A FRAMEWORK FOR VISUAL SPECIFICATION AND SIMULATION OF CELLULAR SYSTEMS

Ernesto POSSE
McGill University
School of Computer Science
3480 University Street
H3A 2A7 Montreal, Canada

Alexandre MUZY
Università di Corsica
SPE lab, UMR CNRS 6134
Campus Grossetti, BP 52
20250 Corti, FRANCE

Hans VANGHELUWE
McGill University
School of Computer Science
3480 University Street
H3A 2A7 Montreal, Canada

ABSTRACT

Cellular systems specification is an important research topic as it allows modeling structure and behavior of many real-world systems. We use meta-modeling, model-transformation, and the Discrete-EVent system Specification (DEVS), for the automatic construction and simulation of cellular models. Our focus here is on the generation of those models through graphical means (using meta-modeling and model-transformation concepts), rather than on their efficient simulation. Through the presented framework, complexity of systems can be adaptively and precisely tackled.

Keywords: Cellular models, Meta-modeling, graph-transformation, DEVS, AToM^3.

1 INTRODUCTION

A cellular system consists of a spatial arrangement of interacting sub-systems. Local interactions and behaviors of sub-systems lead to global behavior due to their influence on the state of neighboring sub-systems. At a specification level, these local interactions and behaviors are given once for diverse zones and replicated subsequently throughout the whole system.

To specify certain classes of cellular systems, modelers can benefit from a generic, simple and easily modifiable high-level visual specification of both cellular topology (including sub-system interconnections) and sub-system behaviors. If the visual notations are sufficiently non-technical and possibly domain-specific, it becomes possible for domain-specialists in different disciplines to model cellular systems. Furthermore, if a graphical or semi-graphical mechanism is provided to automatically transform such visual specifications into simulation code, the modeler obtains three benefits: first, the generated code is guaranteed to be robust (whereas manually developing the code requires verification); second, the entire process from visual specifications to generated code is in the form of explicit model transformations, which can be modified with little programming involved in order to yield variations of the cellular systems; and third, such transformations constitute a precise documentation of the process.

To faithfully and efficiently model and simulate complex, spatially distributed systems such as cellular systems, an appropriate formalism has to be defined.

Visual (yet rigorously defined) languages can provide an intuitive and explicit means for specifying cellular models. Meta-modeling can be used [1, 16] to quickly design and modify such languages. Subsequently, according to the system's specification, the mapping between the model and a simulate-able representation needs to be specified. Such specification may be done in the form of graph grammars or by means of direct code-generation.

The Discrete-EVent system Specification (DEVS) [22] is a formalism used to describe complex discrete-event systems in a modular fashion. This paper describes an approach to the specification of cellular DEVS models from a graphical description level, which the automatic generation of suitable representations that can be given as input to discrete-event simulators. The discrete-event simulator used here for validation is PyDEVS [3].

The rest of this paper is organized as follows. In the next section the state-of-the-art of meta-modeling, model-transformation, cellular systems and discrete-event modeling and simulation is presented. In Section 3, the cellular framework is described and in Section 4, the whole approach is validated through an application to the Conway's game of life. Conclusions and discussions of this research, including future research directions, are given in Section 5.

2 STATE OF THE ART

Meta-modeling and graph-transformation can be used to describe cellular systems formalisms in terms of discrete-event modeling and simulation concepts. Note that from such descriptions, interactive, visual modelling environments can be synthesized. In the sequel, these concepts and relations between them are identified.

2.1 Meta-modeling and model transformation

Meta-modeling is the process of modeling the syntax of models in a formalism by means of a meta-model, in an appropriate meta-modeling formalism. As such, it is sometimes referred to as a grammar. For example, to specify the general structure of Petri Net models, one can use models in a formalism such as UML Class Diagrams [14] or Entity-Relationship (ER) diagrams [4].

One of the advantages of meta-modeling is that for many useful formalisms, models and meta-models can be represented in a visual, intuitive way. Furthermore, a graphical representation of models allows the explicitly modeling of operations such as model analysis and code generation by means of *graph-transformation*, using *graph-grammars* [5].

A graph-grammar is the generalization of term rewriting systems to the domain of graphs. A graph-grammar consists of a set of rules of the form $L => R$, where L and R are graphs (called LHS for "Left Hand Side" and RHS for "Right Hand Side" respectively.) Such a rule represents the action of replacing the sub-graph L by R in a given *host graph*. A rule can be applied to a host graph only if its LHS occurs in the host graph. Sometimes the rules have an additional condition and an action. In such case, the condition must be satisfied for the rule to be applicable, and the action represents some side-effect. Furthermore, it is common for nodes and arcs in a graph to have attributes. In the context of a graph-grammar, the attributes of the LHS must be matched by the relevant sub-graph in the host graph in order for a rule to be applicable. Informally, given a host graph and a graph-grammar, all the rules in the grammar will be applied until no more rules can be applied. A *graph-rewriting engine* is an algorithm implementing such a procedure.

Combining meta-modeling and graph rewriting enables us to develop modeling environments with relative ease and with little programming involved, as the process is done at the level of modeling. For this project we have used AToM³ [7], a tool that combines the two techniques.

By using meta-modeling we can describe (the syntax of) the DEVS formalism [13, 8], as well as a formalism to specify cellular spaces. From those meta-models, AToM³ automatically generates a visual modeling environment in which we can specify both the local (DEVS) behavior of cells, and the global structure of the cellular space. The behavior of atomic DEVS components is given by means of state-transition diagrams for the modeler's convenience. From these specifications, we generate the actual cellular DEVS model by means of graph-grammars.

2.2 Cellular systems

Cellular Automata (CA) [17] is a well-known formalism for specifying spatially distributed phenomena. Standard CA consist of an infinite lattice of discrete identical sites, each site taking on a finite selection of, for instance, integer values. The values of the sites evolve synchronously in discrete time steps according to deterministic rules that specify the value of each site in terms of the values of neighboring sites. CA are models where space, time and states are discrete.

This definition of basic CA (infinite lattice, neighborhood and rule uniformity of the cells, closure of the system to external events, discrete

states of the cells, etc.) is too limited to specify complicated cellular models. Scientists often need to tailor the CA formalism to their needs.

DEVS can be used as a foundation to devise more powerful cellular formalisms. An approach to formally merge CA and DEVS was presented in [15]. Currently, the more direct approach is Cell-DEVS [19], which is implemented in the CD++ toolkit [18]. This cellular formalism is closed under coupling. High-level structures allow the specification of discrete-event cellular models. Delay constructs are provided to deal with asynchronous time.

Lastly, another approach tries to deal with the Dynamic Structure DEVS (DSDEVS) formalism [2]: the Dynamic Structure Cellular Automata specification [11]. The idea is to take advantage of the Dynamic Structure Discrete Time System Specification (DSDTSS) sub-formalism to deal with discrete-time cellular models. The DSCA focuses on discrete-time and dynamic structure without dealing with timing issues of a discrete-event approach. The latter can be inefficient to deal with large discrete-time cellular models.

2.3 Modeling and simulation of discrete-event systems

A DEVS atomic model is described by a structure $< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$. X is the set of input events, S is the state set, Y is the set of output events, δ_{int} is the internal transition function, δ_{ext} is the external transition function, λ is the output function, and ta is the time advance function. The transition functions are triggered by events, and they operate on inputs and the state of the system when an event occurs.

A DEVS coupled model is described by a structure $<X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select>$. X is the set of input events, and Y is the set of output events. D is an index of components, and for each $i \in D$, M_i is a DEVS model (atomic or coupled). I_i is the set of influencees of model i. For each $j \in I_i$, Z_{ij} is the i to j translation function. *Select* is the tie-breaking function which sequentializes simultaneous internal transitions.

DEVS models can be executed by a plethora of simulators. Many of these simulators use different programming languages, each one with its own strengths and weaknesses.

As simulation mechanisms and specifications of DEVS models can be difficult to understand for non-specialists, visual languages are being developed. [13, 18, 8]). However, to the best of our knowledge, no visual languages have been proposed for the specification of DEVS cellular models.

3 CELLULAR MODELING FRAMEWORK

As described in the previous section, different discrete-event system specifications can be used at different abstraction levels (from abstract mathematical structures to generated code). At every level, choosing the right formalism allows one to focus on relevant features of a problem. If a modeling formalism is not well adapted to a problem specification, analysis, design and verification phases will not be efficient (or even incorrect).

To help modelers to automatically and efficiently switch between modeling (re-use, understanding, exchange) and simulation (run-time efficiency and accuracy) goals, meta-modeling and model transformation can be used.

Our main goal is the (automatic) generation of "Cellular Spaces" models from visual specifications. The basic idea is that the modeler provides a visual specification of the behavior of a cell (in a visual notation for DEVS,) and a "generator" specification which denotes a cellular space. The framework then produces the full cellular space, by replicating the given cell model. This process is carried out by means of model transformation. The main advantages are that: firstly, the transformation guarantees the robustness of the generated models (as opposed to manually coding the DEVS representation of the cellular space in some programming language), and secondly, the model transformations can by modified to yield different topologies without the need for hard-coding.

The environment consists of a tool to visually specify DEVS models as well as general cellular spaces, and to generate appropriate executable representations of these models. The modeler provides the specification for a single cell. Given this as well as the model representing the structure of the cell space, the process of generating a DEVS representation can be done in two ways: 1) via graph-transformation, and 2) via direct code-generation.

In the first approach, the tool generates a template of the cellular space, which is then filled with instances of the DEVS model specified for every individual cell. Once this is done, the

environment generates a code representation of the large cellular DEVS model, which can be processed by a DEVS simulator.

In the second approach, a single class is generated for the DEVS model specified for every individual cell, and a compact representation of the coupled DEVS model representing the cell space is also produced.

While the second approach is better both in execution time and space complexity, we point out that the first one is faster to implement and is purely visual, giving deep insight and requiring little programming knowledge as the whole process is done at the level of models rather than code. In particular, one of the main advantages of using explicit graph-transformations to generate the cell space is that is it straight-forward to change the topology of the space, simply by changing the set of rules, without requiring explicit coding of the topological constraints.

As an illustration, we first describe the process of generating 2D cellular spaces, and then the process of instantiating the sites in the cellular space with the DEVS model given by the modeler.

3.1 Specifying cellular spaces visually

First we start with the meta-model, which is illustrated in Figure 1 as an Entity-Relationship diagram. The main entities involved are the cells, their neighborhood relations, and the *generator* entity which is used to specify the global structure of the space. There are two other relations called *first* and *last*. These are used as pointers during the space generation.

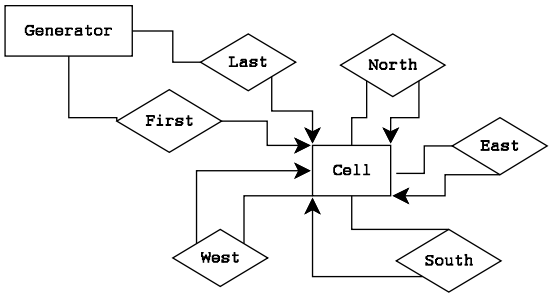


Figure 1. ER diagram of the cell space meta-model

The meta-model describes what constitutes a legal cell-space. Note however that it does not specify the actual topology, it only describes the possible connections. The topology itself is generated by a graph-grammar.

From this ER (meta-)model, AToM³ generates the appropriate modeling environment. Hereby, icons representing the various entities (Generator and Cell in this case) were modeled using AToM³'s visual icon editor (a drawing tool).

Cells act as "place holders." They are not interesting in their own right. Their sole purpose is to be replaced by instances of DEVS models and to provide the topology of their neighborhood. The neighborhood relations explicitly describe such topology. The role of the generator is dual: to specify the dimensions (*M* rows, *N* columns) and to serve as a guide in the generation processed by the graph-grammar. Figure 2 shows a typical model. The neighborhood arrows are labeled *n*, *e*, *s* or *w* to denote the particular spatial relation (North, East, South, and West, respectively, for a von Neumann neighborhood.) The generator entity is the node on top (with a diagonal through it.) It is labeled with the dimensions of the form "*m/M; n/N*". This means that it has *M* rows, *N* columns. The *m* and *n* are explained below.

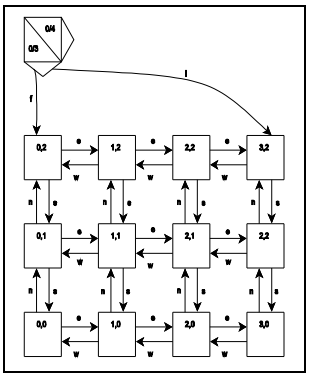
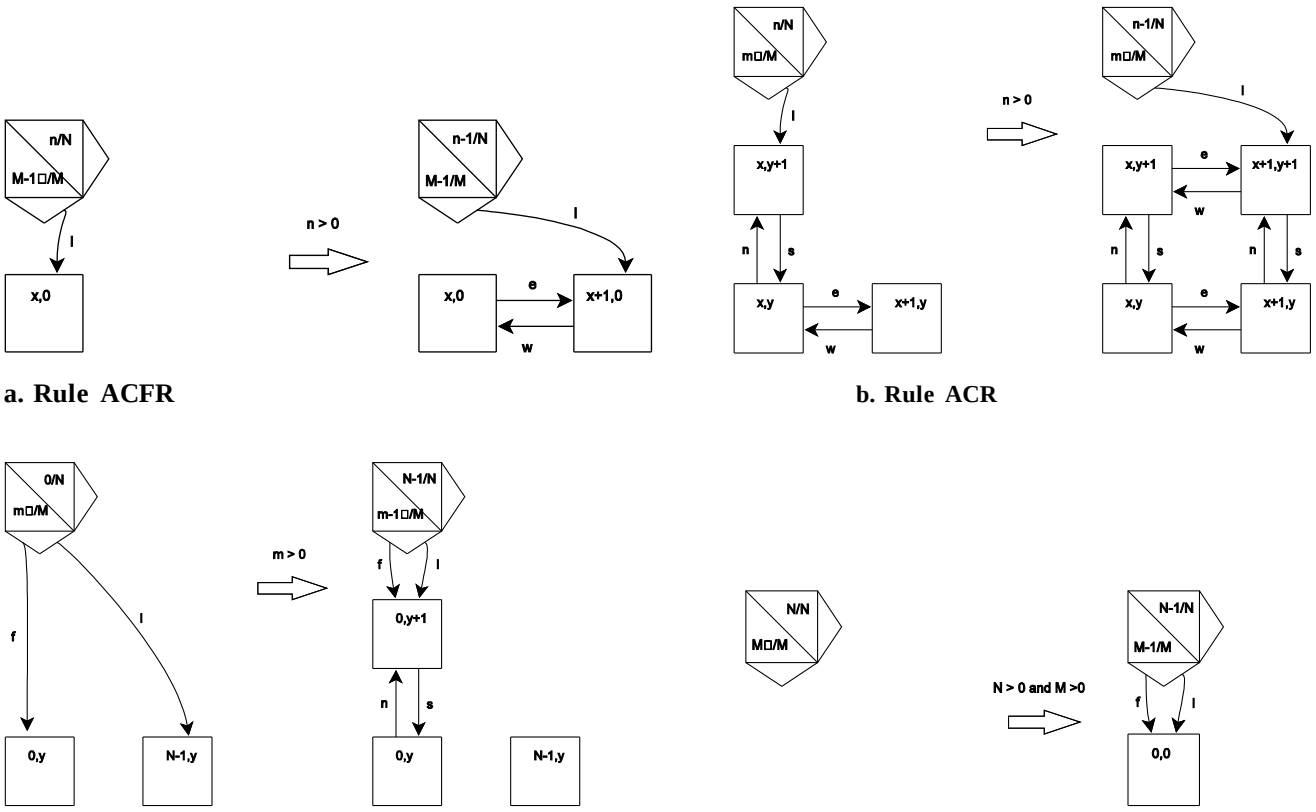


Figure 2. A typical cellular model

Now we specify a graph-grammar (*GenCellSpace*) which is applied to a generator entity and produces the template for the cellular space with the appropriate dimensions. In Figure 3 (a through d), we show the basic rules used to generate a two-dimensional cell-space where each cell has a von Neumann neighborhood, and with a finite boundary. Different topologies can be generated with a different set of rules.



a. Rule ACFR **b. Rule ACR**

c. Rule NR **d.Rule FC**

Figure 3. Basic rules used to generate a two-dimensional cell-space

The rules have the form *L => R* with an associated condition *C*, where *L* and *R* are valid models with respect to the meta-model of Figure 1.

The intuition behind this graph-grammar is to generate the cell-space by constructing one row at a time from the bottom-up, and each row is built cell-by-cell from left (West) to right (East).

As mentioned above, the generator node is labeled *"m/M; n/N"* where *M* and *N* are the dimensions. The *m* and *n* represent counters to keep track of how many rows are left to create (*m*) and how many cells are left to create in the current row (*n*).

In these rules, the arrow labeled *f* acts as a pointer to the first cell in the current column being generated, and the arrow labeled *l* is a pointer to the last cell generated in the current row.

The first rule is *FC* (First Cell). This rule is responsible for creating the first cell. It adds the cell and sets the *f* and *l* arrows to this new cell. Each new cell is given its own new label.

The second rule, *ACFR* (Add Cell First Row) takes care of creating the first row. It requires that *m=M-1* to ensure this is the first row, and that *n>0* to ensure this is not the last column. It adds a new cell to the right of the one pointed by *l* (the last added) and decrements the column counter by *1*.

The third rule is *NR* (New Row). It requires *m>0* so that there are more rows to add, and requires that *n=0* so that there are no more cells to add in the current row. It then creates a new cell adding the *n* and *s* arrows, and moves both *f* and *l* arrows to point to this new cell. It also sets *n* back to *N-1* and decrements *m* by *1* to start the next row.

The last rule is *ACR* (Add Cell in Row). It is responsible for the generation of inner-cells. It requires that *n>0* (as well as the given LHS structure be present) ensuring there are cells to add in this row. It then decrements *n* by *1* (leaving *m* unchanged.)

During graph rewriting cells are labeled with their absolute position *(x,y)* which could be used by other graph-grammars to process cells with

respect to their actual position. Strictly speaking *(x,y)* is not necessary for this generation process.

Other topologies can be described by modifying the rules provided.

3.2 Translating the cell space into a coupled DEVS

Once the cell space has been generated - once none of the rules in the *GenCellSpace* grammar are applicable- we apply a second graph-grammar (*Cells2DEVS*) which transforms the cells into instances of the given DEVS model describing cell behavior.

In Figure 4, we show only one representative rule (the other rules deal with borders.) The rule C2D shown simply rewrites a cell node with an instance of a DEVS model with the appropriate input and output ports (*tn* for 'to North', *fn* for 'from North', etc.)

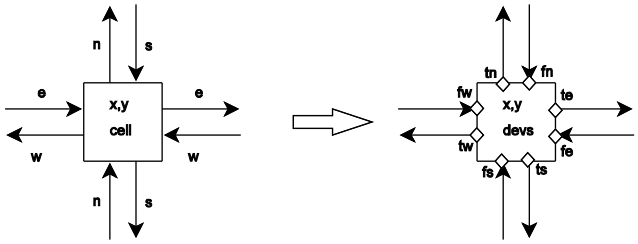


Figure 4. Rule to transform cells into DEVS models

There is a subtle issue here. This graph- grammar transforms a model from the "Cellular Spaces" formalism to one in the DEVS formalism. Thus, the initial model satisfies the meta- model of Section *3.1*, while the resulting model satisfies the DEVS meta- model. However, *while* the graph-grammar is executed, the intermediate steps will be graphs that are a combination of both formalisms, yet without being valid in either. This is not a problem as long as the initial and final models are valid, which this grammar guarantees. However, one has to be aware that the arrows in the "Cell Spaces" formalism only connect cells, and the arrows in the DEVS formalism are channel instances, which only connect ports, not cells. To get around this problem, cell arrows are rewritten in this rule to be "generic" arrows belonging to the GenericGraph formalism. Then we add a simple rule which rewrites these generic arrows to DEVS channels.

It is also possible to specify alternative neighborhood topologies in this graph- grammar.

3.3 Simulation code generation

Once the Cell2DEVS grammar has no more applicable rules, a third grammar (*CodeGen*) is applied. This grammar has been presented elsewhere (see [13, 8]). The main idea is that the main rewriting rule marks each visited DEVS model and has a side- effect action synthesizing the code from the structure provided in the DEVS instance. Another rule collects all connections (marking the ones already visited) in order to generate couplings in the coupled DEVS model.

4 GAME OF LIFE APPLICATION

To illustrate our approach (and our simple "Cellular Spaces" formalism) we developed a model of Conway's Game of Life [6]. In this model, each cell corresponds to an atomic DEVS implementing the rules of a cellular automaton. To show this model we take a conservative approach whereby all cells undergo state transitions simultaneously, coordinated by a central synchronizer. This is not ideal from a discrete- event simulation point of view, but we can guarantee that the behavior of the cellular DEVS model corresponds precisely to that of a classical cellular automaton of the Game of Life. We do not make any claims with respect to the simulation efficiency of this particular model, as our aim is to demonstrate the ease with which new formalisms can be developed and tools synthesized.

The user of the environment specifies the behavior of the cells and that of the synchronizer (visually) as DEVS models. A detailed description of these models is found below. From such models, code representing the final DEVS system is generated. The generated system consists of a (closed) coupled model which contains the synchronizer and another coupled model of the DEVS cellular model.

The behavior of the cell synchronizer and the individual cells is generated from the models depicted in Figures 5.a and Figures 5.b respectively. Both models have been graphically specified through the interface.

The whole structure of the cellular DEVS model has been graphically specified and generated in our (automatically generated) modeling environment, and the state- transition diagrams corresponding to the cells' and synchronizer's behavior.

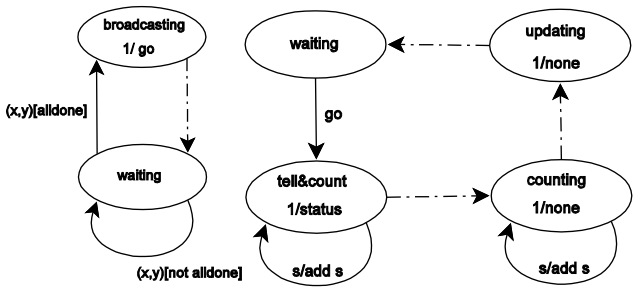


Figure 5. (a) Synchronizer and (b) Cell behavior

We describe the behavior informally below.

The central cell synchronizer is connected to every cell. It goes through a simple loop of broadcasting a message telling every cell to go ahead, and then waits for all cells to finish their turn. The loop continues forever. Each cell goes through a loop of four modes (or phases): *waiting*, *telling_and_counting*, *counting*, and *updating*. The state is a tuple of the form *(mode, status, neighbors_checked, neighbors_alive, remaining_time)* where:

- *mode* is one of the four phases mentioned above,
- *status* is *dead* or *alive*,
- *neighbors_checked* is the number of events embedding neighbors' states received in the current iteration,
- *neighbors_alive* is the number of neighbors who are alive currently,
- *remaining_time* is the time to the next scheduled internal transition.

The cell remains in the *waiting* mode until it receives an event from the global cell synchronizer to go ahead, thus guaranteeing every cell starts the iteration at the same time.

In the *telling_and_counting* mode it sends a message to all its neighbors informing them of its current status (*dead* or *alive*). If it receives a message from a neighbor, it updates the *neighbors_checked* and *neighbors_alive* parameters accordingly. It makes sure that the time of the next internal transition scheduled remains the same (by setting the new *remaining_time* to be the *current remaining_time – elapsed*)

Once in the *counting* mode, it continues to receive messages from its neighbors. When all of them have been checked it moves to the *updating* mode.

In the *updating* mode, it updates the state according to the current status and the *neighbors_alive*. External transitions are ignored.

After updating, it returns to the *waiting* mode, issuing a message to the central cell synchronizer, informing it that it has finished its turn.

5 CONCLUSION

We have proposed a framework for the visual modeling and simulation of "Cellular Spaces" whose semantics is given in terms of DEVS. Conway's game of life has been used to demonstrate the approach. Different software engineering concepts (meta-modeling, graph-transformations, discrete-event modeling and simulation, cellular models, etc.) have been combined to design a flexible and easy-to-use tool. Meta-modeling and graph-transformation allow to efficiently and quickly focus on the elements of interest for the specification of the cellular model. The use of meta-modeling and graph-transformation benefits mostly those building modeling environments, but also provides some benefits to the modelers by allowing the experimentation of different topologies without the need of hardcoding them. At present, the framework is still in its early stages, and it needs more capabilities to tackle complexity at the simulation and specification levels.

In our study, cells have uniform rules and neighborhoods. We are currently defining heterogeneous rules and neighborhoods. Also, dynamic structure representation of cellular systems using the DSDEVS formalism is a promising research area. More efficient discrete-event simulators should be explored soon and adapted to our framework: the adevs simulator ([12]), which is based on new efficient algorithms [23], and the DSCA simulator [24]. This will be easily achieved by modifying the code generator to produce C++ code for adevs instead of Python code for PyDEVS. This demonstrates the flexibility of our framework to deal with different modeling and simulation objectives.

Finally, more complex spatial models will have to be specified to allow dealing efficiently with complexity. We are working in parallel with economists of the Università di Corsica – Pasquale Paoli, to define a spatial economics model which will be able to account for large geographical data of land management.

REFERENCES

[1] C. Atkinson and T. Kuhne. *Rearchitecting the UML infrastructure*. ACM Transactions on Modeling and Computer Simulation (TOMACS). Volume 12, Issue 4. pp 290 - 321. October 2002.

[2] F. J. Barros. Modelling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modelling and Computer Simulation*, v. 7, 1997, p. 501- 515.

[3] J.S. Bolduc and Hans L. Vangheluwe. The modelling and simulation package PythonDEVS for classical hierarchical devs. MSDL technical report MSDL-TR-2001- 01, McGill University, June 2001.

[4] P. P. Chen. *The entity- relationship model - toward a unified view of data*. ACM Transactions on Database Systems, 1(1):9- - 36, 1976.

[5] H. Ehrig. Tutorial introduction to the algebraic approach of graph- grammars. In Graph- Grammars and Their Application to Computer Science. v. 291. 3-14. LNCS. Springer. 1986.

[6] M. Gardner. The fantastic combinations of John Conway' s new solitaire game `Life'. *Scientific American*, 223(4):120- 123, October 1970.

[7] J. de Lara and H. Vangheluwe. *AToM3: A tool for multi- formalism and meta- modelling*. In European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science 2306, pages 174 - 188. Springer- Verlag, April 2002. Grenoble, France.

[8] A. Levytskyy, E. J.H. Kerckhoffs, E. Posse, and H. Vangheluwe. *Creating DEVS components with the meta- modelling tool AToM3*. In Alexander Verbraeck and Vlatka Hlupic, editors, 15th European Simulation Symposium (ESS), pages 97 - 103. Society for Modeling and Simulation International (SCS), October 2003. Delft, The Netherlands.

[9] A. Muzy, E. Innocenti, J. F. Santucci, D. R. C. Hill. *Optimization of Cell Spaces Simulation for the Modeling of Fire Spreading*. Annual Simulation Symposium 2003: 289- 296

[10] A., Muzy, A. Aïello, P.- A. Santoni, B. P. Zeigler, J. J. Nutaro, and R. Jammalamadaka. Discrete event simulation of large- scale spatial continuous systems. *International Conference on Systems, Man and Cybernetics (SMC), IEEE*, 2005.

[11] A. Muzy, E. Innocenti, A. Aiello, J. F. Santucci, P- A Santoni, D. R. C. Hill: Dynamic Structure Cellular Automata in a Fire Spreading Application. ICINCO (3) 2004: 143- 151

[12] J. Nutaro. aDEVS-0.2. C++ library for parallel DEVS. University of Arizona, Tucson, 1999. http://www.ec.arizona.edu/~nutaro.

[13] E. Posse and J.S. Bolduc. *Generation of DEVS modelling and simulation environments*. In A. Bruzzone and Mhamed Itmi, editors, Summer Computer Simulation Conference. Student Workshop, pages S139 - S146. Society for Computer Simulation International (SCS), July 2003. Montréal, Canada.

[14] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual* (2nd Edition). Addison- Wesley Object Technology Series, 2004.

[15] H. Vangheluwe and G. C. Vansteenkiste. *The cellular automata formalism and its relationship to DEVS*. In Rik Van Landeghem, editor, 14th European Simulation Multi- conference (ESM), pages 800- 810. Society for Computer Simulation International (SCS), May 2000. Ghent, Belgium.

[16] D. Varro and A. Pataricza. *VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics)*. Software and Systems Modeling. Volume 2, No. 3 pp. 187 - 210.

[17] J. von Neumann. Theory of Self- reproducing Automata. University of Illinois Press, 1966. Edited and completed by Arthur W.Burks.

[18] G. A. Wainer, G. Christen, A. Dobniewski. *Defining DEVS models with the CD++ tool*. In Proceedings of European Simulation Symposium. Marseilles, France. 2001

[19] Wainer, G., and N. Giambiasi, "Application of the cell- DEVS paradigm for cell spaces modeling and sim- ulation". Simulation, v. 76, 2001, p. 22- 39.

[20] L.v. Bertalanffy, *General system theory. Foundations, development, applications.*, George Braziller. New York, 1968.

[21] B. P. Zeigler. Multifaceted Modeling and Discrete- Event Simulation. Academic Press. London. 1984.

[22] B. P. Zeigler, Tag Gon Kim, Herbert Praehofer. Theory of Modeling and Simulation. Second Edition. Academic Press Inc. 2000.

[23] A., Muzy, and J. J. Nutaro, "Algorithms for efficient implementation of the DEVS & DSDEVS abstract simulators". *1st Open International Conference on Modeling and Simulation (OICMS)*, 2005, p. 273- 279.

[24] A., Muzy, E. Innocenti, F. Barros, A. Aïello, and J. F. Santucci. Efficient simulation of large- scale dynamic structure cell spaces. *Summer Computer Simulation Conference*, 2003, p. 378- 383.

[25] Hu X, Zeigler BP. 2004. *A high performance simulation engine for large- scale cellular DEVS models*. Presented at High Performance Computing Symposium (HPC'04)

[26] Nutaro, J., B. P. Zeigler, R. Jammalamadaka, and S. Akrekar, "Discrete event solution of gaz dynamics winthin the DEVS framework: exploiting spatiotemporal heterogeneity". *International Conference for Computational Science*, 2003.

[27] Wainer, G., "Modeling and simulation of complex systems with Cell- DEVS". *Winter Simulation Conference*, 2004, p. Tutorial.