IODA: An interaction-oriented approach for Multi-Agent Based Simulations

Yoann Kubera, Philippe Mathieu, Sébastien Picault

▶ **To cite this version:**

Yoann Kubera, Philippe Mathieu, Sébastien Picault. IODA: An interaction-oriented approach for Multi-Agent Based Simulations. Journal of Autonomous Agents and Multi-Agent Systems, 2011, 23 (3), pp.303-343. 10.1007/s10458-010-9164-z . hal-00825534

HAL Id: hal-00825534
https://hal.science/hal-00825534v1

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.

Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-010-9164-z

IODA: an interaction-oriented approach for multi-agent based simulations

Yoann Kubera · Philippe Mathieu · Sébastien Picault

© The Author(s) 2011

Abstract Multi-Agent Systems (MAS) design methodologies and Integrated Development Environments exhibit many interesting properties that also support simulation design. Yet, in their current form, they are not appropriate enough to model Multi-Agent Based Simulations (MABS). Indeed, their design is focused on the functionalities to be achieved by the MAS and the allocation of these functionalities among software agents. In that context, the most important point of design is the organization of the agents and how they communicate with each other. On the opposite, MABS aim at studying emergent phenomena, the origin of which lies in the interactions between entities and their interaction with the environment. In that context, the interactions are not limited to exchanging messages but can also be fundamental physical interactions or any other actions involving simultaneously the environment and one or several agents. To deal with this issue, this paper presents the core notions of the *Interaction-Oriented Design of Agent simulations* (*IODA*) approach to simulation design. It includes a design methodology, a model, an architecture and also *JEDI*, a simple implementation of *IODA* concepts for reactive agents. First of all, our approach focuses on the design of an agent-independent specification of behaviors, called interactions. These interactions are not limited to the analysis phase of simulation: they are made concrete both in the model and at the implementation stage. In addition, no distinction is made between agents and objects: all entities of the simulation are agents. Owing to this principle, designing which interactions occur between agents, as well as how agents act, is achieved by means of an intuitive plug-and-play process, where interaction abilities are distributed among the agents. Besides, the guidelines provided by *IODA* are not limited to the specification of the model as they help the designer from the very beginning towards a concrete implementation of the simulation.

Y. Kubera (✉) · P. Mathieu · S. Picault
Laboratoire d'Informatique Fondamentale de Lille, LIFL CNRS UMR 8022, Université Lille 1—Sciences et technologies, Cité Scientifique, 59655 Villeneuve d'Ascq Cedex, France
e-mail: yoann.kubera@lifl.fr

P. Mathieu
e-mail: philippe.mathieu@lifl.fr

S. Picault
e-mail: sebastien.picault@lifl.fr

Published online: 15 January 2011

Keywords Architecture · Framework · Interaction · Design methodology · Model · Multi-agent based simulation

1 Introduction

Computer simulation is a powerful tool for various applications, including understanding or explaining the mechanisms at the origin of an emergent phenomenon, predicting how a phenomenon will evolve, building artificial life/societies inspired by an observed phenomenon (e.g., immersion in a driving simulator) or mimicking a phenomenon for leisure use (e.g., video games).

Despite having different goals, all these simulations are built using the same general process called in this paper the *simulation design process*. First, relevant elements from the phenomenon are made abstract and captured in a formal representation called a *model of the phenomenon*. Then, this model is implemented as a computer simulation using a particular language or simulation framework.

1.1 Major issues of the simulation design process

The simulation design process induces many major issues. Firstly, models are designed by domain specialists—like biologists or social scientists—whereas implementation is handled by computer scientists. Because the model and the implementation are built by two different actors, the construction of simulation requires a translation to go from model to implementation. During the translation process, information can be missed or wrongly interpreted. Both lead to erroneous computer simulations that must be avoided.

The simulation of a complex system may involve many different entities and a great variety of interactions between them (such simulations are called in this paper *large scale simulations*). Modifications ad revisions of both the model and the implementation are usual due to the iterative simulation design (see Sect. 2). It becomes a major issue if modifications in the model implies a complete re-implementation of the simulation.

Different unexpected results might be observed when running a simulation. In such cases, domain specialists and computer scientists have to check whether the differences comes from a wrong implementation of the model, from a wrong or incomplete model or from a wrong abstraction of the phenomenon. This phase is made easier if the structure and the terminology of the model is kept for the implementation.

The only complete and not ambiguous description of a simulation is its implementation in a programming language. These languages are inseparable from computer science and are consequently poorly accessible to domain-specialists. Therefore, the simulation design process has to involve such languages as late as possible in order to involve domain specialists as much as possible during the design process. To reach this goal, generic abstractions of phenomena have to be identified and used to build the model and the implementation.

These issues have made Multi-Agent Based Simulations (MABS) preponderant among simulation tools. Indeed, the notion of "*agent*" matches the notion of "*entity in the phenomenon*" and provide to domain specialists "*(potentially) formal yet more natural and transparent descriptions of the target system*" [29].

1.2 MABS design issues

Many different frameworks are used to design agents. Frameworks like Netlogo [65] or Repast Simphony [52] are based on a simple programming paradigm and are designed for non-computer scientists. These frameworks remove computer scientists from the design process and thus comply with some requirements of simulation design. Yet, they do not provide means to facilitate the revision of the model. Thus, such approaches are not fit to design large scale simulations since without an appropriate implementation structure, a revision is likely to involve an almost complete reimplementation of the simulation.

On the opposite, generic and open agent frameworks like Swarm [64], Madkit [35], JADE [4] or Magique [5] offer software refinements to design agents—like design patterns, inheritance, etc.—but require to design the behavior of agents directly with algorithms. Since large scale simulations involve a great variety of agents and interactions, the model to build contains a great amount of information. To ease its construction, model design has to be guided by an appropriate abstraction.

In this context, which abstraction is the most appropriate for the simulation of a phenomenon having emergent properties, knowing that such emergent properties generally arise from the—direct or indirect—influence the agents have on each others? Current approaches are task-centric, which means they place the focus on the design of the agents and the tasks the agents can perform. The influence of one agent on another is left to be implicitly expressed within the action during the simulation. Usually, these approaches are well suited for the design of a MABS (see Sect. 2). However, a large-scale simulation implies a large variety of agents and subsequently a great complexity of influences. In order to facilitate the design process, these influences must be easily represented and intuitively allocated to the agents. We answer this particular problematic by introducing an interaction-based methodology where the influences between the agents (referred to as interactions) constitute the core of the design. In this methodology, interactions are not limited to speech acts or communication protocols but consider any influence an agent can have on another (physical or chemical reaction, procreation, eating, etc).

1.3 Towards an interaction-oriented design of agent simulations

We describe in this paper an interaction-oriented approach to simulation design where interactions are predominant. This approach considers that every action of an agent is part of an interaction—i.e., a structured set of actions involving simultaneously two agents of the simulation or an agent and its environment. Moreover, we uphold that interactions have to be designed independently from the agents and have to be made concrete independently from the agents at the implementation stage. In that context, every entity of the simulation is an agent endowed with interaction capabilities.

These features have many advantages regarding the requirements described above. Indeed, an agent may participate in a set of interactions which are not specifically developed for it. Most interactions can be re-used in many other simulations of the same application field. Moreover, the declaration of the agents' abilities (what an agent can do) is separated from the declaration of its (inter)action selection process (how it chooses what interaction and with whom to interact). The agents—and more generally all the entities in the simulation—are almost empty shells where interaction abilities and an interaction selection process are plugged. This modular structure enables the design of heterogeneous agents (i.e., purely reactive agents, purely cognitive agents or hybrid agents) yet using a homogeneous structure.

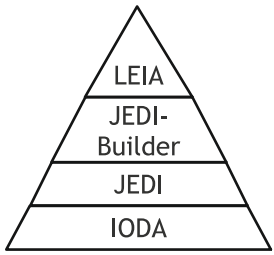


Fig. 1 Our interaction-oriented approach to simulation design consists of the four tools presented in this figure. It contains the *IODA* formal model and methodology, the *JEDI* simulation framework, the *JEDI-Builder* design IDE and the *LEIA* simulation space explorer

The concrete separation between the agents and their interactions brings computer simulations closer to the domain-specific terminology. Besides, it enables the definition of the interactions occurring in the simulation by means of filling a matrix, in a plug and play fashion. This matrix simplifies the definition and the modification of the layout of all interactions in a simulation.

A set of key features have to be defined in order to provide usable tools for MABS design (see Sect. 2). In our interaction oriented approach, called *Interaction-Oriented Design of Agent simulations* (*IODA*), these features are (see Fig. 1):

- the *IODA model* describes what elements are contained in an interaction-oriented model, how they are managed, and more generally how knowledge is represented in the model;
- the *IODA modeling methodology* provides intuitive means for domain specialists to build interaction-oriented formal models of a phenomenon;
- the *JEDI framework* (which stands for *Java Environment for the Design of agent Interaction*) illustrates the interaction-oriented approach in a simulation framework dedicated to reactive agents and discrete time simulations. It shows that an interaction-oriented design can really be kept at the implementation level and thus benefit of *IODA* features, even at runtime;[1]
- the *JEDI-Builder IDE* where *IODA* formal models can be built with the *IODA* modeling methodology using a Graphical User Interface and then automatically implemented onto the *JEDI* simulation framework.

The *IODA* approach represents knowledge differently from the usual action-centric specification of agents. It makes possible to consider simulation design from a different perspective and leads to both theoretical and practical results.

The most significant result is the *LEIA* [27] (for *LEIA* lets you Explore your Interactions for your Agents) simulation space explorer, which aims at reversing the design process used to build simulations. In classical simulation design, a candidate explanation of the origin of an emergent phenomenon is expressed in a model and then assessed with simulations. On the opposite, *LEIA* offers tools to create models randomly from an interaction library (i.e. the interaction layout between entities and the behavior of entities are generated randomly) until one of them produces results similar to the data extracted from the studied phenomenon. It is then possible to backtrack directly to the model that produced these results and find the candidate explanation of the origin of the emergent phenomenon.

The design perspective of *IODA* also enables eliciting and explicitly handling some biases underlying any simulations that otherwise are implicitly handled in existing approaches [43,44].

This paper focuses on the description of the *IODA* approach, which is at the origin of these results. More precisely, it presents the *IODA* formal model, the *IODA* modeling methodology,

[1] See http://www2.lifl.fr/SMAC/projects/ioda/jedi/demonstration.php.

and finally the *JEDI* simulation framework. It is structured as following. First, related works are studied to underline the main issues of simulation and the motivations of our proposition. Next, the formal model of *IODA* is described. The *IODA* modeling methodology used to build models is then described and *IODA* features are illustrated on a short case study. Finally, the algorithms implementing *IODA* models and the *JEDI IODA*-compliant framework are described.

2 Related work

In this section, we introduce the main notions related to simulations, we identify the issues that designers have to face in order to build simulations and we define the context of our proposition. Then, existing solutions to build simulations are discussed along two points of view: the process used to build simulations and the knowledge representation used in existing Agent-Based frameworks. Relying on this study, we finally present our motivations to create an interaction-oriented approach to simulation design and what features this approach has to exhibit.

2.1 Short introduction to computer simulation

Computer simulation is a tool used for various applications [3,22]. Because of these multiple applications, no agreement is reached regarding the process used to conduct simulations or the taxonomy used to describe simulations [25,55]. Consequently, *we focus on the identification of the general layout of the activities involved in simulation rather than on a precise and dogmatic simulation process*.

The main idea of simulating [3,37,61,62] is to "*design a model of a real system, and conduct experiments with this model*". The model provides an abstraction of what is being modeled (i.e., the real system), retains only certain features considered as relevant, makes assumptions about unknown aspects and simplifies other aspects [29]. The model is the core notion of simulations: it provides a formal description containing enough information to be implemented on a simulation framework.

Like real experiments, virtual experiments with computer simulations require an experimental protocol. We call this protocol *the simulation process* in this paper. This protocol involves many different activities, including the construction of the model. Its comprehension is a prerequisite for the identification of simulation issues.

2.2 The simulation process

No consensus exists about the description of the simulation process [1,33,34,63]. However, these articles altogether provide an insight into the activities the simulation process contains.[2] (see Fig. 2) This process starts with identifying the aims of the simulation ("*problem definition*"). Secondly, elements from the real system that are relevant for that problem are captured into an informal description ("*system definition*") and then into a formal model ("*conceptual model formulation*"). Thirdly, the nature of the model input data is studied along with observations of the real system and used to refine the formal model ("*preliminary experimental design*"). Then, the model is translated into a computer program ("*model trans-*

[2] Please note that this representation makes no consensus and is only used for illustration.

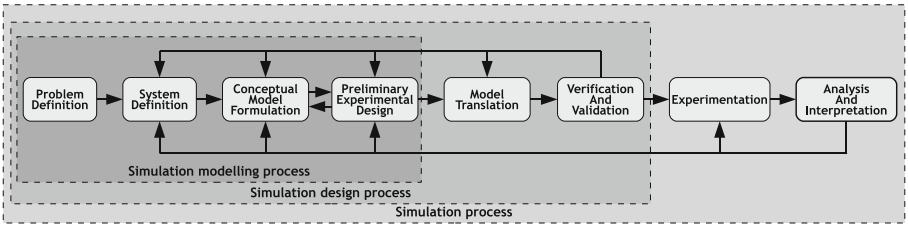


Fig. 2 Summary of the simulation process mainly inspired by Shannon's description in [63]. The decomposition in the "simulation modeling process", "simulation design process" and "simulation process" comes from our interpretation

lation"). Once the implementation is assumed to be correct ("*verification and validation*"), simulations can be carried out like real experiments by domain specialists, without the intervention of computer scientists.

The focus in this paper on a subset of the simulation process: the activities going from problem specification to the concrete implementation—i.e., the result of verification and validation. We label this subset the *simulation design process*. In the next section, we present the main issues encountered during this process.

2.3 Main issues of the simulation design process

Two major issues have to be considered during the simulation design process: (1) to avoid unexpected results or at least to detect their origin in order to correct them; (2) to support efficiently the design iterations.

2.3.1 Unexpected results issue

Unexpected results may be observed during the analysis of experiments. At best, the cause of these results would only be wrong assumptions in the model, i.e., a model that provides a wrong abstraction of the real system. In practice, this is not the only cause. These results can also come from wrong assumptions made during the implementation, because the model does not provide enough information to be implemented. They also may come from a wrong translation of the model which was not detected by the verification and validation phases [3,23,28,29]. Identifying and avoiding unexpected results coming from these additional sources is the first major issue of simulation design since it is caused by an inappropriate communication between domain specialists and computer scientists.

A first solution to this issue is found in [6,12,21,29]. It advocates that many different actors (also called roles) have to be involved in the simulation design process. These actors have specific knowledge about simulation, computer science, or both. They build different models that gradually lead to the concrete implementation of the simulation. Using models as communication support, they avoid errors by sharing their experience. A second solution is found in [65]. It upholds that a single actor (the domain expert) has to be involved in the simulation design process. This actor has to be guided from the problem definition to the implementation with simple guidelines and intuitive tools which prevent him making errors.

Although the ideas underlying these solutions are contradictory, they lead to the same requirements. First, the simulation design process has to be supported from problem definition to the implementation. Secondly, model and implementation have to use abstrac-

tions from real systems as much as possible. Indeed, it will provide common and intuitive concepts for all actors and an almost direct implementation of the model.

2.3.2 Design iterations issue

The second issue is encountered because a simulation is not a straightforward process leading from a real phenomenon to results diffusion (see Fig. 2). During the "*verification and validation*" phases, errors may be detected either in the model or in the implementation of this model [60]. It leads to model revision and consequently to changes in its implementation. A revision of the model also occurs during the "*analysis and interpretation*" phase. Primarily to deal with unexpected results, and secondly to remove from the model elements that have no influence on emergent properties of the phenomenon [3,59].

Model revision must be made simple and efficient. This requires a software structure fit to reuse as many elements as possible [38].

2.4 Why using multi-agent based simulation?

Multi-Agent Systems (MAS) provide a "*powerful suite of metaphors, concepts and techniques for conceptualizing, designing and implementing complex systems*" [40]. They provide significant support to software reuse, and hence to iterations of the simulation process. Moreover, the notion of agent is close to the notion of entity in a real system and provides a common abstraction to domain specialists and computer scientists. This abstraction is kept in the model and in its implementation as well.

Because they help solving both major issues of simulations, MABS have become outstanding among simulation tools.

2.5 The simulation design process for multi-agent based simulations

To provide guidelines supporting MABS design, some works add MAS design related activities to the simulation process [12,16,34]. The description of these activities being general and not providing any description of the formal model, no simulation framework or IDE implement explicitly these steps.

Notations introduced within the FIPA [24] specifications or AUML [53] can be considered as means to express formal models. Their implementation in frameworks like JADE [4] is a well addressed problem. These solutions rely on representations close to UML and are consequently dedicated to computer scientists. It hardens the construction of formal models for domain specialists.

Other frameworks like Repast Simphony [52], SeSAm [41] or Netlogo [65] provide simple programming languages or graphical tools to specify the behavior of agents, which are accessible to non-computer scientists. These intuitive specifications remove computer scientists from the design process and partially solve the translation issues mentioned in the previous sections. However, this kind of framework can only be used for the design of small projects. Indeed, in bigger simulations containing a great variety of agents, which have a great variety of interactions, directly designing the agents' behavior is a difficult task, because of the many elements that have to be taken into account all at once. The design of these simulations requires appropriate design guidelines that support the gradual construction of a model and its implementation. In this paper, we refer to this kind of "big" simulations with the term "large scale simulations"

In the last decade, some efforts have been made in that respect, with approaches relying on modeling methodologies like Tropos [8], Prometheus [66], O-MAsE [30], Gaia [67], Agile-PASSI [14], INGENIAS [57], or ADELFE [6]. These approaches both provide precise models to design a MAS and guidelines that describe how such models can gradually be built from very abstract specifications to the concrete model of each agent. Most of these methodologies are supported by Integrated Development Environments (IDE) which provide graphical tools to make model design more intuitive and which implement automatically such models on agent frameworks. For instance, consider the TAOM4E framework [26] for Tropos, the PDT framework [56] for Prometheus, the agentTool III framework [31] for O-MAsE, etc. Some works like the CAFnE toolkit [39] also showed that a concrete and functional application could be completely designed by domain specialists.

The methodologies we mentioned above are firstly meant to design applications that provide services to users. Consequently, in these methodologies, the model of the MAS can be filled freely as long as functionalities of the MAS operate correctly. Such an approach has flaws when it comes to simulations in application domains like biochemistry or ethology. Indeed, in that context, a simulation is used as a tool to reinforce or to weaken hypothesis about the origin of an emergent phenomenon. Such hypothesis are expressed by a model, and then implemented as a simulator. Consequently, the contents of the model (i.e., a formal representation of an hypothesis and of known aspects of a phenomenon) and its accurate implementation are as important as the results of the experiments.

Some works led by Pavón [58] show that MAS design methodologies and IDEs can be used to model and implement a simulator. Yet, the authors acknowledge that MAS design methodologies are not appropriate enough to build a valid model and that MABS specific methodologies and IDEs are needed. According to them, "agent concepts [have to be] wrapped and represented in terms of application domain concepts", especially when it comes to action modeling [58]. We also subscribe to this point of view, but believe that this issue is not only due to the way actions are handled in the model and the implementation, but also to the way interactions are. Indeed, the methodologies mentioned above are meant to design applications that provide services to users or that exhibit specific functionalities. In this context, agents are heterogeneous entities having to cooperate and interactions between agents are structured message exchanges using interaction protocols. In simulations, interactions have a wider meaning. For instance, in the study of an ecosystem, "a predator eats a prey" is also an interaction between a predator and a prey. Describing this interaction as mere exchanges of messages between the predator, the prey and the environment is restrictive. We consider that interactions are a set of structured actions involving simultaneously the environment and one or more agents. Therefore we think they have to be at the core both simulation design and agents' behavior specification.

To find a more suitable abstraction of the interactions for MABS, we have to pay attention on how existing MAS and MABS architectures represent interactions and how interactions are supported in the behavior of agents.

2.6 Interactions in MABS

Many agent frameworks—like Madkit [35], Mason [47] or Swarm [64]—are open, generic and ready to implement any kind of agents. However, no guidelines are provided to design their behavior: it can be any algorithm of any complexity, with no constraint on implementation. As a consequence, the probability of obtaining erroneous implementation is high.

The frameworks where agents' behavior is designed without directly using algorithms are separating what the agents are able to do (i.e., actions and interaction declaration) from how they choose what to do (i.e., action selection process).

Finding a suitable abstraction of interactions for MABS requires an explicit—and action-selection independent—representation of the interactions and a definition of agents behavior that relies on this representation. To achieve this, we have to understand how the behavior of agents is usually designed and how interactions must be defined to support action-selection processes.

2.6.1 Existing agent behavior architectures

The representation of an agent's actions depends on its architecture. Our main concern here is to describe how actions are represented in those architectures. We do not focus on the action selection process.

In cognitive architectures, agents have to reason about what they are able to do, in order to fulfill goals they have. In this case, the separation between the declaration of the agents' abilities—i.e., what agents are able to do—and the action selection process—i.e., how agents choose what to do—is natural.

There are mainly two kinds of cognitive architectures. The first one is the *deliberative planning* architecture. It is found in frameworks like Act-R [2], Icarus [15] or Soar [46]. It represents an action with rules (also called procedural rule, operator, etc.), containing at least two elements: *preconditions* that describe the conditions required to perform the action, and *effects* that describe the state reached after performing the action. Agents have goals to achieve (a state to reach, or an action to perform) and build dynamic plans (action trees) to reach them. The action they perform is chosen among those moving them forward in their plans.

The second one is called the *reactive planning* architecture and appears in PRS-based [32] frameworks like JAM [36], JACK [11] or Jadex [7]. It represents actions by patterns of behavior containing at least three elements: a *goal* satisfied by this pattern, a *context* required to be true in order for the pattern to be triggered and a *body* containing all the actions performed by the pattern. Unlike deliberative planning architectures, where plans are built automatically by the system from actions, plans are static and written by humans as patterns of behavior. A pattern of behavior consumes a goal of the agent (the "goal" part of the pattern) and produces sub goals (when performing the "body" of the pattern).

In reactive architectures, agents act only in response to stimuli and do not reason about the actions they perform. As a consequence, even though some architectures keep the representation of action abilities as condition/actions rules (*e.g.* in the MANTA project [20]), most reactive architectures do either not define an explicit representation to agents' action abilities or are based on a representation of entities behavior as a set of interconnected elements. The most famous is the subsumption architecture [10], where entities' behavior is modeled as a set of interconnected computational machines called modules. In this architecture, modules are similar to the action abilities of an entity. Thus entities' behavior design consists in interconnecting modules in order to define how modules are activated – or inhibited—after the activation of another module.

This architecture is aimed at designing robots. It is adapted in more abstract architectures like the component-based architecture MALEVA [9], in order to design agent based simulations. In these architectures, an action ability of an entity is modeled as a software component describing the effect of an action. These components provide a structure able to express and reuse complex actions, which is a rare property for reactive agent architectures. Other reac-

tive architectures, like SeSAm [41], depict the behavior of agents as a control flow graph, where action abilities are nodes represented as computational machines. Therefore, unlike cognitive architectures where action abilities are represented as rules, reactive architectures rely on very different action representations.

MABS make no restrictions on the nature of agents: they might be reactive, cognitive, or have both characteristics at the same time. Reactive and cognitive agents might also coexist in the same simulation. Using very different action representations to design these agents makes the simulation design process less robust to model revisions and action reuse. Indeed, both a cognitive agent and a reactive agent might be endowed with the ability to "pick up" an object in the environment. Having two different action representations requires to define, for the same action, two different representations, which causes consistency issues. Moreover, when the simulation process iterates, a reactive entity might become a cognitive one (and conversely). In such a case, all action abilities have to be defined again with the appropriate representation.

Layered architectures like InteRRaP [51] bridge the gap between these architectures to build agents both reactive and cognitive. These frameworks allow for the design of heterogeneous agents within the same simulation, yet relying on a single architecture. However, the representation of each action depends on which layer it is used: on the reactive layer, agents manipulate patterns of behavior whereas they manipulate rules on the deliberative layer. That way, knowledge representation depends on the layer on which it is declared.

Although action selection in reactive agents, deliberative planning agents or reactive planing agents are different, a homogeneous structure of actions can be defined. Indeed, the MANTA project shows that even reactive architectures can rely on *(condition, actions)* like rules. Yet, the contents of the rules have different specificities depending on the reactive or cognitive nature of the action selection process using it. For instance, patterns of behavior require the identification of the goal consumed by the action whereas reactive actions do not.

Although the representation of an action has to contain information related to cognitive or reactive action selection, its interpretation in the model remains the same. For instance, "to eat" keeps the same meaning whether it is used in a complex plan built by an agent or used by a hungry animal: it reduces the hunger sensation and either harm the eaten entity or remove it from the environment. We uphold that any action can have an abstract description remaining valid for all kinds of action selection processes. In addition, this description can be made independent from agent specificities. For instance, the previous definition of "to eat" makes no assumptions on how the hunger sensation is reduced. It can consist in incrementing the energy of the agent, resetting its period of fasting, *etc.* Providing an action representation that keeps this property favors the gradual conception of simulations. Actions can be designed independently from agents specificities and are re-usable in many contexts. Currently, no architecture supports this property.

Actions in a simulation might be designed using a homogeneous representation, almost independently from the reactive or cognitive nature of agents. It favors action reuse during the revision of the model or the implementation.

2.6.2 Existing interaction representations

The rare systems to explicitly design the interactions are organization-centered multi-agent system (OCMAS) like Madkit [35] and FIPA-compliant [24] platforms like JADE [4]. In those approaches, an interaction is *an action performed by an agent which consists in sending a message to one or more other agents*. The aim of interactions is to provide a common support for interoperability between heterogeneous distributed systems. An inter-

action follows a particular syntax like the KQML language and is part of a communication protocol (an interaction protocol). Such a protocols is used by the agents to provide and use services through the coordination of distributed heterogeneous agents. In these protocols, the agents have different roles and send or react to different messages depending on their roles.

This definition of interaction as speech acts is also used in MABS, for instance in coordination, cooperation or collaboration between agents [50]. It is also found in methodologies like SODA [54]. In this case, an interaction is part of "*the joint efforts of independent communicating actors towards mutually defined goals*" [48].

A protocol is a semantic block of informations whose consistency has to be preserved at the implementation stage in order to avoid code dispersion. Thus, protocols have to be integrated in these MAS as reified entities, independently from agents. As Doi states in [19], most implementations of interaction protocols lack readability because only the reply to particular messages is considered, without providing a view on the whole organization of the protocol. As for the description of actions mentioned in the previous section, interactions can have an overall and agent-independent description. IOM/T [19] takes this into account in interaction description to provide generic, reusable and easy to interpret interactions for agents. However, this description lacks the necessary information to be part of the agents' behavior: no conditions to the execution of interaction are mentioned.

Other approaches, mostly in application fields like biochemistry, consider interactions in their literal meaning, as the fundamental interactions from the physical laws. This is for instance the case of [13,17]. Although these interactions are represented as (*condition/action*) rules and represent actions involving simultaneously two or more agents, they are not part of the action selection process of the agents. Indeed, physical laws always apply independently from agent decisions. Thus, this representation cannot be used in the general case to design agents behavior.

2.7 Summary

In this section, the study of works related to MAS, simulation MABS identified what simulation consists in and what issues have to be dealt with during their construction.

Efficient tools do exist to design small simulations. However, they cannot be used to design large scale simulations—i.e., simulations containing a great variety of entities and interactions. Studying existing works emphasized properties that potential solutions have to exhibit to tackle more intuitively and efficiently the construction of large scale simulations:

- gradual design support during the whole simulation design process;
- a formal model that uses abstractions from real systems as much as possible;
- ways to keep the abstractions of the formal model at implementation;

The notion of agent is an intuitive abstraction of the entities found in real systems and made MABS preponderant among simulation design approaches. MAS design methodologies offer gradual design support for the whole simulation design process, but focus mainly on organizations and agents messages exchange. Yet, simulations are inseparable from emergent phenomena, which originate from the interactions between entities, in a broader sense.

Current solutions to MAS and MABS design refer to interactions either as communication or coordination between entities or as the result of physical laws like gravitation. Any other interaction is modelled as an action. Furthermore, in classical approaches, the only agent explicitly identified in an action is the agent at the origin of the action references to other agents must be made manually throughout the algorithm coding the action.

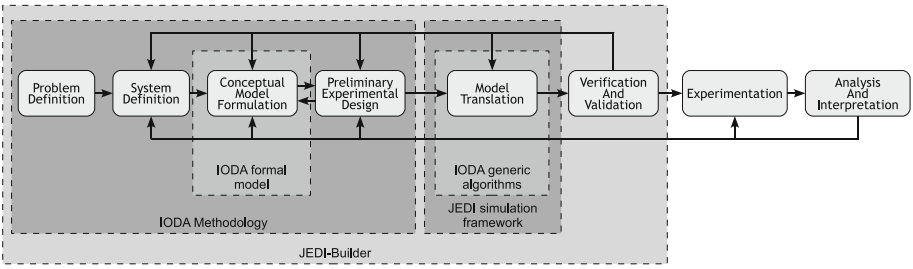


Fig. 3 Depiction of the different elements of the *IODA* approach and the way they relate to each other

On the opposite, we maintain that an interaction must be generic as to support the communication, coordination or any actions between two or more agents. Furthermore, we maintain that the agents participating in such an interaction must be explicitly identified. Besides, We uphold that such interactions have to be considered first during the simulation design process.

The study of related works identified what properties for the representation of interaction are mandatory to be as important as agents during the simulation design process:

- interactions must have a homogeneous representation, as a *(condition, actions)* or *(objective, context, actions)* rule;
- interactions must be declared independently from agents at implementation;
- interactions must be independent from the action selection process of agents;
- interactions must give an agent-independent description of the entities it involves;
- agents must have the ability to choose whether to initiate or not an interaction.

There is currently no such description of interactions and no methodological frame to exploit this representation is outlined in current works. In this paper, we propose an interaction-oriented approach to MABS design that provides a better abstraction for models and implementation. This approach—called *Interaction-Oriented Design of Agent simulations* (*IODA*)—features a set of components listed below and depicted in Fig. 3.

- an interaction-oriented formal model called *IODA*;
- the *IODA* modeling methodology which provides design guidelines and graphical specifications to involve domain specialists as much as possible during the simulation design process;
- algorithms that keep the structure of the model even at implementation and almost make automatic the implementation of the models built with the *IODA* modeling methodology;
- a simulation framework—called *Java Environment for the Design of agent Interaction* (*JEDI*)—that illustrates *IODA* features in a framework dedicated to reactive agents and discrete time simulations;
- a MABS design IDE that relies on the *IODA* methodology to support simulation design during the whole simulation process (from model specification to the implementation on the *JEDI* framework).

3 Presentation assumptions and presentation context

The *Interaction-Oriented Design of Agent simulations* (*IODA*) approach provides a fundamentally different way of specifying agents than the traditional approaches mentioned in the previous section. This paper builds on [42] in which the design principles of the *IODA*

approach were outlined. This paper provides a more detailed description of the formal model updated with definitions of the environment and the update matrix. An illustration of the modeling methodology is also provided, along with descriptions regarding the graphical representations of the model within this methodology.

The only requirement to use such a model is that the environment has a metric (*i.e.* the notion of distance between two entities can be defined).

Building a very different approach to MABS design requires to define many notions, algorithms and design tools. To avoid their proliferation, we describe the *IODA* approach in a simplified context and focus on its core notions:

- simulations are in discrete time;
- conflicts—for instance two agents trying to pick up the same item—are solved with an up stream measure: during a time step, agents act in a random sequence;
- agents initiate at most one interaction per simulation time step;
- interactions only involve up to two agents;
- the decisions of agents are made with a reactive process.

These statements are not a limitation to the interaction-oriented approach but merely a choice of representation. The simplified context avoids issues that have obviously to be dealt with, but are out of the scope of this article.

In this paper, we focus on the description on how the *IODA* approach can be used to design reactive agents. Some previous references from our team deal with how an interaction-oriented design can be applied to other kinds of simulations: [18] supports an interaction-oriented design of cognitive agents; [44] addresses simultaneous interaction issues; and [43] addresses more complex reactive selection processes.

4 The *IODA* simulation model

This section describes the core of the *IODA* approach: the *IODA* formal model. Since no accurate terminology makes consensus in MABS, a definition to every key notion is provided for clarity purpose.

4.1 What is an environment?

In *IODA*, an environment is a space populated with entities. The environment defines a metric in order to define the distance between two agents. The distance is not limited to Euclidean spaces: it can represent a social distance between agents (for instance a difference of wealth), a distance in a directed graph (the minimal number of links separating the nodes) or any other distance measure in a topological space.

Since the environment can be modified by interactions (for instance through the addition or the removal of an agent, through moves, *etc.*), we consider that an environment defines a set of primitives which make possible to manipulate its contents. These primitives have to contain at least four elements: a primitive that computes the distance between two agents, a primitive that adds an agent to the environment, a primitive that removes an agent from the environment and a primitive that returns the set of all agents within the environment.

Definition 1 (*Environment*) The *environment* $\mathcal{E}$ of a simulation is a space populated with agents. It is defined as a monomial $\mathcal{E} = \langle primitives \rangle$, where *primitives* is a set of specified primitives.

In *IODA*, primitives are defined with a structure similar to methods in Object-Oriented Programming.

Definition 2 (*Primitive and Specified Primitive*) A *primitive* is a computational machine identified as a tuple $\langle name, returnType, parameters \rangle$, which is similar to the signature of a method in Object Oriented Programming.
A *Specified primitive* p is a pair $p = \langle primitive, specification \rangle$, where $primitive$ is a primitive, and $specification$ is an algorithm describing what the primitive does.

A call to a primitive (e.g., "getWidth()") of the environment is written "Environment. getWidth()".

4.2 What is an interaction?

IODA relies on a homogeneous representation of the actions performed by the agents, called *Interactions*. An interaction is a semantic sequence of actions involving a fixed number of agents simultaneously. It describes how and under what conditions agents may interact with each others or with the environment. The conditions of an interaction are the conjunction of:

- *preconditions* that describe the logical or physical conditions required to initiate the interaction. For instance, "*The position where the source wants to move is empty*" for a MOVE interaction, or "*The target is not rotten*" for an EAT interaction;
- *triggers* that describe the teleonomic part of the conditions—i.e., explicit or implicit goals this interaction aims at:
  - *explicit goals* are agents memory elements consumed by the interaction in order to move the reactive plan forward, like in reactive planning architectures;
  - *implicit goals* are stimuli reduced the interaction. For instance "*The source is hungry*" for the EAT interaction, since its implicit goal is to reduce the hunger sensation of the source.

An interaction can be performed only if both its trigger and its preconditions are true. The overall structure of an interaction is presented in Fig. 4a.

4.3 Interactions polymorphism

Despite their differences, two behaviors observed in a real system can have the same abstract definition. Indeed, anyone can tell if a wolf and a human are eating just by observing their behavior on a video, even though they do not act exactly the same. This identification is

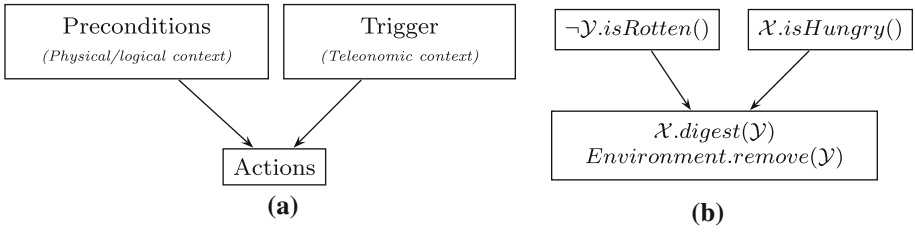


Fig. 4 General structure of an interaction **a** applied to the example of the EAT interaction **b**, which describes how an agent $\mathcal{X}$ eats an agent $\mathcal{Y}$. The ¬ symbol corresponds to a negation

possible because the overall layout of interactions can be described independently from the specificities of entities. We uphold that interactions in MABS should be designed according to this principle.

For instance, even if the two interactions described in Fig. 5 have agent dependent specificities, their general semantics follow the same pattern (described in Fig. 4b): they check if the source agent is hungry and if the target is not rotten. The actions include to remove the hunger sensation and to remove the target agent from the environment. Consequently, they might have a common generic description.

The generic description of interactions manipulate abstract pirmitives instead of directly using agent states. Agents that can participate in an interaction will implement these primitives according to their own specificities. For instance, in Fig. 4b, the abstract primitive `isHungry` may mean:

- "*To have its energy under a particular threshold*", like in INTERACTION1 in Fig. 5;
- "*To have been starving for the last four days*", like in INTERACTION2 in Fig. 5;
- "*To have a stress attribute over a particular threshold*", etc.

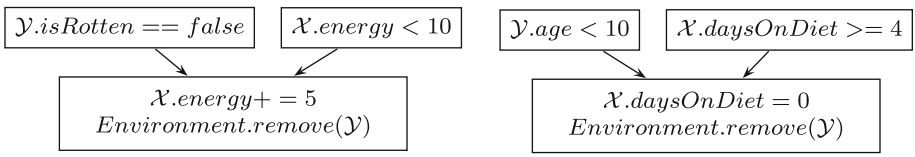
Interactions are described as generic entities, independently from agents. At the same time, they can be tuned with agent specificities–i.e., provide a kind of polymorphism—without loosing their generic description. Thus, building simulations leads to the construction of generic interaction libraries. These libraries can be re-used in a wide set of simulations from the same application field, even if agents have very different specificities. Consequently, the more simulations are designed, the faster further simulation design becomes (see Sect. 5.1).

4.4 Interaction typology

Agents involved in an interaction generally do not play the same role. Difference is made between agents that initiate the interaction—for which the participation in the interaction comes from their own action selection process—and agents that undergo the interaction—which are chosen by the action selection process of source agents.

Definition 3 (*Source/Target Agents*) The *Source* agents of an interaction are agents participate in an interaction because of their own action selection process—i.e. the agents that initiate the interaction. The *Target* agents are chosen by the action selection process of source agents—i.e. the agents that undergo the interaction.

The cardinality defines how many source and target agents participate in an interaction:



(a) Description of INTERACTION1 **(b)** Description of INTERACTION2

Fig. 5 Description of two agent dependent interactions—called INTERACTION1 and INTERACTION2—which occur between two entities X and Y. These figures **are not** a *IODA*-compliant description. An equivalent *IODA*-compliant description is presented in Fig. 4b

Definition 4 (*Interaction Cardinality*) The *cardinality* of an interaction $\mathcal{I}$ is the pair $(card_S(\mathcal{I}), card_T(\mathcal{I}))$ where $card_S(\mathcal{I})$ (resp. $card_T(\mathcal{I})$) is the number of source (resp. target) agents involved in $\mathcal{I}$.

The most usual case of interaction is an interaction with $(1, 1)$ cardinality—i.e., interactions involving a single source agent and a single target agent. For instance, interactions like EAT, where a source agent eats a single target agent to reduce its hunger (see Fig. 4), or PICKUP, where a source agent gets a target agent and puts it in its inventory.

Complex problems may define other situations like the interaction of an agent with itself—for instance to REST—or with the environment—for instance to MOVE. Calling these interactions may seem questionable since they involve a single agent—i.e., their cardinality is $(1, 0)$. However, their structure is identical to regular interactions and they do have an implicit target. To homogenize the definition of agents abilities, such elements are called *degenerate interactions*.

Definition 5 (*Degenerate Interaction*) In a reflexive interaction—i.e., an interaction that targets the source itself—or an interaction involving only the environment, the target is implicit. Such an interaction, which cardinality is (1,0), is called a *Degenerate Interactions*.

Situations where interactions involve more than one target agent may occur:

– a Pheromone that DIFFUSES onto its 8 neighboring Pheromones ((1, 8) cardinality);
– an Enzyme that JOINS together two Molecules ((1, 2) cardinality).

Others may require more than one source agent:

– a heavy Table that has to be CARRIED by four Movers ((4, 1) cardinality);
– two Wolves may PROCREATE ((2, 0) cardinality). As a result, one of the sources becomes gravid.

IODA is a formal methodology fit to design simulations with any kind of interactions, as long as they are expressed in the *normal form*. This restriction is necessary to express the behavior of an agent in the generic algorithms of Sect. 6.

Definition 6 (*Normal Form*) An interaction $\mathcal{I}$ is in the *normal form* iff $card_S(\mathcal{I}) = 1$.

Interactions not in the normal form—e.g., involving more than one source—denote a coordination problem. Turning an interaction into the normal form requires maintaining one source as the source of the interaction and moving the $card_S(\mathcal{I}) - 1$ remaining sources as targets. This seems to violate the principle of agent's autonomy and might look as an inappropriate choice. However, coordination is never met spontaneously between agents: an agent first has to initiate the coordination process with the other agents. This agent will be the source of the normal form interaction. Thus, transforming interactions with a (n, m) cardinality in normal form makes sense as long as the interaction is triggered under the agreement of the $n - 1$ other sources.

Particular interactions can involve a number of target agents depending on the situation of the source agent: for instance broadcast or multi-cast interactions. Their only difference with fixed cardinality interactions is that the amount of target agents is dynamic. The targets of a broadcast interaction are all perceived agents and the targets of a multi-cast interaction are a subset of perceived agents. This subset is identified with an acceptation criterion defined in the interaction. Apart from this, their description is managed exactly as interactions with an $(1, n)$ cardinality, where $n \in \mathbb{N}$.

IODA is currently designed for interactions with $(1, 0)$ and $(1, 1)$ cardinalities. Consequently, from this point on, notions are presented for only these two cardinalities. We are

currently extending *IODA* to other cardinalities. Although using only interactions with (1, 0) and (1, 1) cardinalities seems restrictive, a wide set of simulations can be designed this way.

4.5 Formal definition of an interaction

According to the notions described in the previous sections, we define an interaction as in Def. 7. This structure is similar to team plans that can be found in JACK [11], yet used differently. On the opposite of JACK, where interactions are limited to the coordination within an explicit team of agents, interactions in *IODA* model anything an agent initiates.

Definition 7 (*Interaction*) An interaction $\mathcal{I}$ is a semantic block that describes how and under which conditions an agent can interact with another agent or with the environment. It is represented as a tuple $\mathcal{I} = \langle name, card, labels, primitives, preconditions, trigger, actions \rangle$ where:

- $name$ is an explicit name which qualifies what the interaction;
- $card = (card_S(\mathcal{I}), card_T(\mathcal{I}))$ is the cardinality of the interaction;
- $labels$ is a set of names used to identify each agent involved in the interaction. Thus, $|labels| = card_S(\mathcal{I}) + card_T(\mathcal{I})$;
- $primitives$ is a function that associates each element in $labels$ to a set of primitives it has to specify;
- $preconditions$ is an algorithm manipulating primitives which describes the preconditions of the interaction;
- $trigger$ is an algorithm manipulating primitives which describes how the interaction is triggered;
- $actions$ is an algorithm manipulating primitives which describes the actions of the interaction.

In our context, interactions have only one source and at most one target. To design homogeneously interactions, we impose that $labels = \{Source, Target\}$ in interactions with (1, 1) cardinality, and that $labels = \{Source\}$ in degenerate interactions. In the preconditions, the actions and the trigger of the interaction, calling primitive $isHungry$ defined in the agent labeled $Source$ is written "Source is Hungry()".

The interaction's cardinality can be identified from the number of elements in $labels$. Moreover the $primitives$ function can be identified by analyzing the contents of the preconditions, the trigger and the actions of the interaction. Thus, interactions can be represented with a graphical element such as illustrated by Table 1.

Table 1 Graphical representation of an interaction $\mathcal{I}$, whose name is "eat", whose labels are $\{Source, Target\}$ and whose preconditions, trigger and actions are described as algorithms

EAT($Source, Target$)	
Trigger	Source is Hungry()
Preconditions	¬ Target is Rotten()
Actions	Source digest(Target);
	Environment remove(Target);

Table 2 Raw Interaction Matrix of an ecosystem simulation, where wolves, sheep, goats and grass agents evolve

Source \ Target	$\emptyset$	Grass	Sheep	Goat	Wolf
Grass	(Grow)				
Sheep	(Move)	(Eat, d = 0)	(Procreate, d = 1)		
Goat	(Move)	(Eat, d = 0)		(Procreate, d = 1)	
Wolf	(Move)		(Eat, d = 3)	(Eat, d = 3)	(Procreate, d = 1)

The element *(Eat, d = 0)* at the intersection of the row starting with *Sheep* and the column starting with *Grass* is read "Sheep agents are able to initiate the Eat interaction with a target Grass agent at a maximal distance of 0 from the source"

4.6 The interaction matrix

Owing to the agent-independent description of interactions, the separation between source and target agents, and the restriction to normal form interactions, an overview of the interactions occurring in the simulation can be displayed in an *Interaction Matrix*. The matrix defines what interactions can be initiated by an instance of an agent as a source together with another instance of an agent as a target (see Table 2).

The interaction matrix sums up all the interactions occurring in the simulation. Degenerate interactions have no explicit target. Consequently, a particular column—labeled $\emptyset$—is added in order to display all degenerate interactions a source agent might initiate.

4.6.1 Limit distance

An agent only interacts with agents that are close enough. This distance—called *limit distance*—not only depends on the interaction but also on the agents that participate in the interaction. For instance, a sheep may EAT grass agents only at a distance of 0 (only at their feet), but a chameleon may EAT insect agents at a greater distance (using its tongue). Consequently, each cell of the raw interaction matrix does not contain interactions but *assignation elements*.

Definition 8 (*Assignation Element*) An *assignation element* corresponds to the representation, in the interaction matrix, of an interaction that a source agent is able to initiate. Assignation elements can either:

- represent a degenerate interaction $\mathcal{I}$ that the source agent is able to initiate. These assignation elements are called *degenerate assignation elements*, and are written $(\mathcal{I})$ in the matrix, in the column $\emptyset$ or,
- represent a non-degenerate interaction $\mathcal{I}$ that the source agent is able to initiate with a particular target agent. The interaction is possible only if the distance between these agents is lower or equal to a *limit distance* dist. This assignation element is written $(\mathcal{I}, d = dist)$—or $(\mathcal{I}, dist)$.

4.6.2 Agent families and instances of agents

In *IODA*, difference is made between *instances of agents* and *agent families*.

Definition 9 (*Separation between Agent Family and Instance of Agents*) An *agent family* is an abstract specification of agents having the same primitives, interaction abilities and behavior. An *instance of agent*[3] instantiates at least one agent family. From this point on, let $\mathbb{F}$ be the set of all agent families of a simulation.

[3] In the remaining sections of this paper, the word "agent" alone means "instance of agent".

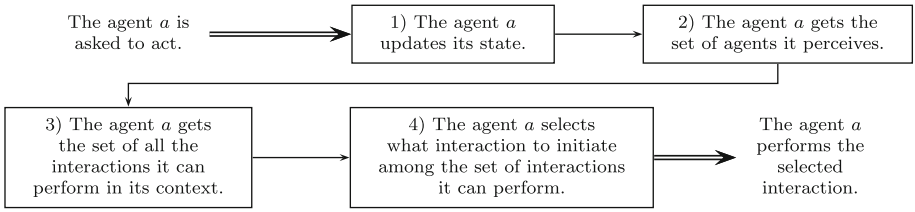


Fig. 6 Outline of the process that rules the behavior of an agent

During a simulation, interactions occur between instances of agents. Nevertheless, the interactions they are able to initiate or undergo are defined by their agent family. Thus, only agent families are specified in the interaction matrix as sources or targets.

The word "family" is used instead of "type" or "class" to describe this concept because "family" is part of a terminology used in ethology and is close to the notion of "species".

4.6.3 Matrix definition

The *raw interaction matrix* is the set of all possible *assignations* of the simulation:

Definition 10 (*Assignation*) The *assignation* $a_{\mathcal{S}/\mathcal{T}}$ represents the set of interactions that an agent from the family $\mathcal{S}$ is able to initiate with a target agent from the family $\mathcal{T}$. The elements of $a_{\mathcal{S}/\mathcal{T}}$ are non-degenerate assignation elements.
The set of degenerate interactions an agent from the family $\mathcal{S}$ is able to initiate is labeled $a_{\mathcal{S}/\emptyset}$, and contains only degenerate assignation elements.

An assignation represents a cell of the raw interaction matrix.

Definition 11 (*Raw Interaction Matrix*) The *raw interaction matrix* $\mathcal{M}$ represents all the interactions that occur between the different agents during a simulation. It is the aggregation of all possible assignations of the simulation: $\mathcal{M} = (a_{\mathcal{S}/\mathcal{T}})_{\mathcal{S}\in\mathbb{F},\mathcal{T}\in\mathbb{F}} \cup (a_{\mathcal{S}/\emptyset})_{\mathcal{S}\in\mathbb{F}}$.

An illustration of he interaction matrix of a toy problem is provided on Table 2. This matrix defines the possible interactions between entities in an ecosystem simulation where the different species are Grass, Sheep, Goat and Wolf.

4.7 Agent families in *IODA*

In *IODA*, agents are represented homogeneously. It enables the design of a wide set of interactions independently from the reactive, cognitive or hybrid nature of the agents. An agent is an autonomous entity instantiated from an agent family. Agents act according to a particular generic process in four steps, which is outlined in Fig. 6.

4.7.1 State update (step 1)

The state of the agents may evolve independently from the interactions in which they participate. For instance, an agent may grow old, or keep track of what it has perceived since its birth—i.e., have an internal representation of the environment—etc. In *IODA*, such a process is defined in an agent family with an update matrix. Such a matrix is similar to a raw interaction matrix where only degenerate assignation are defined.

Definition 12 (*Update matrix*) The *update matrix* $\mathcal{U} = (u_S)_{S \in \mathbb{F}}$ describes which degenerate interactions are performed in order to update the state of the agents. This matrix contains degenerate assignation elements.

4.7.2 Perception (step 2)

An agent only interacts with the agents it perceives. In *IODA*, the perception process relies on the notion of *halo*. An agent only perceives its neighborhood—i.e., the agents in its halo.

Definition 13 (*Halo/Neighborhood*) The *Halo* of an agent $\mathcal{A}$ is the process that tells how to compute the set of agents perceived by $\mathcal{A}$. The *Neighborhood* of an agent $\mathcal{A}$ is the set of all agents retrieved by its halo—i.e, it is the set of perceived agents.

The description of an agent's halo depends on what kind of environment is used in the simulation. In most cases, the halo defines the subset of the environment in which agents are perceived. This halo depends on the nature of environment's topology: a surface in a two dimensional space, a volume in a three dimensional space, acquaintances in a contact network, etc. The perception is not restricted to direct perception in the environment. It can also include the memory of the agent, the inventory of the agent, *etc.* Since the structures of the inventory, the memory or the environment are domain-dependent, *IODA* does not constrain the specification of the halo.

4.7.3 Get the set of all interactions the agent might initiate (step 3)

The set of all the interactions an agent can perform in a particular context depends on both its abilities—i.e., what interactions it is able to initiate with target agent families—and the agents in its neighborhood. This set is called *realizable tuples* set. A *realizable tuple* represents either:

- a degenerate interaction which conditions hold true for the source agent, or
- a pair containing an interaction and a target agent, so that the conditions of the interaction hold true for the source agent and target agents.

A formal definition of a realizable tuple is provided in Sect. 6.1.1.

4.7.4 Select the interaction the agent will initiate (step 4)

The interaction selection step consists in determining what interaction is performed by the source agent and with which agent the interaction is performed. It corresponds to the selection of a particular realizable tuple. The modalities of this selection depend on whether the agent is cognitive, hybrid or reactive.

In *IODA*, we have implemented a reactive interaction selection process (see Sect. 4.10). Yet, *IODA* is compatible with any interaction selection processes. We illustrate this point in Sect. 4.11, by showing how a reactive planning interaction selection process can be modeled without changing the structure of an agent family.

4.7.5 Formal definition

In *IODA*, cognitive, reactive and hybrid agents are designed with the same modular structure. This structure is meant to be implemented as such, in order to easily modify parts of an agent family only (*e.g.* only change its halo) without modifying the other parts of the agent family.

Definition 14 (*Agent Family*) An *agent family* $\mathcal{F}$ is defined as a tuple $\mathcal{F} = \langle name, halo, interactionMatrixRow, updateCell, primitives, attributes, selection \rangle$, where:

- $name$ is the *name* of the agent family;
- $halo$ is the *halo* of the agent family, which defines how the agent perceives its neighbors;
- $interactionMatrixRow$ is the *row of the interaction matrix* that defines the interactions the agent is able to initiate as a source;
- $updateCell$ is the *Cell of the update matrix* that defines how the state of the agent evolves independently from the interactions the entity initiates or undergoes;
- $primitives$ is the set of specified primitives this agent family defines. The specification of these primitives manipulates agent attributes;
- $attributes$ is the set of *attributes* this agent family defines;
- $selection$ is the *model of the interaction selection process* of this agent family. It defines how agents from this family act.

The primitives an agent family has to specify are defined by the interactions in the raw interaction matrix. If the family can be the source of an interaction then it has to specify the primitives from the $primitives(Source)$ set of that interaction. If the family can be the target of an interaction then it has to specify the primitives from the $primitives(Target)$ set of that interaction.

The interactions are summarized as a whole matrix during the model specification phase only. Each row is thereafter dispatched into the corresponding source agent family. Since the rows only are required to define what a source agent initiates (see Fig. 6.1.2), the columns of the matrix are not included in agent families.

This definition is independent from how these families are handled in the framework. They can be a concrete class in an object language, prototypes in prototype-based programming, a data structure in procedural programming, a component, etc.

4.8 Agents and inheritance

In object-oriented languages, the inheritance has two meanings. It is firstly meant to denote a *is-a* relationship. This relationship is also used for a class to inherit from *all* methods and attributes of its parent class—i.e., to do code factoring.

In *IODA*, to avoid any misinterpretation the *is-a* relationship is called *agent specialization*. This relationship is used to describe a semantic link between two agent families and to create abstract groups of agent families. For instance a Wolf and a Sheep are both Animals. Thus the Wolf agent family specializes the Animal agent family. This semantic link between agent families enables the use of a single assignation element to describe interactions that might otherwise require many assignation elements. For instance, a Hunter might HUNT Animals indiscriminately.

The relation between the specialization of an agent's family and the interactions that can be initiated or undergone by the instance of an agent family is as follows. If an agent family $\mathcal{F}_A$ specializes an agent family $\mathcal{F}_B$ (*i.e.* $\mathcal{F}_A$ is-a $\mathcal{F}_B$) then each agent from the family $\mathcal{F}_A$ can initiate and undergo at least the same interactions as agents from the family $\mathcal{F}_B$. This relationship is represented in the interaction matrix by the string "$\mathcal{F}_A : \mathcal{F}_B$".

Property 1 Let $\mathcal{F}_A \in \mathbb{F}$ and $\mathcal{F}_B \in \mathbb{F}$ be agent families. Then:

$$\mathcal{F}_A : \mathcal{F}_B \Rightarrow \begin{cases} \forall \mathcal{F} \in \mathbb{F}, a \in a_{\mathcal{F}/\mathcal{F}_B} \Rightarrow a \in a_{\mathcal{F}/\mathcal{F}_A} \\ \forall \mathcal{F} \in \mathbb{F}, a \in a_{\mathcal{F}_B/\mathcal{F}} \Rightarrow a \in a_{\mathcal{F}_A/\mathcal{F}} \\ a \in a_{\mathcal{F}_B/\emptyset} \Rightarrow a \in a_{\mathcal{F}_A/\emptyset} \end{cases}$$

An application of this relationship to model design is described in Sect. 5.2.

4.9 Time representation

Time representation has an influence on the contents of the model. Indeed, in accurate time representations like the approximation of continuous time with a discrete event model, an agent has to define precisely the dates when it has to act. Every time it performs an interaction, the agent has to anticipate when the interaction will be finished, and thus when it will have to decide again what to do. This time representation requires to mention the duration of an interaction and thus requires a more complex model. In this article, we made the choice not to introduce such specifications. Consequently, *IODA* is described in the context of discrete time simulations where time is divided in atomic time intervals called time steps. During a time step, each agent has the opportunity once to initiate an interaction.

During a time step, interactions might interfere one with another. For instance, two agents might want to pick up the same object. Solving such conflicts first requires detecting which interactions are in conflict and then solving the conflicts with specific policies. Conflict detection and solving has to be defined in the model since the outcome of a simulation can greatly change depending on how conflicts are solved. Solving a conflict often means selecting randomly one among the interactions in conflict. In *IODA*, we choose to model time with an almost equivalent representation: entities act in a sequence which is shuffled at each time step. This solves in a similar way conflicts, yet without having to detect them.

If an agent can perform more than one interaction per simulation time step, then the model has to be completed with information concerning how much interactions it can initiate or under which conditions more than one interaction can be performed. In order to avoid such specifications, we consider in this article that an agent can initiate at most one interaction per simulation time step. This is a presentation choice, not a limitation of *IODA*.

4.10 Reactive interaction selection

In this section, a reactive interaction selection process is described. This process is inspired from the subsumption architecture [10] and EMF [20]. Every assignation element of the interaction matrix is given a *priority*—an integer number—whose value represents the priority of the interaction for the source agent. The higher the integer, the higher the priority.

The behavior of an agent starts with the computation of all realizable tuples in the context. Since every tuple is related to a particular assignation element of the interaction matrix, every tuple has a priority. A reactive interaction selection consists in selecting the tuple that has the highest priority among the set of realizable tuples. If more than one tuple share the highest priority, a tuple is selected through a selection policy. By default, this policy selects a random tuple.

Consequently, the behavior of reactive agents is defined by providing a priority for every assignation element of the raw interaction matrix. It leads to the construction of more complete matrix, called the *Refined Interaction Matrix*.

Table 3 Refined interaction matrix for the ecosystem simulation presented in Table 2

Source \ Target	$\emptyset$	Grass	Sheep	Goat	Wolf
Grass	(Grow, p = 0)				
Sheep	(Move, p = 1)	(Eat, d = 0, p = 3)	(Procreate, d = 1, p = 2)		
Goat	(Move, p = 1)	(Eat, d = 0, p = 3)		(Procreate, d = 1, p = 2)	
Wolf	(Move, p = 1)		(Eat, d = 3, p = 3)	(Eat, d = 3, p = 4)	(Procreate, d = 1, p = 2)

Definition 15 (*Refined Interaction Matrix*) A *refined interaction matrix* is a matrix where each assignation element is given an integer value as priority. The higher the number, the higher the priority.

In case of the ecosystem simulation presented in Table 2, the refined interaction matrix as presented in Table 3 shows that if a Sheep agent first tries to EAT Grass agents. If it cannot EAT at least one Grass agent, it tries to PROCREATE with another Sheep agent. Finally, if it cannot PROCREATE with at least one other Sheep agent, it tries to MOVE.

Priorities can be interpreted as preferences between targets of the same interaction. For instance, in Table 3, a Wolf agent prefers EATing Goat agents (priority 4) rather than Sheep agents (priority 3).

4.11 How to design other interaction selection processes?

In this section, we illustrate shortly how the *IODA* formal model can be used to define a model of interaction selection based on the reactive planning architecture.

This architecture is centered on the concept of goal, which models something the agent wants to achieve. Goals are modeled as tokens. An interaction is performed either because it fulfills a goal of the agent or because it helps decomposing a goal of the agent into subgoals.

This interaction selection process can be modeled as a tuple $\langle goals, consume, produce, goalPref, assignEltPref \rangle$. In this structure, *goals* are the goals the entity wants to achieve. If the structure contains more than one goal, the agent uses a function called *goalPref* to determine the goal it will try to achieve. Once the goal is selected, the *consume* function is used in order to get the set of assignation elements that can achieve the goal. Its implementation can be a data structure that maps assignation elements to a goal. Then, realizable tuples for these assignation elements are listed and one of them is selected with the $assignEltPref$. Once this selection is made, the actions of the interaction for this tuple are performed. Then, the function $produce$ is used to identify (with a principle similar to $consume$) the subgoals introduced by the interaction. These goals are finally added to the current goals of the agents.

This structure enables designing reactive planning agents using the same agent family model than for reactive agents. Indeed, only the interaction selection model changes. Thus, *IODA* is able to design both reactive and cognitive agents.

4.12 Conclusion

This section has presented the structure of an interaction-oriented model of a MABS.

As for any MABS model, this formal model is worthwhile as long as domain specialists—i.e., biologists, chemists, etc.—are involved as much as possible in its conception. To cope with this issue, *IODA* provides a modeling methodology that guides the design of this model with simple guidelines. These guidelines were used in particular to build an applica-

tion—called *JEDI-Builder*—that helps the construction and the implementation of a *IODA* model for reactive simulations.

5 The *IODA* modeling methodology

The formal model—as a technical specification of the simulation – cannot be defined directly by domain experts. Yet, their involvement in its construction is a prerequisite to obtain valid simulation results. The *IODA* modeling methodology copes with this issue with simple specifications that gradually build the model of the simulation.

5.1 Problem analysis

The *IODA* modeling methodology guides the construction of a *IODA*-compliant model with abstract representations that are gradually refined. This process consists of 7 steps:

1. Specify:
   - how time is represented in the simulation;
   - what distance and halo stand for—i.e., how the environment is represented;
2. List the different interactions occurring in the simulation;
3. List the entities involved in the simulation. Each entity is represented by an agent family. During this step, the specialization relationship between agent families has to be identified and represented by a graph where nodes are agent families and edges represent specializations between two agent families;
4. Fill the raw interaction matrix of the simulation:
   - with interactions between source and target agent families and set their corresponding limit distance;
   - with interactions for a source agent family into the $\emptyset$ column of the matrix;
   - an interaction in the $\emptyset$ column of the matrix becomes a degenerate interaction. Otherwise, it becomes an interaction with (1, 1) cardinality;
5. Fill the update matrix:
   - with interactions for a source agent family into the corresponding cell of the update matrix;
   - an interaction in the update matrix becomes a degenerate interaction;
6. Specify the interactions:
   - write the interactions trigger, preconditions and actions. This leads to the identification of the different abstract primitives that agents have to implement in order to participate in the interaction. *Accurate interactions are easier to interpret, but at the same time less generic. Thus, during that process, the modeler has to find a trade-off between generic and expressive interactions* or,
   - get an already existing interaction from the interaction library for which the trigger, the preconditions, the actions and what primitives agents have to implement are already specified;
7. Specify the interaction selection process of the agents. For reactive agents, it corresponds to providing a priority to every assignation element of the interaction matrix;
8. Specify for the agent families involved in the simulation:

- the abstract primitives the family has to implement with algorithms. These primitives depend on the interactions the agent family can initiate or undergo. If the family can initiate an interaction then it has to specify all primitives contained in the set $primitives(Source)$ of that interaction. Conversely, if the family can undergo an interaction then it has to specify all primitives contained in the set $primitives(Target)$ of that interaction;
- the attributes of the agents manipulated by these primitives;
- the halo of that family, depending on how the environment is represented;
- the initialization primitive of this family, using algorithms.

These steps do not represent an absolute and dogmatic sequential process. Although every step has to be clearly identified, all are in practice intermingled during the specification of the simulation. For instance, an alternative sequence can be as follows. Two agent families and one interaction are first identified. The interaction is added to the matrix and specified in detail. Subsequently, other interactions are identified, specified, added to the matrix and so on. The only requirement is to build the interaction matrices and specify the interactions before focusing on the agents structure.

5.2 Case study: a strategy game in a nutshell

This section illustrates how the *IODA* methodology can be used to design a simple strategy game with reactive agents.

As Axelrod states in [3], "*the basic problem is that it is hard to present a social science simulation briefly*"[4] and "*Articles and chapters are often just not long enough to present the full details of the model*". As mentioned in the introduction, *IODA* aims at easing the design of large scale simulations. Such simulations are described using different graphical representations for each agent families and interactions. The amount of these graphical representations—and the length of the whole thought process leading from the definition of the problem to the resulting model—are incompatible with such space constraints. Consequently, we deliberately choose in this section to provide an insight on how the modeling process is dealt within *IODA* with the following restrictions:

- the results of each step of the methodology are presented only;
- the simulation contains 16 interactions and 9 agent families (this simulation barely qualifies as a large scale one);
- only a subset of these interactions are accurately described;
- the initialization of the simulations is not described.

Please refer to our website[5] for a more didactic application of this methodology without a straightforward application of the seven steps of section 5.1. *IODA* is currently used to design the behavior of customers in a virtual store [45]. Since these applications require knowledge in specific application fields, they are not described in this paper.

5.2.1 Brief description of the simulations

The simulation reproduces a simple strategy game where towns compete against each other for the total control of the land. Each town gathers resources to build new buildings and to train people for particular professions.

[4] Although this statement is aimed at social sciences, we believe that it remains valid for other application fields.

[5] http://www2.lifl.fr/SMAC/projects/ioda/ioda/models.php.

A town is characterized by a forum where persons are trained to particular functions like peasantry or soldiery. The forum can also decide where buildings like farms should be located in the environment. Peasants harvest resources they perceive—food stored in farms, gold or trees lying in the environment—and store them into their forum. If they perceive a farm being constructed, they will try to finish its construction—and report it to the forum—before harvesting resources. Warriors attack persons or buildings from other towns. Trees replicate themselves in the environment (to represent forest growth).

5.2.2 Interaction-oriented problem analysis

This section summarizes how the simulation described above is modeled using the *IODA* modeling methodology.

Step 1: Specify what distance stands for. Since the interaction matrix manipulates a notion of limit distance, knowing what distance stands for is required. In our case, the environment is a two dimensional space where agents occupy a ground surface. Thus, the distance between two agents is the Euclidean distance between their respective surfaces.

Step 1: Specify what time stands for. In this case study, we make the arbitrary choice that time is discrete as described in Sect. 4.9.

Steps 2 and 3: identify agent families and interactions. An interaction oriented interpretation of the description made in Sect. 5.2.1 reads as follows:

A town is characterized by a Forum where all Resources are STORED. The Forum can TRAIN Persons for professions like Peasant or Warrior. It can also PLACE A NEW FARM in the environment. All Persons are able to WANDER in the environment. If Peasants perceive a Resource, they GO TOWARDS it, and when they are close enough they HARVEST the Resource. Once they cannot carry any more Resources, they GO TOWARDS their Forum in order to STORE the Resources they are carrying. If they perceive a Farm in construction they will no longer search for Resources, GO TOWARDS the Farm and then BUILD it. Once the Farm is built, they will GO TOWARDS the Forum, and REPORT that the Farm is built. Warriors WANDER in the environment until they perceive an enemy Building or an enemy Person. If and when so, they will GO TOWARDS it and ATTACK it. The Resources in the environment are Trees and Gold Ore. Tree agents are able to EXPAND on a nearby place and create another Tree. The Farm is both a Building and a Resource that Peasants are able to HARVEST, to get some food.

This description leads to the identification of the agent families, interactions and specialization displayed in Fig. 7. It presents the behavior of agents as a hierarchy of interactions.

Peasants and Warriors can initiate the GO TOWARDS interaction. They do not initiate this interaction with an other agent as a target. They GO TOWARDS an agent only if they have the motivation to do so. For instance, a Warrior GOES TOWARDS a Person to ATTACK it. Consequently, the target has to be an enemy. A Peasant GOES TOWARDS a Resource to HARVEST it. Consequently, the source cannot be already carrying something.

Step 4: Specify the interaction matrix. With the specifications provided in Sect. 5.2.1 and during the steps 2 and 3, the raw interaction matrix is obtained as illustrated on Table 4.

Step 5: Specify the update matrix. In this simulation, agents do not update their state independently from the interactions they initiate or undergo. As a result, the update matrix is empty.

Step 6: Specify the interactions. In this step, the modeler has to provide a more accurate semantic to the interactions. For this simulation, some interactions are described from Tables 6, 7, 8. For the sake of concision, some interactions are not described.

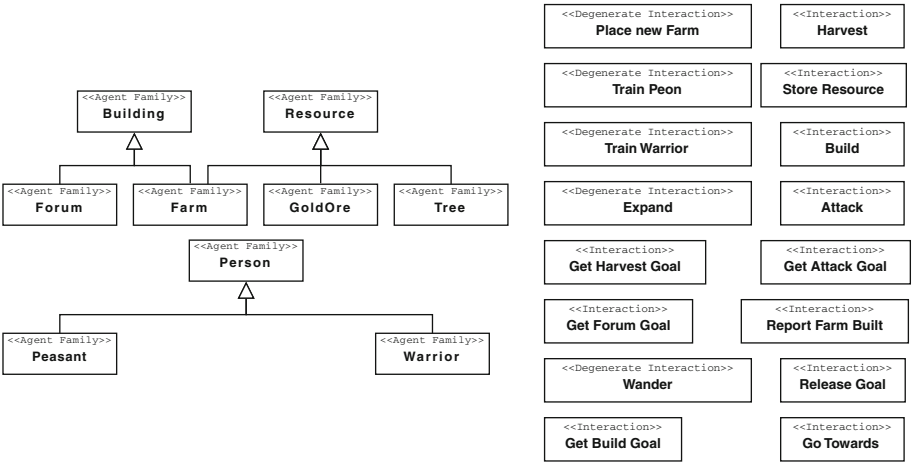


Fig. 7 Interactions and agent families involved in the strategy game simulation described in Sect. 5.2.1. An edge from a family $\mathcal{X}$ towards a family $\mathcal{Y}$ stands for "$\mathcal{X}$ specializes $\mathcal{Y}$". This enables the creation abstract groups of agent families.

Interactions related to moves like WANDER or GO TOWARDS are common interactions in a two-dimensional space. Thus, they are likely to be found in a generic interaction library. In that case, the specification phase of these interactions in the *IODA* methodology consists in using the specification described in the library. In our strategy game model, this is the case of the WANDER and GO TOWARDS interactions, which are also used in the context of a simulation of customer behavior in a virtual store [45].

Step 7: Specify the interaction selection process of the agents. This step of the methodology consists in giving a priority to every assignation element of the interaction matrix described in Table 4. The resulting refined interaction matrix is depicted in Table 5.

As mentioned in step 2, the motivation for an agent to move originates from an interaction. Whenever a Peasant or a Warrior identifies another agent that can be the target of an interaction, a GET…GOAL interaction is first triggered, if necessary, for the source to reach the target. For this, the target of the interaction is simply set as the current goal of the source. Once the source is close enough to the target, the interactions that firstly originated the move s performed—for instance HARVEST or BUILD. Once that interaction is over, the source performs the RELEASE GOAL interaction to acknowledge that the goal is now achieved.

One interaction only can be performed per time step. Thus, to ensure that the interaction sequence GET…GOAL → GO TOWARDS → HARVESTING (or ATTACK, or REPORT, *etc.*) → RELEASE GOAL is achieved, priorities have to be carefully defined while taking into account the meaning of each interactions. The lowest priority has to be set to the first interaction of the sequence. The following interactions are then given increasing priority values, so that the last interaction has the highest priority.

For instance, in case of Peasants HARVESTING Resources, the priorities order among the interaction is (HARVEST, Resource) > (RELEASE GOAL, Resource) > (GO TOWARDS, Resource) > (GET HARVEST GOAL, Resource).

In addition, Peasants favor STORING the Resources they have previously harvested and REPORTING the construction of a Farm over any other interaction. They also give priority to BUILDING a perceived Farm over HARVESTING Resources. If none of these interactions are possible, they WANDER in the environment. Warriors prefer to ATTACK Persons than to ATTACK Buildings. If they cannot ATTACK any of these, they WANDER in the environment.

Table 4 Raw interaction matrix of the strategy game simulation described in Sect. 5.2.1

Source \ Target	$\emptyset$	Forum: Building	Building	Person	Resource	Farm: Building,Resource
Forum: Building	(Train Peasant) (Train Warrior) (Place new Farm)					
Peasant: Person	(Wander)	(Get Forum Goal, $+\infty$) (Go Towards, $+\infty$) (Release Goal, 1) (Store Resource, 1) (Report Farm Built, 1)			(Get Harvest Goal, $+\infty$) (Go Towards, $+\infty$) (Release Goal, 1) (Harvest, 1)	(Get Build Goal, $+\infty$) (Go Towards, $+\infty$) (Release Goal, 1) (Build, 1)
Warrior: Person	(Wander)		(Get Attack Goal, $+\infty$) (Go Towards, $+\infty$) (Release Goal, 1) (Attack, 1)	(Get Attack Goal, $+\infty$) (Go Towards, $+\infty$) (Release Goal, 1) (Attack, 1)		
Tree: Resource	(Expand)					

The limit distance $+\infty$ can be interpreted as "as long the target is perceived". The element "Forum: building" in the first column, is read "Forum specializes—i.e., is a sub-family of—building".

In this simulation, if an agent $\mathcal{A}$ perceives at least one agent $\mathcal{B}$ towards which it wants to go—for instance to Harvest it—then $\mathcal{A}$ will first perform the Get Harvest Target with agent $\mathcal{B}$ as a target. Owing to this interaction, $\mathcal{A}$ Goes Towards $\mathcal{B}$ only. Once $\mathcal{A}$ is next to $\mathcal{B}$, it has reached its goal. To tell the agent that it has no longer a motivation to Go Towards $\mathcal{B}$, $\mathcal{A}$ initiates the Release Goal interaction with $\mathcal{B}$ as target

Table 5 Refined interaction matrix of the strategy game simulation described in Sect. 5.2.1

Source \ Target	∅	Forum: Building	Building	Person	Resource	Farm: Building,Resource
Forum: Building	(Train Peasant, p=15) (Train Warrior, p=10) (Place new Farm, p=5)					
Peasant: Person	(Wander, p=0)	(Get Forum Goal, +∞, p=45) (Go Towards, +∞, p=50) (Store Resource, 1, p=55) (Report Farm Built, 1, p=55) (Release Goal, 1, p=60)			(Get Harvest Goal, +∞, p=5) (Go Towards, +∞, p=10) (Harvest, 1, p=15) (Release Goal, 1, p=20)	(Get Build Goal, +∞, p=25) (Go Towards, +∞, p=30) (Build, 1, p=35) (Release Goal, 1, p=40)
Warrior: Person	(Wander, p=0)		(Get Attack Goal, +∞, p=5) (Go Towards, +∞, p=10) (Attack, 1, p=15) (Release Goal, 1, p=20)	(Get Attack Goal, +∞, p=25) (Go Towards, +∞, p=30) (Attack, 1, p=35) (Release Goal, 1, p=40)		
Tree: Resource	(Expand, p=15)					

Table 6 Description of the GO TOWARDS interaction, that occurs between a source agent Source and a target agent Target

GO TOWARDS(Source, Target)	
Trigger	Source has To Get Closer To(Target)
Preconditions	none
Actions	Source turn Towards(Target);
	Environment move Forward(Source, Source get Speed());

Table 7 Description of the HARVEST interaction, that occurs between a source agent Source and a target agent Target

HARVEST(Source, Target)	
Trigger	¬ Source carries A Resource()
Preconditions	¬ Source is Over loaded() **and** ¬ Target is Empty()
Actions	Source set Carried Resource Kind(Target get Kind());
	int i = Source get Afforded Load(Target);
	Source add Resource Qty(i);
	Target remove Qty(i);

Table 8 Description of the GET HARVEST GOAL interaction, that occurs between a source agent called "Source" and a target agent called "Target"

GET HARVEST GOAL(Source, Target)	
Trigger	¬ S carries A Resource() **and** ¬ Source has A Goal()
Preconditions	¬ S is Over loaded() **and** ¬ Target is Empty()
Actions	S set Goal(T);

Table 9 Primitives that a Peasant or a Warrior have to implement as the Source of GO TOWARDS interaction

Name	Description
Has to get closer to(Target)	The Source agent has a goal attribute which is an agent. This primitive returns true only if the Target is equal to the goal attribute of the Source

Step 8: Specify the agent families involved in the simulation. This step involves the concrete specification of abstract primitives, for every agent family involved in an interaction. This paragraph does not provide an exhaustive specification of these primitives since most of them have an explicit name, for instance primitives like `removeFoodQty(integer q)`, `getSpeed():integer`, etc. We focus instead on non-trivial primitives in Table 9 and Table 10.

Forums and Resources do not need a halo, since they initiate at best degenerate interaction. The halo of Peasants and Warriors is defined by a circle centered on the agent with a 10 units long radius. Neighbors are the agents in the environment, whose surface intersects this circle.

Table 10 Primitives that a Peasant or a Warrior have to implement as the Source of GET FORUM GOAL, GET HARVEST GOAL, GET BUILD GOAL or GET ATTACK GOAL

Name	Description
Set goal(Target)	The Source agent has a goal attribute which is an agent. This primitive sets the goal attribute of the Source to the value Target
Has a goal()	The Source agent has a goal attribute which is an agent This primitive returns true if the goal attribute is not null

This section has presented how to build interaction-oriented models with the guidelines provided by the *IODA* modeling methodology. The methodology was illustrated on a simple strategy game simulation. In order to provide runnable simulations, the model information have to be exploited by generic algorithms. The next section focuses on these implementation related issues, and briefly presents a simulation framework called *JEDI*, which accurately implements the *IODA* concepts.

6 From the model to the implementation

Maintaining the structure of the model at the implementation stage is not mandatory when using open frameworks like Madkit or Netlogo. However, we believe it can represent a real benefit and therefore *IODA* provides a set of algorithms for that matter. These algorithms altogether build a simulation framework called *JEDI*. The are presented in this section.

6.1 How does a *IODA*-compliant simulation framework work?

This subsection provides the set of algorithms enabling to run simulations containing generic agents designed with *IODA*. It is described how the single declaration of a refined interaction matrix makes the definition of reactive agents behavior possible.

In *IODA*, agents act according to a generic process outlined in Fig. 6. That algorithm is a sequence of four different steps: "Update the agents state", "Perceive neighbors", "Get the set of interactions the agent can initiate" and "Select the interaction the agent initiates". Although the definition of the two first steps are domain-dependent, the latter steps can be described with generic algorithms.

6.1.1 Get the set of interactions the agent can initiate in the current context

The construction of the interactions set an agent can perform in a particular context depends on both its abilities and on the agents in its neighborhood. Both the *eligibility* syntactic criterion and *realizability* semantic criterion are defined in this section to help listing all the possible interactions a source agent might perform.

Definition 16 (*Eligible Tuple*) Let $\mathcal{S}$ be an agent from the $\mathcal{F}_S$ family that is asked to act, $\mathcal{T}$ an agent from the $\mathcal{F}_T$ family lying in the neighborhood of $\mathcal{S}$, $a = (\mathcal{I}, lim)$ an assignation element and $\mathcal{M} = (a_{\mathcal{X}/\mathcal{Y}})_{\mathcal{X}\in\mathbb{F},\mathcal{Y}\in\mathbb{F}} \bigcup (a_{\mathcal{X}/\emptyset})_{\mathcal{X}\in\mathbb{F}}$ the interaction matrix of the simulation. The tuple $(a, \mathcal{S}, \mathcal{T})$ is said *eligible* if:

- $\mathcal{S}$ is able to initiate $\mathcal{I}$ together with $\mathcal{T}$—i.e., the cell of the matrix at the intersection of the row of $\mathcal{F}_S$ and the column of $\mathcal{F}_T$ contains $a : a \in a_{\mathcal{F}_S/\mathcal{F}_T}$;
- the distance separating $\mathcal{S}$ and $\mathcal{T}$ is less than lim.

Definition 17 (*Eligible Degenerate Tuple*) Let $\mathcal{S}$ be an agent from the $\mathcal{F}_S$ family that is asked to act, $a = (\mathcal{I})$ an assignation element, $\mathcal{I}$ a degenerate interaction, and $\mathcal{M} = (a_{\mathcal{X}/\mathcal{Y}})_{\mathcal{X}\in\mathbb{F},\mathcal{Y}\in\mathbb{F}} \bigcup (a_{\mathcal{X}/\emptyset})_{\mathcal{X}\in\mathbb{F}}$ the interaction matrix of the simulation. The tuple $(a, \mathcal{S})$ is said *eligible* if $\mathcal{S}$ is able to initiate the degenerate interaction $\mathcal{I}$—i.e., $a \in a_{\mathcal{F}_S/\emptyset}$.

Tuple eligibility is a syntactic criterion: it denotes whether a source agent has the ability to interact with a target agent. This criterion is not semantic and thus the interaction trigger or the preconditions might not be verified for the source and the target agents. To check the semantic validity of a tuple, a *tuple realizability* criterion is defined below.

Definition 18 (*Realizable Tuple*) Let $\mathcal{S}$ be an agent asked to act, $\mathcal{T}$ an agent lying in the neighborhood of $\mathcal{S}$, $a = (\mathcal{I}, lim)$ an assignation element and $\mathcal{I}$ an interaction. The tuple $(a, \mathcal{S}, \mathcal{T})$ is said *realizable* if:

- $(a, \mathcal{S}, \mathcal{T})$ is eligible and
- the trigger of $\mathcal{I}$ is valid for the agents $\mathcal{S}$ and $\mathcal{T}$ and
- the preconditions of $\mathcal{I}$ are valid for the agents $\mathcal{S}$ and $\mathcal{T}$.

Definition 19 (*Realizable Degenerate Tuple*) Let $\mathcal{S}$ be an agent asked to act, $a = (\mathcal{I})$ an assignation element and $\mathcal{I}$ a degenerate interaction. The tuple $(a, \mathcal{S})$ is said *realizable* if:

- $(a, \mathcal{S})$ is eligible and
- the trigger of $\mathcal{I}$ is valid for the agent $\mathcal{S}$ and
- the preconditions of $\mathcal{I}$ are valid for the agent $\mathcal{S}$.

The set of all realizable tuples and realizable degenerate tuples where $\mathcal{S}$ is the source represents all the interactions agent $\mathcal{S}$ can perform in its current context. The generic algorithm listing the realizable tuples of an agent x is presented in Fig. 8.

6.1.2 Select the interaction the agent initiates

Once the set of all the interactions a source agent can initiate has been listed, the agent must determine which one will actually be performed. For that matter, it depends on the reactive, cognitive or hybrid nature of the agent.

In this paper, we define a reactive interaction selection process centered on the priority of the assignation elements. It ensures that the interaction initiated by an agent belongs to a realizable tuple and that its assignation element has the highest priority. If more than one realizable tuple has the highest priority, then the interaction performed is selected at random. Using this random criterion is not an obligation. Designers can define, for a specific priority, a specific way to select a tuple, for instance by providing a preference value to tuples. This point is argued in detail in [44]. The corresponding generic algorithm is defined in Fig. 9.

In large scale simulations, the increased number of agents might have an impact on the number of possible interactions and therefore on the priority values assigned to them. In turn, this affects the algorithm described in Fig. 9. In that case, it is possible to avoid a loss of efficiency by slightly tweaking the algorithm. If a realizable tuple (or degenerate realizable tuple) with a priority p is selected then realizable tuples of lower priorities do not have to be considered. The census of realizable tuples can be gradually built during interaction selection—i.e., each priority at a time—to avoid unnecessary computations of tuples realizability.

realizable_tuples(x):
| **Let** $\mathcal{R}(x) = \emptyset$.
| %% *Census of interactions*
| **For All** y **in** $\mathcal{N}(x)$ **Do :**
| | **For All** $\mathcal{F}$ **in** {the family of y} $\cup$ {all parent families of y} **Do :**
| | | **For All** $a = (\mathcal{I}, lim)$ **in** $a_{\mathcal{X}/\mathcal{F}}$ **Do :**
| | | | **If** (a, x, y) is realizable **Then :**
| | | | | **Set** $\mathcal{R}(x) = \mathcal{R}(x) \cup \{(a, x, y)\}$.
| | | | **End If.**
| | | **End For.**
| | **End For.**
| **End For.**
| %% *Census of degenerate interactions*
| **For All** $a = (\mathcal{I})$ **in** $a_{\mathcal{X}/\emptyset}$ **Do :**
| | **If** (a, x) is realizable **Then :**
| | | **Set** $\mathcal{R}(x) = \mathcal{R}(x) \cup \{.(a, x)\}$
| | **End If.**
| **End For.**
| **Return** $\mathcal{R}(x)$.

Fig. 8 Generic algorithm used to list all realizable tuple of an agent x. Let $\mathcal{N}(x)$ be the set of neighboring agents of x, $\mathbb{F}$ the set of all the agent families, $\chi, \in \mathbb{F}$ be the family of x and $\mathcal{M} = (a_{S/T})_{S \in \mathbb{F}, T \in \mathbb{F} \cup \emptyset}$ the interaction matrix of the simulation

performed_interaction(x):
| **Let** $\mathcal{R} = realizable\_tuples(x)$.
| **Let** $\mathcal{P} = \mathbb{P}$.
| **Let** $selected = null$.
| **While** $selected == null$ **and** $\mathcal{P} \neq \emptyset$ **Do :**
| | %% *Get the next highest priority and remove it from* $\mathcal{P}$
| | **Let** $p = max(\mathcal{P})$.
| | **Set** $\mathcal{P} = \mathcal{P} \backslash \{p\}$.
| | %% *Get the set of all realizable tuple of* p *priority in* $\mathcal{R}_p$
| | **Let** $\mathcal{R}_p = \emptyset$.
| | **For All** $\mathcal{E}$ **in** $\mathcal{R}$ **Do :**
| | | **If** $priority(element(\mathcal{E})) == p$ **Then :**
| | | | **Set** $\mathcal{R}_p = \mathcal{R}_p \cup \{\mathcal{E}\}$.
| | | **End If.**
| | **End For.**
| | **If** $\mathcal{R}_p \neq \emptyset$ **Then :**
| | | %% *Select a realizable tuple at random*
| | | **Set** $selected = random\_element(\mathcal{R}_p)$.
| | **End If.**
| **End While.**
| **Return** $selected$.

Fig. 9 Generic algorithm of a reactive interaction selection process for an agent x in *IODA*. It selects the realizable tuple that corresponds to the interaction an agent x chooses to perform. Let x be an agent, *priority* a function getting the priority of an assignation element, $\mathbb{P}$ the set of all priorities and *element* a function returning the assignation element contained in a realizable tuple

6.1.3 Perform the selected interaction

Performing the interaction associated to a tuple corresponds to performing the algorithm contained in the action part of the interaction. When a primitive is called in the algorithm, the corresponding primitive of the source (or target) agent is called.

6.2 A concrete implementation of *IODA*: the *JEDI* simulation framework

The algorithms described in previous sections have been used to build an interaction-oriented simulation framework dedicated to reactive agents called *Java Environment for the Design of agent Interaction* (*JEDI*). This framework is based on the java programming language and accurately implements the concepts defined in *IODA* for the reactive interaction selection process described in Sect. 4.10.

6.2.1 Choices for the implementation

Some elements of the formal model *IODA* are domain-dependent: time and halo representation, or the definition of a distance notion. *JEDI* makes assumptions concerning these domain-dependent elements. Such choices imply that a model built with *IODA* can only be transformed into code on the *JEDI* simulation framework if both make the same assumptions concerning time, distance and halo representation.

This section aims at providing an overall description of *JEDI*. The optimizations made to improve simulations efficiency are not mentioned.

Euclidean Space. Simulations occur in a two dimensional Euclidean space environment. Each agent $\mathcal{A}$ occupies a rectangular ground surface $surface(\mathcal{A}) \subset \mathbb{R}^2$. This rectangle is a bounding box of its real surface. An agent also has a position $pos(\mathcal{A}) \in \mathbb{R}^2$ in that space, which is the center of $surface(\mathcal{A})$. Such implementation choices have consequences on both the halo and the expression of distance:

- $halo(\mathcal{A})$ of an agent $\mathcal{A}$ is a surface centered on the agent's position. Another agent $\mathcal{B}$ is in the halo of $\mathcal{A}$ iff $surface(\mathcal{B}) \cap halo(\mathcal{A}) \neq \emptyset$;
- the *distance* between two agents is the minimal Euclidean distance that separates the agent's surfaces.

Discrete Time Simulations. In *JEDI*, time is discretized into simulation steps, during each of which every agent is asked to act in a sequential randomized order. It solves concurrent interaction issues, like two predator agents trying to eat the same prey, with a simple up stream measure.

To figure out when the simulation has to end, a stop criterion has to be defined: for instance, to run a particular number of time steps, to reach a particular number of agents, to never end, etc.

Consequently, a simulation runs according to the algorithm in Fig. 10 and a simulation time step follows the algorithm in Fig. 11.

Let $i = 0$.
Initialize the simulation :
| Create the set of agents that lie in the envrionement
| Add these agents in the environment
While The ending criterion is not met **Do :**
| Perform the i^{th} simulation step (see Fig. 21)
| **Set** $i = i + 1$.
End While.

Fig. 10 General algorithm of a simulation in the *JEDI* simulation framework

```
perform_simulation_time_step() :
| %% Agents update
| For All 𝒜 in 𝔸 Do :
| %% Get all degenerate interactions of the agent from its cell in the update matrix
| | For All ℐ in 𝒰(family of 𝒜) Do :
| | | If ℐ.preconditions(𝒜) and ℐ.trigger(𝒜) Then :
| | | | ℐ.actions(𝒜)
| | | End If.
| | End For.
| End For.
| %% Agents behavior
| Let 𝔸_act = 𝔸.
| Reorder 𝔸_act with a particular criterion (see Sect. 6.2.2). By default at random.
| For All 𝒜 in 𝔸_act Do :
| | %% Compute the neighborhood of 𝒜.
| | Let 𝒩(𝒜) = ∅.
| | For All ℬ in 𝔸\{𝒜} Do :
| | | If halo(𝒜) ∩ surface(ℬ) ≠ ∅ Then :
| | | | Set 𝒩(𝒜) = 𝒩(𝒜) ∪ {ℬ}.
| | | End If.
| | | Let r = performed_interaction(𝒜) – see Fig. 19.
| | | If r ≠ null Then :
| | | | Perform the interaction associated with the realizable tuple r
| | | End If.
| | End For.
| End For.
```

Fig. 11 Algorithm of a time step in *JEDI*

6.2.2 JEDI parameters

A simulation framework has to provide means to tune its computational complexity, in order to increase simulations efficiency if necessary. The modular decomposition of *JEDI* allows tuning a set of parameters. Reducing the computational complexity with these parameters might introduce simulation biases if choices do not capture correctly the actual complexity of the model. Thus, choices concerning these parameters are left to the experimenter.

Point vs. Surface agents: In most generic frameworks, agents are considered as points[6] and therefore do not have an explicit surface when distance and perception are computed. Although *JEDI* promotes the use of surfaces to allow the definition of complex halo, agents can be represented as points as well, to improve the computation efficiency of the set of neighbors of an agent or the computation of the distances.

Agents Sequence Order: The order following which agents act during a simulation step has a great influence on the simulation results. In *JEDI*,this sequence is shuffled at the beginning of every time step. By default, this order is changed randomly to ensure equity among agents. Users can define their own sorting policy: for instance to order agents according to their "speed" attribute or not to reorder that set to improve simulations efficiency at the expense of possibly biased results.

Realizable tuples census: The census of realizable tuples can also be tuned, in order to improve simulation efficiency. Indeed, listing every realizable tuple from a given priority might require a large amount of operations (distance, trigger and preconditions computation) if many agent lie in the neighborhood. Some policies might improve the efficiency of this

[6] Agents might be *graphically* represented as shapes, but they are considered as points when the distance between them is computed.

census although introducing biases. For instance, only getting the first realizable tuple met introduces a bias depending on the order the assignation elements and the neighbors are checked. Similarly, getting one realizable tuple only per assignation element puts a bias on the order the tuples are checked.

Reproducibility: For implementation validation purpose, the ability to reproduce twice the exact same simulation is required in order to detect bugs. For that purpose, *JEDI* enables to specify the seed value used in the default random number generator or to define a custom random number generator.

6.3 Conclusion

This section has defined how the structure of a model designed with *IODA* can be kept at the implementation stage, which is a fundamental property for facilitating iterations of the simulation design process. Algorithms were provided to make these properties available in a concrete simulation framework. Eventually, it has briefly presented a simulation framework called *JEDI* that implements accurately these algorithms and the concepts defined in *IODA* models. Besides, *JEDI* supports large scale simulations owing to the tuning of its parameters set.

7 Conclusion and further works

MAS design methodologies exhibit many appealing features desirable within the scope of designing MABS. Indeed, they provide an explicit formal model to the system and guidelines to gradually build large models. Besides, they support the automation of models' implementation. Moreover, they rely on the concept of agent, which is an abstraction close to the notion of entity in a real phenomenon. Yet, these approaches are not appropriate enough for designing MABS, mainly because they focus on the functionalities the MAS has to achieve and on social aspects like organization and agent communication. On the opposite, MABS aim at studying emergent phenomena, whose origin lies in the interactions between entities. In that context, interactions are not restricted to communication between entities, but also include fundamental physical interaction or any other action involving simultaneously the environment and one or several agents. This article presents the formal model and the design methodology of the *IODA* approach where the design of a simulation is centered on this general definition of an interaction.

IODA models describe interactions—i.e., actions involving simultaneously two or more elements from the simulation—as generic and agent-independent elements. However, they allow taking into account agents specificities—i.e., give access to polymorphism—through the definition of abstract primitives. Consequently, reusable interaction ontologies are built along with simulations, which ease the design of further simulations. Moreover, the declaration of agents interaction abilities is separated from the interaction selection process, which allows representing heterogeneous entities in the simulation through homogeneous *IODA* agents. Entities can change their nature even at runtime and without any loss in the efficiency of the simulation. For instance, an "object" can be given an interaction ability, even during a running simulation, and thus become an "agent" (in the classical meaning).

In addition to these software engineering features, *IODA* models use notions related to real systems, including the notion of agent, interaction or halo. Moreover, the *IODA* methodology provides an intuitive tool called the interaction matrix, by means of which the interacting abilities of the agents can be defined and parameterized. These elements allow for incremental

design and implementation from high level specifications to fine-grained descriptions. The design of large scale simulations—i.e., involving a large variety of agents and interactions—is therefore facilitated.

The *IODA* methodology is not limited to the specification of simulations model: this paper provides the algorithms required to build a simulation framework that maintains the structure of the model. Thus, it guides the simulation design process from the specification to the implementation of the model. These algorithms are part of a simulation framework called *Java Environment for the Design of agent Interaction* (*JEDI*) which provides a simple yet accurate implementation of *IODA* concepts for reactive simulations. *JEDI* also facilitates the design of large scale simulations by providing a leverage on the simulation computational efficiency. By means of a set of parameters, simulations can run faster or include more agents, yet at the expense of introducing biases.

Simplifying assumptions were made in this paper to ease the comprehension of *IODA* core concepts. Actually, *IODA* provides a more complete set of features to design simulations. These features include tools to solve the following issues:

- "is an agent able to participate in an interaction if it already participates in another one ?", with the concept of agents activation/deactivation (see [44]);
- "how following a pheromone track is handled as an interaction?", with an enhanced specification of the interaction selection process (see [43]);
- "how pheromone diffusion is handled as an interaction?", with the enrichment of interactions cardinality;
- "how specialization between agent families is managed?", and especially "how to design agents families that do not inherit from all features of their parent family?", with the enrichment of agent families and interaction matrix specification.

7.1 Limits of *IODA*

Based on experiments, we found that the *IODA* approach was particularly appropriate to the design of simulations where the agents had a great variety of interactions, like in ethology, chemistry or sociology. *IODA*'s contribution is however less significant in the following cases. Firstly, it is not suited for simulations where each agent interacts with all the others agents using a limited variety of interactions. This is for instance the case of simulations in the application domain of cosmology, where the n-body problem implies that each particle initiates a gravitational interaction with every other particle in the environment. Secondly, *IODA* is not aimed at designing simulations where the complexity does not come from the number of interactions but from the intrinsic cognitive abilities of the agents. Indeed, in such cases the complexity comes from the algorithms responsible for the behavior of the agents (the primitives). This is for instance the case of marketplace simulations where dedicated approaches like in the *ArTificial Open Market (ATOM)* [49] are more appropriate.

7.2 Further works

A currently explored track for future research consists in investigating how to reverse the simulation process with *IODA*. In the classical approach, the simulation is implemented according to a prior hypothesis on the behavior of the agent and their interactions. Hypothesis may be validated through the observation of the resulting emergent phenomena. On the opposite, an interaction-oriented design allows generating random emergent phenomena—i.e., generate the behavior of each agents and the interactions between them randomly—from a given interaction library. When the expected phenomenon is observed, it is possible to backtrack

to the model of the simulation and to figure out what is the hypothesis [27]. It corresponds to *LEIA*[7] in Fig. 1.

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

References

1. Anderson, J. R. (1974). Simulation: Methodology and application in agricultural economics. *Review of Marketing and Agricultural Economics 42*(01), 3–55.
2. Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036–1060.
3. Axelrod, R. (1997). Advancing the art of simulation in the social sciences. *Simulating Social Phenomena, Lecture Notes in Economics and Mathematical Systems, 456*, 21–40.
4. Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE—a FIPA-compliant agent framework. In *Proceedings of the fourth international conference on the practical application of intelligent agents and multi agent technology (PAAM'99)*, London, pp. 97–108.
5. Bensaid, N., & Mathieu, P. (1997). MAGIQUE: A hybrid and hierarchical multi-agent architecture model. In *Proceedings of the second international conference on the practical application of intelligent agents and multi agent technology (PAAM'97)* (pp. 145–155).
6. Bernon, C., Camps, V., Gleizes, M. P., & Picard, G. (2005) *Engineering self-adaptive multi-agent systems: The ADELFE methodology*. USA: Idea Group Publishing, Chap. 7, pp. 172–202.
7. Braubach, L., Pokahr, A., & Lamersdorf, W. (2005). Jadex: A bdi agent system combining middleware and reasoning. In R. Unland, & M. K. M. Calisti (Eds.), *Software agent-based applications, platforms and development kits* (pp. 143–168). Basel: Birkhäuser-Verlag.
8. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems, 8*(3), 203–236.
9. Briot, J. P., & Meurisse, T. (2007). An experience in using components to construct and compose agent behaviors for agent-based simulation. In Fernando Barros, N. G. Claudia Frydman, & B. Ziegler (Eds.), *Proceedings of the international modeling and simulation multiconference (IMSM'07)*, The Society for Modeling & Simulation International (SCS) (pp. 207–212).
10. Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2*(1), 14–23.
11. Busetta, P., Howden, N., Rönnquist, R., & Hodgson, A. (2000). Structuring BDI agents in functional clusters. In *Proceedings of the 6th international workshop on intelligent agents VI, agent theories, architectures, and languages (ATAL'99)* (pp. 277–289). London: Springer.
12. Campos, A. M., Canuto, A. M., Fernandes, J. H., & de Moura, E. C. (2004). Masim: A methodology for the development of agent-based simulations. In *Proceedings of 16th European simulation symposium*, Budapest, Hungary.
13. Cannata, N., Corradini, F., & Merelli, E. (2008). Multiagent modelling and simulation of carbohydrate oxidation in cell. *International Journal of Modelling, Identification and Control (IJMIC), 3*(1), 17–28. doi:10.1504/IJMIC.2008.018191.
14. Chella, A., Cossentino, M., Sabatucci, L., & Seidita, V. (2004). From passi to agile passi: Tailoring a design process to meet new needs. In *Proceedings of the international conference on intelligent agent technology (IAT'04)* (pp. 471–474), IEEE Computer Society, Los Alamitos, CA, USA.
15. Choi, D., Kaufman, M., Langley, P., Nejati, N., & Shapiro, D. (2004). An architecture for persistent reactive behavior. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems (AAMAS'04)* (pp. 988–995), IEEE Computer Society.
16. Christley, S., Xiang, X., & Madey, G. (2004). Ontology for agent-based modeling and simulation. In *Proceedings of the agent 2004 conference on social dynamics: Interaction, reflexivity and emergence*, (pp. 115–125).
17. Desmeulles, G., Querrec, G., Redou, P., Kerdélo, S., Misery, L., Rodin, V., & Tisseau, J. (2006). The virtual reality applied to biology understanding: The in virtuo experimentation. *Expert Systems with Applications*, *30*(1), 82–92. http://dx.doi.org/10.1016/j.eswa.2005-09-051.

[7] Which stands for: *LEIA lets you Explore your Interactions for your Agents*.

18. Devigne, D., Mathieu, P., & Routier, J. C. (2005). Interaction-based approach for game agents. In: Y. Merkuryev, R. Zobel, E. Kerckhoffs (Eds.), *Proceedings of the 19th European conference on modelling and simulation (ECMS'05)* (pp. 705–714).
19. Doi, T., Tahara, Y., & Honiden, S. (2005) IOM/T: An interaction description language for multi-agent systems. *In: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS'05)* (pp. 778–785) ACM, Utrecht, The Netherlands.
20. Drogoul, A., Corbara, B., & Fresneau, D. (1992). Applying ethomodelling to social organization in ants. In J. Billen (Ed.), *Biology and evolution of social insects* (pp. 375–383). Belgium: Leuven University Press.
21. Drogoul, A., Vanbergue, D., & Meurisse, T. (2003). Multi-agent based simulation: Where are the agents? *Lecture Notes in Computer Science, 2581*, 43–49.
22. Edmonds, B. (2005). Simulation and complexity—how they can relate. In *Virtual worlds of precision—computer-based simulations in the sciences and social sciences, feldmann, valerie and mühlfeld* (katrin edn., pp. 5–32). Münster: Lit Verlag.
23. Edmonds, B., & Hales, D. (2003). Replication, replication and replication: Some hard lessons from model alignment. *Journal of Artificial Societies and Social Simulation, 6*(4), 11.
24. FIPA. (2002). Foundation for intelligent physical agents (FIPA) specification. http://www.fipa.org/.
25. Fishwick, P. A. (1995). *Simulation model design and execution: Building digital worlds*. Upper Saddle River, NJ, USA: Prentice Hall PTR.
26. SE Division at Fondazione Bruno Kessler. (2009). TAOM4E: Tool for agent oriented modeling. Accessed September 12, 2009, from http://sra.itc.it/tools/taom4e/.
27. Gaillard, F., Kubera, Y., Mathieu, P., & Picault, S. (2008). A reverse engineering form for multi agent systems. In A. Artikis, G. Picard, & L. Vercouter (Eds.) *Proceedings of the 9th international workshop engineering societies in the agents world (ESAW'2008)*, Lecture Notes on Artificial Intelligence, Vol. 5485, pp. 137–153.
28. Galán, J. M., & Izquierdo, L. R. (2005). Appearances can be deceiving: Lessons learned re-implementing axelrod's evolutionary approach to norms. *Journal of Artificial Societies and Social Simulation, 8*(3), 2.
29. Galán, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A., & Edmonds, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation 12*(1):1. http://jasss.soc.surrey.ac.uk/12/1/1.html.
30. Garcia-Ojeda, J. C., DeLoach, S. A., Robby, Oyenan, W. H., & Valenzuela, J. (2008). O-MaSE: A customizable approach to developing multiagent development processes. In *Proceedings of the 8th international workshop on agent oriented software engineering (AOSE 2007)*, pp. 1–15.
31. Garcia-Ojeda, J. C., DeLoach S. A., Robby, (2009). agentTool III: from process definition to code generation. In *AAMAS '09: Proceedings of the 8th international conference on autonomous agents and multiagent systems, international foundation for autonomous agents and multiagent systems* (pp. 1393–1394), Richland, SC.
32. Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the sixth national conference on artificial intelligence (AAAI'87)* (pp. 677–682), Seattle, WA.
33. Gilbert, N., & Troitzsch, K. G. (2005). *Simulation for the social scientist* (2nd ed.). Maidenhead and New York: Open University Press.
34. Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S.K., Huse, G., Huth, A., Jepsen, J. U., Jørgensen, C., Mooij, W. M., Müller, B., Pe'er, G., Piou, C., Railsback, S. F., Robbins, A. M., Robbins, M.M., Rossmanith, E., Rüger, N., Strand, E., Souissi, S., Stillman, R. A., Vabø, R., Visser, U., & Deangelis, D. L. (2006). A standard protocol for describing individual-based and agent-based models. *Ecological Modelling, 198*(1-2), 115–126.
35. Gutknecht, O., Ferber, J., & Michel, F. (2001). Integrating tools and infrastructures for generic multi-agent systems. In J. P. Müller, E. Andre, S. Sen, & C. Frasson (Eds.) *Proceedings of the 5th international conference on autonomous agents (AGENTS'01)*. Montreal, Canada: ACM.
36. Huber, M. J. (1999). Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the third annual conference on autonomous agents (AGENTS'99)* (pp. 236–243). New York, NY: ACM.
37. Ingalls, R. G. (2001). Introduction to simulation. In *WSC '01: Proceedings of the 33nd conference on Winter simulation* (pp. 7–16). Washington, DC: IEEE Computer Society.
38. Janssen, M. A., Alessa, L. N., Barton, M., Bergin, S., & Lee, A. (2008). Towards a community framework for agent-based modelling. *Journal of Artificial Societies and Social Simulation, 11*(2), 6.
39. Jayatilleke, G. B., Padgham, L., & Winikoff, M. (2007). Evaluating a model driven development toolkit for domain experts to modify agent based systems. In *Proceedings of the 7th international workshop on agent-oriented software engineering (AOSE-07)* (Vol. 4405/2007, pp. 190–207). Hakodate, Japan: Springer.

40. Jennings, N. R. (1999). Agent-based computing: Promise and perils. In *Proceedings of the sixteenth international joint conference on artificial intelligence (IJCAI'99)* (pp. 1429–1436). San Francisco, CA: Morgan Kaufmann Publishers Inc.
41. Klügl, F., Herrler, R., & Fehler, M. (2006). Sesam: Implementation of agent-based simulation using visual programming. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS '06)* (pp. 1439–1440). New York, NY: ACM. http://doi.acm.org/10.1145/1160633.1160904.
42. Kubera, Y., Mathieu, P., & Picault, S. (2008). Interaction-oriented agent simulations: From theory to implementation. In *Proceedings of ECAI 08*, Patras, Greece.
43. Kubera, Y., Mathieu, P., & Picault, S. (2009a). How to avoid biases in reactive simulations. In Y. Demazeau, J. Pavón, J. Corchado, & J. Bajo (Eds.), *Proceedings of 7th international conference on practical applications of agents and multi-agents systems (PAAMS 2009)* (pp. 100–109). Advances in intelligent and soft computing. New York: Springer.
44. Kubera, Y., Mathieu, P., & Picault, S. (2009b). Interaction biases in multi-agent simulations: An experimental study. In: G. P. Alexander Artikis, & L. Vercouter (Eds.) *Post-proceedings of the 9th international workshop engineering societies in the agents world (ESAW'2008)*. LNAI, Vol. 5485, pp. 217–235. New York: Springer.
45. Kubera, Y., Mathieu, P., & Picault, S. (2010). An interaction-oriented model of customer behavior for the simulation of supermarkets. In N. Cercone, & J. Huang (Eds.), *Proceedings of IEEE/WIC/ACM international conference on intelligent agent technology (IAT'10)*. http://www.yorku.ca/wiiat10/?2.
46. Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1–64.
47. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation, 81*(7), 517–527. http://dx.doi.org/10.1177/0037549705058073.
48. Malone, T. W., & Crowston, K. (1994) The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR) 26*(1), 87–119.
49. Mathieu, P., & Brandouy, O. (2010). A generic architecture for realistic simulations of complex financial dynamics. In: Y. Demazeau, F. Dignum, J. Corchado, & J. Bajo (Eds.), *Advances in practical applications of agents and multiagent systems, 8th international conference on practical applications of agents and multi-agents systems (PAAMS' 2010).* Advances in intelligent and soft computing, Vol. 70, pp. 185–197. Springer. ISBN: 978-3-642-12383-2.
50. Miles, S., Joy, M., & Luck, M. (2001). Designing agent-oriented systems by analysing agent interactions. In *Agent-oriented software engineering* (pp. 377–405). New York: Springer.
51. Müller, J. P., & Pischel, M. (1994). Modelling interacting agents in dynamic environments. In *Proceedings of the eleventh European conference on artificial intelligence (ECAI'94)* (pp. 709–713), Amsterdam, The Netherlands.
52. North, M., Tatara, E., Collier, N., & Ozik, J. (2007). Visual agent-based model development with repast simphony. In *Proceedings of the agent 2007 conference on complex interaction and social emergence* (pp. 173–192).
53. Odell, J., Parunak, H., & Bauer, B. (2000). Extending uml for agents. In *Proceedings of the agent-oriented information systems workshop (AOIS'00)*, pp. 3–17.
54. Ominici, A. (2001). Soda: Societies and infrastructures in the analysis and design of agent-based systems. In A*gent-oriented software engineering* (Vol. 1957/2001, pp. 311–326). New York: Springer
55. Ören, T. (1987). Simulation: Taxonomy. In *Systems and control encyclopedia*, (pp. 4411–4414). Boston: Pergamon Press.
56. Padgham, L., Thangarajah, J., & Winikoff, M. (2005). Tool support for agent development using the prometheus methodology. In *Proceedings of the fifth international conference on quality software (QSIC '05)* (pp. 383–388). Washington, DC: IEEE Computer Society. http://dx.doi.org/10.1109/QSIC.2005.66.
57. Pavón, J., & Gómez-Sanz, J. J. (2003). *Agent oriented software engineering with INGENIAS*, (Vol. 2691, pp. 394–403). Springer-Verlag: Prague, Czech Republic.
58. Pavon, J., Sansores, C., Gomez-Sanz, J. J., & Wang, F.-Y. (2008). Modelling and simulation of social systems with ingenias. *International Journal of Agent-Oriented Software Engineering, 2*(2), 196–221. http://dx.doi.org/10.1504/IJAOSE.2008.017315.
59. Robinson, S. (2006). Conceptual modeling for simulation: Issues and research requirements. In *Proceedings of the winter simulation conference, 2006 (WSC'06)*, pp. 792–800.
60. Sargent, R. G. (1998). Verification and validation of simulation models. In *Proceedings of the 30th conference on Winter simulation (WSC'98)* (pp. 121–130). Los Alamitos, CA: IEEE Computer Society Press.

61. Sargent, R. G. (2005). Verification and validation of simulation models. In *Proceedings of the 37th Winter simulation conference (WSC '05)* (pp. 130–143).
62. Shannon, R. E. (1976). Simulation modeling and methodology. In *Proceedings of the 76 bicentennial conference on Winter simulation (WSC '76)* (pp. 9–15).
63. Shannon, R. E. (1998). Introduction to the art and science of simulation. In: *Proceedings of the 30th Winter simulation conference (WSC'98)* (pp. 7–14). Los Alamitos, CA: IEEE Computer Society Press.
64. Terna, P. (1998). Simulation tools for social scientists: Building agent based models with Swarm. *Journal of Artificial Societies and Social Simulation, 1*(2). http://jasss.soc.surrey.ac.uk/1/2/4.html.
65. Wilenski, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
66. Winikoff, M., & Padgham, L. (2004). *Developing Intelligent agent systems: A practical guide*. New York, NY: Halsted Press.
67. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology, 12*(3), 317–370. http://doi.acm.org/10.1145/958961.958963.