1976.

[Zei84] ZEIGLER, B. "Multifaceted Modeling and discrete event simulation". Academic Press, 1984.

[Zei90] ZEIGLER, B. "Object-oriented simulation with hierarchical modular models". Academic Press, 1990.

[Zei97] ZEIGLER, B. "Objects and systems: principled design with Java/C++ Implementation". Springer-Verlag. New-York, 1997.

**APPENDIX - Cell behavior specification language (BNF)**

**Notation**

| Notation | Syntax definition |
|---|---|
| [a] | Optional |
| {a} | Unlimited repetition (0 to n) |
| a \| b | Selection |
| A | Synctactic element |
| **A** | Language terminal |
| A ::= | Rule specification |

alpha ::= **A | B | ... | Z**
digit ::= **0 | 1 | ... | 9**
char ::= alpha | digit
sign ::= **[+] | -**
int_const ::= [sign] digit {digit}
variable ::= **a** index index **| ALL | Row** index **| Column** index **| input**

expression ::= bool_exp
bool_exp ::= [**NOT**] int_exp rel_op int_exp | bool_exp rel_op bool_exp | delay

rel_op ::= **= | != | > | < | >= | <= | AND | OR**

int_exp ::= [sign] int_sent {ar_op int_sent}
int_sent ::= variable | int_const | func | (expression)

ar_op ::= **+ | - | * | / | ^**

func ::= **one_count() | zero_count() | fi_count()**

sentence ::= **0 | 1 | ϕ** (expression);

specification ::= sentence {sentence}

delay ::= **tr_delay**(int_sent) **| in_delay**(int_sent);

index ::= 1 | 2 | 3