Interaction-Oriented Agent Simulations : From Theory to Implementation

Yoann Kubera, Philippe Mathieu, Sébastien Picault

▶ **To cite this version:**

Yoann Kubera, Philippe Mathieu, Sébastien Picault. Interaction-Oriented Agent Simulations : From Theory to Implementation. 18th European Conference on Artificial Intelligence (ECAI'08), Jul 2008, Patras, Greece. pp.383-387. ⟨hal-00731987⟩

HAL Id: hal-00731987

https://hal.science/hal-00731987v1

Submitted on 29 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Interaction-Oriented Agent Simulations: From Theory to Implementation

Yoann Kubera and **Philippe Mathieu** and **Sébastien Picault**[1]

Abstract. This paper deals with the software architecture for individual-centered simulations, i.e. involving many entities interacting together. Many software architectures have been developped in this context, especially many advanced – but domain specific – frameworks. Yet those frameworks imply tight software dependencies between agents, behaviors and action selection mechanisms, which leads to many difficulties in modelling and programming. We propose a method and an architecture where interactions are reified regardless of agents, in order to obtain a complete interaction-oriented design process for simulations. Then, an agent is only an entity that can perform or undergo a set of interactions, even not specifically developped for it. Thus most interactions can be re-used in many contexts. In addition, our method clearly separates knowledge about behaviors from its processing, and thus makes the design of simulations easier. Moreover, this new and user-friendly approach helps programmers to build simulations with a large number of different behaviors at the same time, especially in the context of large-scale simulations.

1 Introduction

In recent years, agent-based simulations became preponderant among living beings simulation tools, either to understand their mechanisms or to copy them for leisure use (video games, animation in films, etc.). It links up experts from both specific domains (biology, sociology, etc.) and computer science. Its multidisciplinary nature has given birth to more-or-less domain-specific platforms.

A large subset of those – like Swarm [4], Madkit [6] or Magique [2] – are open, and thus enable the user to freely implement agents, behaviors and environments. They offer different levels of software refinement and allow the use of many engineering tools – design patterns, components, inheritance, *etc.* Moreover, the platforms cited above are not only dedicated to simulations, but can also be used to build agent-based applications. Others – like Netlogo [12] – are based on a simple programming paradigm designed for non-computer scientists. The generic aspect of all those open platforms is obtained at the expense of a formal way to guide the design of behaviors. Data is indeed mixed with its processing – i.e. the action selection mechanism is mixed with behavior representation – which implies a complete reimplementation of the agent when adding or deleting an interaction in which it is involved.

On the opposite, many formalisms – like Petri nets, subsumption, rules sets, artificial neural networks – may strongly guide agent architecture, at the expense of reusability in other formalisms. Some of the rare ones that make possible behavior reuse are the cognitive architectures with plans like in Act-R [1] where knowledge is separated from its processing. However they are often not fitted neither to build multiagent simulations because of their poor performances nor to design reactive agents.

In order to build reusable and generic behaviors, we promote in this paper the *Interaction-Oriented Design of Agent simulations* (IODA) formal method and architecture based on [9, 8] works. It consists in abstracting from the agents the actions they participate in, by reifying them into the notion of interaction. An agent may perform or undergo a set of interactions which are not specifically developped for it. Thus most interactions can be re-used in many contexts. In addition, this architecture clearly separates data from processing, and thus makes the design of simulations easier. We also describe the *Java Environment for the Design of agent Interactions* (JEDI) platform, which is a Java implementation of IODA for simulations with reactive situated agents.

The second section contains a brief introduction to related work on generic agent behavior achitectures. The third section describes the IODA methodology and its advantages – like the separation between data and its processing, interactions libraries, or large-scale simulation construction. The fourth section presents the generic features of IODA concepts, through an easy to customize simulation platform called JEDI. Eventually, the last section concludes about IODA and JEDI.

2 Additional Related Work

Research on multiagent systems and on agent design is very active, and many generic agent description models do exist. Formal description methods and generic architectures for agents behavior can be examined from two points of view.

The first one is about interactions design : the way agents communicate with each others are extracted from their model into abstract communication patterns and protocols. Generally, this abstraction is limited to the model design step, and the interaction protocol and the agent's behavior are mixed together during implementation – like in JADE, AgenTalk, Swarm, etc. This leads to a decreased maintainability due to dispersal of the protocol's implementation. As proposed in [5], one solution is to abstract the interaction protocol from agents, and then reify it as a single entity defined by roles and messages sequences, which use functionalities that agents implement on their own according to their role.

[1] University of Lille, France, email: name.surname@lifl.fr

The second one is about agents behavior itself. Many generic methodologies stop at the formal specification of a simulation, giving place at worse to implementation errors and at best to mixing data (i.e. actions an agent can perform) and its processing (i.e. selection of an action given a particular valuation of the global state of the simulation).

Definition 1 *The* **global state of the simulation** *is the union of the set of all states of the environment and the states of all agents in the environment.*

Formal methods and architectures allow to keep the separation between data and processing with agent-independent actions, like in [3] where actions are agent-independent components, so that the behavior of an agent is defined by a set of interconnected components. This kind of solution is well suited to complex action scheduling, but the connectivity of these components decrease the maintainability of the agents, especially if their behavior change during simulation, or if the simulation is using a large scale knowledge representation.

Definition 2 *A simulation is called* **large scale simulation** *if its environment contains a great amount of agents (namely* **simulation with large scale computations***) or if it contains a large number of agents with different behavior and a large number of actions per agent (namely* **simulation with large scale knowledge representation***).*

In the following sections, we propose a formal method and an architecture providing the advantages of both interaction reification and separation between knowledge and processing, fitting large scale knowledge representation requirements with an homogeneous design of agents and interactions.

3 The IODA Methodology

In general, a communication protocol is used in order to describe a particular abstract process involving many agents, for instance *"to exchange goods"*. In order to build reusable and generic behavior, we present in this paper the IODA formal method and architecture. It relies on an homogeneous representation of actions performed by agents, called *Interaction*, close to the concept of *design/perceived affordance* of Norman [11]. This formal representation is adapted to represent actions involving only one agent as well as complex actions involving many communicating agents.

3.1 An Interaction-centered Methodology

The behavior of an agent is defined by a specific arrangement of semantic blocks called interactions (see § 3.5). An interaction is itself a set of primitives simultaneously involving a fixed number of agents, which describes how and under what kind of conditions agents may interact one with others or with the environment.

An agent owns a set of perception primitives – used to get information from the global state of the simulation – and a set of action primitives – used to change this global state (change the environment's, other agent's or his own local state). These are the atomic elements of interactions.

Definition 3 *An* **Interaction** *is a structured set of action primitives involving simultaneously a fixed number of agents. An interaction can occur only if the activation conditions – a boolean expression of perception primitives – are met.*

Definition 4 *Agents involved in an interaction generally do not play the same role. We make a difference between* **Source agents** *that may* perform *the interaction, and* **Target agents** *that may* undergo *it.*

As described in Def. 3, an interaction sets the logical sequence of primitives required in order to make agents interact. These primitives may be implemented differently according to the agent specificities. As a consequence, it leads to a more enhanced and easier-to-use polymorphism in agent behavior compared to other agent architectures like [3] where close behaviors cannot be expressed without complex means.

An interaction is not agent-dependent and may be re-used in other simulations. Thus, building simulations leads to the construction of interaction and agent libraries, and facilitates further simulation design.

3.2 IODA Agents

In IODA, agents follow a simple architecture which makes possible to design homogeneously agents with different specificities in the same simulation.

Definition 5 *An* **agent** x *is an autonomous entity of a simulation. Its minimal specification :*

- *has* **properties***;*
- *has a* **local state***, which is a valuation of its properties;*
- *implements a set of action and perception primitives;*
- *perceives other agents and the state of the environment only in a subset of the environment $\mathcal{H}(x)$ called* **halo***. The set $\mathcal{N}(x)$ of agents present in $\mathcal{H}(x)$ is called* **neighborhood***;*
- *is assigned a set of interactions it can perform or undergo (see § 3.5);*
- *implements an interaction selection process (see § 3.7).*

Definition 6 *An* **agent family** *(or* **agents equivalence class** *or* **agent class***) is an abstract set of agents, in which all agents share all or part of their properties, action or interaction primitives, or behavior.*

From this point on, if $S \in \mathbb{F}, x \in S$ means x is an agent from the S agent family.

A IODA agent is not restricted to a particular kind of agent. Programmers may freely define a cognitive or reactive interaction selection process, reactive or cognitive perception primitives, more or less complex neighborhood computations. Besides, neighborhood computation taken apart, the interaction selection process is independent from the environment's topology, and needs only a notion of distance between agents.

3.3 Interactions and cardinality

As its name implies, an interaction may occur between a source agent and a target agent. However, complex problems need to define other situations like the interaction of an agent with itself (*to sleep*, *to think*) or with the environment (*to move*, *to die*). Even more complex situations may occur, where interactions involve more than one source or target (for instance *to burst* involving many casualties). Cardinality (see Def. 7) unifies those notions.

Definition 7 *The* **cardinality** *of an interaction I is the pair $(card_S(I), card_T(I))$ where $card_S(I)$ (resp. $card_T(I)$) is the number of source agents (resp. target agents) involved in the interaction. Particular interactions where an agent interact with itself or with the environment, i.e. with $T = \emptyset$, are called* **degenerate interactions**.

Definition 8 *An interaction I is in* **normal form** *if and only if $card_S(I) = 1$.*

It has been shown that any interaction can be expressed into normal form [8]. Thus, in the following sections of this paper, interactions are supposed in normal form, mainly for complexity matters [8].

3.4 Problem analysis

In addition to the formal specification of simulations, IODA provides a set of algorithms to go from model analysis to concrete implementation. Those algorithms are demonstrated in the JEDI platform (see § 4) in the context of reactive and situated agents, but could also be implemented for any other kind of multi-agent system as well.

According to our methodology, the design of a simulation follows 5 steps :

1. Identify all ***agent families*** as well as all ***interactions*** of the simulation. It leads to the definition of a matrix between source agents and target agents containing interactions. This step is called "assignation of interactions to source and target agents".
2. Define all primitives needed to write the ***activation conditions*** and the ***action sequence*** of the interactions.
3. Identify the ***action and perception primitives*** that will be implemented by each agent family, and how they will be implemented.
4. Define for each assigned interaction I a ***priority*** $p(I)$ and a ***limit distance*** $d(I)$ (see § 3.7). It implies refining the initial matrix.
5. Define how the matrix evolves during simulation, i.e. if agents can change their own or other's behavior by changing a line or a colmun of the matrix.

To help the design of simulations, the assignation of interactions to source and target agents is summarized into a matrix called ***Interaction Matrix***.

3.5 The Interaction Matrix

Agents may interact only if target agents are present into the neighborhood of the source agent, but interaction is also constrained by a ***limit distance***. Indeed, seeing a target doesn't means a source agent may perform the interaction *to slap target* with it : it has to be close enough to the target, and this distance depends on the source agent's properties. This notion is independent of grid-like environments : it may be a Minkowski distance as well as a social distance, etc.

Additionally, every assigned interaction is endowed with a priority, so to build a hierarchy between them from the viewpoint of the source agent, which is used in the interaction selection process (see § 3.7). These priorities may be constant or dynamic, depending on the nature of the source agent.

Definition 9 *The* **assignation** $a_{S/T}$ *of an interaction set $(I_j)_{j\in[1,n]}$ between a source agent family S and a set of target agent families T describes the set of interactions that agents belonging to S may perform as sources together with sets of agents from T as targets. It is defined by a set of tuples $(I_j, p_j, c_j, d_j)_{j\in[1,n]}$, named* **assignation elements**, *where :*

- *I_j is an interaction that S can perform and all $x \in T$ can undergo;*
- *p_j is the* **priority** *of this assignation of interaction I_j;*
- *c_j is the* **interaction's cardinality** *(i.e. the number of awaited targets);*
- *d_j is the* **limit distance** *allowed between S and all $x \in T$ so that S may perform the interaction with T.*

N.B.: Elements of the assignation $a_{S/\emptyset}$ of degenerate interactions are (I_j, p_j) pairs.

Definition 10 *If $\mathbb{F}$ is the set of all agent families in a simulation, then the* **interaction matrix** *of the simulation is the set $M = (a_{S/T})_{S\in\mathbb{F},T\subseteq\mathbb{F}}$ of all assignations between all relevant source agent family S and target agent family set T, according to the behaviors to be modeled (see Fig. 1).*

3.6 Agent libraries

Because agents from different families may have some similar behavior, agents from an A agent family may be a particular subset of a B agent family. Thus, if $M = (a_{S/T})_{S\in\mathbb{F},T\subseteq\mathbb{F}}$ is the interaction matrix of a simulation, S and T may be abstract sets of agent families like groups, teams, etc. We define a particular algebra to specify the relations between agent families, especially how they share their assignation elements through 3 matrix modification operators :

Definition 11 *Let $\mathbb{F}$ be the set of all agent families.*

- *The* **specialization** *of an agent family X by a agent family Y is noted $Y : X$. It means agents of the Y family inherit all assignation elements, perception process, primitives and properties of the X family.*
- *The* **addition of an assignation element** *e with source agent family $S \in \mathbb{F}$ and target agent families $T \subseteq \mathbb{F}$ to the interaction matrix is noted $+(a_{S/T}, e)$.*
- *The* **suppression of an inherited assignation element** *e with source agent family $S \in \mathbb{F}$ and target agent families $T \subseteq \mathbb{F}$ is noted $-\left(a_{S/T}, e\right)$.*
- *The* **modification of an inherited assignation element** *$e = (I, p, c, d)$ with source agent family $S \in \mathbb{F}$ and target agent families $T \subseteq \mathbb{F}$ is noted $*\left(a_{S/T}, e, I', p', c', d'\right)$.*
- *The* **modification of an inherited assignation element** *$e = (I, p)$ with source agent family $S \in \mathbb{F}$ and target agent families $T \subseteq \mathbb{F}$ is noted $*\left(a_{S/T}, e, I', p'\right)$.*

Property 1 *Let $\mathbb{F}$ be the set of all agent families, $X, S, Y \in \mathbb{F}$, $T \subseteq \mathbb{F}$, e an assignation element, I, I' two interactions, $d, d' \in \mathbb{R}$ and $c, c', p, p' \in \mathbb{N}$.*

- *Generally, $(Y : X) \Rightarrow \left(\forall T \subseteq \mathbb{F}, a_{X/T} \subseteq a_{Y/T}\right)$*
- *$+\left(a_{S/T}, e\right) \Rightarrow e \in a_{S/T}$*
- *$-\left(a_{S/T}, e\right) \Rightarrow e \notin a_{S/T}$*
- *$*(a_{S/T}, (I, p, c, d), I', p', c', d') \Rightarrow ((I, p, c, d) \notin a_{S/T} \wedge (I', p', c', d') \in a_{S/T})$*

source \ target	$\emptyset$	Grass	Sheep	Goat	Wolf
Grass	+(Grow;0)				
Animal	+(Die;3) +(Move;0)				
Herbivore		+(Eat;2;1;0)			
Sheep:Animal,Herbivore			+(Breed;1;1;1)		
Goat:Animal,Herbivore				+(Breed;1;1;1)	
Wolf:Animal	*((Die;3),Die,4)		+(Eat;2;1;0)	+(Eat;3;1;0)	+(Breed;1;1;1)

Figure 1. Example of an interaction matrix for a predator/prey simulation with 4 species. The '$\emptyset$' column contains degenerate interactions. In this example, the '+' operator uses either one integer representing the degenerate assignation element's priority, or three integers representing the assignation element's priority, its cardinality and its limit distance. The '$*$' operator, in this case, is used to modify the priority of the inherited "Die" interaction for wolves.

- $*(a_{S/T}, (I,p), I', p') \Rightarrow ((I,p) \notin a_{S/T} \wedge (I',p') \in a_{S/T})$

In the interaction matrix, a cell is the intersection of a line, corresponding to the interactions that an agent of S family can perform, and a column, corresponding to the interactions that a set of agents of T families can undergo. Thus $a_{S/T}$ is implicit in the operators used in the matrix on Fig. 1. Such a formalism is platform-independent, especially the specialization notion which meaning changes along the programming language : inheritance for a language object, *kind of* in a frame language, etc.

3.7 Interaction Selection Basics

The core of an agent's behavior is the interaction selection process (see Def. 12). This process checks if activation conditions are met, finds targets to interact with, selects a particular set of targets, considers interactions with the correct priorities, and finally performs the sequence of actions.

Definition 12 Interaction selection *is the process an agent uses in order to select an interaction to perform (i.e. as a source) on particular targets given a particular valuation of the global state of the simulation.*

Both the ***eligibility*** syntaxic criterion and ***realizability*** semantic criterion, as well as the ***Interaction potential*** set are defined in this section to help the census of all possible interactions for a source agent x.

Definition 13 *Let $dist(x,y)$ be the distance between two agents x and y.*

The assignation element $e = (I_j, p_j, c_j, d_j)$ is said **eligible** *for the source agent x and the set T_{arg} of target agents – written $eligible(e, x, T_{arg})$ – if and only if $e \in a_{x/T_{arg}}$ and $card_T(I_j) \neq 0 \Rightarrow \Big(\forall y \in T_{arg}, y \in \mathcal{N}(x) \wedge dist(x,y) \leq d_j\Big)$.*

Definition 14 *Let $cond(I, x, T_{arg})$ be the activation conditions of the interaction I applied to the source agent x and the set of target agents T_{arg}.*

The assignation element $e = (I_j, p_j, c_j, d_j)$ is said **realizable** *for the source x and the set T_{arg} of targets – written $realizable(e, x, T_{arg})$ – if and only if: $eligible(e, x, T_{arg}) \wedge cond(e, x, T_{arg})$*

Definition 15 *The* **"p-level interaction potential" of an agent** *x – written $\mathcal{P}_p(x)$ – is the set of all realizable assignation elements with x as a source for any target set :*

$$\mathcal{P}_p(x) = \{(e,T), e = (I_e, p_e, c_e, d_e) \,|\, T \subseteq \mathcal{N}(x) \wedge p = p_e \wedge realizable(e,x,T)\}$$

As a consequence, interaction selection is the process where an x agent selects an element from $\mathcal{P}_{p_{max}}(x)$ where p_{max} is the highest priority such that $\mathcal{P}_{p_{max}}(x) \neq \emptyset$.

All the definitions and properties defined in this section are platform independent. Their implementation on a specific programming language implies many choices. We propose in the following section a possible implementation of IODA concepts in the Java language.

4 From Methodology to Implementation

The ***JEDI*** platform implements the formal concepts defined in IODA, which means there is an univocal path from problem analysis in IODA to implementation in JEDI. Besides, this transition between model and implementation is automated by a generator called ***JEDI-Builder***. Note that JEDI is more a proof of usefulness of IODA concepts than a regular simulation platform : the IODA methodology may be implemented in other languages, so we did in *Netlogo*.

4.1 Implementation Choices

Implementation choices define the scope of simulation models supported by a platform. Their consequences are displayed in [7], therefore this section does not argue in details about the reasons of those choices. In JEDI, these are :

- **Interaction cardinality is restricted.**
- **Simulation is in discrete time.**
- **Situated : simulation is in a two-dimensional grid.**
- **Everything is agent**, which allows an uniform treatment of *things* (called artifacts, objects, tools, patches, etc.) and *"true" agents* at implementation.

Definition 16 *An agent is said* **active** *if he can perform at least one interaction. Otherwise he is said* **passive**.

In JEDI, the only difference between passive and active agents depends on the interaction matrix. This homogeneous representation of agents makes transition of agents between passive and active easier.

Interactions are reified in a Java abstract class called `Interaction`. Each agent family $S \in \mathbb{F}$ – represented by a class inheriting from `Agent` – contains a set ***canPerform*** which is a part of the interaction matrix. It is defined such that $\forall x \in S, canPerform(x) = \{a_{S/T} \forall T \subseteq \mathbb{F}\}$. Thus each line of the interaction matrix is defined in an agent family. The abstract class `Moteur` is the core of the simulation, where the `run()` method executes the main algorithm of the simulation, i.e. performs every steps of the simulation (see Fig. 2).

Let A be the set of agents in the environment and $A_{act} \subseteq A$ the set of active agents.

1. Reorder A_{act} according to a particular criterion (see Sect. 4.2), for instance a random order (equitable choice);
2. Set all agents in A ***operative***;
3. For each operative agent $a \in A_{act}$ do :
   (a) Define the part of the environment $\mathcal{H}(a)$ perceived by a;
   (b) Define the set of all neighboring agents $\mathcal{N}(a)$, and remove from it all non-operative agents;
   (c) Let $p =$ maximal priority in canPerform(a);
   (d) Compute $\mathcal{P}_p(a)$; while $\mathcal{P}_p(a) = \emptyset$, decrement p and compute again;
   (e) If $p = 0$ and $P_0(a) = \emptyset$, then a cannot perform any interaction. It remains operative but ends its simulation step;
   (f) Else, select at an element from $P_p(a)$, i.e. elements $((I, p, c, d), T_{arg})$ containing an assignation element and a set of target agents, using the interaction selection process of the agent; for instance a random choice;
   (g) Perform the interaction I with a as source and T_{arg} as targets;
   (h) Deactivate a and all agents in T_{arg}.

Figure 2. Algorithm of a simulation step.

4.2 JEDI Tuning

In order to build simulations with large-scale computations, the programmer has to control the complexity of many parts of the simulation platform in order to find a tradeoff between performances and implementation bias. JEDI's modular decomposition defines a set of parameters for this purpose :

- **Agents' halo** $\mathcal{H}(x)$ may be defined at will as a set of cells.
- **"P-level interaction potential" computing complexity** (3f in Fig. 2) may be reduced if needed, though it may introduce a bias in the evaluation order of assignation elements and target sets; for instance a census of only one target set T_{arg} per assignation element e.
- **Interaction Selection process** may easily be customized by writing how to select an element from $\mathcal{P}_p(x)$.
- **Pseudo parallelism** may be tuned by the order according to which agents are evaluated (1 in Fig. 2), knowing what kind of bias are introduced [10].
- **Interaction matrix** is a shared object between agents when is not modified during the simulation.

5 Conclusion

Designing a simulation is the art of finding a tradeoff between model precision – in order to implement the model without any ambiguities – and model universality – in order to easily implement it on any simulation platform. Most simulation platforms neglect one of those points and sometimes do not even clearly define the model they use.

In this paper we have presented a formal method and an architecture for the design of multiagent simulations, called IODA, which uses an homogeneous representation of actions performed by agents named *Interaction*. Actions involving a single agent, or complex actions involving many communicating agents, are both represented with the same formalism. As a consequence of this, the interaction selection process is also the same for all agents, and can be defined independently from both agents and interactions. Knowledge and processing are not mixed, therefore the user is able to build reusable agent and interaction libraries along with simulations. Moreover, the interaction matrix helps to design simulations with large-scale knowledge representation, and to build automatically the corresponding implementation through a code generator.

The JEDI simulation platform provides a simple implementation tool of IODA models, and defines an interaction selection process suitable to reactive, cognitive or any other kind of agents. In addition, it points up a set of parameters that can be tuned at will. This aims at controlling implementation bias when adapting the complexity of the platform to match with large-scale computations requirements.

Acknowledgements

This research is supported by the FEDER and the "Contrat-Plan État Région TAC" of Nord-Pas de Calais.

REFERENCES

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y Qin, 'An integrated theory of the mind', *Psychological Review*, **111**(4), (2004).
[2] Nourredine Bensaid and Philippe Mathieu, 'A hybrid and hierarchical multi-agent architecture model', in *Proceedings of the Second International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, (april 1997).
[3] Jean-Pierre Briot, Thomas Meurisse, and Frédéric Peschanski, 'Une expérience de conception et de composition de comportements d'agents à l'aide de composants', *L'Objet, Revue des Sciences et Technologies de l'Information*, **12**(4), (2006).
[4] R. Burkhart, 'The swarm multi-agent simulation system', in *Position Paper for OOPSLA'94 Workshop on 'The Object Engine'*, (1994).
[5] Takuo Doi, Yasuyuki Tahara, and Shinichi Honiden, 'IOM/T: an interaction description language for multi-agent systems', in *AAMAS'05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM, (2005).
[6] Olivier Gutknecht, Jacques Ferber, and Fabien Michel, 'Integrating tools and infrastructures for generic multi-agent systems', in *Proceedings of the Fifth International Conference on Autonomous Agents*, eds., Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, Montreal, Canada, (2001). ACM Press.
[7] Yoann Kubera, Philippe Mathieu, and Sébastien Picault, 'La complexité dans les simulations multi-agents', in *Actes des Journées Francophones sur les Systèmes Multi-Agents (JFSMA07)*, ed., Cépaduès-Editions, Carcassonne, France, (2007).
[8] Philippe Mathieu, Sébastien Picault, and Jean-Christophe Routier, 'Donner corps aux interactions (l'interaction enfin concrétisée)', in *Actes de la conférence MFI'07*, Paris, France, (2007).
[9] Philippe Mathieu, Jean-Christophe Routier, and Pascal Urro, 'Un modèle de simulation agent basé sur les interactions', in *Actes des Premières Journées Francophones sur les Modèles Formels de l'Interaction (MFI'01)*, Toulouse, France, (2001).
[10] Fabien Michel, Jacques Ferber, and Olivier Gutknecht, 'Generic simulation tools based on mas organization', in *Proceedings of the 10 European Workshop on Modelling Autonomous Agents in a Multi Agent World MAMAAW'2001, Annecy, France*, (2001).
[11] Donald A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988.
[12] Uri Wilenski, 'Netlogo', Technical report, Center for Connected Learning and Computer-Based Modeling, (1999).