Interaction Biases in Multi-Agent Simulations: An Experimental Study

Yoann Kubera, Philippe Mathieu, Sébastien Picault

Interaction Biases in Multi-Agent Simulations : An Experimental Study

Yoann Kubera, Philippe Mathieu, and Sébastien Picault

LIFL UMR USTL-CNRS 8022
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
Cité Scientifique - 59655 Villeneuve d'Ascq Cedex – FRANCE
{yoann.kubera,philippe.mathieu,sebastien.picault}@lifl.fr

Abstract. How to ensure that two different implementations of a simulation will produce the same results ? In order to assure simulation reproducibility, some domain-independent functional unit must be precisely described. We show in this paper that the management unit that rules the participation of an agent in simultaneous interactions is one of them. Usually, many choices concerning this unit are made implicitly, even if they might lead to many simulation biases. We illustrate this issue through a study of biases that appear even in simple cases, due to a specification lack, and we propose as a solution a classification of interactions that makes those choices explicit.

1 Introduction

Multi-Agent Based Simulations (MABS) – and more generally computer simulations – are a tool used to reinforce, invalidate or compare hypothesis on the origin of a particular emergent phenomenon. The model of a simulation needs these hypothesis become concrete, and has to provide all the information required to perform experiments. Consequently, implementations of this model made by different persons have to produce results with similar nature – *i.e.* results of such a model have to be reproducible.

Building a simulation is a process leading from a domain-specific model to an operational model – through knowledge representation formalisms – and then from the operational model to its implementation in a given programming language, on a given simulation platform [1]. Sadly, there is no consensus about what information each model should contain, since the separation between domain-specific model, operational model and implementation – *i.e.* computer science-specific model – is ambiguous itself [2, 3]. Thence, each step of the simulation design process involves choices – both explicit or implicit – regarding ambiguous parts of the previous step model. Those choices have a more or less dramatic influence on the execution and outcomes of the simulation. Since simulations have to be reproducible, the biases these choices may introduce must be studied, and issues about how and to what extent each choice may change simulations outcomes have to be handled.

We uphold that the modeler has to be aware of – and to understand – each possible choice, and has to specify the ones he chooses for its model. Without this specification, the ambiguity of the models leads to implementations that do not behave as it was initially expected and thus produce unexploitable results.

The spectrum of implicit choices is wide, and concerns very different parts of agent's and simulation's architecture. To make their study easier, the architecture of a multi-agent simulation is considered here through three almost independent functional units that underlie any kind of simulation. These units are the Activation Unit that manages time related elements of the simulation like *when agents trigger their behavior*, or *in which interactions an agent may participate simultaneously*, the Definition Unit that specifies all the interactions the agents are able to initiate, and the Selection Unit that corresponds to interaction selection. In this paper the Activation Unit is studied.

Interactions between agents – *i.e.* actions involving simultaneously two or more agents – are the source of simulation's emergent properties. Thus, they have a major role in MABS. But, because current MABS design methodologies focus only on the behavior of independent agents, many design choices concerning interactions are not explicit. In particular, the participation of an agent in interactions occurring at the same time is almost never tackled, because of not adapted knowledge representation.

This paper aims at studying implementation choices concerning the Activation Unit, and more precisely on how simultaneous interactions are handled. In order to make the study of this issue possible, we use the knowledge representation provided by the IODA methodology [4], which is fit to model such problems. Thanks to a study of some experiments, we present the two main patterns – called *interaction classes* – used to handle simultaneous interactions in any kind of simulation. This study also illustrates the consequences of wrong implementation choices – *i.e.* the misuse of interaction classes or wrong time representations. We uphold that defining what simultaneously means in the model, and providing a class for every interaction in the model determines precisely how the Activation Unit is supposed to manage simultaneous interactions without ambiguities. Thus, it makes sure that the model is implemented without biases.

This paper is organized as following. First, related work concerning the studied functional unit, the Activation Unit, are presented in section 2. Then, the functional decomposition that underlies any multi-agent based simulation, on which this paper's studies are based, and the knowledge representation of the IODA methodology are presented in section 3. Section 4 describes the protocol followed by the experiments of the study. Section 5 to 6 describe two experiments , which results are interpreted to identify interaction classes, and to illustrate the consequences of erroneous implementation choices. Then, section 7 summarizes the results of experiments, and proposes a classification of interactions in order to avoid implementation biases of the Activation Unit. Eventually, section 8 discusses about these results, and emphasizes the need to understand what *"simultaneous interactions"* means in the Activation Unit.

2 Related Works

In many simulation platforms, design is centered on agents and the actions they initiate, rather than on the interactions that may occur between them. Consequently, the definition of which interaction an agent may initiate at a particular time does :

- neither take into account in which interaction it already participates as a target (see section 3.1);
- nor take into account in which interaction the other agents already participate in.

This leads to many biases in the simulation and its outcomes.

For instance, Epstein and Axtell [5] presented an ecosystem simulation where an agent may reproduce more than twice at the same time (once as a the initiator of the interaction, and one or more times as a target). This bias was identified by Lawson [6], and corrected with **a modification of the model** through the addition of a gestation period.

Yet, this issue is not restricted to ecosystem simulation. Indeed, it applies to every simulation where agents have to perform particular interactions at most once at a time (for instance agents that trade goods). Thus, it has to be dealt with **in the domain-independent architecture of the simulation** rather than in the models.

This problem leads Michel [3] to manage interactions depending on their *Strong* or *Weak* nature. This solution is adequate if agents are the only participants in interactions. Thus, it does not solve the problem for interactions between an agent and an object. Indeed, interactions like *Withdraw cash from an automated teller machine* may be performed simultaneously by two different agents with the same machine.

Weyns [7] proposes a more refined solution through the qualification of the relationship between two actions[1]. Two actions may be *Independent*, *Joint*, *Concurrent* or *Influencing*. This solution manages interactions by getting from each agent *"I intend to perform the* I *interaction with the* A *agent as target"* like messages. A mastering unit then gathers these messages, finds out the relationship between them and executes compatible ones.

In spite of its undeniable advantage of concurrent and influencing interaction handling, this solution has a major issue. Indeed, because interactions not compatible with already occurring interactions are not considered during decision making, an agent may try to perform an impossible interaction. Thus, the agent performs nothing at that time, even if another interaction is possible.

To fill this gap, simultaneous interactions have to be considered at decision making. This requires an interaction-oriented design of decision making, like the one shortly presented in the next section.

[1] Although the author uses the term "action", it keeps the same meaning than our "interaction" (see section 3.1).

3 Multi-agent Simulations

Even if the application domains of multi-agent simulations are heterogeneous, they can be split into different and weakly dependent functional units [8, 9], like agents scheduling, communications, modification conflicts solving, *etc.*

We consider here a particular decomposition of a simulation (see Fig. 2) in three main units, called Activation Unit, Definition Unit and Selection Unit, which respectively drive time related elements in the simulation, declaration of what agents are able to perform, and finally how interaction selection is made. This decomposition underlies any kind of simulation.

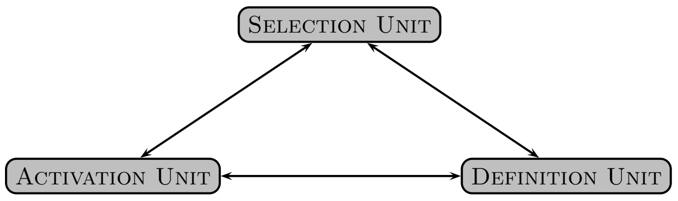


Fig. 1. The three main functional units of a multi-agent simulation.

This kind of separation in different software units is usual in cognitive agent architectures with plans like the *Act-R* [10] or *Soar* [11] language, where knowledge representation is at the center of the simulation, but does not exist in reactive simulation platforms. Moreover the notion of interaction – *i.e.* semantic block of actions involving simultaneously a fixed number of agents (see Sect. 3.1) – is generally hard-coded in the behavior of agents. Because the design of simulations implies crucial choices about those three units, we claim that it is important to make this separation clear, even in reactive simulations, in order to make modeling choices explicit.

3.1 An Interaction-Oriented Design of Simulations

The definition of interactions, and how they are integrated in the knowledge of agents, are based on *IODA* concepts [4]. Please note that *IODA* provides advanced methodological and software engineering tools to design interactions in MABS. Since we do not need all refinements it provides, we use a simplified version of its definitions.

To make the difference between the abstract concept of agent (for instance Wolves), and agent instances (a particular Wolf), we use the notion of ***agent families*** as abstract concept of agent. Thus, the word ***agent*** refers to an agent instance.

Definition 1. *An* **agent family** *is an abstract set of agent instances, which share all or part of their properties and behavior.*

Definition 2. *An* **interaction** *is a structured set of actions involving simultaneously a fixed number of agents instances that can occur only if some conditions are met.*

An interaction is represented as a couple (*conditions*, *actions*), where *condition* is a boolean function and *action* is a procedure. Both have agent instances as parameters. Agents that are involved in an interaction play different roles. We make a difference between ***Source*** agents that may initiate the interaction (in general the one selected by the ACTIVATION UNIT) and ***Target*** agents that may undergo it.

Definition 3. *Let $\mathbb{F}$ be the set of all agent families. Let $\mathcal{S} \in \mathbb{F}$ and $\mathcal{T} \in \mathbb{F}$ be agent families.*
We note $a_{\mathcal{S}/\mathcal{T}}$ the **set of all interactions** *that an instance of the $\mathcal{S}$ agent family is able to initiate with an instance of the $\mathcal{T}$ agent family as a target.*

Thanks to these definitions, we can specify the knowledge of an agent family $\mathcal{S} \in \mathbb{F}$ as the set $\bigcup\limits_{\mathcal{T} \in \mathbb{F}} a_{\mathcal{S}/\mathcal{T}}$, which contains every interactions it is able to initiate as source with any agent family as target.

To unify knowledge, actions (for instance WANDER or DIE) are considered as interactions, which target is implicit (either the environment, or the agent itself). This kind of interactions is called **degenerate interaction**. We do not add this to our notations, please see [4] for more information.

The definition of perceived affordances uses the notion of realizable interaction, in order to determine if two agents can participate in an interaction.

Definition 4. *Let I be an interaction, and $x \prec \mathcal{S}$, $y \prec \mathcal{T}$ two agents. The tuple (I, x, y) is* **realizable** *(written $r(I, x, y)$) if and only if :*

- *$I \in a_{\mathcal{S}/\mathcal{T}}$, i.e. agents of $\mathcal{S}$ family are able to perform I with agents of $\mathcal{T}$ family;*
- *the conditions of I hold true with x as source and y as target.*

A realizable tuple represents one interaction that an agent can initiate with a particular target agent. Moreover, the agent's perceived affordances are the set of all interactions it can initiate in a given context. Thus, at a time t, the perceived affordances of the x agent are the set of all realizable tuples that x may perform.

Definition 5. *Let $\mathbb{A}_t$ be the set of all agents in the simulation at a time t, and $x \in \mathbb{A}_t$.*
Then, the **perceived affordances** *$\mathbb{R}_t(x)$ that x may perform at time t is the set :*

$$\mathbb{R}_t(x) = \bigcup_{y \in \mathbb{A}_t} \bigcup_{I \in a_{x/y}} \{(I, x, y) | r(I, x, y)\}$$

3.2 Discussion about this knowledge representation

In some cases, the distinction between source and target agents might appear as a restriction. This implies that the knowledge representation of IODA cannot be used to model any kind of simulation.

For instance the HANDSHAKING interaction does not seem to make the difference between the source and the target agent – *i.e.* source and target roles are symmetric.

The IODA methodology upholds that, even if the roles in the interaction are symmetric, the two agents do not spontaneously choose to perform the HANDSHAKING together. One agent is at the origin of the HANDSHAKE, and requests the other agent if it wants to perform that interaction.

Consequently, interactions have always an initiator – *i.e.* a source. Thus, the knowledge representation provided by IODA can be used to model any kind of simulation.

3.3 Time Representation and Simultaneous Interactions

The model of a simulation has to define how time is represented in the simulation. Indeed, this representation has a deep influence on how the ACTIVATION UNIT is supposed to work. Moreover, the representation of time defines directly the meaning of *simultaneous interactions*.

Mainly, two different time representation are used[2] in MABS. Both are based on the discrete event paradigm, where time is considered as a number that increases during simulation, given a particular evolution process. Time can :

– either evolve by steps of fixed length. Usually, these kind of simulation are called *Discrete*, because time can be considered as a finite set of integers called *time steps*. At each time step, the ACTIVATION UNIT will ask one or more agents to perform their SELECTION UNIT. The choice of asked agents is up to the policy used in the ACTIVATION UNIT. For instance, ask all agents sequentially in random order, or ask one agent chosen randomly, *etc*;
– or evolve event by event. Usually, these kind of simulation are called *Continuous*, or at least *Pseudo-Continuous*, because the time elapsed between two event is not fixed. At each event, an agent performs its SELECTION UNIT, and schedules an event for its next activation, depending on the interaction it initiated.

For each time representation, interactions have a duration – *i.e.* a time interval during which the interaction is considered as being performed. The side-effects of the interaction are considered consistent only at the end of this interval.

In the case of continuous time, the duration has to be defined by the modeler, because the scheduling of events depends on the date agents finish their current interaction. Thus, duration is explicit in the model.

[2] these two are used for illustration purposes, in order to elicit the notion of interaction duration. Other time representation might exist.

For discrete time, this notion of duration is not as obvious. Implicitly, the duration of actions and interactions is the length of a time step. Sadly, the lack of specification concerning this point causes critical issues in simulation (see section 8).

Thanks to this definition, simultaneous interactions can be defined :

Definition 6. *Two interactions are considered as* **simultaneous** *iff their duration time interval are intersecting.*

We uphold that the meaning of simultaneous interactions has to be explicitly defined in the model, in order to assure simulation reproducibility.

3.4 Functional Decomposition of a Multi-agent Simulation

Each functional unit is in charge of a specific feature of a multi-agent simulation.

The Activation Unit tells when agents may act, the time elapsed between their actions/interactions, what to do if an agent tries to interact with an already acting agent, *etc.* It describes all time-related elements in the simulation.

The Definition Unit lists all interactions in the simulation, under what conditions and between what kind of agents they are possible, and what actions they launch. It is the set of all possible behaviors, defined independently from agents specificities. This unit provides information required to build the space of all possible interactions the selected agent may initiate as a source – *i.e.* all realizable tuples *(Interaction, Selected Agent, Target agents)*, also corresponding to the perceived affordances of the source, as defined in [12].

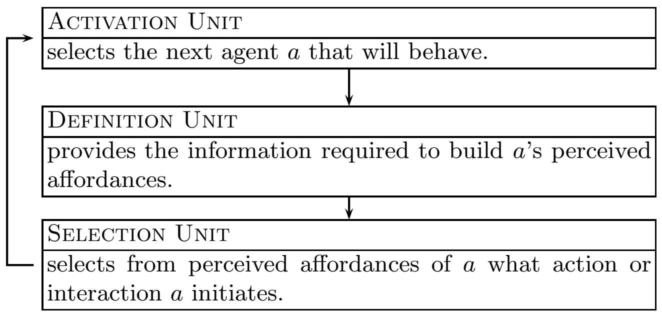
The Selection Unit describes the cognitive or reactive process an agent uses to select which interaction it initiates, and, if many are possible, decides among them the one to initiate.

A simulation is a repetition of 3-steps sequences, where each step exploits a different functional unit (see Fig. 2).

We already argued in [4] for the advantage of agent-independent defined interactions and proposed a formal definition for it, thus creating a software separation between the Definition Unit – which is domain dependent – and both Selection Unit and Activation Unit.

4 Experimental Frame

The goal of this paper is to measure to what extent modifications of the Activation Unit may change simulation outcomes, and how an adequate one may avoid simulation biases. This point is illustrated through two experiments, each confronting two different implementations of the Activation Unit. Thus,


ACTIVATION UNIT

selects the next agent a that will behave.

DEFINITION UNIT

provides the information required to build a's perceived affordances.

SELECTION UNIT

selects from perceived affordances of a what action or interaction a initiates.

Fig. 2. How the three main functional units of a multi-agent simulation are used to run a simulation.

the only variating parameter in an experiment is the ACTIVATION UNIT : its DEFINITION UNIT and SELECTION UNIT remain the same.

This section presents the DEFINITION UNIT, SELECTION UNIT, the protocol used in our experiments, and preliminary discussions concerning the ACTIVATION UNIT.

4.1 The Definition Unit

The DEFINITION UNIT, which defines all domain-dependent information, will change from one experiment to the other. In order to make the comprehension of our examples easier, every experiment deals with the same overall simulation problem : the evolution of an ecosystem containing predators and preys. Please note that experiments provide only an illustration of the general issue we deal with. The solutions presented in this paper are obviously not restricted to that particular simulation, and do not avoid only the biases emphasized in this paper.

4.2 The Selection Unit

The SELECTION UNIT, which corresponds to agent's decision making process, will keep the same architecture in all our experiments.

The architecture we use is the most used one for reactive agents : a subsumption-like architecture [13] that tries every interaction sequentially until a realizable one is found. Every source agent gives to every interaction it can initiate a priority value, which denotes the order it tries the interactions (see Fig. 1).

4.3 The Activation Unit

The model of a simulation has to define how time is represented in the simulation.

In the experiments of this paper, we consider that :

Algorithm 1 SELECTION UNIT used in the experiments of this paper. It defines how an agent a chooses what interaction it initiates at a time z.

```
$\mathbb{R}_z(a) \Leftarrow$ the set of all realizable interactions $a$ may initiate
$\mathbb{P} \Leftarrow$ the decreasing set of all priorities $a$ gives to the interactions it can initiate
$\mathbb{L} \Leftarrow \emptyset$
for $p$ in $\mathbb{P}$ do
  for $(I, a, t)$ in $\mathbb{R}_z(a)$ do
    if $I$ has $p$ priority for $a$ then
      $\mathbb{L} \Leftarrow \mathbb{L} \cup \{(I, a, t)\}$
    end if
  end for
  if $\mathbb{L} \neq \emptyset$ then
    $a$ initiates the interaction of a tuple of $\mathbb{L}$ chosen at random
    stop
  end if
end for
$a$ initiates nothing
```

- time is *Discrete* (see Sect. 3.3);
- during every time step, the agents trigger their SELECTION UNIT in sequence. The order of this sequence is defined at random for each time step, in order to keep equity between agents.
- the duration of interactions is the length of a time step. Thus, two interactions occurring at the same time step are considered simultaneous.

These choices are the most usual ones in classical reactive simulations. Moreover, we make the assumption that an agent may initiate at most one interaction at a time – *i.e.* it cannot be simultaneously the source of two or more interactions. This issue will be discussed in section 8.

4.4 Experimental Protocol

The experimental protocol used in this paper is :

1. first, the aim of the experiment is outlined;
2. then, the DEFINITION UNIT and SELECTION UNIT used by both implementations of this experiment are defined;
3. next, the two ACTIVATION UNIT used in the experiment, and experiment's initial conditions, are described;
4. eventually, the results of the execution of both implementation of the experiment are presented and discussed. From this discussion, an interaction class is emphasized to avoid a possible simulation bias.

Please note that all experiments presented below are voluntary basic to stress out where the problems lie : they obviously do exist in more complex situations as well.

5 First Experiment : Multiple Participation to Interactions Bias

This experiment studies the limits of the usual naive algorithm and introduces as a solution a first interaction class called *exclusive* interaction.

Model used : This simulation studies an ecosystem composed by grass and sheep. Because sheep can move, classical analytical models cannot be used to model the population of species : this simulation requires multi-agent systems.

The environment is a two dimensions toroidal continuous space split into unitary square parcels. Every parcel $\mathcal{P}$ has an attribute $q(\mathcal{P})$ that increases of one unit at every simulation step. $\mathcal{P}$ is said containing grass when $q(\mathcal{P}) > 0$. If $\mathcal{P}$ is emptied by an agent, then $q(\mathcal{P}) = 1 - r_{grass}$ (*i.e.* r_{grass} is the time grass needs to grow) . A sheep $\mathcal{S}$ is an agent with an energy attribute $e(\mathcal{S})$ representing its health, which can initiate the interactions :

1. To **Die** if $e(\mathcal{S}) \leq 0$. Then :
   - $\mathcal{S}$ is removed from the environment.
2. To **Reproduce** with another sheep $\mathcal{S}'$ at a maximal distance of 1 from $\mathcal{S}$ if $e(\mathcal{S}) > 0$ and $e(\mathcal{S}') > 0$. Then :
   - A new sheep $\mathcal{S}''$ is created at $\mathcal{S}$'s location, and $e(\mathcal{S}'') = Min(e(\mathcal{S}), e_{repr}) + Min(e(\mathcal{S}'), e_{repr})$, where e_{repr} stands for the energy consumed by reproduction.
   - $e(\mathcal{S})$ and $e(\mathcal{S}')$ are decreased by e_{repr}.
   - $\mathcal{S}$, $\mathcal{S}'$ and $\mathcal{S}''$ execute the Wander interaction (see below).
3. To **Eat** grass on $\mathcal{S}$'s parcel if it contains some. Then :
   - $e(\mathcal{S})$ increases from e_{eat}, where e_{eat} is the energy gained by eating.
   - $\mathcal{S}$ empties the parcel he is onto.
4. To **Wander** with no conditions. Then :
   - $\mathcal{S}$ turns itself from an angle in $[-\pi, \pi[$ and moves forward from 1 unit.
   - $e(\mathcal{S})$ decreases from e_{wan}, where e_{wan} is the energy consumed by moving.

Sheep behave by using the order $1 > 2 > 3 > 4$ in their Selection Unit, thus they first have to Die, if they don't, they try to Reproduce, if they don't, they try to Eat, . . .

Experimental design : We used this model in a 33×33 environment containing 1089 parcels, where 30% have $q(\mathcal{P}) = 0$ and 70% $q(\mathcal{P}) \in] - r_{grass}, -1]$, and 70 sheep such that $e(\mathcal{S}) = 2 \times e_{repr}$. We also set $r_{grass} = 10$, $e_{repr} = 15$, $e_{wan} = 2$ and $e_{eat} = 7$.

Figure 3 compares the evolution of the sheep population of this model implemented with respectively the *naive* (*single interaction*) Activation Unit, from Algorithm. 2 (from Algorithm. 3).

Algorithm 2 *"Naive* ACTIVATION UNIT*"*. In this implementation, an agent does not take into account simultaneous interactions. MAX is the duration of the simulation, in time steps.

```
for $i = 1$ to $MAX$ do
    Update the environment
    for $a$ in agents in the environment at time $z$ do
        $\mathbb{R}_z(a) \Leftarrow$ all realizable tuples that $a$ may initiate
        $(I, a, t) \Leftarrow$ a tuple of $\mathbb{R}_z(a)$ selected with a particular SELECTION UNIT
        if $(I, a, t) \neq$ null then
            Execute $I$ with $a$ as source and $t$ as target.
        end if
    end for
end for
```

Algorithm 3 *"Single Interaction* ACTIVATION UNIT*"* In this implementation, an agent participates only in one interaction at a time, either as the source or as the target. MAX is the duration of the simulation, in time steps.

```
for $i = 1$ to $MAX$ do
    Update the environment
    for $a$ in agents in the environment at time $z$ do
        Tag $a$ as operative
    end for
    for $a$ in agents in the environment at time $z$ do
        $\mathbb{R}_z(a) \Leftarrow$ all realizable tuples that $a$ may initiate
        for $(I, a, t)$ in $\mathbb{R}_z(a)$ do
            if $a$ is not operative or $t$ is not operative then
                Remove $(I, a, t)$ from $\mathbb{R}_z(a)$
            end if
        end for
        $(I, a, t) \Leftarrow$ a tuple of $\mathbb{R}_z(a)$ selected with a particular SELECTION UNIT
        if $(I, a, t) \neq$ null then
            Execute $I$ with $a$ as source and $t$ as target.
            Tag $a$ and $t$ as not operative
        end if
    end for
end for
```

Results and Discussion : With exactly the same initial environment, the experiment using the *naive* algorithm (see Algorithm 2) produces in overall 68 more sheep than the one using the *single interaction* algorithm (see Algorithm 3).

This difference lies in the number of interactions an agent may participate in during a simulation step. In the *naive* algorithm, a sheep targeted by a REPRODUCE interaction can be the source of another interaction. On the opposite, in the *single interaction* algorithm an agent participates at maximum in one interaction, either as a source or as a target. This difference has a great impact on

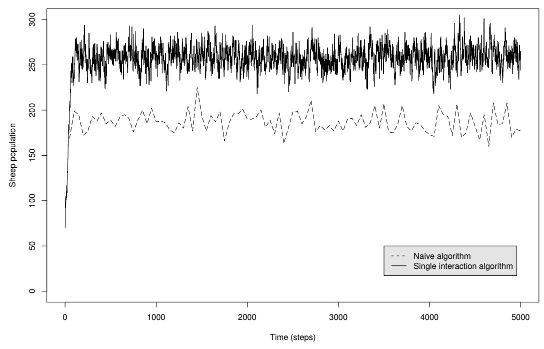


Fig. 3. Sheep population evolution against time (in simulation steps) respectively displayed in dots (in line), for the model presented in Sect. 5 implemented with the *naive* (*single interaction*) algorithm of Algorithm. 2 (Algorithm. 3).

the sheep energy dynamics, their reproduction and death rate, and their density, and, as a consequence, on the sheep population dynamics.

In this simple experience, this difference may be considered as the result of different interpretations of the model, thus it cannot be considered as a bias. Nevertheless, it corresponds to a bias in other experiences, as shown in [3] : if a sheep participates in more than one reproduction per time step, then the sheep reproduction probability is different from the one designed in the model.

We outlined with this experience that the number of interactions an agent can simultaneously participate in has to be restricted. To cope with this problem, we identified a first interaction class called *exclusive* interactions. An agent can participate in such an interaction only once at a time, either as source or as target.

6 Second Experiment : Single Participation to Interactions as Target Bias

This section illustrates a simulation problem concerning the target of an interaction that *exclusive* interactions alone are too restrictive to solve.

Model used : This model adds to the one of experiment 5 a new agent named wolf, and a new interaction to a sheep $\mathcal{S}$:

5. To **Flee** from a wolf $\mathcal{W}$ at a maximal distance of 10 from $\mathcal{S}$. Then :
   - $\mathcal{S}$ turns its back towards where $\mathcal{W}$ is and moves forward from 1 unit.
   - $e(\mathcal{S})$ decreases from e_{wan}.

Sheep behave using the order $1 > 5 > 2 > 3 > 4$ in their Selection Unit, and wolves only Wander in the environment.

In this simulation, sheep Flee systematically wolves. As a consequence, wolves are supposed to be at the center of an empty area.

Experimental design : They are the same as in the experiment of Sect. 5, except that there is a wolf at a random position, and that the simulation is implemented with :

– firstly with the *single interaction* ACTIVATION UNIT (from Algorithm. 3);
– then with the *parallel interactions* ACTIVATION UNIT (from Algorithm. 4) where
  - FLEE is from $\mathcal{I}_2$ interaction class;
  - other interactions are from $\mathcal{I}_1$ class.

Algorithm 4 *"Parallel Interaction* ACTIVATION UNIT*"*. In this implementation, an agent participates only in one interaction of $\mathcal{I}_1$ at a time, either as the source or as the target. An agent can be the target of as many interaction of $\mathcal{I}_2$ as necessary. MAX is the duration of the simulation, in time steps.

```
for $i = 1$ to $MAX$ do
  Update the environment
  for $a$ in agents in the environment at time $z$ do
    Tag $a$ as operative
  end for
  for $a$ in agents in the environment at time $z$ do
    $\mathbb{R}_z(a) \Leftarrow$ all realizable tuples that $a$ may initiate
    for $(I, a, t)$ in $\mathbb{R}_z(a)$ do
      if $I$ is from $\mathcal{I}_1$ class and ($a$ is not operative or $t$ is not operative) then
        Remove $(I, a, t)$ from $\mathbb{R}_z(a)$
      else if $I$ is from $\mathcal{I}_2$ class and $a$ is not operative then
        Remove $(I, a, t)$ from $\mathbb{R}_z(a)$
      end if
    end for
    $(I, a, t) \Leftarrow$ a tuple of $\mathbb{R}_z(a)$ selected with a particular SELECTION UNIT
    if $(I, a, t) \neq$ null then
      Execute $I$ with $a$ as source and $t$ as target.
      if $I$ is from $\mathcal{I}_1$ class then
        Tag $a$ and $t$ as not operative
      else if $I$ is from $\mathcal{I}_2$ class then
        Tag $a$ as not operative
      end if
    end if
  end for
end for
```

The outcomes of such implementations of this model are displayed in Fig. 4.

Results and Discussion : The *parallel interactions* ACTIVATION UNIT produces the expected result (right on Fig. 4), and the *single interaction* ACTIVATION UNIT (left on Fig. 4) is obviously biased : there is no empty halo around the wolf.

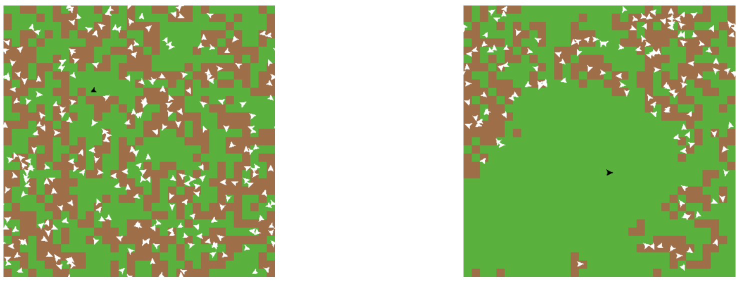


Fig. 4. Outcomes screenshot of the experiment presented in Sect. 6. This model was respectively implemented with the *single interaction* (*parallel interactions*) Activation Unit from Algorithm. 3 (from Algorithm. 4), displayed to the left (right), where respectively light (dark) arrows are sheep (wolves), and light (dark) squares are empty (full) parcels.

The difference lies in the number of interactions a wolf can undergo simultaneously. When a wolf is the target of a Flee interaction, it is set not operative, thus it cannot be the target of another Flee interaction. Consequently, a wolf is fled once per simulation step, and other sheep behave as if there was no wolf.

We outlined with this experiment that all interactions do not put the same restrictions onto their target agent. Interaction classes have to reflect this difference, thus, in addition to *exclusive* interactions, we introduce *parallel* interactions, where agents may simultaneously be targeted as many times as needed.

7 Experiments Summary : A Classification of Interactions

Through two experiments, we have shown that the model has to answer the question *"is the source (or target) agent of an interaction allowed to be simultaneously the source (or target) of another interaction ?"*. Otherwise the lack of specifications leads to ambiguities from which many biases may result.

7.1 Interaction classes

As a solution, we identified two interaction classes, each answering differently to the question above. Considering our modeling experience, these classes correspond to the two main recurrent patterns used to handle simultaneous interactions. The association of a class to each interaction in a model describes explicitly how they are managed, and thus avoids many biases at implementation.

The two interaction classes are :

exclusive interaction : An agent is allowed to participate only to one exclusive interaction at a time, whether as source or as target. In the experiments and in the Algorithm. 3 and Algorithm. 4, it corresponds to $\mathcal{I}_1$ interaction class. It is the case of the Reproduce interaction.

Table 1. Summary of what interactions an agent may participate in simultaneously, depending on its role in them. The cross at the intersection of the line (Exclusive, S) and column (Parallel, T) is read *"An agent can simultaneously be the source (S) of an exclusive interaction and the target (T) of a parallel interaction"*. The empty cell at the intersection of the line (Exclusive, T) and column (Exclusive, T) is read *"An agent cannot simultaneously be the target (T) of an exclusive interaction and the target (T) of another exclusive interaction"*.

		Exclusive		Parallel	
		S	T	S	T
Exclusive	S				×
	T				×
Parallel	S				×
	T	×	×	×	×

parallel interaction : An agent is allowed to be simultaneously the target of as many parallel interaction as needed. In the experiments and in the Algorithm. 4, it corresponds to $\mathcal{I}_2$ interaction class. It is the case of the FLEE interaction.

Moreover, we consider that an agent can initiate only one interaction at a time. Thus an agent cannot simultaneously be the source of a parallel interaction, and participate to an exclusive interaction (either as source or target).

Table. 1 provides a summary of what interaction classes allow and forbid.

7.2 Use of interaction classes

In practice, interaction classes are exploited before the SELECTION UNIT executes. The agent that performs the SELECTION UNIT cannot initiate all the affordances – *i.e.* all realizable tuples – it listed. Indeed, this agent might already be involved in some interactions. Thus, it has to remove from its affordances all interactions that cannot be initiated simultaneously with the interactions already occurring. This removal is based on the table 1 that summarizes what interactions are allowed simultaneously. For instance, an agent cannot initiate an exclusive interaction with an agent that is already the target of an exclusive interaction (see the intersection of the line $(Exclusive, S)$ and the column $(Exclusive, T)$ in Fig. 1).

Thus, the 3-steps sequence of Fig. 2 in section 3.4 becomes as displayed in Fig. 5. The algorithm 4 provides an implementation of such an ACTIVATION UNIT, where :

– time is *Discrete* (see Sect. 3.3);
– during every time step, the agents trigger their SELECTION UNIT in sequence. The order of this sequence is defined at random for each time step, in order to keep equity between agents.
– the duration of interactions is the length of a time step. Thus, two interactions occurring at the same time step are considered simultaneous.
– an agent may initiate at most one interaction at a time, *i.e.* it cannot be simultaneously the source of two or more interactions.

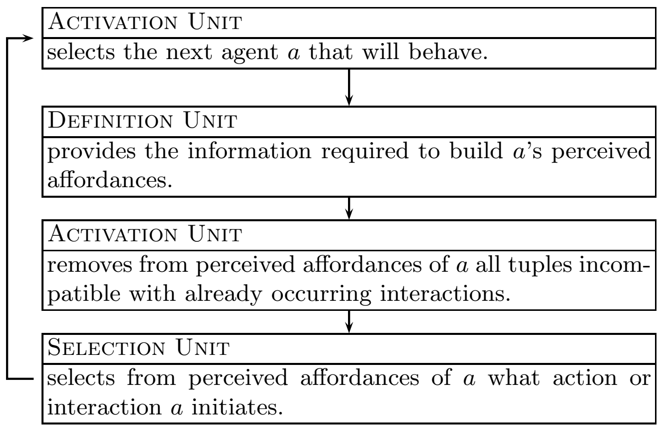

ACTIVATION UNIT
selects the next agent a that will behave.
DEFINITION UNIT
provides the information required to build a's perceived affordances.
ACTIVATION UNIT
removes from perceived affordances of a all tuples incompatible with already occurring interactions.
SELECTION UNIT
selects from perceived affordances of a what action or interaction a initiates.

Fig. 5. How the three main functional units of a multi-agent simulation are used to run a simulation, This takes into account simultaneous interactions.

7.3 Why defining interaction classes in the model ?

Interaction classes answer the question *"is the source (or target) agent of an interaction allowed to be simultaneously the source (or target) of another interaction ?"*.

These interaction classes are deeply bound with the algorithms used to process interactions at implementation. Thus, knowledge on which interaction classes are present in an operational model :

- removes ambiguities found during implementation.
- determines if a simulation platform is fit to implement the model. For instance, a simulation platform like *Netlogo*[14] is no fit by default to implement models containing exclusive interactions : the user has to develop his own ACTIVATION UNIT;

The implementation of such specifications is made easier by a software separation between ACTIVATION UNIT, DEFINITION UNIT and SELECTION UNIT. Indeed, it forces the user to choose interaction classes explicitly, and thus forces to understand the underlying algorithms. It is the case of the *IODA* methodology and the *JEDI* simulation platform [4] where this separation is made by the reification of interactions through the whole simulation process.

Note that even if these classes are defined in the context of discrete simulations – *i.e.* with simulation steps of fixed length – they remain valid for other simulations.

8 Discussion about our solution

Our solution makes the assumption that an agent can simultaneously be the source of at most one interaction. This seems to be an hindrance for some simulations. We illustrate this point on an example.

8.1 Issues about simultaneous initiation of many interactions

Many reactive simulations, such as the *Wolf Sheep Predation* of *Netlogo* (see Algorithm. 5) seem to make the assumption that the Selection Unit can select and initiate more than one interaction during a time step.

For instance, in the *Wolf Sheep Predation* :

– the Activation Unit is defined by the *"go"* block. This block ends with a *"tick"* command, which tells that the Activation Unit uses discrete time. The code *"ask sheep"* tells that for each simulation time step, sheep agents are asked in a random sequence to perform once the content of the *"ask sheep"* block.
– the Definition Unit and Selection Unit are mixed, and defined in the content of the *"ask sheep"* block. In this block sheep may initiate a move interaction, then a eat-grass interaction, then a reproduce-sheep interaction and finally a death interaction.

Sheep seem to be able to initiate up to four interactions during a time step.

8.2 Discussion about this simulation

Time steps are the most easiest way to build an Activation Unit. In simulation using this kind of Activation Unit, the most atomic representation of time is the time step. Thus, interactions are considered as simultaneous if they occur during the same time step.

In the simulation presented above, this means that an agent is able to initiate up to four interactions simultaneously. This seems to invalidate our hypothesis that an agent is able to initiate at most one interaction at a time, and consequently seems to invalidate our solution.

In fact, there is no such thing like initiating simultaneously more than one interaction. The issue is rather related to the wrong use of discrete time, and its underlying definition of simultaneous interactions. Indeed, it makes no sense to uphold that a sheep can initiate Die and Reproduce-Sheep simultaneously. The same holds for any other combination containing two interactions among Die, Reproduce-Sheep, Eat-Grass and Move. These interactions are meant to be executed separately, in sequence. Thus, they are not simultaneous, and have to occur during different time steps. In this case, the implementation does not reflect what the model means.

We uphold that simulations that let agents initiate more than one interaction during the same time step are misusing discrete time Activation Unit, and do not implement the model as it was meant. Thus, initiating at most one interaction at a time is sufficient to model all kinds of simulations.

Algorithm 5 Part of the implementation in *Netlogo* of a Wolf Sheep predation simulation.

```
to go
  (...)
  ask sheep [
    move
    if grass? [
      set energy energy - 1
      eat-grass
    ]
    reproduce-sheep
    death
  ]
  (...)
  tick
  (...)
end

to move  ;; turtle procedure
  rt random 50
  lt random 50
  fd 1
end
```

```
to eat-grass
  if pcolor = green [
    set pcolor brown
    set energy energy + sheep-gain-from-food
  ]
end

to reproduce-sheep
  if random-float 100 < sheep-reproduce [
    set energy (energy / 2)
    hatch 1 [ rt random-float 360 fd 1 ]
  ]
end

to death
  if energy < 0 [ die ]
end
```

9 Conclusion

Most simulations assume and compare hypothesis on a given phenomenon. The model is the mirror of such hypothesis. Thus, it has to contain enough information to assure simulations reproducibility – *i.e.* two implementations of the same model have to produce outcomes with similar natures. Sadly, many modeling and implementation choices are left implicit. This can lead to biases, and thus to non-reproducible simulations. Thus the biases that may result from modeling and implementation choices must be identified and quantified.

In this paper we have shown that the reproducibility of a simulation is not possible without specifying a particular domain-independent functional unit that underlies any simulation, called Activation Unit.

This unit specifies all time related elements in the simulation. In particular, it indicates to what interactions an agent can participate simultaneously. Experiments showed that the lack of specifications concerning the particularities of these interactions may introduce biases in simulation outcomes. Indeed, as an example, the target of a reproduction behavior cannot initiate simultaneously another interaction, otherwise an agent may reproduce twice at the same time.

To solve this kind of problem, we uphold that the model must specify :

– what *simultaneous interaction* means. For discrete simulations, where time is divided in time steps of the same length, interactions are simultaneous if they occur during the same time step;

– the interaction class of each interaction of the simulation, among *exclusive* and *parallel*. Thanks to these classes, an agent can determine which interactions it can initiate – *i.e.* be the source – at a particular time, according to all the interactions already occurring at that time.

We illustrated on an experiment that the lack of knowledge on what simultaneous interactions means may lead to implementations that do not correspond what the model meant. Thus, the specification of what interactions agents may participate in simultaneously have meaning only if the notion of *simultaneous interactions* is known and understood.

Taking into consideration time representation and these classes while conceiving the model removes ambiguities that would have led to biases. Without the specification of these two points, two different developers will likely obtain very different outcomes for the same model.

References

1. Fishwick, P.A.: Computer simulation: growth through extension. Trans. Soc. Comput. Simul. Int. **14**(1) (1997) 3–20
2. Nuno, D., Sichman, J.S.a., Coelho, H.: Towards an emergence-driven software process for agent-based simulation. In: Proceedings of MABS 2002. (2002) 89–104
3. Michel, F., Gouach, A., Ferber, J.: Weak interaction and strong interaction in agent based simulations. In: Proceedings of MABS 2003, Melbourne, Australia (2003) 43–56
4. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations : From theory to implementation. In: Proceedings of ECAI 08, Patras Greece (July 2008) 383–387
5. Epstein, J., Axtell, R.: Growing Artificial Societies. Brookings Institution Press, Washington D.C. (1996)
6. Lawson, B., Park, S.: Asynchronous time evolution in an artificial society mode. Journal of Artificial Societies and Social Simulation (2000)
7. Weyns, D., Holvoet, T.: Model for simultaneous actions in situated multi-agent systems. In: Proceedings of MATES 2003, Erfurt, Germany (2003) 105–119
8. Demazeau, Y.: From interactions to collective behaviour in agent-based systems. In: Proceedings of ECCS'95, Saint-Malo, France (1995) 117–132
9. Weyns, D., Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems: State-of-the-art and research challenges. In: Environments for Multiagent Systems, New York, NY, USA (2004) 1–47
10. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychological Review **111**(4) (2004) 1036–1060
11. Newell, A.: Unified theories of cognition. Harvard University Press, Cambridge, MA, USA (1994)
12. Norman, D.A.: The Psychology of Everyday Things. Basic Books (1988)
13. Brooks, R.A.: A robust layered control system for a mobile robot. iEEE journal of robotics and automation **2**(1) (March 1986) 14–23
14. Wilenski, U.: Netlogo. Center for connected learning and computer-based modeling, http://ccl.northwestern.edu/netlogo/ (1999)