Environment updating and agent scheduling policies in agent-based simulators

Philippe MATHIEU[1] and Yann SECQ[1]
[1] *Université Lille 1, LIFL (UMR CNRS 8022), FRANCE*
{*first_name.last_name*}*@univ-lille1.fr*

Keywords: agent-based simulation, agent scheduling, simultaneity, simulation bias, parameter sensibility

Abstract: Since Schelling's segregation model, the ability to represent individual behaviours and to execute them to produce emergent collective behaviour has enabled interesting studies in diverse domains, like artificial financial markets, crowd simulation or biological simulations. Nevertheless, the description of such experiments are focused on the agents behaviours, and seldom clarify the exact process used to execute the simulation. In other words, little details are known on the assumptions, the choices and the design that have been done on the simulator on fundamental notions like time, simultaneity, agent scheduling or sequential/parallel execution. Though, these choices are crucial because they impact simulation results. This paper is focused on parameter sensitivity of agent-based simulators implementations, specifically on environment updating and agent scheduling policies. We highlight concepts that simulator designers have to define and presents several possible implementations and their impact.

1 INTRODUCTION

When building a simulation infrastructure, several criteria have to be taken into account by the simulator designer. (Arunachalam et al., 2008) defines criteria to compare some existing agent-based simulators: *design* (environment complexity, environment distribution, agent/environment coupling), *model execution* (quality and nature of visualisation and ease of dynamical properties evolution at runtime), *model specification* (expected level of programming skills and effort needed to create a given toy example) and *documentation* (quality and effectiveness). These criteria are mainly concerned by the user point of view, stressing the flexibility and richness of models that can be expressed, but also involve some aspects linked to the implementation of the proposed multi-agent models (specification and execution).

In this paper, we want to stress the importance of design and implementation choices that are made by simulator designers. To do so, we define a set of problematics that have to be answered by simulator designers. These problematics can be crucial or irrelevant depending on the application domain. If these problematics are not properly taken into account, it can lead to biased results. It is thus critical to clearly state these design and implementation choices, in order to ease reproducibility and trust in produced results. We stress the fact that this paper is focused on simulators parameter sensitivity and not on domain parameter sensitivity.

In section 2, we define criteria in order to provide a structured set of questions to characterize agent-based simulators models and implementations. We believe that simulator designers should take into account these questions before making their implementation choices. Section 3 characterizes impacts of agents scheduling and environment updating policies on three application domains.

2 AGENT-BASED SIMULATORS PROBLEMATICS

When building an agent-based simulator, there is a large set of problematics that have to be taken into account: time and duration notion, simultaneity handling, agents equity, spatial/non spatial environment, determinism and reproducibility and scalability issues. Even if not all these aspects are pertinent for each application domains, simulator designers should have them in mind before making implementation choices that can deeply impact their conceptual and technical models. To provide some guidelines of the choices that can be made for a given simulation

model, we believe that designers should at least answer precisely these questions:

- Does your model need simultaneity ?
- Do you have a spatial environment or not ?
- Do you need to guarantee equity in talk, in information or in resources ?
- Does your agents need to perceive a lot of information in order to decide ?
- Does your agents have small or high resource usage (memory and/or computation) ?
- Do you have a "human in the loop" ?
- Do you have an open or closed simulator ?

Providing a clear diagram detailing the relation between these problematics could be interesting, but as they are interleaved and because they are also impacted by the application domain, we believe that it is not possible to define a clear taxonomy. Thus, following sections will only focus on policies that can be used in order to update the environment and to schedule agents.

To illustrate the importance of environment updating scheme and agent scheduling mechanism, we study their impact on three well known models (Table 1): prey-predator, Game Of Life and artificial stock market. In these models, the system is composed of active entities that evolve to compute some function. In prey-predator, only one cell is evaluated within a time-step, in cellular automata all cells evolve simultaneously and in an artificial stock market traders act concurrently. If these properties are changed, then these models do not exhibit the same behaviour and loose their emergent properties. We make the assumption that *everything is an agent* (Kubera et al., 2010) but adding an explicit environment update as a last stage in the simulation loop could enable an approach mixing agents and objects.

3 ENVIRONMENT UPDATING AND AGENT SCHEDULING

This section purpose is to define the main aspects that an agent-based simulator designer should have in mind before developing its own tool (Macal and North, 2007). We first introduce the main schema of agent-based simulators before diving into environment updating policies and agent scheduling complexities.

3.1 A classical agent-based simulation loop

- first, environment and agents have to be defined,
- then, during the simulation, the simulator has to:
  - **mix agents** to determine in which order agents will be queried to retrieve their chosen action. The agents list is often shuffled but it can also be sorted to give priorities to specific agent classes,
  - **choose agents** to decide which agents will be able to act at this time-step (useful if some agents have to wait or to enable action durations spanning on more than one time-step),
  - **query agents** to retrieve their action,
  - **execute actions and update environment**, some conflicts can appear, in such case the designer has to define a tie-break rule,
  - **update agent population**, ie. removing or creating agents if necessary,
  - finally **update probes** so observations can be made on the environment or agents in order to analyze some representative criteria.

These basic steps enable the definition of a large range evaluation scheme: with evolving population or not, with random or specific agent querying ordering, with synchronous or asynchronous environment update.

3.2 Environment updating policies and agent perception

Agents act on the environment, thus environment updating scheme linked with agent scheduling policy, are important in order to guarantee that agents have equal access to the same environment state (equity in information). The two main environment handling schemes are synchronous and asynchronous updating. On one hand, with synchronous updating scheme, all modifications are done simultaneously. This mechanism is generally implemented by relying on the availability of another environment representing the next generation. This approach enables agents to access the same information (by looking at the current environment) and allows to switch between current and next environment in order to give the illusion of simultaneous update of the whole environment. On the other hand, with asynchronous update scheme, there is only one environment and modifications are done directly. This means that during a time-step, elements from different time-steps are mixed, which can break the equity in information. Indeed, the first agent only handle information from the current time-step

Table 1: Three well-known agent-based models: Prey-predator, Game of Life and Artificial Stock Market

	Prey-predator	**Game of Life**	**Stock Market**
Environment	2D Grid or continuous	2D Grid	Non spatial
Agents	Grass / Sheep / Wolf	Cell	Trader
Human in the loop	No	No	Yes
Simultaneity	None	Yes	None
Agent per timestep	1	all cells	N
Emergence	Lotka-Voltera graph of population evolution	stable, growing or cyclic patterns	stylised facts
Equity in talk	Yes or No	Yes	Yes
Equity in information	No	Yes	Yes/No
Equity in resources	No	Yes	No

while the last one to act can perceive modifications that have been done by previous agents.

Environment updating is also deeply linked with agents perception handling. The two main approaches are pushing or pulling information. In the pushing scheme, information is automatically given by the environment to the agent, while in the pulling scheme, it is the agent that initiate the request to access some information within the environment. It is important to distinguish these two schemes because if a simulator rely on a pushing scheme, it can ensure information equity by pushing information to agents at the beginning of the time-step and after query agents for their action. If a pulling mechanism is used, agents can request information to the environment while modifications have already been done during the current time-step. Thus, it becomes harder to maintain information equity in a pulling scheme with an asynchronous environment updating mecanism.

Table 1 presents the differences of the three models concerning these problematics of environment updating and notification scheme. In prey-predator, there are several agent families with different properties and behaviours: the grass just grows, sheeps and wolves can eat, move and reproduce themselves. In the Game of Life (or GoL), there is only one kind of agent, a cell with only two states: dead or alive. In artificial stock markets, there are several traders families which define different trading strategies. There is no simultaneity in prey-predator: a time-step is reduced to one agent action. This is in fact to prevent a sheep and a wolf to act simultaneously and to have a wolf trying to eat a sheep but being unable to catch it. On the opposite, in GoL, cells evolution rely on the crowding in their close neighbourhood. Thus, each cell compute how many alive cells surround her and all cells switch their state simultaneously. If simultaneity is not properly handled, emergent patterns do not appear. Finally, in artificial stock markets, equity in talk is enforced but there is no simultaneity in the model: as soon as an order can match another, a new price is fixed. Nevertheless, this do not mean that notification is done immediately. To enforce information equity, notification occurs only at the end of a time-step, or if agents use a pulling scheme and price history is kept, it is possible to ensure that the agent access the previous fixed price and not the one produced during this time-step.

3.3 Agent scheduling policies

The Table 2 express two dimensions useful to describe precisely agent scheduling policies: agent logical scheduling and underlying sequential or parallel execution. We believe that simulator designer should describe precisely how they implement their simulation engine in order for the simulation designer to know exactly which bias could be observed. An experimentation to reproduce the Sugarscape model (Epstein and Axtell, 1996) with the MASON simulator insists also on the importance of action scheduling (Bigbee et al., 2005).
Indeed, one aspect is to know how agents will be able to act at each time-step, in sequential or in parallel, and how the simulation engine will indeed give the computational resources to agents. To understand the differences and implications of these choices, we have to detail the different cases.

3.4 Purely sequential and fair

Controlled scheduling with one process is concerned with a round-robin agent scheduling to retrieve their actions and a sequential execution of these actions. This approach is often used in simulators as it is easy to understand and implement. Nevertheless, the simulator designer has to make a choice between two action execution schemes: a direct execution of agents actions or a deferred one. The first scheme allows agents that are last in the round to take into account

Table 2: Agent logical scheduling and physical execution context

	Controlled scheduling *Explicit agent handling by the simulation engine*	**Undefined scheduling** *Delegated to the language or operating system*
Only 1 process all sequential	Simulated simultaneity possible (influence/reaction) Fair access to info	no simultaneity
N physical processes real execution simultaneity	increase execution time for agents	real time reasoning

what others agents have done (and thus break information equity), while the second scheme ensure that the information available to all agents is the same.
This approach is fair as each agent has the opportunity to act at each time-step. It can lead to some distortion in equity if the simulator designer do not distinguish action gathering from action execution. The figure 1 illustrates a classic implementation. The main restriction of this approach is a loss of performance while running on parallel hardware because agents reasoning and the main simulation loop are purely sequential.

Figure 1: A purely sequential and fair classical implementation

```
Environment env = createAndInit();
List<Agent> agents = createAndInit();
List<Action> actions = init();
while !simulationFinished() {
  List<Agent> activeAgents =
    choose(mix(agents));
  actions.clear();
  for (Agent agent : activeAgents) {
    actions.add(agent.act());
  }
  env.apply(actions);
  updatePopulation(agents);
  updateProbes(environment, agents);
}
```

Prey-predator: to handle correctly simultaneity in this model, the time-step has to be reduced to only one agent selection. Equity in talk can be guaranteed through the agent selection policy (ie. *choose* primitive). *Game of Life*: information equity has to be enforced in order to provide simultaneity between all cells. It can be done by separating the perception stage from the cell state evolution. One approach consists in gathering all cells actions before applying them, or through a costly environment duplication (one handling the current time-step for perception and the other one for the next time-step to store new cells states). *Stock Market*: if information equity is required, an approach similar to the GoL has to be implemented. Otherwise, the only important aspect is to shuffle agents at each time-step, unless the designer want to enforce a priority in talk, for example to simulate the fact that some traders are inside the market, while others are remote.

3.5 Purely sequential and unfair

With undefined scheduling and one process, all actions are evaluated sequentially but there is no guarantee that each agent will talk the same number of time during a simulation. In fact, the timestep is reduced to querying and executing only one agent chosen randomly (or with a specific strategy that can take into account talk equity between agents). In this approach, simultaneity is impossible because only one agent act within a timestep. The figure illustrate a classic implementation.

Figure 2: A purely sequential and unfair classical implementation

```
Environment env = createAndInit();
List<Agent> agents = createAndInit();
while !simulationFinished() {
  Agent current = choose(agents);
  env.apply(current.act());
  updatePopulation(agents);
  updateProbes(environment, agents);
}
```

Prey-predator: this approach is particularly suited for the prey-predator model. It is simpler to implement that the first cell because no simultaneity is required in this specific simulation. *Game of Life*: this approach is not suitable because of the mandatory simultaneity of the model. If this scheme is used, cells do not evolve simultaneously and thus cells from different time-steps are mixed, leading to a mismatch with rules model. With this constraints, classical patterns of the Game of Life cannot be reproduced. *Stock Market*: if equity in talk is not needed, like for prey-predator, this approach is easier to implement. But if equity in talk is required, it implies that the de-

signer has to implement a policy enforcing that agent selection for each time-step check that no agent can be more than one time-step in future than the others.

3.6 Parallel and fair

Controlled scheduling with multiple processes ensures fairness by constraining agents actions through an explicit synchronisation at each time-step. But, as several (physical) processes are available, performances are generally increased because computations implied by agents reasoning can now be executed in parallel. In this implementation, the main simulation loop is restricted to a simple synchronisation barrier that waits after all agents actions. The choice made on straight or deferred action execution implies the same consequences as in the case of controlled scheduling with one process.

With the advent of multi-core architecture, this approach can leverage the raw computing power available in current hardware infrastructure. Nevertheless, creation and switching process costs have to be measured and balanced with behaviours evaluation costs in order to really obtain interesting speedup. In case of multi-core CPU, this approach do not imply important code refactoring, but if the execution target is a GPU, the translation is not easy. If special care is taken on agent's reasoning to simplify it as a finite state automata, this approach can scale on GPU infrastructure as demonstrated by the FLAME-GPU framework (Richmond Paul, 2009).

The figure 3 illustrates a classical implementation where each agent has its own thread and where a synchronisation barrier is used to guarantee equity in talk. Thus, in each loop, all agents are waken and have to proactively store their action in a shared resource.

Figure 3: A parallel and fair classical implementation

```
Environment env = createAndInit();
List<Agent> agents = createAndInit();
List<Action> actions = init();
// Launching all agents threads
for (Agent agent : current){
  new Thread(agent).start();
}
while !simulationFinished() {
  List<Agent> activeAgents= mix(agents);
  List<Agent> current =
    choose(activeAgents);
  waitAllAgentsActions(activeAgents);
  env.apply(actions);
  updatePopulation(agents);
  updateProbes(environment, agents);
}
```

Prey-predator: as this model do not need simultaneity, special care should be taken to avoid that two agents of the same neighbourhood act in parallel. It could lead to some artefacts like a wolf trying to eat a sheep that is no more present at execution because it has simultaneously moved. This problem can be easily solved by decoupling action gathering from action execution and by giving priorities to wolves over sheep. *Game of Life*: as with prey-predator, it is necessary to defer action execution otherwise cells from different time-step are mixed. Speedup should not be so interesting in this specific model because cells computation are not costly. Unless a specific implementation under a GPU with an environment completely embedded within GPU memory (Perumalla and Aaby, 2008) is used, sequential versions should be faster than a parallel one. *Stock Market*: in contrary to prey-predator, as no simultaneity is possible in this model, there are no issue if two agents act simultaneously as there will always have an order that arrive before another within an order book. But the fact that traders can run in parallel imply that some gain could be observed for costly trading behaviours. Nevertheless, processes synchronisation costs reduce the gain that could be obtain with the last approach.

3.7 Parallel and unfair

Finally, uncontrolled scheduling and multiple processes can be seen as a special kind of individual-based simulators where focus is put on real-time simulation. As no scheduling is done on agents and actions can occur simultaneously, this approach is adapted to real-time interactive simulations. Illustrations of this special kind of simulations are mainly related to Massively Multi-player Online Role Playing Game (MMORPG) or serious games (pedagogical games). This level of parallelism enable scaling in agents number and in response time. It should be noted that in such settings, questions of reproducibility or fairness are no more pertinent. This context should mainly be used to enable *human-in-the-loop* simulations in a real-time setting, which is the case in games and serious games. Equity in talk has another meaning here, and virtual agents should be slow down in order to enable humans to react in the same timing as virtual agents.

Prey-predator: in this setting, the only problem that can occur is simultaneous modification of adjacent agents. It could be solved by some locking mechanism to ensure that simultaneity cannot occur in these situations. *Game of Life*: again, in this context, the model cannot be guaranteed unless strong synchronisation and deferred action execution is enforced. Such

Figure 4: A parallel and unfair classical implementation

```
Environment env = createAndInit();
List<Agent> agents = createAndInit();
List<Action> actions = init();
// Launching all agents threads
for (Agent agent : agents){
  new Thread(agent).start();
}
// Main simulation loop
while !simulationFinished() {
  List<Agent> active = choose(agents);
  env.apply(next(active,actions));
  updatePopulation(agents);
  updateProbes(environment, agents);
}
```

move would reduce any gains that could be obtain from parallel action execution. *Stock Market*: this approach is clearly fitted to artificial stock market simulation, particularly in serious games context where human agents interact with virtual agents. The main issue is then to slow down virtual agents so humans can react in the same timing.

4 CONCLUSION

Agent-based simulator designers should take into account multiple notions: time and simultaneity, the multiple notions of equity (in talk/information/resources) between agents, the environment nature, centralised or distributed, batch or interactive execution, reproducibility and scalability issues. In this paper, we focused our study on parameter sensitivity of agent-based simulators to highlight the impact of design choices made by simulator builders.

We have restrained our study on agent scheduling and environment updating. To do so, we have proposed in section 2 two criteria that help to divide conceptual and implementations choices in four distinct approaches: purely sequential and fair, purely sequential and unfair, parallel and fair, and real-time. We have shown the advantages and problems that are implied by these conceptual and implementation related choices and we have presented some guidelines that should help simulator designers to choose the right approach for the right simulation model.

The question we are left with is "Is-it possible to define an universal simulator ?", that could be configured and customised in order to allow the whole range of approaches. The heterogeneity of simulation models requirements and also the diversity of technological choices, let us think that it is improbable that such a tool will appear.

Future works will be focused on the formalisation of these notions of equity and simultaneity in order to provide a conceptual framework to characterise more precisely simulation models and their implementations.

REFERENCES

Arunachalam, S., Zalila-Wenkstern, R., and Steiner, R. (2008). Environment mediated multi agent simulation tools; a comparison. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 43 –48.

Bigbee, A., Cioffi-Revilla, C., and Luke, S. (2005). Replication of sugarscape using mason. In *4th International Workshop on Agent-based Approaches in Economic and Social Complex Systems (AESCS 2005)*.

Epstein, J. M. and Axtell, R. L. (1996). *Growing Artificial Societies: Social Science from the Bottom Up (Complex Adaptive Systems)*. The MIT Press, 1st printing edition.

Kubera, Y., Mathieu, P., and Picault, S. (2010). Everything can be agent! In van der Hoek, W., Kaminka, G., Lespérance, Y., Luck, M., and Sen, S., editors, *Proceedings of the ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2010)*, pages 1547–1548. International Foundation for Autonomous Agents and Multiagent Systems.

Macal, C. M. and North, M. J. (2007). Agent-based modeling and simulation: desktop abms. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, WSC '07, pages 95–106, Piscataway, NJ, USA. IEEE Press.

Perumalla, K. S. and Aaby, B. G. (2008). Data parallel execution challenges and runtime performance of agent simulations on gpus. In *Proceedings of the 2008 Spring simulation multiconference*, SpringSim '08, pages 116–123, San Diego, CA, USA. Society for Computer Simulation International.

Richmond Paul, Coakley Simon, R. D. (2009). Cellular level agent based modelling on the graphics processing unit. In *High Performance Computational Systems Biology*, Trento, Italy.