Multi-level agent-based simulations: Four design patterns

Philippe Mathieu, Gildas Morvan, Sébastien Picault

Simulation Modelling Practice and Theory 83 (2018) 51–64

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Multi-level agent-based simulations: Four design patterns

Philippe Mathieu [a], Gildas Morvan [b,*], Sebastien Picault [a,c]

[a] Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL (équipe SMAC) Centre de Recherche en Informatique Signal et Automatique de Lille, Lille F-59000, France
[b] Univ. Artois, EA 3926, Laboratoire de Génie Informatique et d'Automatique de l'Artois (LGI2A), Béthune, France
[c] BIOEPAR, INRA, Oniris, La Chantrerie, Nantes 44307, France

ARTICLE INFO

Article history:
Available online 10 January 2018

Keywords:
Agent-based simulation
Multi-level
Design patterns
Architecture

ABSTRACT

This paper describes four design patterns that aim at systematizing and simplifying the modelling and the implementation of multi-level agent-based simulations. Such simulations are meant to handle entities belonging to different, yet coupled, abstractions or organization levels. The patterns we propose are based on minimal typical situations drawn from the literature. For each pattern, we present use cases, associated data structures and algorithms. For genericity purposes, these patterns rely on a unified description of the capabilities for action and change of the agents. Thus, we propose a precise conceptual and operational framework for the designers of multi-level simulations.

1. Introduction

Multi-Agent Systems (MAS) are intrinsically built as two-level systems: the "microscopic" level, where the agents are endowed with a specific behaviour, and the "macroscopic" level, where the system is seen as a whole. But often this latter level remains *implicit* in the sense that it is "outside" the agents (heterogeneous to them): it is of a *different nature* (set of specifications or issues to be resolved before the design of the MAS, set of descriptors aggregated during or after the functioning of the MAS), and when it exerts a *feedback* on the behaviour of the agents, it is often only in an *implicit* way.

Making the link between the microscopic and macroscopic levels explicit remains an open question [1].

Not only do we not have at the present time any general method to answer it, but the scientific questions underlying this issue are also still rather widely ignored. However, in the current period, the use of MAS is changing and so are the corresponding research subjects. From disciplines such as ethology and sociology, which required relatively small numbers of agents (up to hundreds or thousands), new applications of MAS are now moving to large-scale systems (ecology, molecular biology or cellular, social networks, financial markets, transportation, etc.) where classical approaches encounter many limitations. Issues related to the organization of systems, and specifically their organization into subsystems, take precedence over those for individual decision architectures.

Those new needs are responsible for the development of *Multi-Level Agent-Based Simulations* (MLABS), i.e. MAS which try to provide an explicit representation of the macroscopic level and to each relevant intermediary level, possibly as new agents. This can be done through many forms to address very diverse objectives, such as: introducing in the model abstraction levels which are "useful" or "relevant" to experts in the field [2]; dynamically adaptating the level of detail of the simulation, for instance to save computation time when possible [3]; coupling heterogeneous models to simulate processes

* Corresponding author.
E-mail addresses: philippe.mathieu@univ-lille.fr (P. Mathieu), gildas.morvan@univ-artois.fr (G. Morvan), sebastien.picault@univ-lille.fr (S. Picault).

https://doi.org/10.1016/j.simpat.2017.12.015
1569-190X/

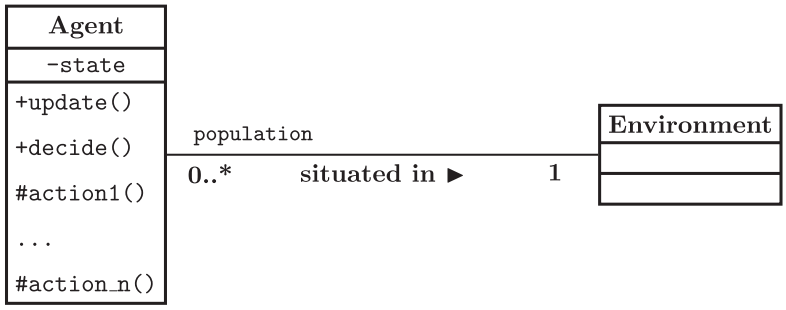

Fig. 1. Elementary structure of an agent and its relationship to the environment.

at different scales [4–6]; or even detecting interesting collective behaviours or emerging spatial structures, either with, or without, reification (agentification) [7].

As observed by [8], there is currently no precise or unambiguous characterization of what is covered by the "multi-level" concept, either in terms of objectives, or system structure, or agent architecture, or algorithms. At best, recurrent problems can be identified, for which many implementations are proposed.

Our goal here is to show that multi-level agent-based simulations involve the combination of elementary situations, which can be characterized by a small number of general criteria, and for which recurrent implementation solutions can be proposed. In other words, we seek to identify and describe *Design Patterns* for multi-level agent-based simulations, in the same sense as in Software Engineering [9].

This paper is organized as follows: in the next section, we present several operational prerequisites on which we rely, and describe our method for classifying concrete problems and identifying patterns. Then, we provide a detailed description of each pattern (name, features, algorithms, use cases) and show how they can be composed in a single simulation. Finally, we discuss methodological, theoretical or practical issues before concluding.

2. Proposed approach

The approach we advocate here consists in characterizing recurrent situations observed in practice in MLABS, in order to build a typology, with the purpose of determining which answers are the most appropriate for each kind of needs. To do so, it is necessary to settle in advance some minimal assumptions regarding our conceptual and operational tools, namely agents and simulation engines. Any multi-agent simulation that supports these criteria should be able to implement the patterns we propose here without any other prerequisites.

2.1. Operational requirements

What we present here is not intended to provide a final definition of an agent in a MAS, or even just in a multi-agent simulation. Instead, our purpose is to resort to the *parsimony principle* (Occam's razor) in order to identify, within multi-agent simulations, a set of features which can be a common foundation for building consistent design patterns in this field.

Then, in what follows, an agent is considered as an entity situated in an environment, with a state (not directly reachable) affected by its actions and the actions of other agents, and which furthermore may be subject to a dynamic evolution of its own (*thereafter $s_t(a)$ denotes the state of agent a at time t*). An agent has several primitives, reflecting its elementary (atomic) capabilities of perception and action. To trigger appropriate changes in its state over time, the agent provides *at most one* public entry point (method, procedure, function...), which we call `update` by convention. Similarly, to trigger action selection, i.e. the choice of primitive actions as a response to its state and its perceptions, the agent provides *at most one* public entry point which we call `decide`. The aim of this entry point is to perform the classical perception-decision-action sequence. By "public entry point", we refer to platform-dependent software control mechanisms, which cannot in general be reduced to language-dependent features such as the well-known `public` vs. `private` Java keywords.

We also focus on situations, quite prevalent even in the MLABS literature, where agents belong to *only one environment* (Fig. 1), without any further assumption regarding the nature of that environment or its implementation (which can also follow design patterns, as shown in [10]). Alternative situations, allowing agents to belong to several environments at the same time, are addressed in e.g. [11–13].

The action primitives of the agents are not intended to be executed directly by the simulation engine, but their execution must occur during the interactions between agents. Thus, we can assume that their access is only allowed for the effective realization of the behaviours of agents.

The autonomy of an agent essentially lies in the fact that the only public entry points are `update` and `decide`. This ensures that there is no other way to manipulate the state of the agent, or distort its perceptions, or cause its actions. In addition, these two entry points should be managed by a *scheduler* which is the only authority to determine when the

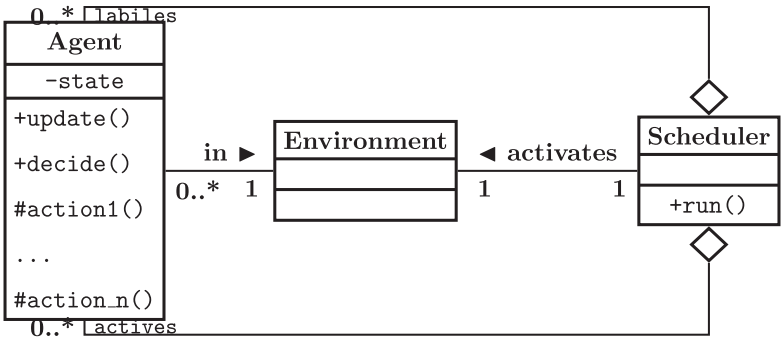

Fig. 2. The scheduler is in charge of maintaining the lists of active and labile agents.

status of the agent must be recalculated and when a decision must be made. This scheduler can be global to the system, as is often the case in simulation, or local to the agent, for example the internal clock of a robot.

Obviously an agent can provide no `update` entry point if it never changes its state spontaneously, or no `decide` entry point if it can do nothing by itself. In the latter case, it is customary to call such an entity a "resource", or an "artefact", or an "active object", etc. instead of an "agent". We assume in the present context that such distinctions are not particularly relevant. They are indeed a *restriction* of the agent concept, all the more arbitrary since the passage from classical multi-agent simulations to multi-level simulations may involve, as we shall show, some loss of autonomy of the agents. Thus, the classical terminology would lead, during the process of aggregation or disaggregation, to transforming agents to artifacts or resources or vice versa.

We therefore prefer to benefit from a wide homogeneity of entities in the model. It has been shown that such a uniform handling, if taken properly into account in the simulation engine, induces almost no additional cost in computation time [14,15]. The only counterpart is the capability to identify agents with an `udpate` entry point (which we call *labile agents*) and those with a `decide` entry point (which we call *active agents*). In what follows for instance, we assume that the scheduler is in charge of maintaining two corresponding lists (Fig. 2).

The management of the scheduling of the activation of the agents is a key choice when designing a simulator. This component of the simulator, called the "scheduler", may, if built without proper thinking, bring great variability in the outcomes [16]. In the present case, two operations have to be taken into account: updating the state of labile agents (through `update`) and asking agents to make a decision (through `decide`).

The scheduler on which we rely here is a discrete-time, fair, sequential scheduler [16] based on the explicit identification of *labile* and *active* agents. It follows this rather simple procedure: at each simulation time t, it triggers the `update` entry point for all labile agents, then it triggers the `decide` entry point for all active agents (Algorithm 1).

Algorithm 1: `run()` : Scheduling of labile and active agents in a simulation

```
for each simulation time t do
    for a ∈ shuffle(labiles) do
        a.update()
    end
    for a ∈ shuffle(actives) do
        a.decide()
    end
end
```

2.2. Analysis method

Since the early 2000s, the number of research works related to MLABS has been growing exponentially. We worked on a large number of concrete simulation cases found in the MAS literature, from various application domains. Thus, a classification and characterization effort was necessary to reduce the complexity of particular cases to a few key criteria, in order to provide a univocal description of actually implemented processes.

We attempted to identify and prioritize relevant issues from an operational point of view for the design of MLABS. This approach necessarily assumes some degree of methodological reductionism: in order to overcome the domain-specific features of the studied simulations, and gain in abstraction, generality and reusability, we were eager to delineate minimal "typical situations". To do so, we focused on every micro/macro relation encountered in the reviewed models, so as to build *minimal schemes* based on two **relative, intertwined micro and macro levels**, the macro level representing an aggregation,

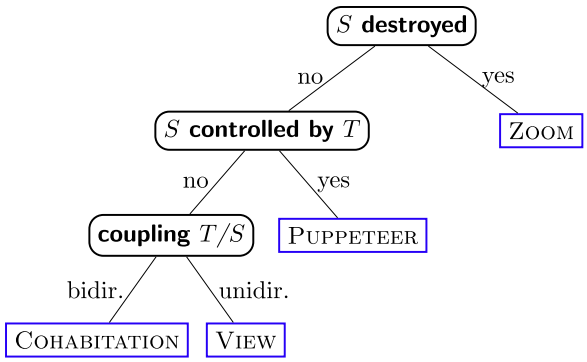

Fig. 3. Decision tree for pattern identification, based on the relationship between a source level S and a target level T.

a composition, a potential group of micro agents (or conversely the micro agents being the product of a disaggregation of the macro agent). These refined situations we obtained can then be characterized by the *nature of the relationship between those two levels*.

The patterns we have identified can be characterized by a decision tree, i.e. a succession of discriminating choices regarding the relationship between micro and macro agents (see Fig. 3). Each pattern is indeed primarily determined by the *direction of level transition*: it can be either an *aggregation* where micro agents create or join a macro agent, or a *disaggregation* producing micro agents from macro agents. This distinction allows us to distinguish between the *source level* and the *target level*. Then, the classification criteria rely upon the relationship between source and target levels, and not upon the direction of level transition itself.

The first issue is whether agents from source level are *destroyed* (which leads directly to the Zoom pattern) or *kept in the simulation* (possibly undergoing some alterations). In the latter case, agents from the source level may be *controlled* by the new agents of the target level (Puppeteer pattern) or, on the contrary, stay *autonomous* from the target level. Finally, the coupling between the two levels can be either *unidirectional*, forcing the target level to merely *reflect* the state of the initial level (View pattern), or *bidirectional*, providing each level with a total behavioural autonomy (Cohabitation pattern).

Of course, these patterns are intended to be *combined* in multi-level simulation applications for each pair of micro/macro agents. In addition, when level changes can occur in both directions, one pattern can be used to perform the aggregation and another one for disaggregation.

3. Pattern description

In this section we provide a systematic description of each of the four patterns mentioned above, according to the following organization: first, we exhibit a (non-exhaustive) set of examples of literature which we consider a representative implementation of the pattern; then, we provide the general definition of the pattern, followed by the algorithms involved in aggregation and disaggregation; finally, we discuss the advantages and limitations of the pattern.

To do so in a univocal way, we hereafter use the following denotations. Micro-level agents are denoted μ_i; the agent of the corresponding macro level, M; $s_t(a)$ denotes the state of the agent a at time t. $\mathcal{S}$ and $\mathbb{S}$ are the respective state spaces of the micro and macro agents, i.e. $\forall \mu_i, s(\mu_i) \in \mathcal{S}$ and $\forall M_j, s(M_j) \in \mathbb{S}$; $\wp(\mathcal{S})$ is the power set of $\mathcal{S}$ (its elements are vectors of micro agents states). Finally, in order to represent the coupling between the micro and macro levels, there must be a function which computes the state of a macro agent from a set of micro agents (in the case of aggregation) or vice versa, to reconstruct n states of micro agents from the state of a macro agent (in the case of disaggregation), respectively:

$$\textbf{\textit{compose}}: \begin{array}{rcl} \wp(\mathcal{S}) & \rightarrow & \mathbb{S} \\ \{s(\mu_1), \ldots, s(\mu_n)\} & \mapsto & s(M) \end{array}$$

$$\text{and } \textbf{\textit{decompose}}: \begin{array}{rcl} \mathbb{S} & \rightarrow & \wp(\mathcal{S}) \\ s(M) & \mapsto & \{s(\mu_1), \ldots, s(\mu_n)\} \end{array}$$

Those two functions are naturally dictated by the application field, and even more specifically by the very nature and modelling hypotheses of the micro and macro agents considered.

Furthermore, the quantity of information corresponding to the state variables of a macroscopic agent is often less than that contained in the state variables of n micro agents that compose it: several individual parameters (e.g. simply the positions) are lost. This is not usually an issue for the aggregation process: on the contrary, it is rather what is expected from it. But, conversely for the disaggregation process, most of the concrete *decompose* functions are not able to affect *all* characteristics of micro agents in a deterministic way from the state of the macro agent. They must then integrate an *assignment policy* for indeterminate variables: for example, through a default value (constant or periodic), a probability law, or statistical information.

Fig. 4. The Zoom pattern.

3.1. The ZOOM pattern

3.1.1. Examples

Traffic modelling often involves three observation levels named "micro" (vehicles), "meso" (group of vehicles) and "macro" (vehicle flow). Micro models are generally used to simulate urban areas while meso and macro models are more adapted to motorway networks. To simulate large-size heterogeneous networks, an efficient approach consists in coupling micro (agent-based) and macro (equation-based) models. This coupling must especially ensure the consistency of simulations [17–23].

The coupling used for the aggregation of individual vehicles is generally destructive. Indeed, micro agents (vehicles) are destroyed when they enter an area managed by a macro agent, which modifies its internal state (characterized by the speed and average flow density) as a function of the density and the speed of micro agents.

Conversely, the dynamic hybrid simulation of traffic flow [24] allows dynamically disaggregating a macro area into a set of micro agents (vehicles). The coupling used for the disaggregation is here once again destructive. When one wishes to dynamically switch between a macro representation and a micro representation in a given area, the associated macro agent in this area is destroyed and disaggregated into a set of micro agents representing vehicles.

Though most of the models of this kind have been developed in the field of road traffic, some other applications, such as crowd simulation, have been published in recent years [25–28].

3.1.2. Definition

Those examples are typical of the Zoom pattern, which corresponds to cases of **destructive aggregation or disaggregation**: the creation of agents at the target level implies the destruction of the original agents in the source level (Fig. 4).

In the case of aggregation, micro agents "dissolve" into the macro agent. The process can be either sudden (all micro agents replaced by a macro agent) or progressive (micro agents being "absorbed" by an existing macro agent), but in both cases the principle is the same. Similarly, in the case of disaggregation, the macro agent is ultimately destroyed after the creation (either gradual or in one step) of micro agents. In the following, we always consider the case of one-step transitions.

3.1.3. Algorithms

The algorithms involved in the Zoom pattern, whether for aggregation (Algorithm 2) or disaggregation (Algorithm 3) are quite simple, because the agents in the source level are destroyed and replaced by agents in the target level. The main issue lies in the consistency between both levels, which is the responsibility of the *compose* and *decompose* functions.

Algorithm 2: Aggregation in the Zoom pattern

create macro agent M
$s(M) \leftarrow compose(\{s(\mu_1), \ldots, s(\mu_n)\})$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
  `population`.remove(μ_i)
  `labiles`.remove(μ_i)
  `actives`.remove(μ_i)
end

3.1.4. Advantages and limitations

This coupling between levels has the advantages of simplicity and extreme computational economy, since only the agents considered relevant to a given observation level are kept in the system.

Algorithm 3: Disaggregation in the Zoom pattern

create n micro agents $\mu_1, \ldots, \mu_n$
$\{s(\mu_1), \ldots, s(\mu_n)\} \leftarrow decompose(s(M))$
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end
`population`.remove(M)
`labiles`.remove(M)
`actives`.remove(M)

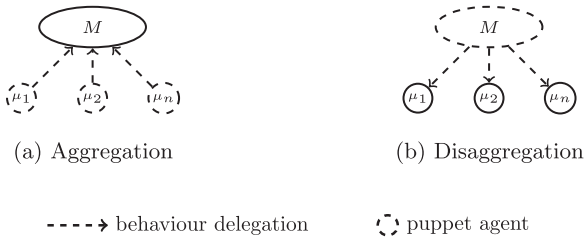

Fig. 5. The Puppeteer pattern.

However, in the aggregation process a rather important quantity of information, namely most of the individual characteristics, is lost. Of course it is always possible to build a *compose* function that would preserve the collection of individual states of the micro level as a dedicated state variable of the macro agent, but it is usually not the case, and that would to some extent counterbalance the computational efficiency with a high memory cost.

Conversely, in the disaggregation process, the *decompose* function has to introduce information to provide fully specified micro state variables from the macro state variables. The designer of the simulation has to make explicit choices, such as using probability laws or statistical information.

3.2. The Puppeteer pattern

3.2.1. Examples

The Rivage project [2] can be considered a precursor of MLABS. In this hydrology project, the micro agents are moving and interacting *waterballs*.

Depending on the similarity of their state variables, they can aggregate into macroscopic agents such as a *ravine* or a *pond*. The corresponding micro agents are not destroyed but their behaviour is frozen and managed by the pond or ravine agents.

In a similar way, the modelling of the impact of PAI-1 molecules on the metastasis of avascular tumours [29] explicitly involves two levels, called "micro" (cells) and "meso" (core of the tumour, composed of millions of necrosed or inactive cells). When a cell agent is identified as inactive or necrosed, it is frozen and its behaviour is delegated to the "meso" agent.

In the Galaxian project [30] (a space battle simulation), *fighter* agents may aggregate to become a *squad* agent, which is in charge of handling individual behaviours of hunters, such as controlling their trajectory and triggering fire orders. When the squad agent decides to disband, its disaggregation makes the fighters return to their original autonomy.

Regarding disaggregation examples, Swarm [31], one of the first explicitly multi-level platforms, is based on a *top-down* design methodology. Thus, the behaviour of a macro agent is defined as the result of the behaviour of micro agents that compose it. However, it is not possible to define feedbacks from macro agents to micro agents.

3.2.2. Definition

The Puppeteer pattern is characterized by the situation where agents from the source level are not destroyed but "frozen", i.e. they keep their individual state, which is still able to evolve according to its own dynamic, but *delegate* their behaviour to the agents of the target level (Fig. 5). Thus, agents of the source level("puppets") *lose the ability to execute their* `decide` *entry point on their own* (they are no longer considered active). On the other hand, their state continues evolving, so their `update` entry point can still be executed by the scheduler (they remain labile).

3.2.3. Algorithms

The algorithms used for the Puppeteer pattern, whether for aggregation (Algorithm 4) or disaggregation (Algorithm 5), are built to ensure that the simulation scheduler no longer considers source level agents as *active* agents (in order to

Algorithm 4: Aggregation in the PUPPETEER pattern

create macro agent M
$s(M) \leftarrow compose(\{s(\mu_1), \ldots, s(\mu_n)\})$
M.puppets $\leftarrow \{\mu_1, \ldots, \mu_n\}$
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
  actives.remove(μ_i)
end
population.add(M)
labiles.add(M)
actives.add(M)

Algorithm 5: Disaggregation in the PUPPETEER pattern

create n micro agents $\mu_1, \ldots, \mu_n$
$\{s(\mu_1), \ldots, s(\mu_n)\} \leftarrow decompose(s(M))$
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
  μ_i.puppets $\leftarrow \{M\}$
  population.add(μ_i)
  labiles.add(μ_i)
  actives.add(μ_i)
end
actives.remove(M)

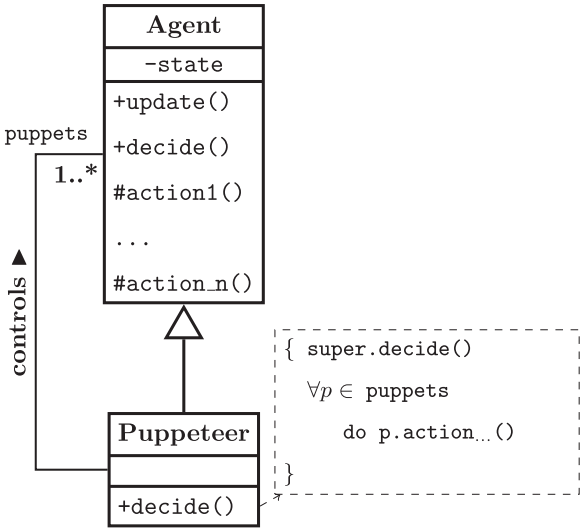

Fig. 6. Modelling of a PUPPETEER agent able to control the actions of its "puppet" agents.

disable their decide entry point). Besides, in order to realize behavioural delegation, the agents of the target level must be specialized (Fig. 6): in addition to their own behaviour, they are responsible for the activation of some (domain-dependant) actions of their puppets.

3.2.4. Advantages and limitations

The PUPPETEER pattern allows explicitly incorporating the target level entities in the model. It also simulates efficiently systems composed of a large number of agents. Unlike the ZOOM pattern, it is non-destructive and therefore avoids loss of information. Moreover, when the puppeteer agents are destroyed, the original autonomy of its puppet agents can easily be restored.

However, in the case of the aggregation process, this pattern is easy to implement only if the behaviours of aggregated agents are similar and the collective behaviour simple. Thus, in the above examples, aggregated agents are either inactive (they do not move and their state does not change) or characterized by a collective dynamics which can be easily controlled by the macro level (e.g. linear motion).

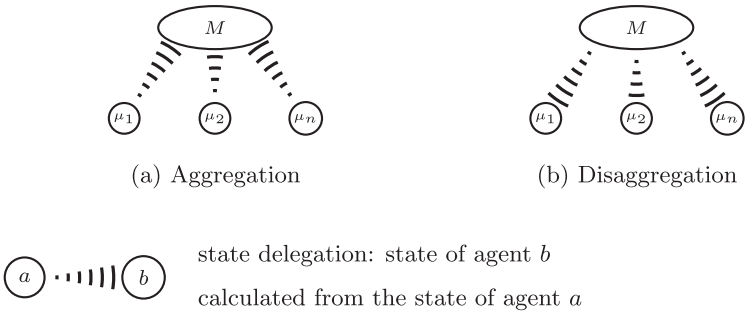

Fig. 7. The VIEW pattern.

Conversely, in the case of the disaggregation process, an extensive analysis of works using SWARM [31] suggests that this approach has only been used at the technical level to separate the design of observation tools, the scheduling of different types of agents and the behaviours of agents. Yet the possibility of designing complex hierarchical models through this approach remained unexploited.

Thus disaggregation cases using the PUPPETEER pattern appear to be confined to relatively simple systems, endowed with an ascending causality chain, and which can be decomposed analytically in autonomous subsystems. This pattern is therefore not really suitable for the modelling of highly complex systems (e.g. biological or social ones), which are characterized by bidirectional interactions or coupling between levels. The COHABITATION pattern (see Section 3.4) will be more suitable in those cases.

3.3. The VIEW pattern

3.3.1. Examples

The design of relevant observation or representation tools is an important issue in multi-agent simulation, since these tools are often the only way to provide the user with a visual opportunity to detect emergent phenomena or structures. Most of the views proposed in multi-agent simulations to detect the emergence of a collective phenomenon are often limited to the visualization of agents characteristics, possibly grouped by family, by population or mapped on to the environment.

Thus, emergent phenomena are often neither reified nor agentified, i.e. they are not explicitly represented as entities in the simulation, and therefore have no autonomy or feedback capacity.

To remedy this problem, the agents involved in a collective phenomenon may be represented by a "View" agent endowed with a behaviour of its own, reflecting micro agents. This approach was used to agentify or simply to reify emerging phenomena in fluids flow simulations [32], flocking [33] or social simulations [34].

Similarly, interactive hybrid simulations of traffic flow [35] give the user the ability to "magnify" a macroscopic area and thus generate by disaggregation the corresponding microscopic representation composed of vehicles in interaction. These "View" agents are *autonomous* with respect to the visualized macro agent: their behaviour is completely defined by behavioural models (acceleration, lane change) specific to them.

3.3.2. Definition

VIEW agents are intended to reify, in the target level, a representation of agents from the source level. Therefore, their state must be computed to reflect at every moment the state of the agents they emanate from. They are *labile*, but their `update` entry point must be defined as the *composition* or the *decomposition* of the states of the agents they represent. In the general case, they also have the ability to perform `decide`, as long as their state is not affected by their actions (thus without any possible impact on the state of the agents they represent), *e.g.*, to process data that will be displayed to the user. Information exchange (the coupling between levels) is in this case clearly unidirectional, from the agent(s) visualized (level source) to the view agent(s) (target level). The autonomy of visualized agents is not changed. We propose to describe this situation as a form of *state delegation* (Fig. 7).

3.3.3. Algorithms

Many algorithms have been proposed to detect emergent phenomena. For instance, Gil-Quijano used classification methods (self-organizing maps, K-means, particle swarm) to detect group formation in social simulation [36]. Chen proposed a language for describing in an abstract way emergent phenomena, characterized by multi-level dynamics [37]. Our purpose here is not to deal with the large number of solutions proposed to detect group emergence (for a comprehensive overview, see [38]). Instead, the algorithms used for the pattern VIEW, whether for aggregation (Algorithm 6) or disaggregation (Algorithm 7), are primarily intended to ensure "state delegation". Therefore, agents of the target level must be specialized (Fig. 8) to be able to redefine `update` as the composition or decomposition of the agent states from the source level.

Algorithm 6: Aggregation in the VIEW pattern

create macro agent M
M.observed $\leftarrow \{\mu_1, \ldots, \mu_n\}$
redefine M.update()=$\{s(M) \leftarrow compose(\{s(\mu_1), \ldots, s(\mu_n)\})\}$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)

Algorithm 7: Disaggregation in the VIEW pattern

create n micro agents $\mu_1, \ldots, \mu_n$
$\{s(\mu_1), \ldots, s(\mu_n)\} \leftarrow decompose(s(M))$
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
  μ_i.observed $\leftarrow \{M\}$
  redefine μ_i.update()=$\{s(\mu_i) \leftarrow decompose(s(M))[i]\}$
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end

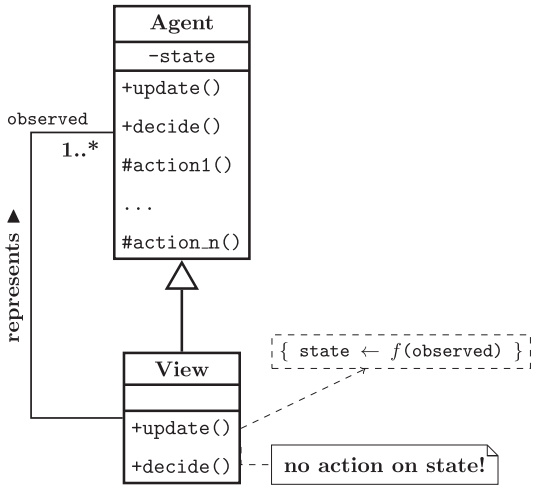

Fig. 8. Modelling of a VIEW agent whose state is computed from the agents it represents. The *f* function to use is either *compose* (aggregation), or *decompose* (disaggregation).

3.3.4. Advantages and limitations

The VIEW pattern provides the observer with relevant representations of collective phenomena, which take part in the simulation. Furthermore, since the first agentification works which were tightly linked to application domains, generic toolboxes appear gradually, like SIMANALYZER [7], allowing a wider usage.

Let us note that the majority of approaches implementing the VIEW pattern for aggregation are only intended to reify detected phenomena, without agentifying them fully: in other words, these "views" objects have an `update` entry point but no `decide`, and therefore are not considered active by the scheduler. The use of this pattern is obviously all the more simple when the *View* agent has no specific behaviours.

Otherwise, the fact that the coupling between levels is unidirectional may raise consistency issues. To overcome this problem, the possible bidirectionality of information exchange has to be taken into account, which can be done through the COHABITATION pattern (presented in next section), which is more complicated to implement and more expensive computationally.

In the case of disaggregation, it is necessary to introduce additional information into the *View* agents, as in the ZOOM pattern. However, it is less problematic here because it does not affect the soundness of simulation results, but only the realism of their visualization.

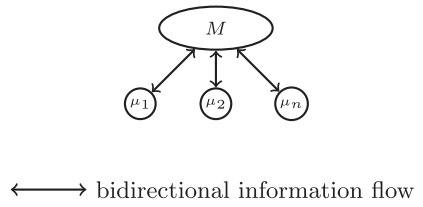

Fig. 9. The COHABITATION pattern.

3.4. COHABITATION

3.4.1. Examples

In the simulation of the development of diptera larvae [39], detected aggregates of micro agents are dynamically agentified, in order to predict correctly the temperature generated by interactions between insects, and therefore accurately simulate their development. The resulting macro agents act autonomously with respect to the micro agents. They may split, merge or disappear. They feed back the temperature felt by micro agents to them, according to the size and density of the aggregate.

Similarly, in the hybrid simulations of traffic flows (described previously, see Section 3.1.1), disaggregation is realized by generating micro agents based on the characteristics of the macro agent upstream and on the micro area downstream. There is in this case, unlike the aggregation process of the same simulation, a bidirectional information exchange between micro and macro levels. Indeed, macro agent generating micro agents must take into account the state of downstream traffic. For example, a congestion reduces the initial velocity of the generated vehicles. Similarly, if the rate of road occupancy is maximal, vehicles cannot be generated, the state (speed, flow, density) of the macro agent is altered.

3.4.2. Definition

The COHABITATION pattern represents the most complex case encountered in MLABS. Indeed, agents from the source and target levels can run `update` and `decide`. The interactions between micro and macro levels are bidirectional, i.e. they influence each other (Fig. 9). Thus, this pattern is particularly used in simulations of very complex systems such as biological systems, e.g. in many works on the multi-level modelling of tumour development [5,6,40,41].

3.4.3. Algorithms

As for the VIEW pattern, emergent phenomena detection algorithms are necessary, but they are usually specific to the application domain. The operations to perform for the aggregation (Algorithm 8) or disaggregation (Algorithm 9) are in

Algorithm 8: Aggregation in the COHABITATION pattern

create macro agent M
$s(M) \leftarrow compose(\{s(\mu_1), \ldots, s(\mu_n)\})$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)

Algorithm 9: Disaggregation in the COHABITATION pattern

create n micro agents $\mu_1, \ldots, \mu_n$
$\{s(\mu_1), \ldots, s(\mu_n)\} \leftarrow decompose(s(M))$
for $\mu_i \in \{\mu_1, \ldots, \mu_n\}$ **do**
    `population`.add(μ_i)
    `labiles`.add(μ_i)
    `actives`.add(μ_i)
end

this case rather simple since they simply involve adding agents to the target level simulation. All agents are considered labile and active. The main challenge is therefore, in each case, to appropriately define the domain-dependent *behaviours* of the agents each level must have.

Table 1
Patterns used in different types of hybrid simulation.

	$\mu \rightarrow M$	$M \rightarrow \mu$
Classical [17–21]	Zoom	Cohabitation
Dynamic [24]	Zoom	Cohabitation+Zoom
Interactive [35]	∅	View

3.4.4. Advantages and limitations

The Cohabitation pattern allows the integration in the same simulation of agents belonging to different observation levels or different organizations, by handling them in a homogeneous way. Whether the agents are "atomic", or the result of an aggregation process or a disaggregation process, they are all managed by the scheduler in the same way. Their states evolve following a dynamics of their own, and they may be provided with arbitrarily complex behaviours. The model can then be as accurate as needed.

However, besides the classic issue of emergence detection, it is necessary to provide all the agents with appropriate and consistent behaviours. Interactions can take place between agents from a micro/macro couple, but also between agents from each of these levels and completely different agents. Thus, they are more difficult to design, at least because of the number of combinations involved.

Furthermore, the computational cost associated with the preservation of fully active micro agents can be high. Therefore, we suggest avoiding this pattern if the goal is to save computation time.

3.5. Pattern composition

It may be necessary to compose different patterns in a single model. These patterns can be different depending on whether we aggregate or disaggregate. For instance, in the case of a "classical" hybrid traffic simulation, i.e. in which micro and macro area are static and defined *a priori*, the Zoom pattern is used for aggregation while the Cohabitation pattern is used for disaggregation.

Moreover, in the same field of application, it can also be necessary to integrate multiple patterns to handle different aggregation and disaggregation cases according to the objective of the simulation (Table 1). In the case of a dynamic hybrid simulation, the Zoom pattern is also used for disaggregation (in addition to the Cohabitation pattern) when the representation associated to an area is modified at runtime [24]. In the case of an interactive hybrid simulation, the View pattern is used for disaggregation [35].

4. The patterns in practice

In the last section, we mainly tried to highlight the existence of reccurrent abstract structures and relations underlying existing multi-level agent-based simulations. In the following we present the complementary approach, consisting in using the patterns defined above to *design* new multi-level simulation systems according to specific functional needs.

A first step was achieved recently in the field of computational epidemiology with the design and implementation of EMuLSion (Epidemiological MUlti-Level Simula- tION framework) [42,43]. This framework aims at capturing existing modelling paradims in epidemiology, involving several aggregation levels (e.g. individual-based to compartmental models) and several scales (from individuals to metapopulations at a regional scale).

The four patterns were used to design the set of agent classes required for answering this diversity of domain-related needs (Fig. 10), for instance:

- Zoom pattern: `Compartment` agents are used for representing aggregated populations (amounts) when individual differences amongst micro agents are not relevant,
- View pattern: `EvolvingAtom` agents which represent fully autonomous individuals (micro agents) can be stored inside an `AdaptiveView` agent (macro level), which monitors the similarity of agent states with respect to a specific concern, e.g. their health state; when the state of an `EvolvingAtom` is modified as a consequence of its behaviour, the `AdaptiveView` detects this changes and rejects this agent (as specified in the pattern, this action does not affect the state of the micro agent itself),
- Puppeteer pattern: a `GroupManager` agent is actually a Puppeteer, since it is endowed with a specific behaviour (e.g., the management of an infectious process) and in charge of controlling the actions of lower-level agents (micro-level), which in EMuLSion are other kinds of groups (instances of `GroupAgent` subclasses): for instance, when the infectious process run by the `GroupManager` leads to the contamination of N individuals, the `GroupManager` requires the micro-agent holding "healthy" animals to "transform" N of them into "infectious" animals, which are moved to the group of sick animals (regardless of the nature of these N individuals: true agents or only amounts),
- Cohabitation pattern: the `MultiProcessManager` agent is used in EMuLSion to represent at the same time herds and populations at a regional level: both are fully autonomous, since herds are subject to local processes (demography, infection, herd management decisions), while populations provide them with a feedback (e.g. public health policies, macroeconomic processes).

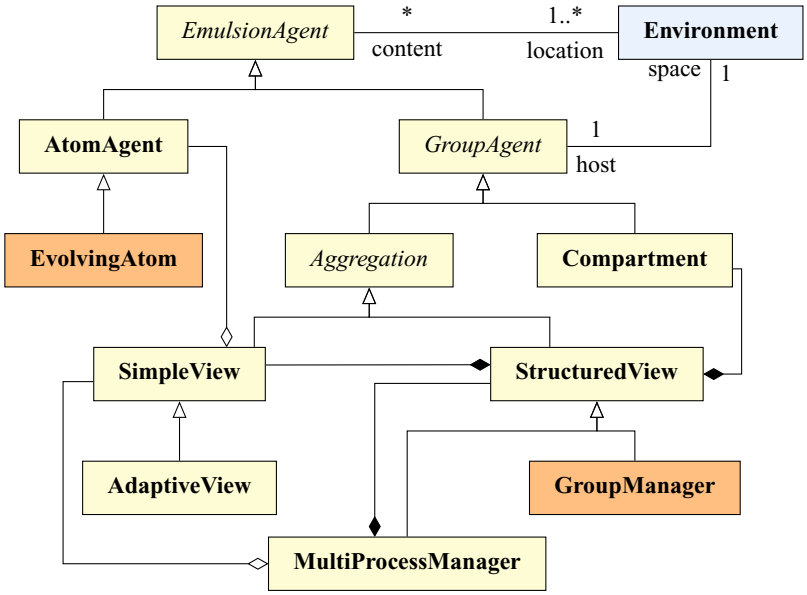

Fig. 10. Overview of the multi-level multi-agent classes used in the Emulsion framework, based on the multi-level patterns described in this article [42].

Table 2
Action/evolution capacities of agents in source and target levels, in each pattern.

Pattern	Source level	Target level
ZOOM	*destroyed*	`update` `decide`
PUPPET.	`update`	`update` extended `decide`
VIEW	`update` `decide`	simplified `update` restricted `decide`
COHAB.	`update` `decide`	`update` `decide`

The relevance of such an architecture was assessed by studying the dynamics of a zoonotic disease (Q Fever) at several scales (within-herd and between-herd models), and proved an efficient way to switch easily from one level of interest to another [44].

5. Conclusion and perspectives

In the previous sections, we have shown that relations between agents in a MLABS can be characterized by a combination of patterns, based on the possible couplings between a "source" level and a "target" level, either in the micro-to-macro direction (called "aggregation"), or in the macro-to-micro direction (called "disaggregation").

The criteria underlying these patterns reflect the diversity of the simulations found in the literature, but also issues that any model designer has to tackle. Besides, these key questions are a hinge between the model and its implementation.

The four patterns that result from this classification correspond to minimal, abstract, domain-free situations, and can therefore be applied generically. They are characterized by the capacity of agents of the source and target levels to execute or not the `update` and `decide` entry points (Table 2), regardless of the direction of the level change.

Yet, our approach is still empirical, as was the original work of pattern identification in Software Engineering. We do not aim at a normative or exhaustive description of multi-level multi-agent simulations. Rather, software engineering patterns are intended to offer efficient solutions in design or implementation issues, and provide a common abstraction among developers. Thus, they can benefit and extend from new proposals.

We are also well aware that the major difficulties which arise in the design of multi-level models is the *functional specification* in the sense of all behaviours specific to each field, in particular the exact nature of couplings that occur between each level.

For instance, the proper decision whether to destroy incoming vehicles or to make them coexist with a "traffic jam" macroscopic agent, is the responsibility of the field expert and depends on the very nature of the behaviours to model. However, the computer scientist may exert an advisory role in this process, for example, advocating the Puppeteer or View

Table 3
Relations between the patterns and the functional goals of MLABS.

	Zoom	Puppeteer	View	Cohabitation
Aggregation: save computation time	×	×		
Disaggregation: gain in precision	×	×		×
Add abstraction levels		×	×	×
Model coupling				×
Visualization	×		×	

patterns for efficiency considerations, or implementing a Cohabitation pattern to enhance the explicit understanding of the coupling between levels.

In this context, the purpose of our contribution is to help the model designer to clarify the most *technical* aspects of the multi-level simulation process, namely those of the micro-macro mechanisms which are not specific to an application domain. Each pattern brings its own advantages and limitations, regarding either its capacity to model some situations better than others, or its implementation simplicity or efficiency.

The patterns classification and characterization work that we conducted here now calls for *methodological developments*. Given the empirical nature of our approach, it is probably premature to propose a turnkey methodology from analysis to implementation for those four patterns. Nevertheless, we can suggest two main outlines.

First, the designer may have a fairly clear idea of the *relationships* that should exist between levels during aggregation or disaggregation: destruction of source level agents, supervision of the target level, etc. In this case, the decision tree that we have defined (Fig. 3) is a good way to identify the best suited pattern.

However, it seems that, in most cases, the identification of the most relevant pattern for a given situation can only come from the *advantages and limitations* of this pattern, according to the constraints on the agents to model: computational efficiency, capability to extend the model easily, compatibility with other patterns already defined between one level and another group of agents, etc.

In this context, it is interesting to note that the structural characteristics of the patterns coined in Fig. 3 meet the functional goals of MLABS: introducing "relevant" abstraction levels in the model, dynamically adaptating the level of detail of the simulation to save computation time or gain in precision, coupling heterogeneous models or visualize interesting collective or individual behaviours (Table 3).

It is possible that some situations do not fit completely our analysis grid, perhaps only because simplifying assumptions were introduced in models or shortcuts used during implementation. Among the four patterns we have described, the richest from a behavioural point of view is obviously the Cohabitation pattern, which already resulted in formal meta-models [12,13] and in the identification of *behavioural patterns* [45].

Ongoing work is now focusing on addressing the methodological issues on the one hand, and on systematizing the identification of *behavioural patterns* on the other hand.

In this article, we focused on models composed of two **relative micro and macro levels**. When a model has more than two levels, the problem of the consistency of the composition of the patterns arises. Therefore, a major perspective of this work is to define a meta-pattern, i.e., a pattern for composing patterns.

References

[1] H. Parunak, Between agents and mean fields, in: Multi-Agent-Based Simulation XII, in: LNCS, 7124, Springer, 2012, pp. 113–126.
[2] D. Servat, E. Perrier, J.-P. Treuil, A. Drogoul, When agents emerge from agents: introducing multi-scale viewpoints in multi-agent simulations, in: Multi-Agent Systems and Agent-Based Simulation, in: LNCS, 1534, Springer, 1998, pp. 183–198.
[3] L. Navarro, F. Flacher, V. Corruble, Dynamic level of detail for large scale agent-based urban simulations, in: 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'11), 2011, pp. 701–708.
[4] A. Hjorth, D. Weintrop, B.C.U. Wilensky, Levelspace: constructing models and explanations across levels, Constructionism, 2016.
[5] Z. Wang, J.-D. Butner, R. Kerketta, V. Cristini, T. Deisboeck, Simulating cancer growth with multiscale agent-based modeling, Semin. Cancer Biol. 30 (2015) 70–78.
[6] L. Zhang, Z. Wang, J. Sagotsky, T. Deisboeck, Multiscale agent-based cancer modeling, J. Math. Biol. 48 (4–5) (2009) 545–559.
[7] P. Caillou, J. Gil-Quijano, Simanalyzer: automated description of groups dynamics in agent-based simulations, in: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'12), 2012, pp. 1353–1354.
[8] J. Gil-Quijano, T. Louail, G. Hutzler, From biological to urban cells: lessons from three multilevel agent-based models, in: Principles and Practice of Multi-Agent Systems, in: LNCS, 7057, Springer, 2012, pp. 620–635.
[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, 1994.
[10] P. Mathieu, S. Picault, Y. Secq, Design patterns for environments in multi-agent simulations, in: 18th Conf. on Principles and Practice of Multi-Agent Systems (PRIMA'15), in: LNCS, 9387, Springer, 2015, pp. 678–686.
[11] J. Ferber, F. Michel, J. Báez, AGRE: integrating environments with organizations, in: D. Weyns, et al. (Eds.), Proc. 1st Int. Workshop Environments for Multi-Agent Systems (E4MAS), LNCS, 3374, Springer, 2005, pp. 48–56.
[12] S. Picault, P. Mathieu, An interaction-oriented model for multi-scale simulation, in: 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), AAAI, 2011, pp. 332–337.
[13] G. Morvan, A. Veremme, D. Dupont, IRM4MLS: the influence reaction model for multi-level simulation, in: Multi-Agent-Based Simulation XI, in: LNCS, 6532, Springer, 2011, pp. 16–27.
[14] Y. Kubera, P. Mathieu, S. Picault, Everything can be agent!, in: 9th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'10), IFAAMAS, 2010, pp. 1547–1548.
[15] Y. Kubera, P. Mathieu, S. Picault, IODA: an interaction-oriented approach for multi-agent based simulations, Auton. Agents Multi-Agent Syst. 23 (3) (2011) 303–343.

[16] P. Mathieu, Y. Secq, Environment updating and agent scheduling policies in agent-based simulators, in: 4th Int. Conf. on Agents and Artificial Intelligence (ICAART), 2012, pp. 170–175.
[17] E. Bourrel, J. Lesort, Mixing micro and macro representations of traffic flow: a hybrid model based on the LWR theory, 82nd Annual Meeting of the Transportation Research Board, 2003.
[18] S. Espié, D. Gattuso, F. Galante, Hybrid traffic model coupling macro- and behavioral microsimulation, 85th Annual Meeting of Transportation Research Board, 2006.
[19] M. El hmam, H. Abouaissa, D. Jolly, A. Benasser, Macro-micro simulation of traffic flow, in: Proc. of the 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM06), 2006, pp. 351–356.
[20] L. Magne, S. Rabut, J. Gabard, Towards an hybrid macro-micro traffic flow simulation model, in: Proc. INFORMS Salt Lake City String 2000 Conf., 2000.
[21] A. Poschinger, R. Kates, H. Keller, Coupling of concurrent macroscopic and microscopic traffic flow models using hybrid stochastic and deterministic disaggregation, Transportation and Traffic Theory for the 21st Century, 2002.
[22] I. Tchappi Haman, V. Kamla, S. Galland, J. Kamgang, Towards a multilevel agent-based model for traffic simulation, Procedia Comput. Sci. 109 (2017) 887–892.
[23] M. Adnan, F. Pereira, C.M. Lima A., K. Basak, M. Lovric, S. Raveau, Y. Zhu, J. Ferreira, C. Zegras, M. Ben-Akiva, Simmobility: a multi-scale integrated agent-based simulation platform, Transportation Research Board 95th Annual Meeting, 2016.
[24] N. Bouha, G. Morvan, H. Abouaïssa, Y. Kubera, A first step towards dynamic hybrid traffic modeling, in: 29th European Conf. on Modelling and Simulation (ECMS'15), 2015, pp. 64–70.
[25] L. Crociani, G. Lämmel, G. Vizzari, Multi-scale simulation for crowd management: a case study in an urban scenario, in: AAMAS 2016 Workshops, Best Papers, in: LNCS, 10002, Springer, 2016, pp. 147–162.
[26] N. Gaud, S. Galland, F. Gechter, V. Hilaire, A. Koukam, Holonic multilevel simulation of complex systems: application to real-time pedestrians simulation in virtual urban environment, Simul. Modell. Pract. Theory 16 (2008) 1659–1676.
[27] A. Kiselev, V. Karbovskii, S. Kovalchuk, Agent-based modelling using ensemble approach with spatial and temporal composition, Procedia Comput. Sci. 80 (2016) 530–541.
[28] T. Nguyen, J.-D. Zucker, D. Nguyen, A. Drogoul, A. Vo, A hybrid macro-micro pedestrians evacuation model to speed up simulation in road networks, in: Advanced Agent Technology, in: LNCS, 7068, Springer, 2012, pp. 371–383.
[29] J. Lepagnot, G. Hutzler, A multiscale agent-based model for the simulation of avascular tumour growth, J. Biol. Phys. Chem. 9 (1) (2009) 17–25.
[30] P. Mathieu, S. Picault, The Galaxian project: a 3D interaction-based animation engine, in: Advances on Practical Applications of Agents and Multi-Agent Systems, Springer, 2013, pp. 312–315.
[31] N. Minar, R. Burkhart, C. Langton, M. Askenazi, The SWARM simulation system: a toolkit for building multi-agent simulations, Technical Report 96-06-042, Santa Fe Institute, 1996.
[32] P. Tranouez, C. Bertelle, D. Olivier, Emergent Properties in Natural and Artificial Dynamical Systems, Springer, pp. 87–99.
[33] T. Moncion, P. Amar, G. Hutzler, Automatic characterization of emergent phenomena in complex systems, J. Biol. Phys. Chem. 10 (2010) 16–23.
[34] G. Prévost, C. Bertelle, Detection and reification of emerging dynamical ecosystems from interaction networks, in: Complex Systems and Self-organization Modelling, in: Understanding Complex Systems, 39, Springer, 2009, pp. 139–161.
[35] J. Sewall, D. Wilkie, M. Lin, Interactive hybrid simulation of large-scale traffic, ACM Trans. Graphics (SIGGRAPH Asia) 30 (6) (2011).
[36] J. Gil-Quijano, M. Piron, A. Drogoul, Social Simulation: Technologies, Advances and New Discoveries, IGI Global, pp. 151–168.
[37] C. Chen, Complex event types for agent-based simulation, University College London, 2009 Ph.D. thesis.
[38] G. Morvan, Multi-level agent-based modeling – a literature survey, CoRR (2013). abs/1205.0561
[39] G. Morvan, D. Jolly, A. Veremme, D. Dupont, D. Charabidze, Vers une méthode de modélisation multi-niveaux, in: 7e Conf. de Modélisation et Simulation MOSIM, 1, 2008, pp. 167–174.
[40] J. Butner, V. Cristini, Z. Wang, Development of a three dimensional, multiscale agent-based model of ductal carcinoma in situ, in: 2017 39th Annual Int. Conf. of the IEEE Proc. of Engineering in Medicine and Biology Society (EMBC), 2017, pp. 86–89.
[41] Z. Wang, P. Maini, Editorial special section on multiscale cancer modeling, IEEE Trans. Biomed. Eng. 64 (3) (2017) 501–503.
[42] S. Picault, Y.-L. Huang, V. Sicard, F. Beaudeau, P. Ezanno, A multi-level multi-agent simulation framework in animal epidemiology, in: International Conference on Practical Applications of Agents and Multi-Agent Systems, in: LNCS, 10349, Springer, 2017, pp. 209–221.
[43] S. Picault, Y.-L. Huang, V. Sicard, P. Ezanno, Enhancing sustainability of complex epidemiological models through a generic multilevel agent-based approach, in: Proc. 26th Int. Joint Conf. on Artificial Intelligence (IJCAI), AAAI, 2017, pp. 374–380.
[44] S. Picault, Y.-L. Huang, V. Sicard, T. Hoch, E. Vergu, F. Beaudeau, P. Ezanno, A generic multi-level modelling and simulation approach in computational epidemiology, J. R. Soc. Interface (2017). submitted
[45] A. Maudet, G. Touya, C. Duchêne, S. Picault, Patterns multi-niveaux pour les SMA, in: 23e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2015), Cépaduès, 2015, pp. 19–28.