An Interaction-Oriented Model for Multi-Scale Simulation

Sébastien Picault, Philippe Mathieu

An Interaction-Oriented Model for Multi-Scale Simulation

Sébastien Picault and **Philippe Mathieu**
University Lille 1
LIFL UMR CNRS 8022
Villeneuve d'Ascq, France
{sebastien.picault, philippe.mathieu}@lifl.fr

Abstract

The design of multiagent simulations devoted to complex systems, addresses the issue of modeling behaviors that are involved at different space, time, behavior scales, each one being relevant so as to represent a feature of the phenomenon. We propose here a generic formalism intended to represent multiple environments, endowed with their own spatiotemporal scales and with behavioral rules for the agents they contain. An environment can be nested inside any agent, which itself is situated in one or more environments. This leads to a lattice decomposition of the global system, which appears to be necessary for an accurate design of multi-scale systems. This uniform representation of entities and behaviors at each abstraction level relies upon an interaction-oriented approach for the design of agent simulations, which clearly separates agents from interactions, from the modeling to the code. We also explain the implementation of our formalism within an existing interaction-based platform.

1 Introduction

Multi-agent simulation has long proved its adequacy to deal with complex systems. The interest of the agent-based approach consists in making concepts, that underlie domain models, concrete – as well as the behavior of those entities. The mapping is not always easy to implement without biases, but it makes an essential contribution to the mutual understanding between domain experts (biologists, economists, sociologists ...) and computer scientists.

Especially, in the context of large-scale simulations (involving large numbers of entities, belonging to many families and exhibiting various behaviors), a major issue is the *coexistence of different scales* [An, 2008]: for instance, the simulation of gene regulation networks involves, at least, the cell level, together with nested organites such as the nucleus, the cytoplasm, and often molecular details such as the transcription factors, the DNA strand, etc. Similarly, the study of client behaviors in commercial places requires to model simultaneously the way people behave in the mall, in the shops, in the shelves, etc.

Approaches such as multi-model simulations offer different paradigms, depending on the level of observation, to represent multi-level phenomena: e.g., they can combine an agent-based simulation at the macroscopic level and a differential equation model at the microscopic level. The disadvantage of these methods is primarily the same as non-agent models, namely the lack of shared representations with experts regarding entities and behaviors. In addition, there is an increased risk of bias by coupling models of different kinds.

We advocate instead a uniform approach in which only *agents* and *behaviors* are used, at each simulation level. Thus, the model we describe in this paper extends a uniform simulation approach, called "IODA" [Kubera *et al.*, 2008; 2011]. IODA is an "interaction-oriented" approach which is based on three key ideas:

- *Each relevant entity is modeled through an agent* [Kubera *et al.*, 2010].
- *Each behavior is a generic interaction* which is defined independently from agents, as a condition/action rule, and implemented through a specific piece of code.
- When an agent, that is able to perform an interaction (a "*source* agent"), encounters another agent that can undergo the same interaction (a "*target* agent"), then this interaction can occur if both source and target agents fulfill the conditions of the interaction.

In this paper, we try to formalize the relationships between the agents and several environments, in order to propose a flexible and unified representation of multiple space and time scales in an agent-based simulation. In addition, we also describe how to take advantage of the interaction-oriented approch, to easily model different patterns of agent behaviors in each simulation level.

In the first section, we examine several models, architectures or simulations that have been proposed to manage either hierarchical structures, or the attachment of an agent to multiple spaces or social groups. Then, we present our formal model, called "PADAWAN", for representing encapsulation and situation relations between agents and environments. This model tries to overcome the limitations of existing approaches and to generalize them. We show that the interaction-oriented approach of simulation facilitates the design of behaviors bound to each observation level. Finally, we explain how this model is implemented within an existing interaction-based platform.

2 Current multi-scale simulation approaches

The simulation of complex systems must take into account all *organization levels* that are considered relevant to the study of a given phenomenon. Thus, many architectures have been long proposed to represent *nested* systems (which usually leads to nested agents or nested environments). More recently, this mere problem of encapsulation recedes, in favor of the issue of agents *multi-membership* (in different groups or areas).

2.1 A hierarchical platform: SWARM

Among many platforms, SWARM [Minar *et al.*, 1996] is one of the first that explicitly addressed the issue of handling multiple space or time scales in individual-based simulations. It is based on a recursive organization of the system, which is composed of *swarms*, that are spaces containing agents, endowed with a scheduler for the actions of those agents: this scheduler is a way to model the time scale attached to the swarm; and *agents* situated in the swarms, and that can themselves *be swarms* (i.e. contain other agents with a scheduler). This approach allows to easily replace a given modeling level by the underlying subsystem, and so ensure that the behavior of an agent is directly produced by the collective behavior of the corresponding swarm.

However, SWARM suffers from severe limitations: it is a strictly hierarchical model with a fixed structure. In additions, it is a mere platform, that does not provide any conceptual framework nor guidelines for the model design. Finally, the fact that a swarm agent has no other behavior than the result of its components is limiting, since behaviors can be often specified at different observation levels, including both the swarm level and the underlying agents level.

2.2 Physical vs. social environments: AGRE

The AGRE approach (Agents, Groups, Roles + Environment) [Ferber *et al.*, 2005] combines conceptual, abstract tools with an operational model. It especially draws a parallel between the physical environment and the social environments (groups and organizations) to which an agent can take part. Thus, the multiagent *world* (organization + physical world) appears as a composition of *spaces* (groups and areas), which are themselves composed of *modes* (roles and bodies) played by the agents.

Thus, in AGRE, agents can interact, both through their body, when they are situated in the same part of the physical space, and through their roles, when they belong to a same group. This emphasizes that, in general, *an agent is situated in several environments* (i.e. physical and social here)

Yet, in AGRE, the body (which is *the visible part of an agent in a physical environment*) is assumed unique. Though it could be argued that this viewpoint is quite intuitive, it actually restricts what the approach allows to represent. In addition, the distinction between social and physical worlds comes from the initial focus, in the earlier model (AGR), on the organization issue in MAS. But, we could also consider mental worlds such as the memory of an agent, or its predictions (a mental simulation), etc. Moreover, AGRE also restricts the structure of the physical world, for instance the nested levels are limited to the world/area/body decomposition.

2.3 Other approaches

In order to cope with the conceptual and computational complexity of large-scale systems, some approaches such as HLA [Fujimoto, 1998] rely on the integration of multiple individual-based simulations. This raises sometimes specific difficulties in order to ensure interoperability between modules or a consistent simulation time, or to solve conflicts in concurrent access to variables [Scerri *et al.*, 2010]. Besides, the lack of uniformity (in agent models, behavior description...) is likely to introduce biases for the same reasons than multi-model simulations.

Besides agent-based approaches, some fruitful features can be drawn from computational models, though they are not especially devoted to simulation. For instance, P-systems [Păun and Rozenberg, 2002] rely on three bio-inspired items: membranes (organized in a tree structure) containing both symbols and rules which determine how to transform symbols and the local membrane structure. Thus, the symbols have no specific behavior by themselves: only the rules of the same membrane can determine their function. That means, on the one hand, that *a clear separation is made between entities and behaviors*, and on the other hand, that *behaviors are defined in a local context*.

2.4 Specifications for a multi-scale model

The comparison of above approaches allows us to sketch the ideal specification of a general approach to multi-scale/multi-level simulation, through the following features:

- **Dynamicity:** An agent must be able to change the level where it operates (i.e. in practice, change its environment) at will; in addition, there should be no limiting principle to the creation or dissolution of any level.
- **Locality:** The behavior of an agent must be the result of its presence in a given environment (i.e. the space that defines its condition of existence), which is itself characterized by a spatial scale and a time scale.
- **Uniformity:** 1) All entities must have a similar software structure (that is, any relevant entity must be an agent) [Kubera *et al.*, 2010] ; 2) All behaviors must be described through the same formalism. 3) The decomposition of the simulation world into environments must also be uniform.
- **Genericity:** The formalism, which describes knowledge about the entities and their behavior in each environment, as well as knowledge about the relationship between levels, has to be as accessible as possible to domain experts, and in various application fields. But, it must also be easily implementable in a generic simulation engine.
- **Transversal approach:** The components of the formal model, that are used during the design phase, should

have a software reality, and this transformation should be as automatic as possible (especially in order to reduce the risk of biases).

The following sections try to apply those principles by defining relations between agents and environments, and then by completing a consistent implementation of the concepts.

3 Formal principles for a multi-scale simulation: the PADAWAN model

In order to fulfill the principles above, the model we advocate in this paper, called "PADAWAN[1]", is based on two principles:

1. *Any agent may dynamically encapsulate an environment.* This is the basis of a recursive nested structure (such as those used in SWARM or P-systems), but this structure must be able to change in time.
2. *Any agent may be situated in several environments at the same time*, without a prior idea of what those environments represent (a micro/macro level of the physical world, a group, an organization, a spatial memory, a social network, etc.). This introduces a generalization to the AGRE approach.

Thus our model relies upon two binary relations between agents and environments: **situation** and **encapsulation**.

In the following sections, we denote by $\mathcal{E}$ the set of environments used in a simulation, and by $\mathcal{A}$ the set of agents. An environment may be any part of a *metric space* (e.g. an euclidian space, a grid, a graph, a point...). In addition, we explain later how to endow it with a specific time scale and with differentiated behavioral rules to apply to the agents it contains.

3.1 Situation of an agent in an environment

The agents we are interested in are *situated agents*. Thus, they belong at least to one environment, in which they can perceive and act. But, we also allow any agent to be situated in *several* environments at the same time. Therefore, we define a first binary relation: the **situation of an agent in an environment**.

Definition 1 *An agent $a \in \mathcal{A}$ **is situated** in an environment $e \in \mathcal{E}$, denoted by: $\mathbf{a} \lhd \mathbf{e}$, iff a can perceive, or be perceived, or act, or undergo actions, in e.*

In the following, we denote: $\forall a \in \mathcal{A}, \text{location}(a) = \{e \in \mathcal{E} | a \lhd e\}$ and: $\forall e \in \mathcal{E}, \text{content}(e) = \{a \in \mathcal{A} | a \lhd e\}$. We also use the notation: $\text{location}(a) \leftarrow \{e_1, e_2\}$, as a shorthand for: "agent a should be placed only in e_1 and e_2", i.e. a is now involved in only two situation relations, namely with environments e_1 and e_2.

Using situated agents means: $\nexists a \in \mathcal{A} \mid \text{location}(a) = \emptyset$.

We call **"usual situated agent"** any agent $a \in \mathcal{A}$ which is situated in a single environment. The set of usual situated agents is: $\mathcal{S} = \{a \in \mathcal{A} | \exists e \in \mathcal{E}, \text{location}(a) = \{e\}\}$.

Conversely, we call **"multi-situated agent"** any agent $a \in \mathcal{A}$ which is situated in several environments at the same time. The set of multi-situated agents is denoted by $\mathcal{M}$: $\mathcal{M} = \mathcal{A} \setminus \mathcal{S}$

According to our aims, usual situated agents and multi-situated agents are characterized in a dynamic way: thus, it is not necessary to assign a prior type to agents, because their status can change during the course of the simulation.

The use of multi-situated agents can fit very different situations: for instance, such an agent can be an edge, an interface between two physical environments (e.g. a door) – in that case it has to be perceived by agents belonging to any of those environments. The concept can also be applied to represent the social belonging of an agent, or its presence in spaces that correspond to distinct scales or organization levels.

3.2 Encapsulation of an environment in an agent

The second binary relation in our model represents the ability, for any agent, to encapsulate an environment (in which other agents can be situated). For the sake of clarity, it is important to keep a conceptual and operational distinction between agents (i.e. entities) and environments (i.e. spaces), but through this second relation we can now formalize that an agent "plays the role" of an environment.

Definition 2 *An agent $a \in \mathcal{A}$ **encapsulates** an environment $e \in \mathcal{E}$, denoted by: $\mathbf{a} \sim \mathbf{e}$, iff a "contains" e. An agent can encapsulate one environment at a time, and similarly an environment can be encapsulated by only one agent.*

In addition, we assume that there is only one environment, denoted by e^0, which is not encapsulated in any agent. In that sense, e^0 corresponds to "the" single environment in a classical MAS. We call it the "root environment" (or "zero-level environment") of the simulation. We denote by $\mathcal{E}^\star = \mathcal{E} \setminus \{e^0\}$ the set of environments that are encapsulated in agents.

Then, we call a **"regular agent"**, any agent that does not encapsulate an environment, and conversely **"compartment agent"**, any agent that encapsulates an environment. The set of compartment agents is denoted by $\mathcal{C}$: $\mathcal{C} = \{a \in \mathcal{A} | \exists e \in \mathcal{E}^\star, a \sim e\}$. The set of regular agents is denoted by $\mathcal{R}$: $\mathcal{R} = \mathcal{A} \setminus \mathcal{C}$. In addition, we denote: $\forall e \in \mathcal{E}^\star, \text{host}(e) = !a \in \mathcal{C} | a \sim e$ and: $\forall a \in \mathcal{C}, \text{space}(a) = !e \in \mathcal{E}^\star | a \sim e$.

3.3 Joint use of both relations

By combining the *situation* in an environment together with the *encapsulation* of an environment in an agent, we are able to define new concepts and the basis of a multi-level architecture.

Thus, for instance, the mere intersection $\mathcal{M} \cap \mathcal{C}$ includes compartment agents that are multi-situated agents at the same time: agents that are able to act or perceive in several environments, and also encapsulate an environment. Far from being a wild invention, this can be applied to many real modeling issues. For instance, it can be used to describe a membrane protein, which has an end inside the cell, and the other end outside: this kind of protein is able to contain other molecules. Similarly, an elevator can be seen as a compartment-agent which is situated at the same time in the shaft and in the successive floors. We call **"pipe agent"** such a multi-situated, compartment agent.

[1]PADAWAN stands for "Pattern for an Accurate Design of Agent Worlds in Agent Nests".

More generally, we can formalize two new relations: **inclusion** and **hosting**.

Definition 3 ***Environment inclusion****. An environment e_1 is* ***included*** *in an environment e_2 (denoteb by $e_1 \subset e_2$) iff: $\exists a \in \mathcal{C} | a \sim e_1 \wedge a \lhd e_2$.*

Definition 4 ***Agent hosting****. An agent a_1 is* ***guest*** *of an agent a_2, or a_2 is* ***host*** *to a_1 (denoted by $a_1 \sqsubset a_2$) iff: $\exists e \in \mathcal{E}^\star | a_2 \sim e \wedge a_1 \lhd e$.*

The first relation can be summarized, for all environments and agents in the simulation, on the **environment graph** (or **inclusion graph**), which is the directed graph defined by all inclusion relations ($\subset$) between environments. This graph is dynamic, since it represents all actual situation and encapsulation relations at a time in the simulation. In the general case, it can also have directed cycles. Similarly, the **agent graph** (or **hosting graph**) of the simulation is the directed graph defined by hosting relations ($\sqsubset$) between agents.

By extension, we define: $e_1 \subseteq e_2$ (transitive inclusion) iff: $e_1 = e_2$ or $e_1 \subset e_2$ or $\exists e \in \mathcal{E} | e_1 \subseteq e \wedge e \subseteq e_2$ (i.e. iff there is a path between e_1 and e_2 in the inclusion graph) ; similarly: $a_1 \sqsubseteq a_2$ (transitive hosting) iff: $a_1 = a_2$ or $a_1 \sqsubset a_2$ or $\exists a \in \mathcal{A} | a_1 \sqsubseteq a \wedge a \sqsubseteq a_2$.

In the general case, any environment is not necessarily included (transitively) in e^0. This could cause paradoxical situations, which have no interest in simulation. In order to avoid them, it is sufficient that, at any time, the environment graph *has no directed cycle*. We call that a **Topologically Regular Simulation**.

Under this assumption, the relation $\subseteq$ is now a **partial order**. Consequently, the environment graph is an upper-semilattice with e^0 as least upper bound: $e^0 = \sup_{\subseteq} \mathcal{E}$. Thus, several rather qualitative concepts such as the "level" of an environment (or of the related compartment) can be now formally defined (in this case, as the *shortest distance to* e^0) and be used to characterize the organization of a system and its subsystems.

4 Implementation in an interaction-oriented simulation approach

The definitions above allow to decompose a complex system into subsystems, which are bound one to another through non-trivial relations (i.e. namely, not only a tree structure). In this section, we explain why an interaction-oriented simulation approach is particularly fruitful in association with situation and encapsulation relations, especially when the behavior of an agent should depend on the environment where it is situated.

Therefore, we rely upon the IODA interaction-oriented approach [Kubera *et al.*, 2011]. Its first advantage is that it is a transversal approach, i.e. the concepts involved during the design phase are implemented as true pieces of code, in a very straightforward way. In addition, IODA assumes a clear separation between agents and behaviors:

- Any relevant entity is an agent (which can be more or less complex) endowed with perception and action *primitives*.
- Any behavior is described by a *generic interaction*, which is a rule involving two agents (a source and a target). Conditions and actions rely upon generic perception and action primitives, thus interactions are independent from the concrete implementation of the agents.
- Interactions are assigned to source and target agents through an *interaction matrix*. Roughly, the interaction matrix is a function $M : \mathcal{A} \times \mathcal{A} \to \wp(\mathcal{I})$, which defines the set of interactions that a source agent can perform on a target agent.

During the course of the simulation, the simulation engine makes each potential source agent (i.e. any agent that can perform interactions, according to the matrix) search, among its neighbors, for potential pairs of interaction/target agent, then selects one of the pairs where the condition of the interaction is fulfilled, and performs the corresponding actions.

4.1 The many faces of an agent

The distinction that the IODA approach makes between entities (agents) and behaviors (interactions) is quite useful, in order to specify behaviors that depend on an organization level. Indeed, the key idea is to *endow each environment with its own interaction matrix*. We explain this mechanism below.

In the following, we consider the general case of an agent a with location$(a) = \{e_1, ..., e_n\}$. Then, we propose to decompose a in a "central", non-situated part, and a set of n "peripheral" parts, each one being situated in e_i.

Thus, we prefer to call **"core of** a**"** the non-situated part of the agent (denoted by $a_{|\bullet}$), and conversely **"face of** a **in** e_i**"** (denoted by $a_{|e_i}$) the part of a that is situated in e_i (i.e. actually a, seen as an agent that would belong only to e_i). The "face" $a_{|e_i}$ is associated with properties, which are specific to its situatedness, such as a location (e.g. coordinates) in e_i, its own perception abilities, so as to determine the neighbors of a *in* e_i, which can be involved in interactions with a according to the interaction matrix M_i defined in e_i.

Now, any agent of our multi-level simulation model can be seen as the union of a "disembodied" agent that cannot perceive nor act (in fact, it is limited to a set of internal states and to computing or decision processes) and several embodied entities which have full perception and action abilities. Each of those embodied entities, when observed in its environment, can be treated as classical agents in usual simulations. The pseudo-agent $a_{|e_i}$ can participate in the native IODA interaction selection process, in the context of environment e_i and interaction matrix M_i. The only difference with a "flat" approach (i.e. with a single environment), is that the primitives performed by $a_{|e_i}$ can access the state and functions of the core pseudo-agent $a_{|\bullet}$.

4.2 Management of time scales and simulation scheduling

In this section, we assume that the simulation is in discrete time, i.e. time is divided in ticks or time steps, which represent a constant duration δt. At time step $t_i \in [0, T_{\max}]$, the simulation time elapsed since the beginning is $t_i \delta t$.

Thus, each environment can easily be endowed with its own *time scale*.

Definition 5 *We call **Rhythm assignation** of environments the function:* $\chi : e \in \mathcal{E} \quad \mapsto \quad (N, \varphi) \in \mathbb{N}^* \times \mathbb{N}$ *where* N *and* φ *represent the **period** and **phase** attached to environment* e.

The period indicates that "something is likely to happen" in e every $N\delta t$, starting at $\varphi\delta t$. Based on that, we say that an environment e is **activated at time** t (or t-activated) iff: $t \equiv \varphi (\text{mod } N)$. We denote by $\mathcal{E}_{\text{act}}(t)$ all t-activated environments. Agents that may act at time t are those situated in t-activated environments. In practice, because agents act according to the interaction matrix of their environment, the simulation engine has to identify faces that belong to t-activated environments, i.e. the pairs: $(a, e) \in \mathcal{A} \times \mathcal{E}$ with $e \in \mathcal{E}_{\text{act}}(t)$, denoted by $\mathcal{A}_{\text{act}}(t)$.

$$\mathcal{A}_{\text{act}}(t) = \bigcup_{e \in \mathcal{E}_{\text{act}}(t)} (\bigcup_{a \in \text{content}(e)} \{(a, e)\})$$

The algorithm we use for agent scheduling is a simplification of the conservative scheduling of HLA [Fujimoto, 1998], since each environment can be seen as a time-stepped simulation. Thus, in order to perform the interaction selection at time t, the pairs of $\mathcal{A}_{\text{act}}(t)$ are first sorted according to a **scheduling policy** (the default policy is a random shuffle). This operation determines the order in which the faces select and perform their interactions. Several scheduling policies can be considered, e.g. sorting by agent families, or by environments, or both. Among others, depending on the application domain, it can be useful to sort environments by level or by period. We cannot focus here on a comprehensive study of frequent "patterns" of scheduling policies, but the point is that this feature is *made explicit* in our simulation model, and thus can be tuned very accurately depending on the purpose and domain of the simulation.

Once the order upon $\mathcal{A}_{\text{act}}(t)$ has been chosen, the only remaining operation is the IODA interaction selection process, applied to the corresponding *faces*, i.e. for each pair (a, e), the face $a_{|e}$ is likely to perform at most one interaction (as a source) with one of its neighbors (as a target). Regarding the core $a_{|\bullet}$, il does not perform nor undergo any interaction by itself: only the activity of the faces is likely to affect the core.

4.3 Primitives specific to the multi-level structure

The interactions we use are described by condition/action rules. Those rules are themselves composed of perception or action *primitives*, i.e. functions or procedures run by the source or target agent involved in the interaction.

Thus, the introduction of situation and encapsulation relations in the interaction-oriented approach requires to define specific primitives, in order to allow the model designer to use those relations in the definition of interactions. We have developed a complete set of perception/action primitives that is sufficient to express atomic operations on the situation and encapsulation relations. We give below several examples, showing how the semantics of those primitives can be defined very precisely.

Test/perception primitives

First, agents are endowed with boolean functions required to characterize them: regular vs. compartment agent, usual situated agent vs. multi-situated agent, etc. An agent can also test if it is the guest of another agent. Those primitives do not raise much difficulties.

Action primitives

Realizable actions must be defined more rigorously, especially because the definition must apply to all kind of agents. In addition, the primitives must keep the simulation "topologically regular", i.e. they must not create directed cycles in the environment graph.

As an example, a primitive like "enter a compartment", `enter(a,c)`, makes sense only if a and c are situated at least in a common environment, and if $c \not\sqsubseteq a$: then, the *location* of a must be modified, by replacing all environments shared with c by the environment which is encapsulated in c: $\text{location}(a) \leftarrow (\text{location}(a) \setminus \text{location}(c)) \cup \{\text{space}(c)\}$.

In some cases, the primitive may use a function as a parameter. For instance, the division of a compartment c assumes that there exists an *allocation policy* π, in order to determine where the agents that were guests of c should be situated after the division. To perform `divide(c,π)`, the first step is to create a compartment c', that is itself situated in $\text{location}(c)$; then, we use the policy, defined as:

$$\pi : \text{content}(\text{space}(c)) \rightarrow \{\text{space}(c), \text{space}(c')\}$$

so that $\forall a \sqsubset c$, all occurrences of $\text{space}(c)$ in $\text{location}(a)$ must be replaced by $\pi(a)$ (which is either $\text{space}(c)$, or $\text{space}(c')$). The default allocation policy is random.

Similarly, we have defined the appropriate primitives to destroy (recursively) an environment or an agent, create agents in specified environments, enter or exit compartments, merge or dissolve compartments, make a regular agent become a compartment and vice versa, go from one environment to another one by "crossing" a multi-situated environment (e.g. a door), etc. All those primitives are available in the current state of our simulation engine.

Extensions of the interaction matrix

In order to allow a compartment agent to interact with its guest agents and vice-versa, we also enhance the IODA interaction matrix by adding both a generic line named "host" to specify the interactions that a compartement may perform on its guests, and a generic column named "host" to specify the interactions that an agent may perform on its host.

4.4 Implementation within a simulation-oriented platform

Since the PADAWAN model relies upon the IODA interaction-oriented approach, the concepts of which are already implemented in a Java simulation framework, called "JEDI" [Kubera *et al.*, 2011]), we developped the appropriate extensions within the JEDI platform. Then, starting with existing classes, we reify the "face" of an agent as shown fig. 1. The entity that appears as an "individual" (the one used in the design phase, and which the domain expert deals with), extends the `PADAWAN_Agent`; but, each time an agent is located in an environment, the simulation engine creates an instance of `PADAWAN_Face`, which behaves exactly like a regular JEDI agent in its single environment.

The scheduler of the simulator has also been rewritten according to the principles explained before. At each time step,

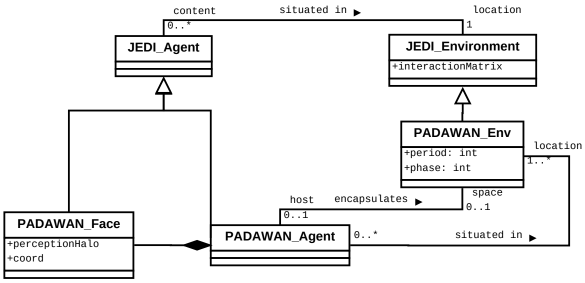

Figure 1: Class diagram which shows the implementation of the PADAWAN model within the JEDI framework. Each "face" agent is situated in a single environment, just like regular JEDI agents, but the "true" entity (used by model designers) is the composition of the faces around a "core".

it computes the faces that are situated in activated environments, applies a *scheduling policy*, and finally makes the faces perform an interaction selection process according to the interaction matrix of their environment. The scheduling policy is chosen as a parameter of the simulation, in a configuration file.

During one time step, each face (as a regular JEDI agent) can interact at most once as a source, and possibly many times as a target. Thus, a `PADAWAN_Agent` that has two faces, each one belonging to an activated environment, is then likely to be the source of one interaction in each environment.

Of course, the design of the *interactions* must take this scheduling into account to ensure a consistent definition of agents behavior. For instance, if a face runs a primitive "grow", while another face runs "shrink", that means that two "opposing" interactions were achievable at the same time: though it could result from a design mistake, it may well be the fusion of opposing forces from different environments – but in any case, this is a domain-dependent issue.

5 Conclusion and perspectives

In this paper, we have presented a double contribution to multi-scale, agent-based simulation. On the first hand, the formal model PADAWAN, based on the situation and encapsulation relations, provides a flexible, dynamical nested structure. This model does not make any assumption regarding what environnements represent, either (physical world, social groups, memory...). We have also characterized "regular" cases which encompass usual MAS. Thus, our situation-encapsulation model is a helpful frame for the decomposition of a multi-layer complex system, whether for simulation or distributed problem solving.

On the other hand, in the very context of interaction-oriented simulation, we have shown how the use of *interactions* (generic rules) to model agent behaviors, makes the design of level-dependent behavior patterns very easy. Indeed, each environment used in the situation-encapsulation model has just to be associated with the approriate *interaction matrix*, in order to specify what interactions an agent can perform on other agents in the corresponding environment.

We have also explained how PADAWAN was implemented within an existing interaction-oriented platform. The current implementation is used for the development of a "Serious Game" devoted to vendor training [Mathieu *et al.*, 2011], which involves the simulation of many levels from malls to shelves.

We also intend to apply our work to cell biology, which is a suitable touchstone for investigating multi-scale issues. In the longer term, we hope that the convergence with other approaches, such as AGRE, which currently focus on organizational issues, could lead to theoretical developments that could be used outside the scope of simulation.

References

[An, 2008] G. An. Introduction of an agent-based multi-scale modular architecture for dynamic knowledge representation of acute inflammation. *Theoretical Biology and Medical Modelling*, 5(11), 2008.

[Ferber *et al.*, 2005] J. Ferber, F. Michel, and J. Báez. AGRE: Integrating environments with organizations. In Weyns *et al.*, editor, *Revised Selected Papers of E4MAS'04*, volume 3374 of *LNAI*, pages 48–56. Springer-Verlag, 2005.

[Fujimoto, 1998] Richard M. Fujimoto. Time management in the high level architecture. *Simulation*, 71:388–400, 1998.

[Kubera *et al.*, 2008] Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations : From theory to implementation. In Ghallab *et al.*, editor, *Proc. of the 18th European Conf. on Artificial Intelligence (ECAI)*, pages 383–387. IOS Press, 2008.

[Kubera *et al.*, 2010] Y. Kubera, P. Mathieu, and S. Picault. Everything can be agent! In van der Hoek *et al.*, editor, *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1547–1548, 2010.

[Kubera *et al.*, 2011] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. IODA: an interaction-oriented approach for multi-agent based simulations. *Journal of Autonomous Agents and Multi-Agent Systems*, 2011. In press.

[Mathieu *et al.*, 2011] Philippe Mathieu, David Panzoli, and Sébastien Picault. Format-store: a multi-agent based approach to experiential learning. In F. Liarokapis, editor, *Proc. of the 3rd Int. Conf. on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 2011.

[Minar *et al.*, 1996] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The SWARM simulation system: a toolkit for building multi-agent simulations. Working Paper 96-06-042, Santa Fe Institute, 1996.

[Păun and Rozenberg, 2002] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.

[Scerri *et al.*, 2010] David Scerri, Sarah Hickmott, Alexis Drogoul, and Lin Padgham. An architecture for modular distributed simulation with agent-based models. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 541–548, 2010.