DEPARTAMENTO DE COMPUTACIÓN

Facultad de Ciencias Exactas y Naturales

UNIVERSIDAD DE BUENOS AIRES

INFORME TÉCNICO

Application of the Cell-DEVS paradigm

Gabriel A. Wainer, Amir Barylko,
Jorge Beyoglonián, Norbert Giambiasi

Report n.: 98-018

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

http://www.dc.uba.ar

Title: Application of the Cell-DEVS paradigm

Authors: Gabriel A. Wainer; Amir Barylko; Jorge Beyoglonián; Norbert Giambiasi.

E-mail: gabrielwl@dc.uba.ar

Report n.: 99-018

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

Abstract: The present article shows the results of the application of a new formalism designed to model and simulate cell spaces. The paradigm is based on the DEVS and Asynchronous Cellular Automata formalisms combined with inertial or transport delays. Efficient development of the simulators is achieved, reducing the cost of the construction of simulators for these systems. Also, the performance execution was improved through the definition of a flat simulation mechanism. The results obtained are thoroughly detailed, allowing the analysis of the advantages of the proposed approach.

Name:...

Address:..

Application of the Cell-DEVS paradigm

Gabriel A. Wainer * § Amir Barylko * Jorge Beyoglonián *
{gabrielw, abarylko, jbeyoglo}@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

The present article shows the results of the application of a new formalism designed to model and simulate cell spaces. The paradigm is based on the DEVS and Asynchronous Cellular Automata formalisms combined with inertial or transport delays. Efficient development of the simulators is achieved, reducing the cost of the construction of simulators for these systems. Also, the performance execution was improved through the definition of a flat simulation mechanism. The results obtained are thoroughly detailed, allowing the analysis of the advantages of the proposed approach.

1. INTRODUCTION

At present, several modelling paradigms have been developed to formally specify dynamic systems. The use of a formal modelling paradigm allows the validation of the behavior of a conceptual model when compared against that of the real system described. The DEVS formalism (Discrete EVents system Specification) is a discrete event paradigm defined by B. Zeigler [Zei76]. It allows the description of the models hierarchically and Object Oriented. A real system modeled with this paradigm can be seen as composed of several submodels, behavioral (atomic) or structural (coupled). The use of this strategy allows the reuse of checked models, improving the security in the simulations, reducing the testing time and improving productivity.

Cellular automata are defined by a formalism using discrete variables for time, space and system states. A cellular automaton consists of an n-dimensional infinite lattice, where each cell includes a computation machine taking a value from a finite set. The states of the cells change in discrete time steps using a local computation function. This function considers the present state for the cell, and the values of a finite number of close cells.

The use of a discrete time base poses constraints into the precision and efficiency of the models. Besides, in most cases a high number of cells in the automata do not need to be updated. The use of asynchronous cellular automata can avoid these problems. In [WG98] the Timed Cell-DEVS formalism was defined, combining both paradigms. Each cell can be defined as an atomic model, and a procedure to couple cells is depicted.

The Figure 1 shows informally the contents of a binary cell, used as a basic construction in this formalism. When an external event is received, a boolean function is executed, consuming the input values for the cell. Then, the output of the cell is delayed during **d** time units, and an internal event is scheduled. Inertial or Transport delays can be included, allowing the definition of preemptive or anticipatory semantics.

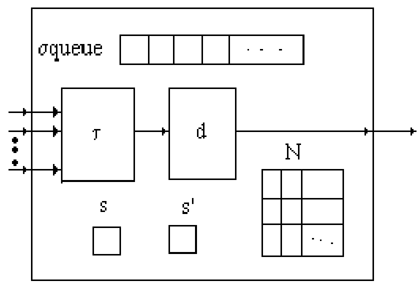


Figure 1. Cell definition with delays.

This article is devoted to show the results obtained using the formalism to develop cellular models. The rest of the work will be devoted to present the main features of the implementation of the Cell-DEVS models, and after, the results of several experiences will be detailed.

2. CELL-DEVS IMPLEMENTATION MODELS

To improve the development time for a simulation, a specified model should be executable. Therefore, the modeler should define them using a formal specification, and an automatic procedure to build a valid implementation model should be included.

The modeler should specify three basic aspects of the cell space: dimension, influences and behavior. Respecting to the dimension, he must be able to define the size of the cell space. After, the influences should be defined by using the neighborhood relationship. The input/output ports and the cell's coupling should be created by following the formal specification. If the cell space must be coupled with other models, the interfacing cells must be considered. Finally, the kind and duration for the delays should be defined.

The local computation function for each cell can be represented by means of a specification language that has been defined with this goal. The specification for the behavior of a cell is defined as a list of sentences with the following structure:

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

Each sentence is a boolean expression returning a binary or three-state value. The expressions can include relational operators and calls to predefined functions. The results are obtained using the present state of a cell and that of its neighbors. The following table presents the main operators and functions available:

Operator	**Functionality**
<, <=, >, >=	Comparators
=, !=	Equality comparison
AND, OR, NOT	Boolean
( )	Precedence
+, -, *, /, ^	Arithmetic
Functions	
one_count();	Number of neighbors with value equal to 1
zero_count();	Number of neighbors with value equal to 0
fi_count();	Number of neighbors with undefined value
Delays	
tr_delay(d);	Transport (d milliseconds)
in_delay(d);	Inertial (d milliseconds)

Table 1. Functions and operators.

A compiler translates the specification into a Cell-DEVS representation. When the cell's behavior is defined, the parameters of dimension and influences are also used. In this way, a modeler can implement a model without the need of programming.

In this way, the local computation function has a fixed behavior and should not be reprogrammed. The specification is parsed, and a dynamic structure is built to define the behavior of the cell. Several errors of the specification can be detected in runtime. If several results exist for one cell with the same state and neighbors, the specification is ambiguous. Instead, if there are no results for a given state, the specification is uncomplete. If several expressions produce the same result, the simulation can continue, but the user should be warned. Instead, if different results are obtained for the same precondition, the simulation should be stopped (the future state of the cell cannot be determined: the model is non-deterministic). This checking allows the detection of inconsistencies in the model's definition. The errors must be reported to the user so as to change the specification. In this way, the validity of the model can be easily checked.

To conclude the definition of a cell space, the border cells should be defined, and the priorities to solve simultaneous events should be determined. The formal specification presented in [WG98] is used to complete the specification of the cell spaces. The interfaces are automatically defined. The time advance function is controlled by the simulation mechanism. Finally, an array consisting of atomic cells is created, and the influences of each cell are defined. The simulation begins by choosing initial states for each cell and the stop conditions for the simulation cycle.

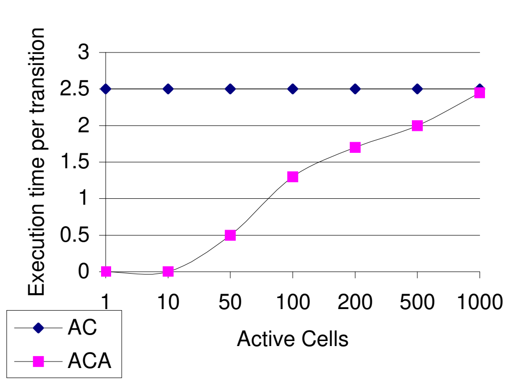
3. DEVELOPMENT EXPERIENCES

This section will focus some of the development experiences with cellular models. Some of the presents the execution of application coded “by hand”, and they are later compared with the use of a development tool following the principles presented in the previous section.

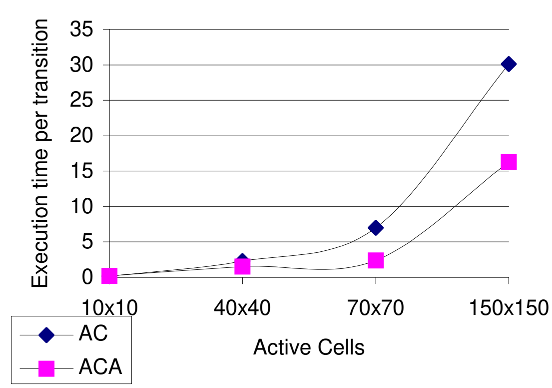
3.1. Experiences with synchronous and asynchronous cellular automata.

The first tests were related with the performance of synchronous and asynchronous automata. The experiments included the execution of the Life game [Gar70] (whose specification will be included in section 3.2). The goal was to measure the execution time to compute the new generations with both approaches, depending on the number of active cells. Initially, the tests considered a fixed size for the cell space (40 x 40 cells), and a variable number of active cells. A second phase of the experience considered the worst execution times for each transition, increasing the size for the cell spaces.

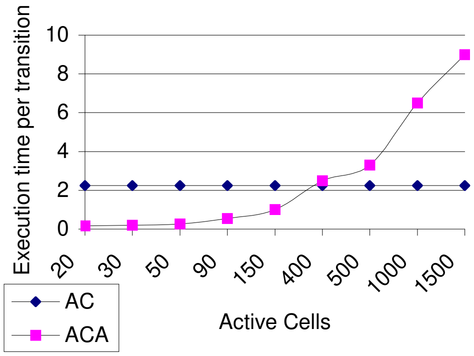
In every case the number of initial active cells was fixed (1000 active initial cells) and the worst execution cases were considered. The goal was to make fair comparisons, because the asynchronous cellular automata behave better depending on the number of active cells. While the worst cases for the synchronous cellular automata did not differ from the average execution times (3% of maximum difference), the asynchronous ones had a standard deviation of 60%.

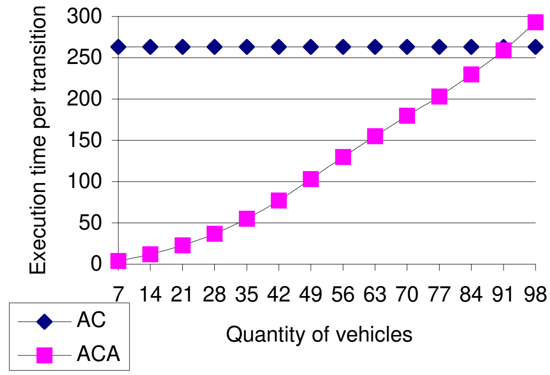


(a)



(b)





(c) (d)

Figure 2. Execution times (in milliseconds) for the Life game (a) Fixed size (b) variation in cell space size. (c) Execution results of the model Wa-Tor. (d) Multiple lane traffic model.

The Figure 2 shows that the synchronous model has a constant delay for each generation (all the cells are scanned, and always in the same order). Instead, the execution time for the asynchronous model has a behavior defined by a function linearly increasing depending on the number of active cells. The time involved in the maintenance of the event list produces this increase of time.

The results obtained when executing the Wa-Tor model [OSH92] were similar. This model is used to study a population of fish and sharks in the ocean considering their lifetimes, birth and death rates. The figure shows the execution time to compute a new generation depending the number of active cells (those with fish or sharks) in a 40 x 40 cell space. In this case, the execution time for the synchronous models is also constant, but the asynchronous model increases exponentially. Approximately when having 400 active cells, the number of neighbors included in the next-events list is so high that makes the synchronous model more efficient than the asynchronous.

The last tests considered a set of cars advancing in multiple lanes in one way. The cars advance with a constant speed. If the cell in front is occupied, the car tries to surpass the other (first using the left lane, otherwise, the right one). The figure shows the execution times for a variable number of vehicles in a grid consisting of 80x10 cells. Another experience analyzed the performance of the traffic by varying the computation time of the local transition functions. This experience allowed studying the influence of having increasing complexity for the local computation functions. The results, showed in the next figure, show that the performance for the asynchronous automata improves in several orders of magnitude depending on this complexity.

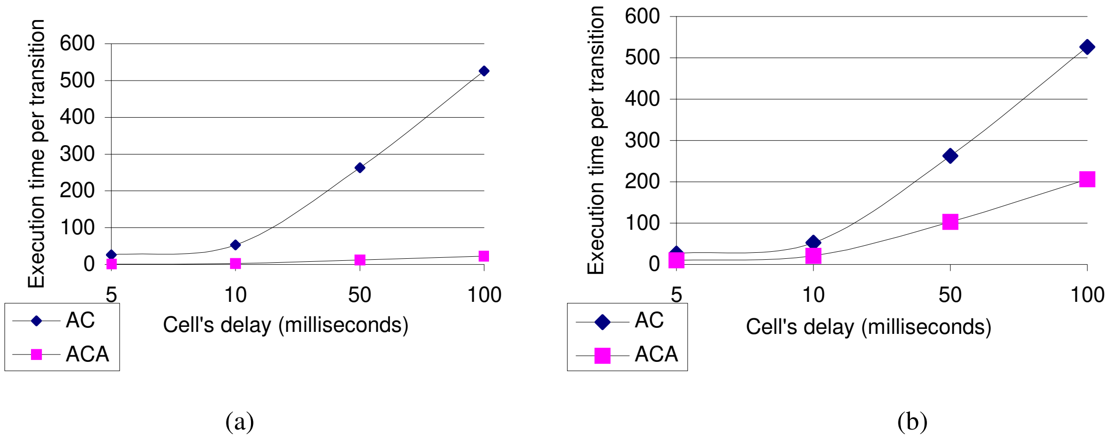


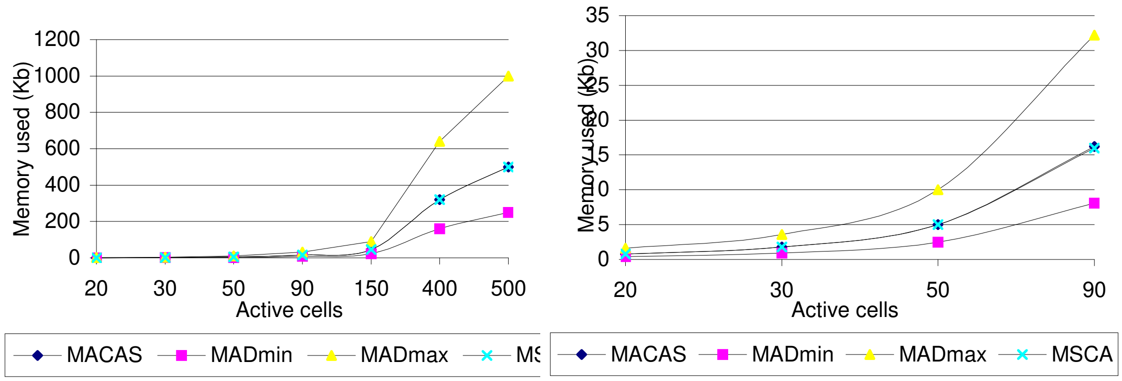
Figure 3. Different execution times for each cell (a) 7 vehicles in the automata; (b) 49 vehicles.

The next studies were focused on the memory utilization for both approaches. Usually, the asynchronous solution is more effective. This happens because a dynamic list is used for the next events, instead of a second state matrix needed for the synchronous solutions. Nevertheless, if the number of active cells grows, it will be necessary to keep information about the non-quiescent cells (in the worst case, the complete cell space). This fact leads to worst use of the memory than with the synchronous approach. Besides, the use of dynamic memory worsens the execution times for the asynchronous solution. The dynamic tables should be implemented using static data structures to reduce the CPU overhead.

The use of memory for all the cases mentioned, was estimated as follows. We will call N to the size of the automata in each dimension, S to the size of a pointer in the system, and T to the size of the data structure used to record a cell's state. Four estimations were computed: MSCA (Memory for a Synchronous Cellular Automaton), MACAS (Memory for an Asynchronous Cellular Automaton using Static data structure), MADmin and MADmax (the minimum and maximum of Memory used by Asynchronous cellular automata using Dynamic data structures). These values are computed as:

- MACAS = $2 * N^2 * T$
- MSCA = $2 * N^2 * T$
- MADmin = $N * T$
  MADmax = $N^2 * (T+S)$

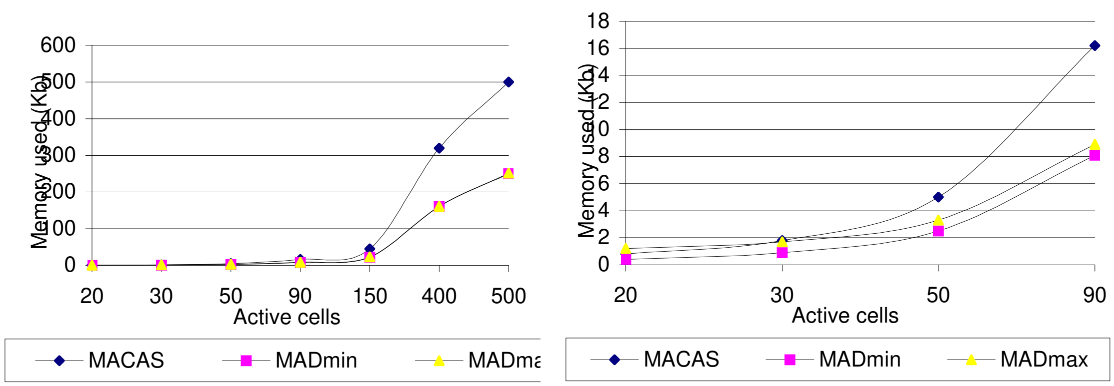
The following figure shows a comparison of these measurements for the Life game using different sizes for the cell spaces.



(a) (b)

Figure 4. (a) Worst cases for the memory use in the Life game (b) Detail.

These results changed in a radical fashion when the average execution times were considered. In this case, the number of elements in the Next-events queue is much smaller than the estimated worst case; therefore, the memory overhead can be reduced.



(a) (b)

Figure 5. (a) Average cases for 1000 initial cells. (b) Detail.

3.2. Experiences with Cell-DEVS models

The CD++ [BBW98] tool allows to specify Cell-DEVS models to be simulated. The cell's behavior is described using a specification language, including the size for the cell's space, their delays, the borders, and the neighborhood. With these basic parameters the tool builds the cellular model and the internal coupling. As a first experience, the Life game was specified using the tool, as seen in the following figure.

```
[top]
components : life
```

```
[life]
type : cell
width : 20
height : 15
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1) life(0,0) life(0,1)
neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialrowvalue :  0  01111000111100011110
initialrowvalue :  1  00010001111000000000
initialrowvalue :  2  00110111100010111100
initialrowvalue :  3  00110000011110000010
initialrowvalue :  4  00101111000111100011
initialrowvalue : 10  01111000111100011110
initialrowvalue : 11  00010001111000000000
initialrowvalue : 12  00110111100010111100
initialrowvalue : 13  00110000011110000010
initialrowvalue : 14  00101111000111100011
localtransition : life-rule
[life-rule]
rule : 1 100 { (0,0) = 1 and truecount = 4 }
rule : 1 100 { (0,0) = 0 and truecount = 3 }
rule : 0 100 { t }
```

Figure 6. Life game specification using CD++.

In the next figure, three execution steps (obtained using the tool) are shown.

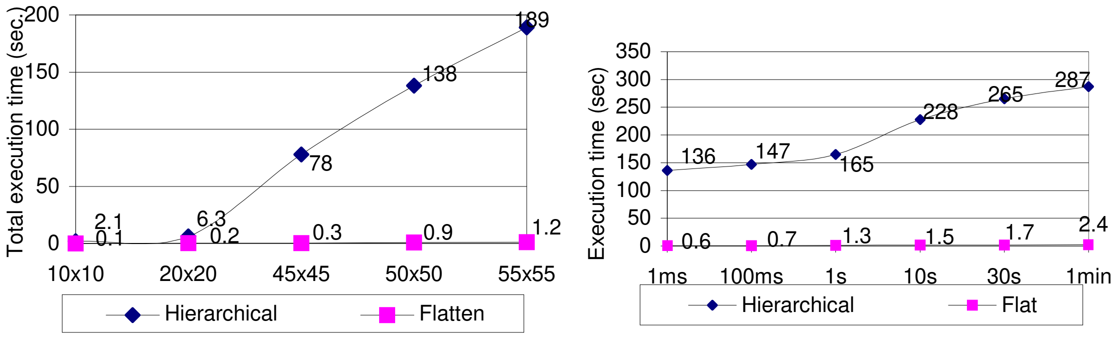
```
Time: 00:00:00:000
    01234567890123456789
   +--------------------+
  0|                    |
  1|                    |
  2|                    |
  3|                    |
  4|                    |
  5|       ***          |
  6|                    |
  7|     *  *  *        |
  8|     * *** *        |
  9|     *  *  *        |
 10|                    |
 11|       ***          |
 12|                    |
 13|                    |
 14|                    |
 15|                    |
 16|                    |
 17|                    |
 18|                    |
 19|                    |
   +--------------------+
```

```
Time: 00:00:01:000
    01234567890123456789
   +--------------------+
  0|                    |
  1|                    |
  2|                    |
  3|                    |
  4|        *           |
  5|        *           |
  6|       * *          |
  7|      *****         |
  8|    ** * * **       |
  9|      *****         |
 10|       * *          |
 11|        *           |
 12|        *           |
 13|                    |
 14|                    |
 15|                    |
 16|                    |
 17|                    |
 18|                    |
 19|                    |
   +--------------------+
```

```
Time: 00:00:02:000
    01234567890123456789
   +--------------------+
  0|                    |
  1|                    |
  2|                    |
  3|                    |
  4|                    |
  5|       ***          |
  6|      *   *         |
  7|     *     *        |
  8|     *     *        |
  9|     *     *        |
 10|      *   *         |
 11|       ***          |
 12|                    |
 13|                    |
 14|                    |
 15|                    |
 16|                    |
 17|                    |
 18|                    |
 19|                    |
   +--------------------+
```

Figure 7. First three steps of execution of the Life game.

The example was simulated using the hierarchical and flat approaches. The number of messages involved in the flat simulation is reduced because interaction only occurs between the higher level coordinator and the Root coordinator. The results seen in the following figure consider ten cycles of simulation applying both approaches and increasing the number of cells in the space. In every case with more than 75% of cells were active initially. The second test used a fixed size for the space (2500 cells), and the duration of the simulation was increased. The results obtained can be seen in the following figure.



(a) (b)

Figure 8. Execution of the Life game. (a) Increasing the size of the space. (b) Simulation duration.

The Figure 9 shows the message interaction obtained by the execution of the models in a hierarchical fashion (obtained by the log file generated by CD++).

It can be seen that the simulation process begins by sending an initialization (**I**) message to the external model (top), that will be in charge to distribute them through the cell space (see (a) in the figure). Each of the cells replies with a **done (D)** message (see (b) in the figure), telling the time for the next internal event (in this case, 100 milliseconds, that is, the delay defined in the specification).

When the top coordinator receives all the **done** messages – (c) in the figure -, it transmits the time of the imminent child to the root coordinator. In this way, the global time of the simulation is advanced. The root coordinator replies using a *-message (see (d) in the figure), that will be sent to the imminent cells. These cells will execute, and their simulators will generate **Y** and **done** messages (see (e) in the figure).

Finally, the coordinator analyzes which output of a non-quiescent cell should be sent to the neighbors (see (f) in the figure), generating **X** messages. The cycle will be repeated up to the moment where all the external events are consumed, or when the simulation time is consumed (in this case, 500 milliseconds, as it can be seen in the item (g) of the figure).

Message I / 00:00:00:000 / Root(00) for top(01) (a)

```
Message I / 00:00:00:000 / top(01) for life(02)
Message I / 00:00:00:000 / life(02) for life(0,0)(03)
Message I / 00:00:00:000 / life(02) for life(0,1)(04)
...
Message I / 00:00:00:000 / life(02) for life(19,19)(402)
Message D / 00:00:00:000 / life(0,0)(03) / ... for life(02)
Message D / 00:00:00:000 / life(0,1)(04) / 00:00:00:100 for life(02)          (b)
Message D / 00:00:00:000 / life(0,2)(05) / 00:00:00:100 for life(02)
...
Message D / 00:00:00:000 / life(19,19)(402) / ... for life(02)
Message D / 00:00:00:000 / life(02) / 00:00:00:100 for top(01)                (c)
Message D / 00:00:00:000 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:100 / Root(00) for top(01)                               (d)
Message * / 00:00:00:100 / top(01) for life(02)
Message * / 00:00:00:100 / life(02) for life(0,1)(04)
...
Message * / 00:00:00:100 / life(02) for life(19,17)(400)
Message Y / 00:00:00:100 / life(0,1)(04) / out / 0.000 for life(02)           (e)
Message D / 00:00:00:100 / life(0,1)(04) / ... for life(02)
Message Y / 00:00:00:100 / life(0,2)(05) / out / 0.000 for life(02)
...
Message D / 00:00:00:100 / life(19,16)(399) / ... for life(02)
Message Y / 00:00:00:100 / life(19,17)(400) / out / 1.000 for life(02)
Message D / 00:00:00:100 / life(19,17)(400) / ... for life(02)
Message X / 00:00:00:100 / life(02) / neighborchange / 0.000 for life(19,0)(383)        (f)
Message X / 00:00:00:100 / life(02) / neighborchange / 0.000 for life(19,1)(384)
...
Message X / 00:00:00:100 / life(02) / neighborchange / 1.000 for life(18,18)(381)
Message D / 00:00:00:100 / life(19,0)(383) / ... for life(02)
Message D / 00:00:00:100 / life(19,1)(384) / ... for life(02)
Message D / 00:00:00:100 / life(19,2)(385) / 00:00:00:100 for life(02)
...
Message D / 00:00:00:100 / life(18,18)(381) / ... for life(02)
Message D / 00:00:00:100 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:100 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:200 / Root(00) for top(01)
Message * / 00:00:00:200 / top(01) for life(02)
Message * / 00:00:00:200 / life(02) for life(19,2)(385)
...
[The simulation cycle is repeated]
Message D / 00:00:00:500 / life(12,8)(251) / ... for life(02)
Message D / 00:00:00:500 / life(12,9)(252) / ... for life(02)
Message D / 00:00:00:500 / life(02) / ... for top(01)                         (g)
Message D / 00:00:00:500 / top(01) / ... for Root(00)
```

Figure 9. Message traffic during the execution of the hierarchical model.

After, the same test was repeated using the flat coordinator. As it can be seen, in this case the message transmission has been greatly reduced, because the only interaction occurs between the higher level coordinators and the root coordinator, avoiding every other interaction.

```
Message I / 00:00:00:000 / top(01) for life(02)
Message D / 00:00:00:000 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:000 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:100 / Root(00) for top(01)
Message * / 00:00:00:100 / top(01) for life(02)
Message D / 00:00:00:100 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:100 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:200 / Root(00) for top(01)
Message * / 00:00:00:200 / top(01) for life(02)
Message D / 00:00:00:200 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:200 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:300 / Root(00) for top(01)
Message * / 00:00:00:300 / top(01) for life(02)
Message D / 00:00:00:300 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:300 / top(01) / 00:00:00:100 for Root(00)
```

```
Message * / 00:00:00:400 / Root(00) for top(01)
Message * / 00:00:00:400 / top(01) for life(02)
Message D / 00:00:00:400 / life(02) / 00:00:00:100 for top(01)
Message D / 00:00:00:400 / top(01) / 00:00:00:100 for Root(00)
Message * / 00:00:00:500 / Root(00) for top(01)
Message * / 00:00:00:500 / top(01) for life(02)
Message D / 00:00:00:500 / life(02) / ... for top(01)
Message D / 00:00:00:500 / top(01) / ... for Root(00)
```

Figure 10. Message traffic during the execution of the hierarchical model.

The one-way traffic model was also implemented. It can be seen that a model more complex than the original can be specified in a simple fashion. This case includes the specification for a bump in the street. The model specification can be seen in the following figure:

```
components : CellCoupled
out : out
in : in
link : in in@CellCoupled
link : out@cellcoupled out

[CellCoupled]
type : flat                    width : 50                   height : 10
delay : transport              defaultDelayTime : 1000      border : wrapped
neighbors :          CellCoupled(-1,0)
neighbors : CellCoupled(0,-1) CellCoupled(0,1)
neighbors :          CellCoupled(1,0)
in : in
out : out
link : in in@CellCoupled(0,0)
link : in in@CellCoupled(7,1)
link : in in@CellCoupled(6,3)
link : out@cellcoupled(9,0) out
link : out@cellcoupled(9,1) out
link : out@cellcoupled(9,2) out
initialvalue : 1
initialrowvalue : 0  10101010101000101110010101010101001?????101000000000
initialrowvalue : 1  0100010101101??0101??0011010101010100010111001010101
initialrowvalue : 2  10101010101000101110010101010101001?????101000000000
initialrowvalue : 3  0010101010101010101001?000000000101010101001?????10101
initialrowvalue : 4  00101010??1010000000000010100010111001010101010101001??
initialrowvalue : 5  1010101010101010101001?000000000101010101001?????10101
initialrowvalue : 6  0010101010101010101001?000000000101010101001?????10101
initialrowvalue : 7  10101010101000101110010101010101001?????101000000000
initialrowvalue : 8  10101010101000101110010101010101001?????101000000000
initialrowvalue : 9  00101010??1010000000000010100010111001010101010101001??
localtransition : transit
zone : bump { (8,8)..(9,9) (1,2)..(4,4) }


[transit]
rule : 1 5 { (0,0) = 0 and (1, 0) = 1  }
rule : 1 5 { ( 0, 0 ) = 0 and (0, -1) = 1  }
rule : 1 5 { (0,0) = 1 and (0,1) = 1  }
rule : 1 5 { ( 0,0) = 1 and (1,0) = 1  }
rule : 0 10 { (0,0) = 1 and (-1, 0) = 0 }
rule : 0 10 { (0,1) = 0 and (0,0) = 0 }
rule : 0 10 { ( 0,0) = 1 and (1,0) = 1 }
rule : 1 5 { t }

[bump]
rule : 0 1 { t }
```

Figure 11. Specification of an urban traffic model using CD++.

A fixed size cell space was considered (60% of active cells), and the length of the simulation was changed. A second test considered a more complex scenario. In this case, the states of certain cells are unknown and there are certain zones with different behavior. Both results can be seen in the following figure.

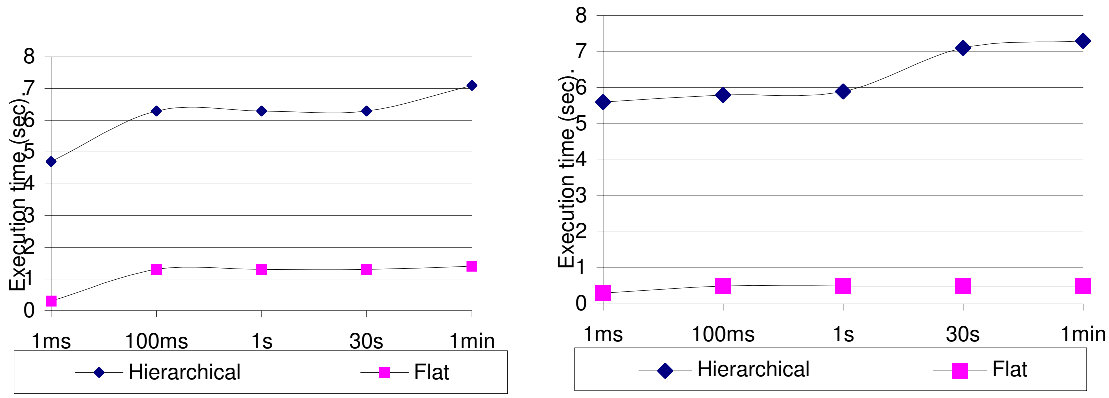


Figure 12. Hierarchical and flatten traffic simulation.

These tests were repeated studying the overhead used in initialization messages, as it can be seen as follows:

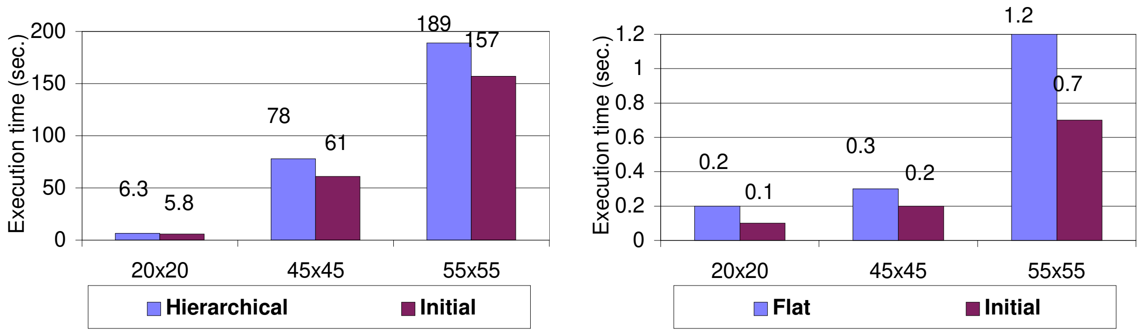


Figure 13. Initialization time (a) Hierarchical (b) Flat.

In this case, as the simulation cycle was short (due to the testing features of the simulations), the initialization time cannot be amortized in the simulation. The case is worst for the hierarchical solutions.

Finally, the performance of the environment was tested against the execution of the same models implemented by hand. Synchronous and asynchronous versions were implemented, and their execution times were compared with those of the flat models. The worst execution cases were considered. That includes a case where the asynchronous automaton behaves worst than the synchronous case. Also, the overhead of initialization was considered, to make a fair comparison between the application coded "by hand" and the execution overhead introduced by the tool. The results can be seen following:

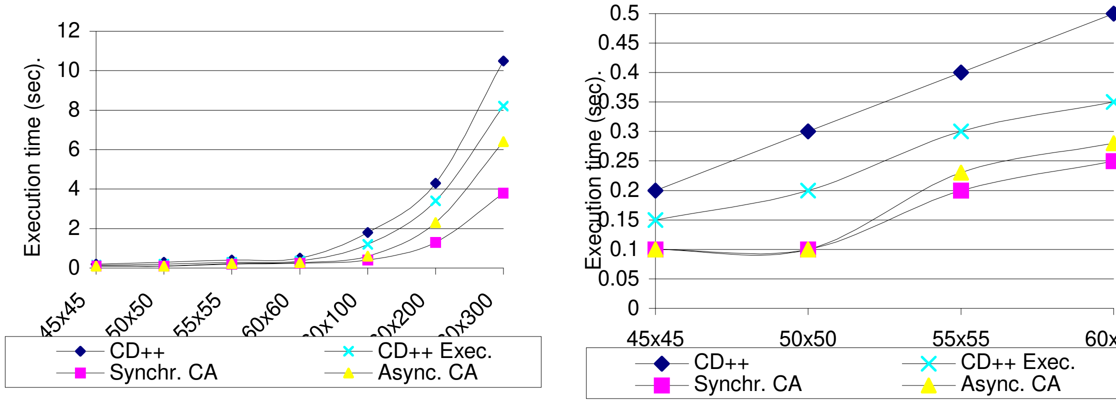


Figure 14. (a) Execution results of the worst cases for the Life game. (b) Detail.

It can be seen that the tool introduced approximately 20% of overhead for this case. Besides, parallel execution can solve this problem without the need of reprogramming the model, as the specification is the same in both cases. Here, the simulation technique is changed independently of the model's specification.

A main goal when defining the new formalism was to reduce the development times for the simulators. The results obtained were promising though the developed experiences were simple prototypes used to check the use of the tools and the formalism. Several data were recorded relating the development times for the different solutions, classifying the different users and their development activities.

Learning time to use the tool: 16 m/h				
Novice developers	Life		Traffic	
	Development	Testing	Development	Testing
Specification of the first problems	0.75 m/h	2.50 m/h	2.50 m/h	4.3 m/h
Specification of the last problems	0.25 m/h	0.75 m/h	1.00 m/h	1.7 m/h
Maintenance	0.25 m/h	0.30 m/h	0.25 m/h	0.3 m/h

Expert developers	Life		Traffic	
	Development	Testing	Development	Testing
Specification of the first problems	0.50 m/h	1.30 m/h	0.50 m/h	0.75 m/h
Specification of the last problems	0.10 m/h	0.30 m/h	0.30 m/h	0.75 m/h
Maintenance	0.25 m/h	0.30 m/h	0.25 m/h	0.50 m/h

Manual programming (Asynch. CAs)	Life		Traffic	
	Development	Testing	Development	Testing
Specification of the first problems	5 m/h	9 m/h	8 m/h	11 m/h
Specification of the last problems	3 m/h	5 m/h	2 m/h	3 m/h
Maintenance	4 m/h	12 m/h	3 m/h	7 m/h

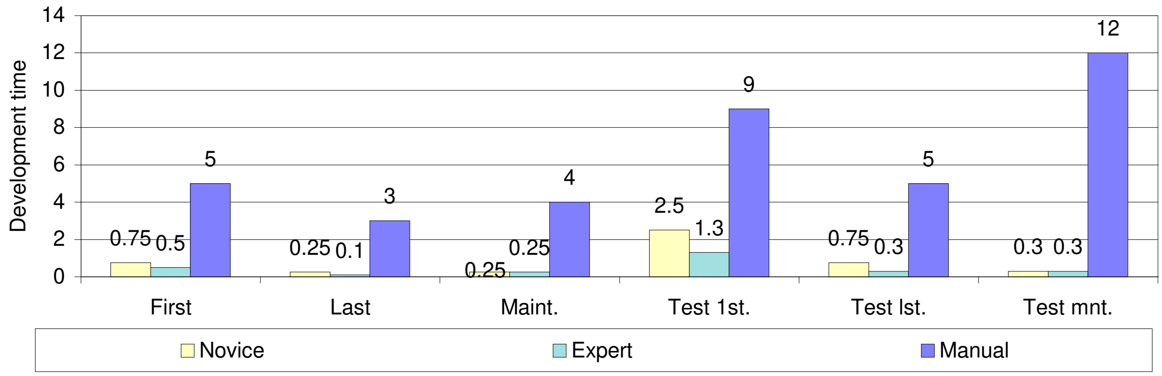


Figure 15. Comparison of development times for the Life game.

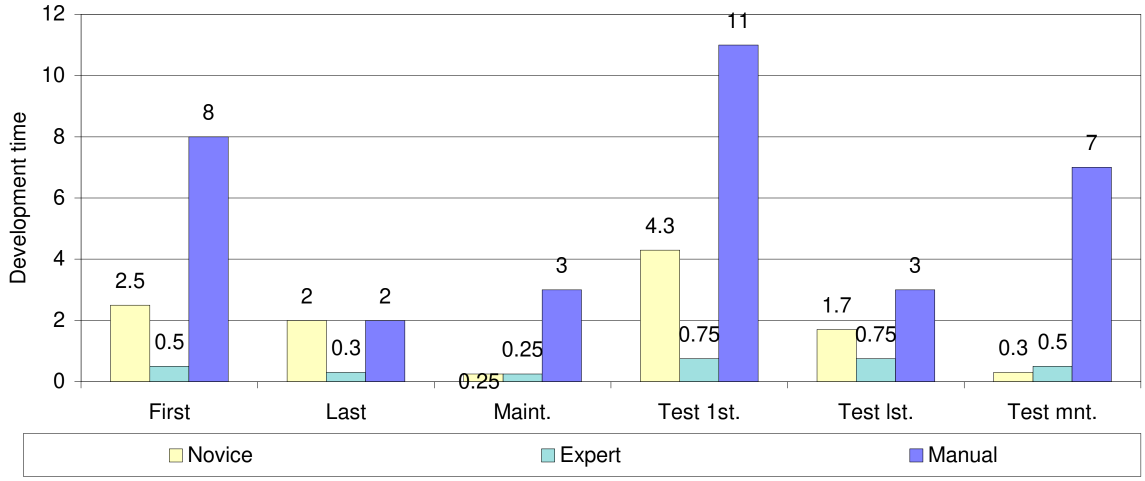


Figure 16. Comparison of development times for the traffic simulation.

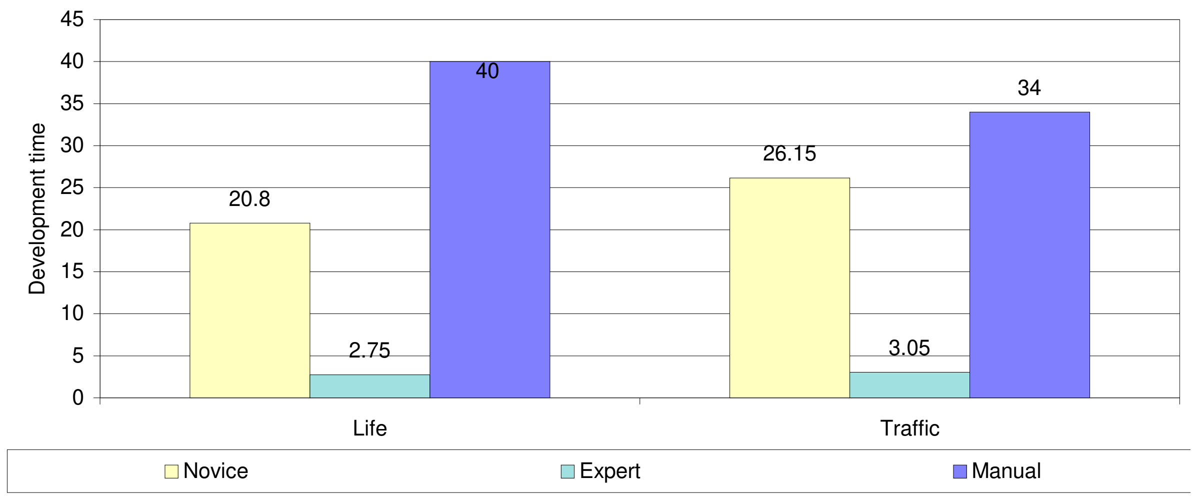


Figure 17. Comparison between total development times for both applications.

4. CONCLUSION

The implementation results of a new paradigm for modelling and simulation of cellular models were analyzed. An asynchronous approach was considered, because in several applications it allows to improve the simulation execution times. Here, it could be seen that the cell spaces do not always meet this goal, but in most of cases, it does. A large number of cells against few active ones produce performance improvements. In the cases that the cell's neighborhood is greater, the management costs of the event lists are increased. Instead, those models where a cell does not affect directly to a neighborhood consisting of many cells, the list management requires little overhead.

In most execution examples, the discrete event cases behoove better than the synchronous ones. This happened because the proportion of active cells was small. The worst results were obtained for the Wa-Tor model, because the increasing in the number of specimen only produces higher activity. Besides this fact, these worst cases are not relevant for the application, due that this situation does not reflect the real activity in the ocean (where the fish concentration is much lower than the tested cases). As a conclusion, the model's activity must be considered prior its implementation as discrete events, because a high ratio of non-quiescent cells can produce bad performance. For these models, a synchronous approach should be used.

The use of a modelling tool introduced a degree of overhead, mostly by the initialization messages. This happened because the tested solutions only executed for short times, and the initialization activity can be better amortized in longer simulations. Besides, executing the models in parallel can enhance the performance.

Apart from the overhead introduced, the main achievements obtained were related with the development times of the simulations. Ten-fold improvements could be achieved for expert users although the tested models were simple prototypes. This occurred because the test and maintenance phases are the more expensive for these systems, and these times were highly reduced by formal specification. Besides, if a parallel coordinator is introduced, an execution speedup of several orders of magnitude can be achieved without changing the specifications. Instead, a parallel implementation for each problem hand-coded can produce ten-fold delays in the development.

5. REFERENCES

[BBW98] BARYLKO, A.; BEYOGLONIAN, J.; WAINER, G. “CD++: a tool to implement binary Cell-DEVS models” (in Spanish). In *Proceedings XXII Latin American Conference on Informatics.* Quito, Ecuador. 1998.

[Gar70] GARDNER, M. "The fantastic combinations of John Conway's New Solitaire Game *Life*". *Scientific American*. 23 (4). pp. 120-123. April 1970.

[OSH92] OVEREINDER, B.; SLOOT, P.; HERZBERGER, L. "Time-Warp on a transputer platform: pilot study with asynchronous cellular automata". In *Parallel computing and transputer applications.* Amsterdam, Netherlands. Washington: IOS Press. September 1992.

[WG98] WAINER, G.; GIAMBIASI, N. "Specification, modelling and simulation of timed Cell-DEVS models". *Technical Report 97-007, Departamento de Computación, FCEN/UBA. Submitted to publication.* 1998.

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