Everything can be Agent!

(Extended Abstract)

Yoann Kubera
Université Lille 1,
59655 Villeneuve d'Ascq,
France
yoann.kubera@lifl.fr

Philippe Mathieu
Université Lille 1,
59655 Villeneuve d'Ascq,
France
philippe.mathieu@lifl.fr

Sébastien Picault
Université Lille 1,
59655 Villeneuve d'Ascq,
France
sebastien.picault@lifl.fr

ABSTRACT

Most Multi-Agent System designers use several notions – like "agent", "artifact", "object", *etc.* – to classify the entities involved in simulations. These notions require different methodologies, data structures and algorithms. In this paper, we show that the representation of entities can be favorably unified. As a consequence, the design and implementation process are made easier, since the designer has no longer to assign a fixed type to each entity during model construction. The implementation handles entities through an unified data structure and algorithm, and is therefore lightweight and more maintainable. Such an unification is performed without efficiency loss in a concrete simulation methodology called Ioda. According to common sense, we propose to call such an unified entity simply "agent"!

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Algorithms, Design

Keywords

Simulation techniques, Software engineering

1. INTRODUCTION

Nowadays, MultiAgent Based Simulations (MABS) became preponderant among computer simulation tools. The origin of this trend comes from the notion of agent, which is said close to the notion of entity in real phenomena.

Paradoxically, many different typologies are used to design the entities MABS involve. For instance, [1] makes the difference between agents and objects, [4] makes the difference between agents and artifacts, [5] makes the difference between agents and patches, *etc.* Each type relies on specific methodologies and data structures to identify entities and on dedicated algorithms to implement them.

Cite as: Everything can be Agent! (Extended Abstract), Y. Kubera, P. Mathieu and S. Picault, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

Even if using types provides some guidelines and representations for entities design, it also includes issues that make simulation design harder. Indeed, identifying of the type of an entity is not an easy and intuitive task, since types do not match intuitive notions of real phenomena. Moreover, prior types are incompatible with a gradual design process: refining the model might imply to change the types of entities, which forces to re-implement all from scratch.

We uphold that these types are just an *a priori* way to characterize the activity of an entity. Relying on a dynamic characterization of entities activity, we propose in this paper to achieve the same thing SmallTalk did with objects in object oriented programming, or Prolog did with terms in logic programming: to unify the representation of entities in MABS, and use them in one single engine.

2. ENTITIES ACTIVITY

We state that the activity of entities in MABS can be characterized by the presence of none, one, two or all of the following abilities:

- **activity:** the entity can initiate actions;
- **passivity:** the entity can undergo actions;
- **lability:** the entity can change its state without acting nor undergoing actions.

Most types used in MABS correspond only to a prior valuation of these abilities. An artifact in [4] is a passive entity – for instance a door used by an entity to go from a room to another. An object in [1] is either a passive entity – see the example above – or an entity without any of these abilities – for instance a curtain that hides the entities behind it. An agent in [1] and in [2] have not the same meaning: in [1] it corresponds to an active and passive entity, and in [2] it corresponds to an active, passive and labile entity.

Other types are used to differentiate some properties of entities. For instance the only difference between turtles and patches in Netlogo [5] lies in their physical representation in the environment: a turtle is a point, and a patch is a square. The use of a property to define the physical representation of entities would have avoided the use of types.

If entities representation makes possible to dynamically identify active, passive, and labile properties of entities, then entities representation becomes unified, and a simulation can run with an unified algorithm. For instance, in the case of discrete time, simulations run with algorithm 1.

Algorithm 1: Outline of the algorithms that define 1) how agents are added in the environment, and 2) how discrete time simulations run, if entities act in sequence.

Let $\mathbb{E}_{lab}$, $\mathbb{E}_{act}$ and $\mathbb{E}_{pas}$ be sets of entities.
1) addEntity(e):
begin
  Add e to the environment;
  if *if e is labile* **then**
    $\mathbb{E}_{lab} \Leftarrow \mathbb{E}_{lab} \cup \{e\}$;
  if *if e is active* **then**
    $\mathbb{E}_{act} \Leftarrow \mathbb{E}_{act} \cup \{e\}$;
  if *if e is passive* **then**
    $\mathbb{E}_{pas} \Leftarrow \mathbb{E}_{pas} \cup \{e\}$;
end
2) performSimulation():
begin
  while *simulation running* **do**
    % Updating labile entities
    for $e \in \mathbb{E}_{lab}$ **do**
      update e;
    Shuffle $\mathbb{E}_{act}$;
    % Asking active entities to act
    for $e \in \mathbb{E}_{act}$ **do**
      % Getting perceived passive entities
      $\mathbb{P} \subseteq \mathbb{E}_{pas} \Leftarrow$ all entities perceived by e;
      $(\mathcal{I}, T \in \mathbb{P}) \Leftarrow \mathcal{I}$ the action e chooses to do, and T the entity that undergoes the action;
      Perform $\mathcal{I}$ with e as source and T as target;
end

3. INTERACTION ORIENTED DESIGN OF AGENT SIMULATIONS (IODA)

In [3], we proposed an interaction-oriented approach to simulation design, called IODA. It relies on an homogeneous representation of actions performed by entities, called **Interaction**. In our work, an interaction is a semantic block of actions involving simultaneously a fixed number of entities, which describes how and under what kind of conditions entities may interact one with others. Difference is made between **source** entities that initiate the interaction – *i.e.* entities that choose to interact with their own action selection process – and **target** entities that undergo the interaction – *i.e.* which are chosen by the action selection process of source entities. The **Interaction Matrix** summarizes which interactions an entity can initiate as a source with another entity as target (see figure 1). An interaction is possible between a source entity x and a target entity y only if the interaction lies in the matrix at the intersection of the line of x, and the column of y.

Interactions are made concrete as software elements, and the interaction matrix is thus kept at implementation. Therefore, entities activity can be characterized as follows:

- an **active entity** owns a not empty line in the interaction matrix: x *is active* $\Leftrightarrow \exists cell \in InteractionMatrix \mid cell \in line(x) \wedge cell \neq \emptyset$
- an **passive entity** owns a not empty column in the interaction matrix: x *is passive* $\Leftrightarrow \exists cell \in InteractionMatrix \mid cell \in column(x) \wedge cell \neq \emptyset$

Entities are able to update with a particular method – which is empty in labile entities.

Consequently, IODA makes possible to express, with an unified representation, entities owning any combination of

Source \ Target	$\emptyset$	Grass	Sheep	Wolf
Sheep	Move Die	Eat	Reproduce	
Wolf	Move Die		Eat	Reproduce

Figure 1: Interaction Matrix of a predator/prey simulation. Interactions involving no target entity – for instance Move – are put in the $\emptyset$ column.

activity properties identified in the second section, and makes possible to dynamically identify active and passive properties. Thus, IODA achieves the representation presented in the latter section, and unifies entities representation.

4. EFFICIENCY ISSUES

Using a dynamic evaluation of entities activity might be less efficient than using types preprocessing. Moreover, IODA has no criterion to differentiate labile from not labile agents. Thus, an efficiency loss can be expected. However, experiments we led show that a simulation using IODA is as efficient as an ad-hoc algorithm to manage entities activity.

5. CONCLUSION

Most existing MABS design methodologies and frameworks make a conceptual distinction between agents, artifacts, objects, *etc.* Entities are given a type, and are designed – and implemented – with very different structures and architectures. Finding out the type to design a particular entity is not always obvious for domain specialists, since these types do not match intuitive notions of real phenomena like "living being", or "animated entity". Moreover, types go against the principle of incremental design of simulations.

In this paper, we show that types are only particular characterization of entities activity. These properties can be dynamically identified so that types are not necessary to design entities: their representation can be unified. Moreover, the resulting implementation can be made efficient and simple.

Our methodology called IODA [3] provides a representation meeting these criterion. For *agent*-based simulations, we propose to call such an unified entity "agent". Consequently, in simulations, everything can be Agent!

6. REFERENCES

[1] J. Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.
[2] J. Ferber and J. Müller. Influences and reaction: a model of situated multiagent systems. In *Proceedings of ICMAS'96*, Kyoto, 1996.
[3] Y. Kubera, P. Mathieu, and S. Picault. Interaction -oriented agent simulations : From theory to implementation. In *Proceedings of ECAI'08*, pages 383–387, Athens, 2008.
[4] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
[5] U. Wilenski. Netlogo. http://ccl.northwestern.edu/ netlogo/, Center for Connected Learning and Computer-Based Modeling, 1999.