A Component-based Model of Agent Behaviors for Multi-Agent-based Simulations

Jean-Pierre Briot[1] and Thomas Meurisse[2]

[1] Laboratoire d'Informatique de Paris 6 (LIP6)
8, rue du Capitaine Scott, 75015 Paris, France
(currently visiting CS Dept., PUC-Rio, Brazil)
Jean-Pierre.Briot@lip6.fr
[2] Sinovia, 93, rue Henri Rochefort, 91000 Evry, France
Thomas.Meurisse@sinovia.com

Abstract. This paper describes a component model aimed at easing the design and construction of agents for multi-agent based simulations (MABS). After first discussing the methodological issues of constructing operational models of agents, we introduce our model of component for agents, named MALEVA. In this model, components encapsulate various units of agent behaviors or activities (e.g., follow gradient, flee, mate, reproduce...). Among its specificities, it provides an explicit notion of control flow between components, for a fine grain control of activation and scheduling. Moreover, a notion of composite component allows complex behaviors to be constructed from simpler ones. Two small case studies, an ecosystem of situated agents and a microsimulation, are presented. We also discuss the benefits on the issue of intra-agent scheduling control.

Keywords: multi-agent based simulation, agent, architecture, component, design, operational model, behavior, activity, composition, scheduling, control.

1 Introduction

Multi-agent based simulation (MABS) is now established as one of the main approach for modeling and simulation of various phenomena. We believe that this is because the concept of agent is both structuring enough (unit of activity, of interaction, concept of organization...) and versatile enough (reactive or cognitive agents, various models of environments...). Meanwhile, we also think, along [5], that the process of constructing operational models from domain models is still often underestimated.

We will start this paper by summarizing a proposition, by Drogoul et al. at MABS'2002 [5], about a first sketch of methodological framework clarifying the different levels, models, steps and roles concerned. Then, we will propose an abstract *component-based model* as the basis for constructing operational models for MABS. Indeed, we think that the design and construction of an agent can benefit from the concepts of component. We believe it can help at an incremental construction of an agent by composition of simple agent behaviors or activities (e.g., flee, follow gradient, mate, reproduce...).

2 From Domain Models to Operational Models

As was discussed in [5], translating conceptual models into computational models for MABS is a critical and not trivial step. That paper has proposed a first sketch of a methodological framework for multi-agent-based simulations. It is illustrated at Figure 1. Let us summarize it briefly.

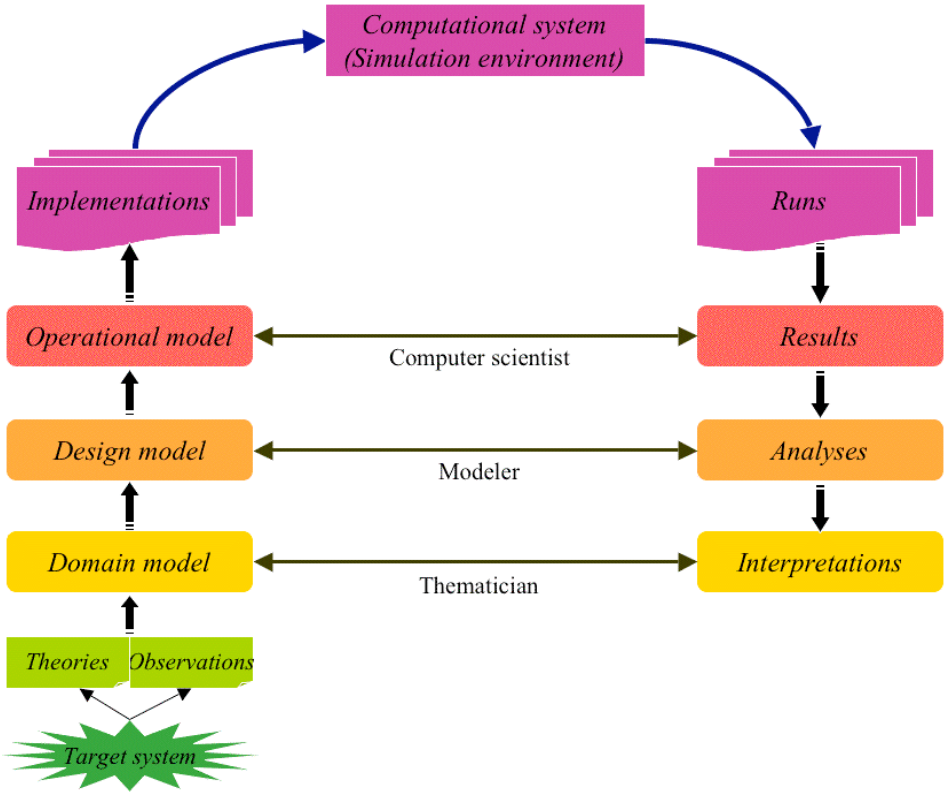

Fig. 1. The proposed methodological framework

We start with a target system which characterizes the phenomenon to predict or the theory that needs explanations. This part involves experts in a particular domain, and they are named *thematicians*. A *thematician* is in charge of defining the *domain model*. It includes three types of information about the target system [13]: theories and assumptions (what he knows or estimates), observations (what he sees or analyzes), and questions (what he wants to understand) [5]. In most cases, the theories associated with the target system may remain partly controversial and ambiguous. Thus, as claimed by Barreteau et al. [2], other intermediary models are often useful, or even necessary, before building the actual simulation program.

Indeed, since the specifications of the thematician do not usually allow for a direct transcription to an operational solution,[3] the domain model has to be translated into something more precise and formal, by clarifying the concepts and removing the ambiguities. The result is named the *design model*, and its definition is the role of the *modeler*.

The design model will be used as an input by the *computer scientist* to shape the *operational model*. Computational agents are used as an implementation technique. Specifying technical properties such as the distribution of the agents, the time scheduling techniques used, etc., and making them explicit in the operational model, will facilitate the comparison between different models and help in understanding the role of computational-specific features in the possible emergence of structures (as shown by, e.g., Axtell [1]) or possible biases (see Section 5) during the simulation.

3 The MALEVA Model of Agent Components

The MALEVA agent component model that we will now describe is a tentative foundation, or more precisely an abstract model, for constructing and implementing operational models. The objective is to help at incrementally design and construct agent behaviors or activities (e.g., flee, follow gradient, mate, reproduce...), by composing simpler behaviors/actions, reified as software components. We therefore assume that there is a library of behavior components associated to the application domains targeted. A component may be primitive (the behavior is written in the underlying language, e.g., Java), or *composite* (as the encapsulation of a composition of components).

3.1 Data flow and control flow

In MALEVA, there is a distinction between the activation control flow and the data flow connecting the components. This characteristic and specificity of our model, which decouples the functional architecture from the activation control architecture, makes components more independent of their activation logic (an example of benefits will be discussed in Section 5) and thus more reusable. Consequently, we consider two different kinds of ports within a component:

- *data ports*. They are used to convey data transfer (one way) between components.
- *control ports*. A behavior encapsulated in a component is activated only when it explicitely receives an activation signal through a control input port. When the execution of the behavior is completed, the activation signal is passed to the control output port.

[3] As shown by Fishwick [7], the conceptual gap between thematicians and computer scientists can be very important. Therefore, identifying a third role in between, the *modeler*, appears to be very useful.

4 1st Case Study: Prey and Predator Situated Agents

This first case study will define behaviors of situated agents within an eco-system. So first step is to define the general (abstract) architecture of a situated agent.

4.1 Abstract Architecture of a Situated Agent

A situated agent senses its environment (e.g., position of the various agents near by, presence of obstacles, presence of pheromones...) through its sensors. These data are used by its (internal) behavior to produce data for its effectors, which will act upon the environment (e.g., move, take food, leave a pheromone, die...). The general architecture of a situated agent usually follows the computational cycle: *sensors* → *behavior* → *effectors* and is shown at Figure 2.

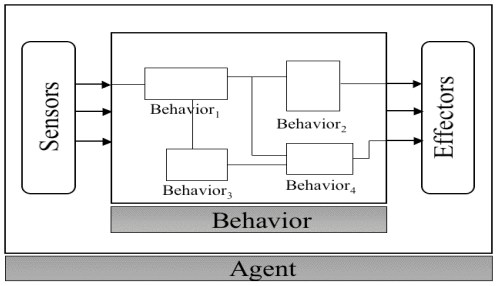

Fig. 2. Abstract architecture of a situated agent

4.2 Prey Behavior

We will now define and construct the basic behaviors of preys and predators. By following a bottom up approach, we first define a set of elementary components, representing the basic behaviors of preys and predators, that we name: `Flee` (fleeing a predator), `Follow` (following a prey), and `RandomMove` (random move for exploration, which represents the default behavior). Then we will compose them, to represent the following agent behaviors: `Prey` and `Predator`.

A `Prey` flees the predators being located within its field of perception. If no predator is close (sensed), the prey moves randomly (basic exploratory behavior). Thus, we construct the `Prey` behavior as the composition of the following three components: `Flee`, `RandomMove`, and a control component named `Switch`.

4.3 Control Components

The `Switch` control component reifies the standard conditional structure into a special kind of primitive component.[4] The condition is the presence or absence

[4] Note that the MALEVA standard library includes other control components, (e.g., repeat loop, barrier synchronization), not described in this paper.

of an input data. The behavior of `Switch`, once being activated (receiving an activation signal), is as follows:

IF *data is received through* `If` *(input data port)*
THEN *transfer control through* `Then` *(output control port)*
AND *send data through* `Then` *(output data port)*
ELSE *transfer control through* `Else` *(output control port)*

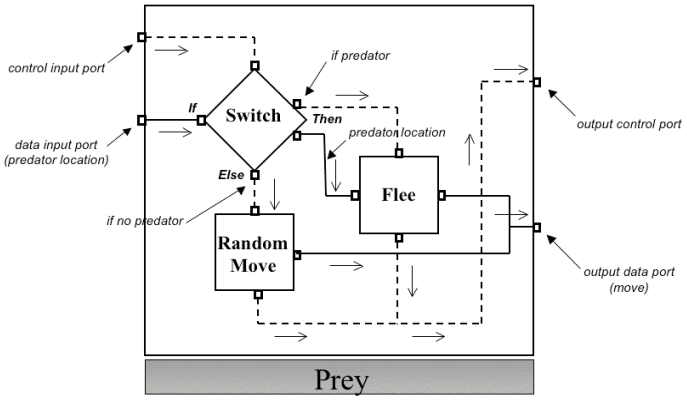

Fig. 3. Architecture of a prey

The architecture of the prey behavior (shown at Figure 3) follows this pattern. If a predator has been detected (some data representing the predator location has been received on the input data port),[5] `Switch` transfers control through its `Then` control output port, which activates the `Flee` behavior. Then `Flee` can compute a move data based on the location of the predator, and send it through its output data port. The move data is finally transfered to `Prey` output data port and then to the effector, to produce a move of the agent on the environment. If no predator has been sensed (no data received), `Switch` transfers control through its `Else` output control port, which activates `RandomMove` behavior.[6]

4.4 Predator Behavior

We may now reuse the `Prey` behavior component to construct the behavior of a `Predator` which follows the preys while fleeing his fellows predators, and moves randomly if he does not sense any other agent. The predator behavior may be defined as a prey behavior (it flees other predators and otherwise carries out a

[5] We assume that the input data port and the output data port of the `Prey` behavior have been connected to the corresponding sensor and effector data ports, along the general architecture of a situated agent, already shown at Figure 2.

[6] Note that `RandomMove` does not need input data to produce a move data.

random movement), to which is added a behavior of predation (it follows the preys). According to our compositional approach, we define `Predator` behavior component as a new composite behavior embedding *as it is* the existing `Prey` behavior component (see the result in Figure 4). Note that in that our current design, hunger (predation) has priority over fear (fleeing), as `Prey` is activated by `Predator`. Other combinations could be possible.

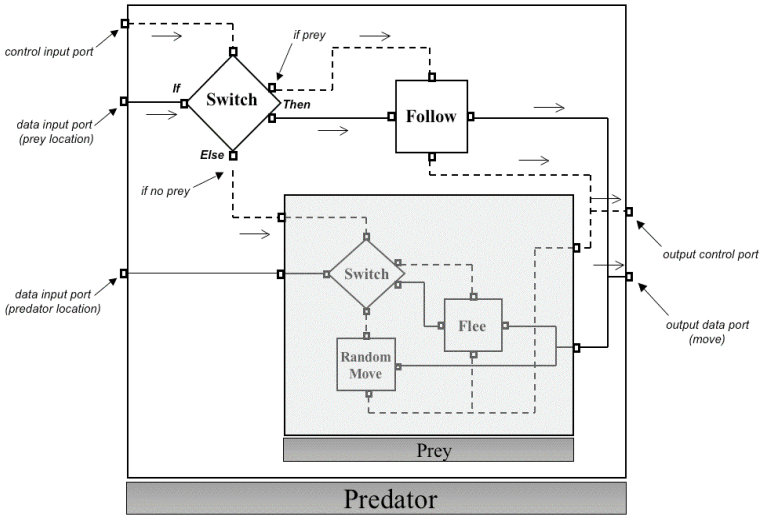

Fig. 4. Architecture of a predator (with prey as a sub-component)

5 2nd Case Study: Population Microsimulation

This second case study is inspired/simplified from an existing application of demography microsimulation, named Destinie [9]. It considers a virtual specy of agents, in which mating of two agents is necessary for reproduction, but without considering sexual differences (i.e. agents are hermaphrodite). We consider three basic behaviors: `Mate`, `Separate` and `Reproduce`. Behaviors may be seen as state changes, governed by probabilistic transition laws.

As a general guideline [8] behaviors are ordered in a sequence, as proposed by domain experts. But the issues are: In what order? And with what impact on the simulation results? Indeed, although not independent, these 3 behaviors are not necessarily bound to a specific sequence, as all possible interleavings are valid. Let us consider the following combinations:

- case *(a)* : *sequence* { `Mate ; Separate ; Reproduce` }
- case *(b)* : *sequence* { `Mate ; Reproduce ; Separate` }
- case *(c)* : *concurrent* { `Mate || Separate || Reproduce` }

The two first strategies *(a)* and *(b)* make explicit an order of activation (sequence) of behaviors. In the third *(c)* strategy, no temporal dependency constraint is specified, leaving the scheduler of the runtime system free of actual scheduling decision. Figure 5 shows he resulting histogram. It displays the number of individuals (y coordinate) having a certain number of children (x coordinate). Results follow the intuition: if reproduction is activated before separation *(b)*, this leads to more babies than if reproduction is activated after separation *(a)*. A fully concurrent strategy *(c)* produces an average number.

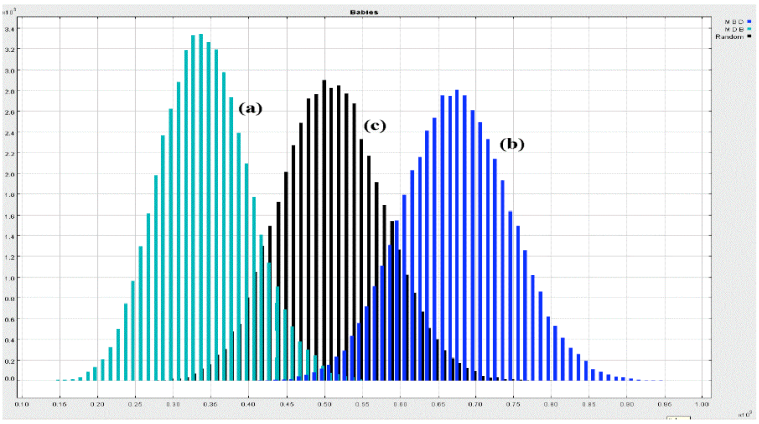

Fig. 5. Histogram of number of babies

This example shows the importance for the experts to be able to experiment with various strategies for ordering behaviors, to compare results with the target models, and to quantify the impact on biases.[7] By considering control flow explicitly, MALEVA helps at specifying and controlling temporal dependences between behaviors - and thus their possible orderings -, via explicit control flow connexions,[8] without any change to the code of behaviors (encapsulation ensured). This helps at: decoupling the functionality of an agent behavior from the logic of activation; making components more generic; and refining incrementally the operational model. By going back to our opening methodological point of view (see Section 2), this eases the relations between the design model and the operational model.

6 Conclusion

In this paper, we presented a component model, named MALEVA, to construct operational models for multi-agent based simulations. This model is relatively

[7] For instance, [10] shows that results of simulations can be found biased in cases where the scheduling of the actions within an agent remains deterministic.

[8] They are usually achieved interactively, via the CGraphGen CASE graphic tool [12].

original in the explicit management of activation control through control ports and connexions, by applying the concept of component also to the specification of control (see [3] for more about architectural issues). Experiments showed that these characteristics improve genericity of components, and help at rationalizing control of intra-agent behavior scheduling, an important issue for simulation. MALEVA has been experimented in different application domains, such as: urban migration [14], automobile trafic simulation [11], and artificial societies, e.g., a reengineering [12] of Sugarscape model [6]. Meanwhile, further experiments are still necessary to confirm and refine what we consider as promising results.

Acknowledgements We would like to thank Marc Lhuillier, Alexandre Guillemet, Grégory Haïk, and Frédéric Peschanski, for their contributions to the MALEVA project, and Alexis Drogoul and Diane Vanbergue for their contributions to the methodological framework that we refered to.

References

1. R. Axtell, Effects of interaction topology and activation regime in several multi-agent systems, *MABS'2000*, No 1979 of LNCS, 2000, pages 33–48.
2. O. Barreteau, F. Bousquet, and J.-M. Attonaty, Role-playing games for opening the black box of multi-agent systems, *Journal of Artificial Societies and Social Simulation*, 4(2), 2001.
3. J.-P. Briot, T. Meurisse and F. Peschanski, Architectural Design of Component-based Agents: A Behavior-based Approach, To appear in pre-Proceedings of 4th International Workshop on Programming Multi-Agent Systems (ProMAS'06), AAMAS'2006, Hakodate, Japan, May 2006.
4. A. Drogoul, B. Corbara, and D. Fresneau, MANTA: Experimental Results on the Emergence of (Artificial) Ant Societies, *Artificial Societies: the computer simulation of social life*, UCL Press, London, U.K., 1995.
5. A. Drogoul, D. Vanbergue, and T. Meurisse, Multi-Agent Based Simulation : Where are the Agents ? *MABS'2002*, LNCS, Springer-Verlag, July 2002.
6. J. M. Epstein and R. L. Axtell, *Growing Artificial Societies : Social Science from the Bottom Up*, MIT Press, 1996.
7. P. Fishwick, *Simulation Model Design and Execution*, Prentice-Hall, 1995.
8. N. Gilbert and K. G. Troitzsch, *Simulation for the Social Scientist*, Open University Press, 1999.
9. INSEE, Le modèle de microsimulation dynamique DESTINIE, Technical Report G9913, Division Redistribution et Politiques Sociales, Institut National de la Statistique et des Etudes Economiques, Paris, France, 1999.
10. B. G. Lawson and S. Park, Asynchronous Time Evolution in an Artificial Society Mode, *Journal of Artificial Societies and Social Simulation*, 3(1), 2000.
11. V. LeCerf and M. Pintado, An adaptive model of Camera-Driven Urban Intersections Observation, *Workshop on Dynamic Scene Recognition from Sensor Data*, edited by C. Tessier, ONERA, Toulouse, France, June 1997.
12. T. Meurisse, Simulation multi-agent : du modèle à l'opérationnalisation, *Thèse de doctorat (PhD thesis)*, Université Paris 6, France, July 2004.
13. K.G. Troitzsch, Methods of empirical social research, *SICSS Summer School*, 2000.
14. D. Vanbergue, J.-P. Treuil, and A. Drogoul, Modelling urban phenomena with cellular automata, *Advances in Complex Systems*, Vol. 3, 2000.