# Towards an interaction-based design of behaviors

Ph. Mathieu S. Picault

*Université des Sciences et Technologies de Lille, SMAC team, LIFL.*
*Cité Scientifique, 59655 Villeneuve-d'Ascq Cedex, FRANCE.*

**Abstract**

In this paper we defend the advantages of pushing the interaction as a concrete means (not just an abstraction) in the representation of agent behaviors. While the behavior of an agent is usually deeply encoded in its architecture and relies on its very abilities, we rather dissociate agents from interactions, regarding as well the conceptual viewpoint as the implementation. Indeed, we associate both agents and interactions with a specific ontology. This approach is especially valuable for increasing the reutilisability of the interactions, which can be written very often in a generic way even when agents are bound to specific domains. We show here that this approach is particularly well adapted to broad simulation scales, regarding as well the number of individuals as the diversity of the behaviors. This paper clearly addresses issues at the intersection between software engineering and knowledge representation in agent simulation systems ; thus we present the IODA methodology which provides guidelines in the design of such simulations.

## Keywords

Simulation, Interaction, Behavior

## 1 Introduction

In recent years, agent-based simulations have been widely used to model living entities, in order to understand the mechanisms of living entities, or to reproduce their features into entertaining applications (video games, movie animation, etc.). The design of such simulations addresses issues that belong to both implementation and modeling.

Thus, a large number of multiagent platforms have been developed. Especially, many open platforms provide the developers with advances in software engineering (reusability, genericity, at least through object-oriented programming, and in some cases design patterns or components), e.g. Swarm [1] or Madkit [2]. On the one hand, some of them are not specifically dedicated to simulation design. Their interest mainly relies on the reusability of the code, which is developed in a generic software context. On the other hand, other platforms, like Netlogo [3], are designed to be used without a lot of knowledge in computer science, and are based on very simple programming languages. Especially in Netlogo simple "turtles" are running procedures in parallel, without any assumption regarding the application field of the simulations. Unfortunately, the openness of such platforms and their genericity are provided to the detriment of a precise, formal modeling scope that would lead the design of agent behaviors. For instance, Netlogo does not provide abstractions; the concept of behavior itself is not even reified in the system.

From the analysis viewpoint, some generic formalisms used to represent agent behaviors (subsumption, neural networks, Petri nets, set of rules...) have a precise formalism, and thus could provide a strong help in the modeling process, to the detriment of behavior reusability. This situation applies to simulations dedicated to the study of a specific phenomenon. Though the agent behaviors developed in this context are shaped by a specific architecture, and thus can be reused by other simulations involving agents with the same architecture, there is often a tight relation between the behaviors and the application field in which the simulation takes place. One exception to mention would be the BDI architecture, which appear to provide generic (logic-based) frame

for representing behaviors independently of the application domain; unfortunately, its logic and symbolic base makes it generally inadequate for simulation purposes.

The ideal approach would consist in designing both generic and reusable behaviors, not only from a software engineering point of view, but also regarding knowledge representation. It also should take place in a formal methodology, precise enough to lead the design of the simulation. Nevertheless, different agent architectures are based on different paradigms, aiming for different qualities, and should not sweeped together into a heap.

This implies to extract all generic features from the agent, such as movement, interactions that can be performed or undergone. The agent should keep only specific perception/action capabilities, and possibly a reasoning engine. For instance, a behavior like *eat* has the same purpose in any simulation, and matches the same function too; it can always be described with the same trigger (hunger) and the same prerequisite (to posses something edible). On the other hand the details in carrying out this interaction depend on the physical particularities of each agent. The same applies to motion: many behaviors have to express that eg. to do $X$ the agent has to go from $A$ to $B$, but often the concrete way of moving from $A$ to $B$ is not the main point (some agent could achieve it using the A* algorithm, others with random search).

In order to work towards this idea, we propose in this article to reify the concept of interaction independently form those of agent and behavior. This leads to define on the one hand the activities that can be performed during a simulation, and on the other hand the agents that will be able to perform them. In the model we propose, interactions are sets of atomic actions, released by prerequisites and triggers; agents are more or less complex entities endowed with the ability of performing or undergoing interactions; and finally, behaviors are all effective interactions that occur during a simulation.

We also demonstrate how the notion of behavior becomes a mere consequence of the interactions abilities given to the agents, each generic interaction being carried out according to the elementary perception, cognition and action capabilities of the agents.

This approach is especially adequate for simulations that require a large number of agents or a large number of agent species (i.e. about thousands or more), and in fields where entities are not important by themselves (as individuals like in ethology, AI, game theory) but rather through their functional relationships: e.g. transport, biology, markets...

Finally, we describe the IODA[1] methodology which provides empirical processes to interaction-based analysis for designing simulations relying on our model. For example, we advocate to translate the intended behavior into dynamic matrixes to describe the interactions between entities, then to refine the structure of agents and their interaction abilities.

## 2 The notion of interaction

We first have to emphasize that we do not intend here to deal with *interaction protocols* (and e.g. problems of local synchronization of actions such as discussed in [5]), nor consider interactions as coordination situations involved in groups, that could be formalized in an algebraic way [6]. Other approaches such as Vowels [4] involve a notion of interaction that is close to ours, but do not lead to a practical implementation. We rather use the term "interaction" in its etymological meaning: "action between" agents, as an alternative representation of agent behavior, not centered on the agent itself.

Among many attempts to introduce abstraction in the development of multi-agent systems, the notion of *role* for example has been used to express and reify a distinction between individual agents and their functional involvement in groups and organization [8].

In the same way, in order to simulate the global behavior of a system, the classical approach consists in designing entities with their own, domain-dependent behaviors ; we propose instead to design separately entities endowed with perception and action primitives on the one hand, and the way those perception and action abilities are carried out in consistent, generically-defined interactions.

Thus, the notion of *interaction* describes sets of elementary primitives (named *actions*) that imply simultaneously some agents and constitute a semantic whole in a given simulation [7]. For

[1] meaning *Interaction-Oriented Design of Agent simulations*

instance, *eat* or *open* are not mere actions, but should be seen rather as structured sets of actions involving two different agents, and that can be performed only under specific conditions.

This definition of interaction makes it completely independent from the agents that are able to use it. It is indeed a concept general enough to be reused in various simulations (the interactions *open* has, of course, the same meaning in a simulation of building emergency evacuation or in a treasure hunt video game) and to be applied to different agents (regarding the prerequisites and the consequences upon the state of the world, the interaction *open* can be described the same way for opening either a door, a chest or a tin). In addition, this distinction is also a practical way to enable the designers to test the consequences of assigning a specific interaction to specific agents, and thus setting up a simulation step by step (it is particularly the case in video games or artificial worlds).

Hence, the design of a simulation consists in first determining the basic action primitives that will be used, then in aggregating them into interactions, and finally in assigning those interactions to agents.

## 2.1 The basic primitives of a simulation.

The basic action primitives define the smallest elements that are to be represented in the simulation, in spatial (resolution), temporal (time steps) and behavioral terms (atomic perception, cognition and action capabilities) at the same time.

Among those primitives, we can cite the perception operations, aimed at looking at the state of the environment, receiving communications from other agents, taking into account internal stimuli (state changes), and sometimes representing also beliefs or goals. We make no particular assumption regarding the cognitive abilities of the agents in our interaction model, so that the notion of "perception" can refer as well to endogenous or exogenous stimuli, as to the result of a sophisticated reasoning process.

The second subset of primitives is related to motion or action in the environment, or on other agents, including destruction or creation of new entities. Those action primitives can also refer as well to mere reactions induced by stimuli, as to deliberate acts resulting for a planning process.

Thus, an interaction can be defined as a sequence of action primitives, applied to several agents, triggered by specific perceptions and liable to prerequisites for running. Those perception and action primitives may be realized differently in distinct agents, but their sequence is described in a generic and consistent way by the interaction (cf. fig. 1).

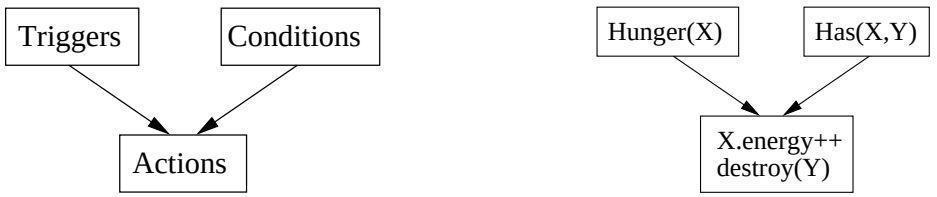


Figure 1: (a) General representation of an interaction. An interaction contains a set of perceptions (on the one hand, a trigger and on the other, prerequisites). When all are fulfilled, they enable the realization of a sequence of actions. (b) Example : the interaction *eat* can be described in a general way through an internal motivation (hunger) which is used as trigger, prerequisites (having something that can undergo the interaction), and the sequence of actions resulting from the interaction, i.e. increase of the energy of the source agent (performing *eat*) and destruction of the target agent (undergoing *eat*).

## 2.2 The notions of trigger and prerequisite.

We now have to define how an interaction can occur between two (or more) agents. First, there is a structural constraint (depending on the definition of the agents): one of the agents has to be able to perform the interaction (it is the *source* of the interaction), and another one must be able to undergo

it (it is the *target*). Then, an interaction, seen as the abstract expression of a behavior, contains also conditions for performing the sequence of action primitives. We have already distinguished between those conditions:

- the *trigger*, which expresses a motivation (either explicit or implicit) for the interaction to be performed: for instance, the trigger of the interaction *eat* is "being hungry". This trigger is indeed a kind of (implicit) goal: the interaction will be performed in order to eliminate the sensation of hunger. More generally, it is likely that the state of the world described by the trigger will be inversed or corrected by the execution of the corresponding sequence of actions.
- the *prerequisite* itself, which expresses physical or logical necessities that must be fulfilled in order to run the action sequence: e.g. in order to *eat*, it is necessary to "possess something edible". If not, it is logically impossible to perform the primitives.

## 2.3 Source and target(s)

In principle, an interaction occurs between a source and a target. The source is an agent that *can perform* the interaction, and the target is one that *can undergo* it. The task of the simulation engine (cf. § 3.1) essentially consists in scanning, for each agent candidate for being a source of interaction, if potential targets can be reached by the possible source (in a given radius), and in chosing the interaction to perform according to a priority order.

The following interaction (coming from a simulation of gene regulation mechanisms) expresses the translation of a gene. In order to undergo the interaction *translate*, the gene has to encounter an enzyme able to perform *translate*. They become then respectively the target and the source of the interaction, according to the code given below. The interaction can effectively occur (and the actions be run) only if both parts of the condition (trigger and prerequisites) are fulfilled at the same time. The prerequisite indicates that the transcription of a gene can occur only in specific states of the DNA, while the trigger uses a stochastic model to determine the activity level of the transcription factor, and whether or not the gene may be translated.

```
TranslateGene :
     Prerequisite(source, target) :
          return (target.state != PACKED)
     Trigger(source, target) :
          if (target.activation == ACTIVE)
            return (random < probaActivation)
          if(target.activation == NEUTRAL)
            return (random < probaNeutral)
          if(target.activation == INHIBITED)
            return (random < probaInhibition)
     Actions(source, target) :
          createNewProtein(target.encodedProtein)
```

Through this model, it is also possible to design interactions without defined targets: in that case they are performed by a single agent (the source), either for state modifications or for actions on the environment.

IODA also allows interactions to cope with multiple targets at the same time, in order to represent activities that require coordination between agents. Yet the interaction uses a single source only. In fact, in our method, we can treat interactions based on one source and one target, and also one source and multiple targets. The symmetric case can be expressed by switching source and targets. For example, in order two represent two agents that have to carry a load simulatenously, we will not use the *carry* interaction (which would involve several sources), but instead the *is_ carried* interaction (which uses only one source and two targets). The only situation that cannot be modelled with IODA is a multiple source/multiple targets interaction (but we did not encounter any need for that).

It is important to keep in mind that the representation used in our model does not require any specific cognition level, since a same generic interaction can be performed through either reactive or cognitive conditions and actions, depending on the nature of the agents involved in the interaction.

### 2.4 Assigning interactions to agents

Once the designer of the simulation has designed the relevant interactions, the entities that could act as source or target are generally quite clear (especially since the usual design methodology starts from the identification of the entities and focuses on the behaviors only in a second stage). There are still two points to elicit:

- Agents usually interact only with other agents when they are "close" enough (may it be a spatial distance in the environment or any measurement in a state space). Thus there is to define a condition for the distance allowed between the source and the target.
- Since each agent is generally able to perform or undergo a large number of interactions, each interaction must be associated, in each agent that is able to perform it, with a priority level, in order to ensure that functional features are assumed in a consistent way

We must emphasize that both features cannot be addressed only in the definition of interactions, nor in the properties of the agents. For instance, the interaction *eat* as defined in fig. 1 can ordinary be performed only when agents are at the same place, but could allow a 3-meter distance for a chameleon! Such an interaction would also have a higher priority than reproduction in human beings than in butterflies.

It is through this matching between generic interactions and concrete agents, and through the determination of priorities and distance conditions associated with each interaction/agent couple, that the behaviors produced during the simulation can be refined.

## 3 The behavior of an agent

In our interaction-based model, the behavior of an agent, defined as the sequence of actions that it effectively carries out during a simulation, appears to be the result of the interactions that are performed. Therefore, we use a simple action selection engine.

### 3.1 A reactive simulation engine

We describe here the mechanism that applies to chose the interactions to run a simulation. The principle is quite simple: at each time step, when two agents "meet" together (with regards to the definition of the distance constraints mentioned above), they examine if one of them can perform an interaction and if the other can undergo it. If so, that interaction becomes candidate, according to the following rules:

$\boxed{I(x,y) \in \textit{Interactions} \text{ can occur if } \exists\, x, y \in \textit{Agents} \text{ so that } \textit{can-perform}(x,I) \wedge \textit{ can-undergo}(y,I)}$

The predicates *can-perform* and *can-undergo* are given through lists of interactions which the agents $x$ and $y$ can play either as source or target.

Concretely, during one cycle in the simulation, we let each agent try to perform an interaction. This choice uses the following protocol:

1. First, the agent tries to find potential targets for each of the interactions it might perform, i.e. other agents within the convenient distance that might undergo the corresponding interaction.
2. Then, interactions for which targets have been found are sorted according to their priority in the potential source agent.
3. The agent inspects their trigger and prerequisites and chooses the first with both conditions fulfilled.
4. Finally, the corresponding actions are run.

If an agent could not perform any interaction, it is still possible that it will undergo one from another source.

### 3.2 Structure of the agents

Generally the agents and their primitives are the most domain-dependent features when designing a simulation. Nevertheless, in our approach, the clear separation between the interactions and the simulation engine reduces the agents to a lower role than in classical SMA engineering methodologies. In our model, the agent is an entity:

- with an internal state
- having specific perception and action primitives
- which is given the lists of interactions that it can perform (with priority levels and distance constraints) or undergo

Any agent which complies to those simple specifications can be integrated in our interaction-based simulation model; we make no particular assumption regarding the cognitive level of the agent itself, so this point can be left to the designer.

Anyway, the action selection engine we use is reactive: interactions are not selected for example as the result of inferences to reach a goal, but only depending on the possibilities to realize them (contact between a source and a target and conditions fulfilled). The process that selects the interaction is the same even if the agent tests its condition through logic proofs in a symbolic model of the world.

### 3.3 The conceptual point of view (the representation of domain-related knowledge)

The "extraction" of generic behaviors from the definition of the agents, in order to define them in separate interactions, provides the designer with a guideline to represent the knowledge of the domain in which the simulation takes place.

In many application fields (chemistry, biology, physics...), the notion of individual behavior (historically drawn from ethology and sociology) is indeed not really relevant. On the one hand, an entity can exhibit very distinct behaviors in its activity. For instance, an enzyme can change its 3D folding and take part thus in various cell functions which have no relation one with another. On the other hand, entities that are structurally different (e.g. in their chemical properties) might provide equivalent functions (a protein or a RNA molecule can both act as a transcription factor in gene expression).

So, the reification of the notion of interaction leads to the definition of two distinct ontologies, one for the entities that take part in a given process, and the other for the functions (= interactions) performed by those entities. The relationship between those two sets is generally not fixed during the simulation.

To represent radical changes in behaviors that can affect some entities, it is sufficient to simply modify the lists of interactions that those entities can perform or undergo. In a classical agent-centered behavior design, the only way to adapt the behavior relys indeed on a sophisticated, individual, action selection mechanism. In the interaction-oriented approach, individual adaptive abilities are still involved, since the agent may "chose" some interactions rather than others through the way its own perception capabilities affect the triggers of the interactions it can perform. But in addition, the global assignment between interactions and agents may be modified during simulation, allowing the representation of major behavioral pattern switches.

### 3.4 The software engineering point of view

The second benefit of our model consists in the reusability, in different simulations, of generic interactions libraries. Of course all interactions of a given simulation are not involved in any other, yet for applications in close enough domains, interactions are expressed in similar ways. Then it remains simply to rewrite, for each specific simulation, the domain-dependent agents with their own perception and action primitives.

We do not claim that our interaction-based approach makes possible some simulations that would not be realized in other platforms. Our point is that, from the software engineering viewpoint,

the design of simulations and the reusing of interactions library is made much easier through our approach. In the kind of problem we address, Swarm, Netlogo and Madkit are amongst the more widely used platforms. In these three frameworks, there is no agent model, nor regarding the behavioral architecture, nor regarding perception and action primitives. The agent is an empty shell in which the designer can/has to put himself all the code to produce the behavior. Consequently, the code of the agent is a mixture of features that concern action upon the environment (e.g. motion and position), interactions with the other agents, and others that take part in action selection. Thus in such a blend it is highly difficult to add or suppress any behavioral ability, and still more to reuse the code in a similar simulation.

```
to go  ;; forever button
  ask turtles [ go-turtles ]
  diffuse chemical (diffusion-rate / 100)
  ask edge-patches
    [ set chemical 0 ] ;; prevent wrapping
  ask patches [ go-patches ]
  do-plotting
  set clock (clock + 1)
end

to go-turtles  ;; turtle procedure
  if (who < clock) ;; delay
  [ ifelse carrying-food?
    [set color orange + 1 return-to-nest ]
    [set color red    look-for-food  ]
  ]
end

to go-patches  ;; patch procedure
  set chemical (chemical*(100-evaporation-rate)/100)
  update-display  ;; Refresh the Display
end
```

The example given above, drawn from the simulation *Ants* in Netlogo libraries, is a good illustration of our argument. Despite the simplicity of Netlogo's programming language, there is a complete mixing between notions which are bound to this specific simulation (e.g. pheromones represented by the `chemical` variable, which can diffuse in the environment), and a makeshift job resulting from the absence of any model/view distinction (e.g. setting `chemical` to 0 on the sides, just because Netlogo's environment is always a torus!). It is clear that if the designer should modify the behavior of ants, it would be puzzled about what procedures he should also rewrite, and would not be able to ensure that no artifacts have been introduced. In general he would probably prefer rewriting all the code.

In open platforms providing all recent advances of software engineering, the reusability applies to agents seen as a whole, e.g. as classes, including the domain-dependent features imprinted in their behavior. On the contrary, true genericity is obtained when the behavior is extracted from the agent and put into abstract interactions.

# 4 Knowledge representation for large-scale simulations

Historically, individual-based simulation involves a few dozen to several hundred agents [9, 11]. Now, many application fields encounter increasing needs for much larger scales, regarding as well the total number of agents (from $10^3$ to $10^5$) and the number of agent classes (several hundred). For instance, in recent movies (such as the battles scenes in P. Jackson's *Lord of the Rings*), about $10^4$ agents have been used in place of extras, in very realistic fights. It is the same in video games, where non-human players have to exhibit more and more rationality and diversity in order to make the impression that they are in some sense taking initiatives instead of waiting for the human player and cheating. Amongst other domains, we also can cite transport (e.g. through projects such as Archisim from the French research institute INRETS [10]), and biology.

The latter field especially illustrates constraints that are specific to large-scale simulations:

- First, constraints regarding the *number of agents*: a single bacteria contains about 4,000 different molecular species, most of them having chemical interactions, and about $10^{10}$ molecules in total. It is obviously unrealistic to represent each molecule with a single agent. Thus, most

biological models (e.g. the study of regulation networks) are based on equation systems, using the concentrations of few molecular species. Though, this approach discards right away the spatial specificities of those biological phenomena, and do not cope with situations involving only few molecules (e.g. those used as signals [12]).

- A more realistic approach should also take into account the existence, among biological processes, of very different *scales* [13]: they concern as well the numbers of agents (from billions of structure proteins vs. dozen signaling enzymes), the space management (from the local environment of cells to the conformation of the DNA), and time (from seconds to hours or days).

In addition, since agent simulations have been first applied to domains such as ethology, economy, sociology, it appears that many concepts drawn from those fields have been widely used as a general basis for simulation, without deeper questions. It is particularly the case with the notion of *behavior* that is usually seen as a property of the entities. Nevertheless, the application of simulation to large scales reduces the importance of the notion of *individual entity*, thus also of *individual behavior*. In physics, biology, ecology of market dynamics, the behavior of each entity taking part to the process is less important than the *interactions* that occur there. For instance, in statistical physics, the gas laws result from the integration of all interactions (i.e. shocks) between gas particles; individual particles do not matter by themselves. In most of such domains, we are not working at this extreme integration level, since agents have an history and thus are not interchangeable; however their activity as mere individuals does not make sense.

# 5 Towards an interaction-based modelling

## 5.1 Sketching a methodology: IODA

In order to address a simulation problem, it is necessary to identify on the one hand the entities which, depending on the model of the application field, are supposed to interact one with another to produce the target phenomenon, and on the other hand those interactions themselves. In an agent-based simulation model, the identification focuses on the entities, which are then given behaviors designed to produce the intended interactions. Thus, the abstract functions associated with interactions are lost and encapsulated in the specificity of the agents.

On the contrary, we suggest to lead the analysis by eliciting the agents and the interactions at the same time, in order to keep an abstract view of the functions provided by the agents.

The methodology we are working on, IODA (*Interaction-Oriented Design of Agent simulations*), proposes a three-step design for an interaction-based simulation:

1. Identify the interactions (abstract functions, elementary processes). This leads to a matrix of potential sources and targets (cf. tab. 1, and tab. 4 for a detailed example), in which generic interactions clearly appear. This matrix can eventually evolve during a simulation, depending on the way the agents are affected by the interactions.

2. Identify the properties of the agents (structure and basic perception/action capabilities), and assign to them the interactions they can perform or undergo. At that stage the relative priority of each interaction and the distance constraints must be specified. The result is a synthesis table (cf. tab. 2, and tab. 5 for a detailed example).

3. Finally, determine the dynamics of the system, i.e. how the characteristics of the agents are likely to evolve through the interactions, including their ability to perform or undergo interactions. This can lead to draw a diagram describing transitions between states of the system (cf. tab. 3 for an example), each state corresponding with different interaction matrixes (for instance in the AOE simulation, switching from a "peace" state with patrol activities and a classical resource management, to a "war" state with military targets and an emergency management).

As a guideline through those three steps, we use arrays that have to be filled before starting the code. They essentially answer the following questions: on the one hand, what are the interactions,

with what sources and what targets? on the other hand, what are the agents, how can they interact? and last, how to assign dynamically interactions to agents?

| Sources \ Targets | $Ag_1$ | $Ag_2$ | ... | $Ag_n$ |
|---|---|---|---|---|
| $Ag_1$ | | | | |
| ... | ... | ... | ... | ... |
| $Ag_n$ | | | | |

Table 1: This first table provides the matrix of the realizable interactions in the studied simulation. An interaction $I$ which is at the crossroad of the $i$ line and the $j$ column can be performed by the $Ag_i$ agent and undergone by $Ag_j$.

| agent | carac | can perform | priority | dist | can undergo |
|---|---|---|---|---|---|
| | | | | | |
| ... | ... | ... | ... | ... | ... |
| | | | | | |

Table 2: This second table gives for each type of agents of the studied simulation, its structural characteristics and the list of the interactions being able to be performed (with, if necessary, their priority and their guard of distance), and to be undergone.

| State | Agent | can perform | can undergo |
|---|---|---|---|
| Peace | Soldier | $\neg Fight$ | $\neg Fight$ |
| Peace | EnnemySoldier | $\neg Fight$ | $\neg Fight$ |
| Peace | EnnemySoldier | $\neg Destroy$ | |
| Peace | Forum | | $\neg Destroy$ |
| War | Forum | $\neg CreatePeasant$ | |

Table 3: The last table lists the possible modifications of assignment of the interactions to the agents, compared to the initial matrix of interactions, according to the dynamics desired for this simulation. In this example, applied to the "Age of Empires" (AOE) simulation, in the "Peace" state the soldiers cannot fight: we remove then from the list of what it can carry out the *Fight* interaction (which stays on the contrary in the "War" state). In the same way, to save resources, the Forum stop producing Peasants during a war period.

## 5.2 Extension of the model : moves

The approach that we described in this paper has been mainly implemented and tested on a platform dedicated to large scale simulations, SimuLE.

This tool is currently used for applications in cellular biology, in particular to evaluate the various assumptions related to the genetic transcription. These simulation require thousands of agents of different families with many interactions between each one.

Several demonstrations of this platform are available on the site of the team. SimuLE is able to make interact until 50 000 individuals simultaneously in real time on the screen with very many interactions.

SimuLE have also an original device of subscription to statistical indicators for each simulation and is able to provide one returned as well 2d as 3d of the simulated model. For optimization and speed reasons, the distance is in the current version specific to each agent (perception size of the potential targets) and nonrelated to the interaction-agent connection like it was described previously. One of the next improvements of this realization will obviously consist in regulating this point.

| Targets → Sources ↓ | Forum | Limit | Mine | Soldier | Ennemy | Peasant | (none) |
|---|---|---|---|---|---|---|---|
| Forum | | | | GiveRole | Fight | GiveRole | CreateSoldier CreatePeasant Doing Nothing |
| Limit | | | | Router | | | |
| Mine | | | | | | | |
| Soldier | | | | | Fight | | Become Chief MoveCheif Move |
| Ennemy Soldier | Destroy | | | Fight | | Fight | Become Chief MoveCheif Move |
| Peasant | TellExhaustedMine PutResource | | Exhausted Mine TakeResource | | | | Move |

Table 4: Matrix of realizable interactions in the "Age of Empires" (AOE) simulation tested with the SimuLE platform. The agents handled in this simulation are : the forum (which create peasants and soldiers, and where one can deposit resources), the mines which contains resources, the peasants which works in mines, the soldiers defending the peasants and the forum of the enemy soldiers, and limits between which soldiers patrol. This matrix indicates which interactions are suitable for be performed and undergone by each class of agents. In case of multi-targets interactions, cardinalities can be added to the interaction name.

We consider several extensions to increase the conceptual and software distinction between the agents and the representation of their activity. In particular, agents moves, and more generally the physical laws of the environment, to which the agents are subjected, are not to be strictly speaking "interactions", insofar as these laws are applied permanently on all the agents, and come to be added to the interactions which the agents have the ones with the others. The agent behavior results to the fact that it is subjected to physical laws exerted by the environment and that it interacts with other agents. Just as we have separated in the software the interactions from the agents, we project to distinguish the physical laws (moves constraints for example) from the agents and/or the interactions in which they are described at this time. Thus, we will have libraries of generic displacements reusable, to which the simulated entities could be subjected independently of their behavioral activities.

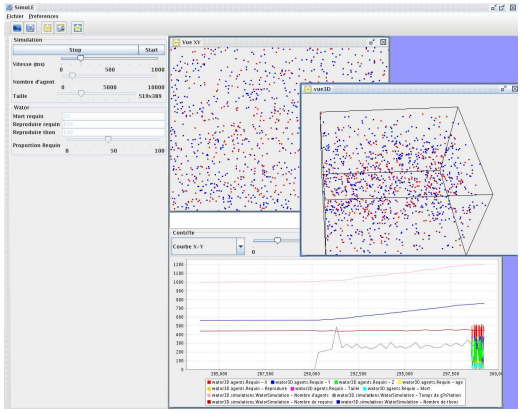

Figure 2: A screenshot of SimuLE, with several sights of the running simulation and a statistical analysis.

# 6 Conclusion

In this paper we have presented a formal model for the interaction between located agents for simulation purpose. This model leaves the report that during the analysis phase of a problem, there is no reason to bind the various possible interactions with the agents which will be put in

| agents | characteristics | can perform | priority | distance | can undergone |
|---|---|---|---|---|---|
| AgentAOE<br>Movement | direction<br>LifePoints<br>camp<br>role<br>ForumPosition | | | | |
| Enemy Soldier | chief<br>attackPoints | BecomeChief<br>Fight<br>Destroy<br>ChiefMove<br>Move | 4<br>3<br>2<br>1<br>0 | <br>$d \leq 5$<br>$d \leq 2$ | Fight |
| Soldier | cheif<br>attackPoints | BecomeChief<br>Fight<br>chiefMove<br>move | 3<br>2<br>1<br>0 | <br>$d \leq 4$ | Fight<br>GiveSoldierRole<br>Router |
| Peasant | resources<br>maxResources<br>resourceTypes<br>exaustedMine<br>provenance | MineExhausted<br>TellExhaustedMine<br>PutResource<br>TakeResource<br>move | 2<br>2<br>1<br>1<br>0 | | GivePeasantRole<br>Fight |
| Forum | mines<br>exhaustedMines<br>limts<br>timerCreation<br>pointsVie<br>resources | Fight<br>GiveSoldierRole<br>GivePeasantRole<br>CreaterSoldier<br>CreatePeasant<br>DoNothing | 4<br>3<br>3<br>2<br>1<br>0 | $d \leq 6$ | Destroy<br>PutResource<br>TellExhaustedMine<br>Destroy |
| Mine | resources<br>resourceTypes | | | | TakeResource<br>ExhaustedMine |
| Limit | direction | Router | 0 | | |
| Obstacle | | | | | |

Table 5: List giving for each type of agents of the "Age of Empires" (AOE) simulation, its characteristics and the list of the interactions being able to be performed (with, if necessary, their priority and their guard of distance), and to be undergone.

presence, as opposed to what offer the current other platforms. This separation of the concepts allows in particular an increased reutilisability of the interactions created and makes it possible without any doubt to conceive in similar contexts several reusable libraries of interactions.

In order to validate this claim, we should evaluate time costs for modifying a simulation, compared to an agent-based encoding a behaviors. This kind of measurement is though hard to perform, as it is difficult to compare object-oriented programming to imperative programming.

We have shown that a rigorous design needs various stages starting with the identification and the creation of the possible interactions thus that of the agents which will be implied, continuing with the assignment of these interactions to the agents with the taking into account of the distance and then the priority between them.

The agent model that we recommend supports on two fundamental properties named *can-undergo* and *can-perform* which contain each one a list of interactions characterizing the behavior of the agent. We finally described precisely how to design the reactive engine which is able to take into account this model. This formal model has been implemented in the SimuLE platform of the SMAC/LIFL team [14] which in particular allows to make interact more 50 000 agents in real time in an environment 2d or 3d. Our IODA methodology has been used and refined during the development of the experiments undertaken on this platform. Currently, SimuLE is in particular used to test various assumptions in several problems of cellular biology, field requiring a large scale platform to be simulated. As well as we have dissociated the agents of their behavior, we plan to distinguish the physical laws from the agents and the interactions, in order to build generic and reusable libraries of moves. These laws are indeed the expression of environmental constraints and apply independently of the nature of the agents or their behavior. We will thus have in the long term triple ontology of agents, interactions and environmental laws which will make it possible to conceive more effectively simulations in changing contexts.

## Acknowledgements

This research work is granted by the French "contrat de plan État-Région" and the EU FEDER.

## References

[1] R. Burkhart. "The Swarm Multi-Agent Simulation System". *Object-Oriented Programming Systems: Langages and Applications (OOPSLA)*, Workshop on The Object Engine, 1994.

[2] O. Gutknecht, J. Ferber, F. Michel, "Integrating tools and infrastructures for generic multi-agent systems". In: *Proceedings of the fifth international conference on Autonomous agents (AA'01)*, ACM Press p. 441–448, 2001.

[3] U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999. `http://ccl.northwestern.edu/netlogo/`.

[4] Y. Demazeau, "From Interactions to Collective Behaviour in Agent-Based Systems". In *Proceedings of the 1st European Conference on Cognitive Science*, Saint-Malo, 1995.

[5] K. Schelfthout, T. Coninx, A. Helleboogh, T. Holvoet, E. Steegmans and D. Weyns, "Agent Implementation Patterns". In *Proceedings of the Workshop on Agent-Oriented Methodologies at OOPSLA'02*, 2002.

[6] A. Gouaich and F. Michel, "MIC*: a Deployent Environment for Autonomous Agents". In *Environments for Multi-Agent Systems, First International Workshop (E4MAS'04)*, 2004. Springer LNAI 3374.

[7] P. Mathieu, S. Picault et J.-C. Routier. "Simulation de comportements pour agents rationnels situés". *Actes de la conférence Modèles Formels pour l'Interaction (MFI'03)*, p. 277–282, 2003.

[8] J. Ferber et O. Gutknecht, "A meta-model for analysis and design of organizations in multi-agent systems". *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, p. 128–137, 1998.

[9] M. Resnick, *Turtles, termites and traffic jams. Explorations in massively parallel microworlds*, MIT Press, 1997.

[10] S. El Hadouaj, S. Espié, A. Drogoul, "A multi-agent road traffic simulation model: validation of the insertion case". *Summer Computer Simulation Conference (SCSC 2004)*, 26-28 July 2004.

[11] J.M. Epstein et R. Axtell, *Growing Artificial Societies. Social Science from the Bottom Up*, MIT Press, 1996.

[12] R.C. Paton, M. Fisher et K. Matsuno. "Intracellular signalling proteins as 'smart' agents in parallel distributed processes". *Biosystems*, 1999.

[13] S. Leibler, L.H. Hartwell, J.J. Hopfield et A.W. Murray. "From molecular to modular cell biology", *Nature* vol. 402, déc. 1999.

[14] The SimuLE project homepage.
`http://www.lifl.fr/SMAC/projects/simule/`.