AN ENVIRONMENT FOR CELLULAR DEVS MODEL SIMULATION

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

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

Claudia Frydman §
Claudia.Frydman@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

In this work we present the features of a simulation framework for cellular DEVS models. The goal is to allow the users the easy development of simulations based on cellular DEVS models with high efficiency. To do so, several changes to the basic DEVS class hierarchy and modifications to the abstract simulation methodology are introduced. The framework will execute these models in parallel or distributed architectures to improve their execution times.

1. Introduction

To build flexible, complex and highly precise automated systems with a cost effective approach, tools and methodologies for computer simulation are frequently used. Simulation allows to manage high degree of complexity, helping researchers and developers to model complex phenomena that otherwise cannot be studied.

At present there are several different formalisms to specify simulation models. In this paper we are interested in one of these paradigms, known as DEVS (Discrete EVents Systems specification). It is a discrete event formalism proposed by Zeigler [Zei76], that allows modular description of the phenomena to model and attacks the complexity using a hierarchical approach. This approach was later integrated with the notions of Object Oriented programming [Zei90].

The model is described as a set consisting of a time base, inputs, states, and outputs. Functions to compute the next states and outputs are also included. The DEVS formal representation of discrete event systems can be mathematically handled (in the same sense of the differential equations for continuous systems).

A DEVS model can be composed of atomic behavioral submodels. Basic models are composed to conform structural models called coupled models. As the formalism is closed under closure, coupled models can be integrated to the simulation hierarchy. The use of hierarchical modeling allows the creation of a model Data Base, permitting the reuse of created and tested models. In this way the security of the simulations is enhanced, reducing the testing time and improving productivity.

The Cellular Automata formalisms [Wol86] are well suited to describe some kind of real complex systems with different description levels. In general, time, space and system states are discrete. A conceptual cellular automaton is an infinite regular n-dimensional lattice where each cell in the lattice can take one 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 transition function using the present cell state and a finite set of nearby cells (called the cell's neighborhood).

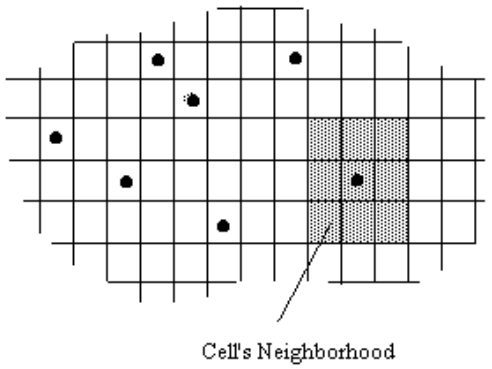

Figure 1. Sketch of a Cellular Automaton

Formally, a cellular automata is defined as CA=<S,N,T>. S $\in$ **Z** is the set of values for the state, N is the neighborhood, and T is the global transition function. The local transition function is defined as follows. If $a_{ij}(t)$ is the state for cell (i,j) in simulated time t (two-dimensional cellular automata), the automaton evolves according with:

$$a_{ij}(t+1) = \tau[a_{i-r,j-r}(t),..., a_{i,j}(t),..., a_{i+r, j+r}(t)]$$

where τ denotes the local transition function for the automaton. The parameter r specifies the region affected by a given site (i.e., the neighborhood).

The use of cellular automata to simulate complex systems requires large amounts of compute time, but in general, at each time step there are several cells where actualization is not needed. The use of a discrete time base also poses restrictions in the precision of the model. The **asynchronous cellular automata** formalism allows to avoid these problems [Ove93].

The main aspect of asynchronous automata is their evolution in continuous time. Events are instantaneous and can occur asynchronously at unpredictable times. Higher time precision can be achieved and periods of inactivity in the simulation are skipped but explicit synchronization of the cells must be done. In some cases this overhead of event list handling can make the synchronous approach behave better than the asynchronous one, specially when there are a high number of active cells.

The goal of our project is to provide a framework to simulate asynchronous cell spaces consisting of DEVS cell models (one atomic model in each cell). The designers should only know the underlying formalisms and build the local transition functions, avoiding other implementation questions. In this way, safety and development costs can be improved.

2. An environment for cell-DEVS simulation

Our goal is to build a simulation framework for cellular DEVS models. Two different points of view have been taken into account. From the user point of view, the idea is to provide a user-friendly environment such that the simulation activities can be accomplished in a safe and cost-effective fashion. To do so, three aspects of Cell-DEVS models have been taken into account: dimension, influencees and behavior.

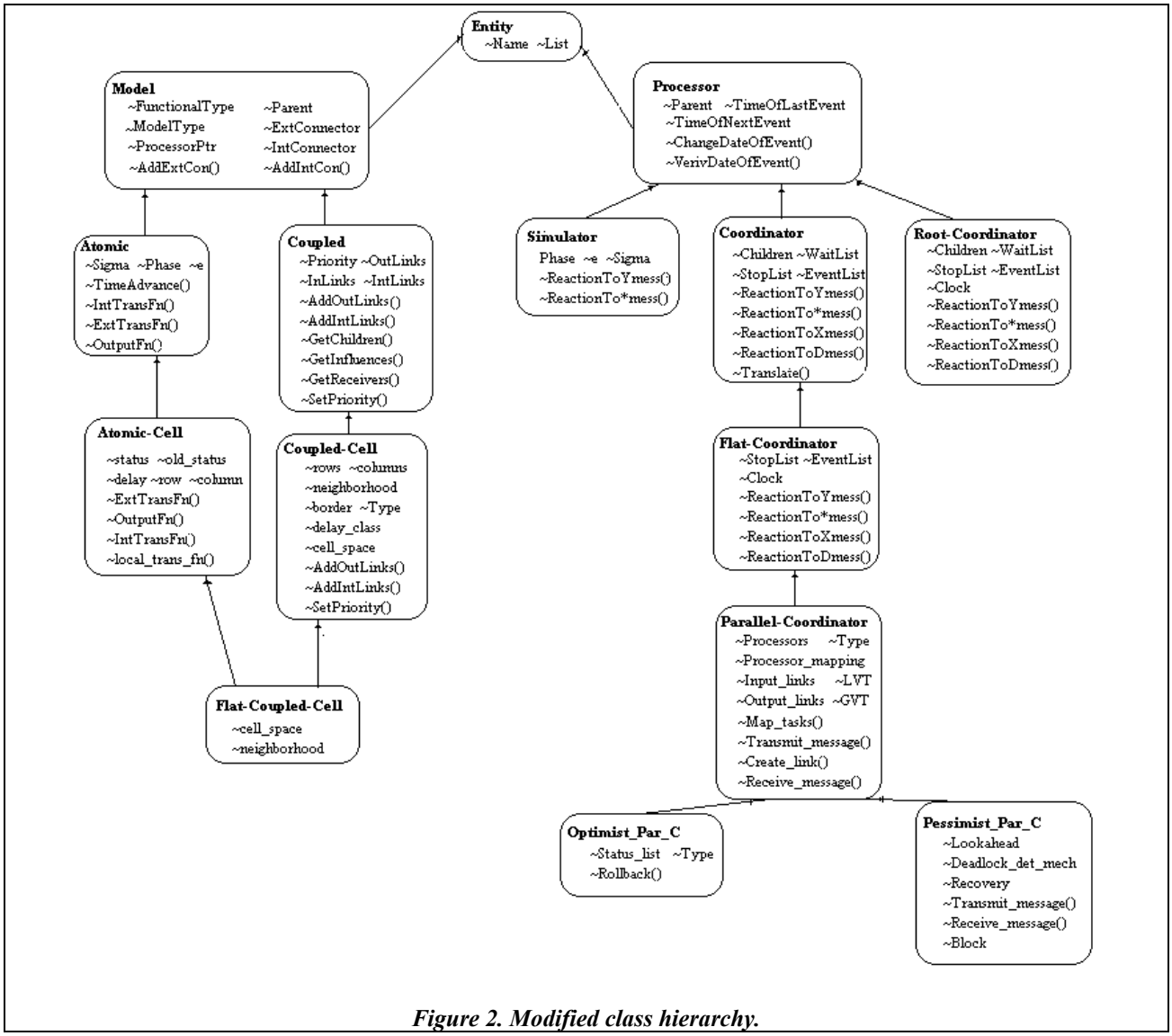

Figure 2. Modified class hierarchy.

From the designer's point of view, the environment is based on the asynchronous automata formalism. This approach has been taken because this formalism allows better representation of the time, and also allows to reduce the CPU time requirements. The DEVS formalism is also used, taking as a basis a previously defined tool to

simulate DEVS models [Gia95]. We have modified the class hierarchy defined in that framework (similar of those of [Zei90]) to allow the definition of a complete cell-space model in a parametric way.

The changes can be seen in the redefinition of the original class hierarchy presented in figure 2. Recalling the functionality of this hierarchy, we can see two main classes derived of Entity class: Models and Processors. Each model can be Atomic or Coupled. The new Atomic Cell class redefines the Atomic one.

2.1. BASIC CELL BEHAVIOR

Each cell has a state set, a behavior and an actualization delay. To represent the behavior of each cell, a simple specification language for local transition functions has been defined. In this way, the user of the environment can define complete simulation models without programming.

At present, this language only allows binary states and transport-delay definition (in the future it will include other discrete states different than binary values, and delays with preemptive semantic). A compiler translates the specification to the internal behavior representation of the cell-DEVS models.

When an input message occurs, the external transition function queries the state of each neighbor, and computes the state of the cell. After that, the cell state affection is delayed according with the transport delay of the cell (it is added in the next-events list).

When the delay time has been consumed, it is verified if the state has changed. In this case, the internal transition function must be executed (otherwise, it is not activated). Before calling the internal transition function, the output function is executed and the new state is sent to the coordinator. This is repeated for every input event stored in the waiting queue. When this queue is empty, the cell passivates. The coordinator sends the changes to every cell in the neighborhood (as changes in one cell influences the neighbors).

2.2. HIERARCHICAL CELL SPACES

2.2.1. Hierarchical description

Following, the question relates with the dimension and coupling of the models. A complete DEVS cell space is automatically filled out by defining:

- Neighborhood: it defines the coupling of model components (cells). The neighborhood can be lately overloaded in the definition for each cell. In this way, general cell spaces with non-uniform neighborhoods can be defined.
- Cell-space size.
- Borders (wrapped cell-spaces or self-state generating borders).
- Initial state for the cell space.
- Priorities to treat simultaneous events: these are used to classify the imminent cells in the cell space. It is used to define the *Select* function of the DEVS specification (or the δcon function if the R-DEVS [Cho94] formalism is used).

With these parameters, an array of atomic cell objects is created and the influences for each cell are defined. To build a simulator the user only need to define these parameters and the local transition function (using the specification language).

The initial parameters for the DEVS cell space are loaded, defining the coupling of the component sub-models. This coupling is carried out using the information provided by the neighborhood. The internal coupling is defined for each cell as a cell pair, with each cell in the inverse neighborhood. This is made because a cell is influenced by its inverse neighborhood [Zei76].

2.2.2 Simulation Processors

Processor sub-classes have also been redefined. As we can see, there are different coordinators. First, the standard coordinator is associated with hierarchical coupled cell models. When a coupled cell model is defined, the internal coupling is set up (as previously stated), and a hierarchical coordinator is associated with the coupled model. Flat coordinators will be explained later.

Model interaction is carried out as in other coupled models. The coordinator chooses the imminent model (in this case, one cell), and sends its simulator a *-message. When the simulator receives the message, the internal transition function is activated. When an external message arrives, an X-message is sent and the external transition function executed. The simulators return done and Y-messages that are converted to new * and X-messages, respectively. The messages are translated using the cell coupling.

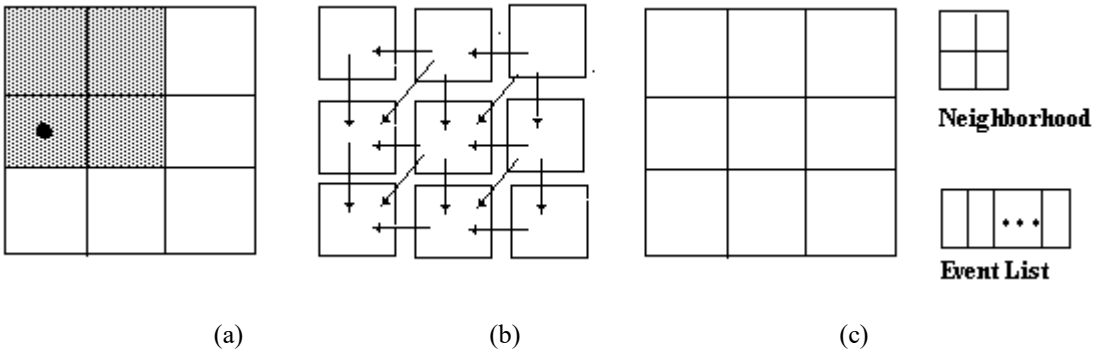

Figure 3 - Cell space structures. (a) Basic cell space. (b) Definition using hierarchical coupled models. (c) Definition using flat models.

2.3. FLAT CELL SPACES

In hierarchical models, the interaction between models is done through coordinators that communicate through message passing. This message interaction increases overhead of a hierarchical simulation.

To avoid this interaction, the model should be flatten. The goal of a flat simulator is to avoid the interaction of the translation functions, and to improve execution times. A flatten simulator is implemented as an array of registers associated with the cells, including states, information about the cell's influencees and other fields indicating change of state. A next-event list is also maintained. Here, the message interaction is not needed, as the cells array is used to detect changes in the cell space, updating the next-event list. The differences between the two approaches can be seen in Figure 3.

The simulator starts detecting quiescent states for the cell space initial state. Non-quiescent cells (v.g., those whose state can change) are added in the next-events list ordered by next-event time.

When the local transition function executes, it is checked if the cell state has changed. In that case, the next event time for the cell is created and added in the next-event list for each cell lin the neighborhood. Previously, a Y-message containing the cell state is created and transmitted to the upper level coordinator. The coupling between cells is done by using the inverse neighborhood (coupling functions are avoided).

2.4. CELL MODELS IN THE MODEL HIERARCHY

To allow the integration of flat models with other DEVS models, a new flat coordinator hass been also defined. The idea is to allow the integration of flat cells models with other models in a model hierarchy. In this way, the outputs of a flat cell model can be the inputs of other DEVS models through model coupling. The other models can be also flat cell models, allowing the integration of several devs models with different local transition functions.

The flat coordinator is activated when a *-message is received, and activates the corresponding cell simulator. X-

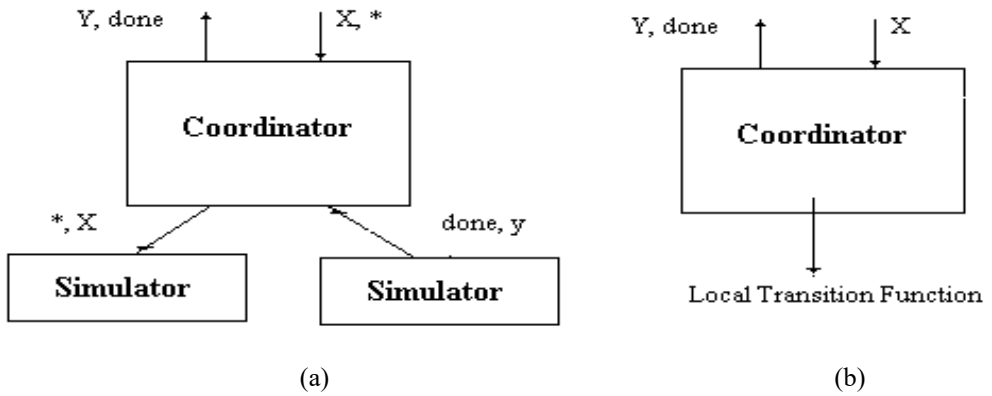

Figure 19. Coordinator reaction to messages (a) Hierarchical (b) Flat.

messages are simply added in the next-event list of the simulator. Y-messages received from the flat simulator

are transmitted to the upper level coordinators (see figure 4).When the simulation finishes, a done-message is created. This message can be sent to other models in the system.

When a cell coupled model is saved in the model data base we only record parameters for the model. When it is reloaded, the model is converted in a flat one.

As the execution of this kind of models can be time consuming, other extensions to allow parallel execution are presented in the following section.

3. Parallel simulation of Cell-DEVS models

When complex systems are simulated, usually a high number of computations are carried out. If the computations are carried out sequentially, it is difficult to obtain a significant sample to study the desired problem. Moreover, in this case, models that are inherently parallel, must be executed sequentially. This is the case for cell-DEVS models.

To improve the utilization of system resources, the environment will allow the execution of the cell models in parallel. At present, many solutions to improve the performance of the simulations using parallel and distributed approaches (see, for instance, [Fuj90]). At present there are two main approaches for asynchronous parallel discrete event simulation: pessimist [Cha79] and optimist mechanisms [Jef85]. Our goal is to study their performance for Cell-DEVS models.

As stated earlier, cellular automata are inherently parallel. As most cellular automata's formalisms rely on discrete time steps we need to use asynchronous approaches such as those ones presented in the previous section. The idea is to improve execution times implementing coordinators in a parallel or distributed fashion.

A flat coordinator is associated with to each processor, syncrhonizing through optimist or pessimist approaches. Each coordinator is coded as a logical process including three event lists: a local event list, and input and output links. The simulation is divided according with the power of each processor to allow balanced distribution.

The coordinator selects locally 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 and transformed as explained in section 2. If the new event belongs to the local processor, it is added in the local 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, it is added as an input message.

As we could see in figure 2, different approaches (optimist and pessimist) will be included using the recently presented strategies. In this way, the chosen approach can be fitted to the application, obtaining improved results for each case.

4. Design and implementation: present status

We have organized the analysis and development of the environment in stages proposing a layered design organized in four layers (see figure 5).
Layers 2 and 3 include the class hierarchy presented in section 2. A main goal of layer 2 will be to provide an interface between the DEVS class hierarchy and the available operating environment. In this way, changing the parallel layer the tool can run in different programming plattforms with few changes. This layer will execute the functions defined in section 3, and optimize the execution of the cellular DEVS models distributing the execution load.

At present, our work is focused in developing Layer 3. The implementation of atomic cellular models and coupled cellular models, and the implementation of the abstract simulator are being finished.

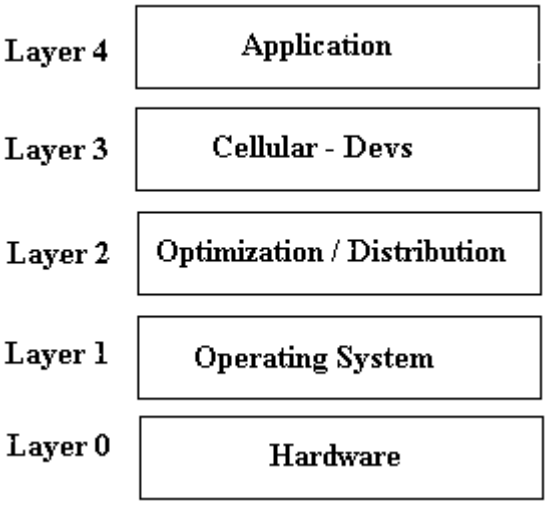

Figure 5. Layered design of the environment

Layer 2 will be implemented as a class library using optimist and pessimist approaches. As stated earlier, the Cell-DEVS coordinator will run in parallel and we will test several of the proposed heuristics. To improve parallel execution, the environment will be implemented using the R-DEVS formalism [Cho94], that allows parallel execution of the DEVS models.
Several conditions will also be tested: Optimist vs. pessimist approaches, influence of the processing order of the local and external event lists, and self-adjustment of parameters of a Rollback Number model that has been defined.

Finally, Layer 4 will be devoted to test the efficiency of the algorithms, using complex cellular models. We have

put into consideration the simulation of cellular communications, urban traffic, hydrology problems and routing in massively parallel computers. This layer will provide an environment used to test a specific problem, and to see the efficiency of the previously implemented algorithms and the parallel framework.

The applications will have as main goal to test the newly developed environment, and to analyze its performance. Special emphasis will be put in significant metrics, such as acceleration of the parallel models, throughput and execution performance. Rollback number, null messages (to study the merit of the parallel algorithms implemented) and finally, application development and maintenance times also will be studied.

5. Conclusion

In this work, the definition of an environment to simulate cellular models in an efficient fashion has been presented. The DEVS and asyncrhonous Cellular Automata formalisms have been employed as a base.

The utilization of a discrete events formalism such as these ones can provide better precision and performance. The use of a formal modelling technique is also crucial to improve development and maintenance costs of the simulators. Its use can also provide better system understanding.

To improve execution times, the hierarchical models has been flattened. They also can run in parallel. To do so, traditional optimist and pessimist approaches will be used, and they will be adapted to run this kind of models.

As a result, a high performance environment for cellular models will be provided. The developer is free of the implementation details of the underlying formalisms. He only must be concerned with the definition of a parameter set and a local transition function defined through a specification language, allowing him to concentrate in high-level decisions and avoiding implementation issues.

References

[Cha79] CHANDY, K; MISRA, J. "Distributed simulation: a case study in design and verification of distributed programs", IEEE TOSE, September 1979.

[Cho94] CHOW, A.; ZEIGLER, B. "Revised DEVS: a parallel, hierarchical, modular modeling formalism". Proceedings of the. Winter Simulation Conference., 1994.

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

[Gia95] GIAMBIASI, N.; FRYDMAN, C.; ESCUDE, B. "Hierarchical/Multi-view modeling and simulation". In Proceedings of the 7th. European Conference on Computer Simulation. 1995.

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

[Ove93] OVEREINDER, B.; SLOOT, P.; HERZBERGER, L. "Time-Warp on a Transputer platform: pilot study with asynchronous cellular automata". Technical Report, University of Amsterdam. 1993.

[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.

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