

From Reactive Multi-Agents Models to Cellular Automata

Antoine Spicher

Nazim Fatès

Olivier Simonin

INRIA Nancy Grand Est - LORIA

Campus Scientifique - BP 239

54506 Vandœuvre-lès-Nancy Cedex, France

{antoine.spicher,nazim.fates,olivier.simonin}@loria.fr

Cellular automata (CA) are a well-known model of massively parallel computing devices; they form a good basis for studying spatially distributed computing. The main advantage of this formalism is its simplicity: it involves static homogeneous computing units that are regularly arranged in space. The drawback of expressing a model with cellular automata appears when one needs to build models with movements of pseudo-independent entities. By contrast, multi-agents systems (MAS) appear more appropriate for modeling systems where autonomous entities, the *agents*, may move in a virtual space, the *environment*, and act on it. Simulating multi-agent systems on CA-like spatially extended computing devices thus requires appropriate tools to switch from one type of description to the other. The purpose of our research is to find a method to translate "the language of multi-agents" into "the language of cellular automata".

At first sight, the two formalisms look very similar and are often confused. One may find several works where the names "cellular automata" and "multi-agent systems" are used without distinction. This is easily understandable since CA are often used to model the environment of a MAS, and, reciprocally, one may see a CA as a particular kind of MAS where agents do not move. Let us first try to clarify the differences between CA and MAS. We compare them at two levels: (1) the modeling level: *What in a model makes CA or MAS more suitable to express it?* (2) the simulation level: *How intuitive is the implementation of CA and MAS?*

MAS and CA, as modeling tools. In their definition, CA are uniform objects: there is a *unique* neighbourhood shape for each cell and a *unique* transition function. As a consequence, CA are fitted to model phenomena that involve *homogeneous* spaces; CA have been used for example to model physical systems, biological systems, spatially embedded computations, etc. Note that it is always possible to take into account inhomogeneities, for example by encoding the heterogeneity in the cell states, but this is generally not straightforward to do.

MAS are preferred for expressing an heterogeneous population of entities. They necessitate to make a distinction between the agents' behaviors and the *environment* where they are embedded [2]. This distinction allows us to focus on the specification of particular and localized events, namely the agents' *actions*. They offer a methodology for designing systems, at the level of algorithms, programming languages, hardware, etc. Examples of MAS applications range from the simulation of natural systems, from ants to human behaviors, to the design of massively distributed software and algorithms like web-services, peer-to-peer technologies, etc. Nevertheless, we must note that, contrarily to CA, no universal definition of MAS has been accepted so far. From the modeling point of view, translating MAS in the cellular automata formalism has (at least) the advantage of fixing the mathematical expression of the model and removing ambiguities of formulation. In this article, we focus on *reactive* MAS, i.e., MAS systems where the agents have no representation of their environment, and no or very limited capacities of reasoning.

MAS and CA, as simulation tools. The key characteristic of complex systems is the difficulty, if not the impossibility, of inferring their global behavior from the local specification of the interactions. Few mathematical tools are available to predict the evolution of complex systems in general, more especially those which involve self-organisation. This gives to simulation a central role to find the mechanisms that can explain how a complex phenomenon results from multiple simple local interactions. We thus have to pay attention to the quality of simulations and to detect ambiguities that may be hidden in the way they are implemented. As far as spatial computing is concerned, this problem is central since it is an emerging field of research, that is mainly explored by means of simulation.

The agent-based programming style is somehow intuitive and natural as the programmer takes the point of view of the agent. There is a form of anthropomorphism that makes MAS programming particularly attractive. Never-

theless, we emphasize that once all the agents behaviors are individually specified, there are still many ways to make the agents interact and play together in the environment. The implementation of such systems raises many questions, like assessing the importance of the synchronicity in simulations: are the agents updated all together or one after the other? The design spatially-extended computing devices will require to imagine a new type of computer science, where the computations do not *necessarily* rely on the existence of a synchronisation between the components.

By contrast with MAS, CA lead to shift the programmer's point of view from the "eyes" of the agents to their environment. The benefits of this shifting effort are twofold. (1) The CA formalism forces the programmer to solve conflicts between concurrent agents actions at the elementary level of the cell and forbids the use of any global procedure. (2) As a consequence, the implementation on massively distributed devices is easy. Indeed, CA provide the programmer a cell-centered programming style where the set of cells represents computing units that are regularly organized. Recent works have shown that it is possible to have a good efficiency by using parallel architecture to run CA simulations for FPGA and for GPU. In other words, *CA provide an easy-to-implement framework, but expressing the local rule necessitates a method to "blend" the different components of a complex system.*

1. CA Expression of a MAS Model

The Synchrony Paradox. In MAS descriptions, emphasis is put on the agents local behaviors. Classically, to avoid collisions between simultaneous actions of the agents, the agents are updated one after the other using a scheduler. However, two objections can be raised:

1. The implementation of this sequential updating on a massively distributed computing device is not impossible, but it requires the introduction of complex procedures to distribute the scheduler. If we are to consider spatial computing on simple entities, for example to binary CA, this distribution of a scheduler may have a prohibitive cost.
2. The use of a scheduler introduces an external form of causality that is generally not specified in the original formulation of the model. For example, looking at the DLA model (see below) ; we observe that a correlation between the order of updating and the initial position of the particle induces a bias in the outcome of simulations (see figure1).

By contrast, the framework of CA demands an early resolution of the conflicts created by simultaneous moves to a given cell. To achieve that, we propose to establish a dialog between cells.

Transactional CA. A particle move requires a *source* cell (that contains a particle at time t) and a *target* cell (that may contain the particle at time $t + 1$). We propose to elaborate a three-step *transactional* process where cells negotiate their requirements:

1. *Request:* *source* cells express their needs to their neighbors.
2. *Approval-rejection:* *target* cells accept or not their neighbors requirements; this decision is done with respect to an exclusion principle policy (for example, an empty cell is an available target iff there is exactly one particle requesting to move to this cell).
3. *Transaction:* *sources* and *targets* separately update their state.

2. An Illustrative Example: the DLA

To illustrate our solution, we focus on the coding of a *diffusion-limited aggregation* (DLA) system within a CA.

The DLA model was introduced to study physical processes where diffusing particles following a Brownian motion aggregate: for instance, zinc ions aggregate onto electrodes in an electrolytic solution. This process leads to interesting self-organized dendritic fractal structures. Different models of DLA have been proposed; we consider here the model where particles stick together forever and where there is no aggregate formation between two mobile particles.

The *environment* is a 2D finite and toric square grid composed of elements called *patches*. The population of agents is composed of the particles. Each particle is localized on a cell of the environment and is characterized by a state: a particle is either Fixed or Mobile. The *exclusion principle* holds: *i.e.*, there cannot be more than one agent on each patch of the grid.

The initial configuration of the system is composed of a population of Mobile particles and some Fixed particles called the *seeds*. The expected behavior is the aggregation of the Mobile particles to build dendrites from the seeds.

We propose to formulate the agent dynamics using the usual *perception-decision-action* cycle. We first describe the perception and action abilities of an agent. The *perception* consists of two functions:

- Γ_1 returns **true** if the agent perceives a Fixed *neighbor* particle, and **false** otherwise;
- Γ_2 computes the set of *directions* that lead to empty *neighbor* patches.

The neighborhood referred in these perceptions corresponds to the four closest positions of ρ_a following North, South, East and West directions. The *actions* are Diffuse(d) (move following direction d), Aggregate (change to the Fixed state), and Stay (do nothing).

Let $\mathcal{U}(S)$ denote the operation of selecting one element in a finite set S with uniform probability, the *decision process* returns an action as a function of the agent perceptions:

$$\begin{array}{lll} \text{if } & \Gamma_1 & \text{then Aggregate} \\ \text{else if } & \Gamma_2 \neq \emptyset & \text{then Diffuse}(\mathcal{U}(\Gamma_2)) \\ \text{else} & & \text{Stay} \end{array} \quad (1)$$

DLA Transactional Model. Figure 2 shows with a graph the local transition function of a transactional CA capturing the DLA model. On this graph, nodes represent the different states of the CA and the arrows specify transitions between states. The states E_0 , F_0 and M_0 are given twice to clarify the figure. States are distributed *vertically*, to segregate the behaviors of sources and targets, and *horizontally*, to distinguish the three steps of the transactional CA.

At the beginning, the cells are either empty (E_0) or contain a particle which is either fixed (F_0) or mobile (M_0).

- The request transition consists in deciding an action for each M_0 cell: depending on the perceptions, a mobile particle either aggregates (state F_1A), or diffuses following a direction d (state $M_1D(d)$), or stays at the same position (state M_1S).
- During the approval step, E_1 cells decide, by reading their neighbors requirements, if they remain empty (state E_2) or become receptors of a particle moving from a direction d' (state $R_2(d')$).
- Finally, the transaction is computed: receptors become particles, moving particles with a receptor target become empty, and aggregating particles become fixed. Other cells remain in their initial state. Note that a particle move is allowed when the direction of the moving particle is opposite to the direction of reception, which we denote by $d = -d'$, for example $d = \text{North}$ and $d' = \text{South}$.

Simulations. Figure 1 presents simulations of the previously described DLA model. Simulations were obtained using a classical sequential framework based on a scheduler, on the first and second lines, and using our synchronous transactional CA, on the third line. The same initial configuration, given on the left column, was used on both simulation platforms. It consists of a 100x100 grid where seeds are localized on the boundaries and where mobile particles are gathered in a 40x40 central square. Qualitatively, we observed that the two systems exhibited the same behavior,

see the right column of figure 1. However, further studies on the dendrites' distribution would be needed to assess the differences between the two approaches. To compare the time scales of the two systems, we define a *simulation time step* as: (a) the three sub-steps of the transactional CA and (b) the update of all the agents in the sequential framework. Using this scale, we observed that the initial square slowly disappears in the synchronous CA approach and quickly disappears in the sequential approach (see the middle column of figure 1). This phenomenon can be explained by the fact that an asynchronous update allows a particle to move to a just evacuated patch *during a simulation time step*, while the synchronous update forbids this behavior. At this stage, more investigations are needed to determine the differences between the two simulations.

3. Towards the Construction of an Automatic Translation Tool

Our goal is to study the expression of other MAS models in the CA formalism. We aim at designing an automatic translation tool that would allow users to specify their multi-agent system and to automatically generate the transactional CA associated with the multi-agent. This of course is impossible for all multi-agent systems and requires to limit the scope of the tool. Our current investigation concerns the definitions of these limits and the construction of the translation tool in the FiatLux CA simulator.

More about this work. This extended abstract sums up a more developed paper: *Simulating a Reactive Multi-Agent Model within a Cellular Automaton*, see www.loria.fr/~fates/Doc/SpicherFatesSimonin-SASO.pdf

References

- [1] Nazim Fatès, *Fiatlux CA simulator in Java*, See <http://nazim.fates.free.fr> for downloading.
- [2] J. Ferber, *Multi-agent systems, an introduction to distributed artificial intelligence*, Addison-Wesley, 1999.
- [3] F Michel, G. Beurier, A. Gouaïch, and J. Ferber, *The turtlekit platform : application to multi-level emergence*, ABS 4 Agent-Based Simulation 4, 2003.

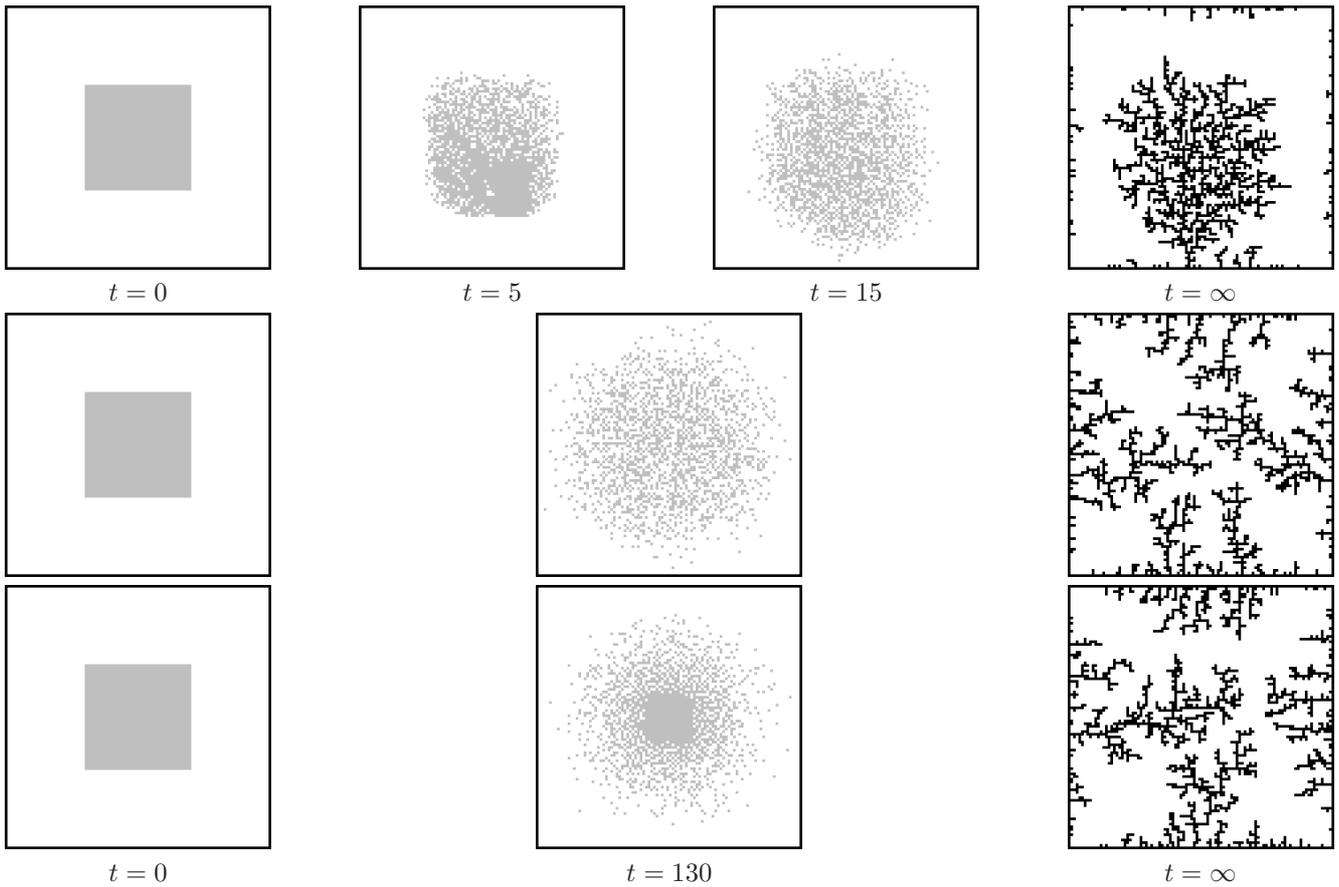


Figure 1. DLA Simulations: the fixed particles are represented in black the mobile particles in grey. In the first line, the scheduler always updates agents in the same order: it corresponds to ordering agents according to their initial positions from bottom-left to up-right. We observe that dendrites mainly grow at the bottom. The second line was obtained using the sequential simulation tool TurtleKit [3]. The third line was obtained using the CA simulation tool FiatLux [1].

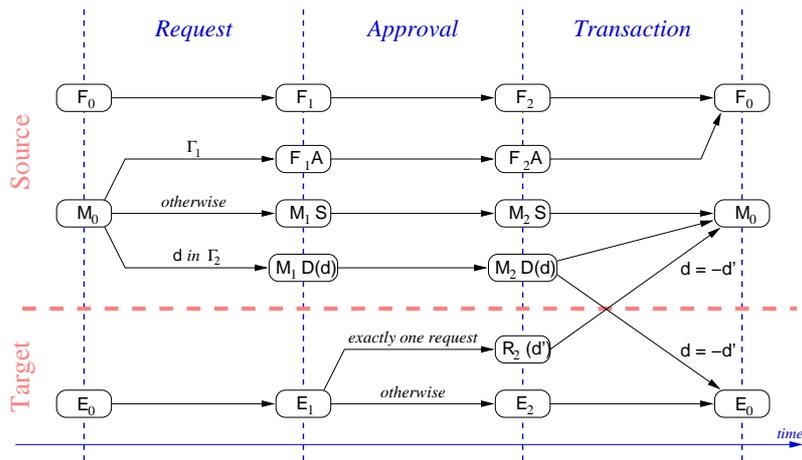


Figure 2. DLA local evolution rule within a transactional CA.