Living Design: Simulation for Self-Designing Agents (ESM 2007)

Sylvain Lemouzy, Carole Bernon, Marie-Pierre Gleizes

LIVING DESIGN: SIMULATION FOR SELF-DESIGNING AGENTS*

Sylvain Lemouzy
Carole Bernon
Marie-Pierre Gleizes
email: {lemouzy, bernon, gleizes}@irit.fr

IRIT - Paul Sabatier University
118 route de Narbonne
31062 Toulouse cedex 09
France

Abstract. Today's challenge is to design complex systems that operate within evolving environments. Self-organisation is a promising paradigm to make these systems adapt to changing conditions. The collective function arises from local interactions and the system design becomes thus bottom-up. The difficulty rests then in defining and coding behaviours at the bottom level in order to make the adequate global function emerge. Multi-agent systems (MAS) are an answer to implement such systems and many agent-oriented methodologies are proposed to guide designers. However few help designers by providing them with tools that build or improve the behaviour of agents. The aim of the work presented here is to reduce the workload of a MAS designer with a simulation tool that would give agents the ability to self-design their behaviour considering that (i) a behaviour can be expressed as a set of rules and (ii) agents interact in the (social and physical) dynamic environment they are situated in. The long-term objective is to improve the development phase of the ADELFE methodology by integrating this tool into it. This paper shows how simulation has contributed to this tool and how applying it has improved the behaviour of simulated foraging ants.

1 INTRODUCTION

Nowadays, computer systems and applications have to cope with a great number of functionalities and a great diversity of users but also they have to operate in environments that are more and more complex and dynamic. Consequently, they have to deal with unexpected situations that prevent their functionality to be totally a priori defined and make their requirements stay vague. Therefore classical software engineering methods are no longer applicable making designing

* Extended version of the paper plublished in European Simulation and Modelling Conference (ESM 2007), Malte, 22-24 octobre 2007, Eurosis, J. Sklenar, A. Tanguy, C. Bertelle, G. Fortino (Eds), pp. 432–436.

such systems a more complex and difficult task. The challenge today is finding new ways to ease designers' task for such systems. A possible answer to implement them are Multi-Agent Systems (MAS) which are made up of interacting autonomous entities which have dynamic relationships. They are mainly inspired from natural systems, composed of autonomous individuals with simple behaviours, that exhibit aptitudes to carry out complex tasks without any global knowledge (for instance, colonies of social insects such as termites and ants [1]). In MAS a non-supervised central control enables a global distributed functionality which essentially depends on the organisation of agents. We are especially interested in a kind of systems in which this global collective function is not a priori given and what makes this organisation appear is the way in which agents interact. Letting agents modify their relationships in an autonomous way, using self-organisation principles, is a means to change their organisation and therefore a way to adjust the global function of the system they belong to. This system becomes then self-adaptive and its design all the more problematical as the complexity of interactions between agents grows. A top-down approach is no longer useful and a bottom-up one has to be adopted: agents have to be thought up before being gathered within a shared system. Helping designers to enable them to think differently becomes mandatory. Several agent-oriented methodologies are offered for guiding MAS designers, however very few focus on adaptive systems and emerging functionalities [2, 3].

We focus this study on self-organising systems design and more especially on those in which agents try to avoid, detect and remove situations (called non cooperative situations, NCS) that prevent them from staying cooperative toward others or themselves [4]. A first work has been done with ADELFE for helping a designer to identify and implement such agents [5, 6]. However, the core difficulty lies in identifying non cooperative situations and [7] proposed an approach, based on simulation, to automatically identify these situations while a prototype of a targeted MAS is executing. Designers had then to manually improve the behaviour of involved agents to remove the detected NCS. However, the long-term objective is to make these modifications in an automatic way in order to reduce a designers' workload. Thus, the work presented here is about giving agents the ability to self-design their behaviour considering that this behaviour can be expressed as a set of rules and that the agents interact in the (social and physical) dynamic environment they are situated in.

The paper is then organised as follows. Self-organising systems we consider and the ADELFE methodology dedicated to their design are presented in the first section. The second section introduces in a more detailed manner the problem we want to solve. How simulation is used to enable agents to self-design their behaviour and related experiments are given in the two following sections. Before drawing some conclusions and envisaging some perspectives, this work is positioned in relation to other existing works.

2 CONTEXT OF THE WORK

As already said, multi-agent systems are a recognised paradigm to deal with the complexity and openness of today's computer systems. Furthermore adaptation is a mandatory characteristic of these systems to make them able to stay operational in today's evolving environments. A possible way for a system to achieve this adaptation ability is to self-organise [8, 9] that is to change its internal organisation without an explicit external control during its execution time.

The work presented in this paper has been done in the context of self-organising systems design, more especially on those that are based on the Adaptive Multi-Agent System (AMAS) theory and designed using the ADELFE methodology. These two concepts are then detailed hereafter.

2.1 Self-organising Systems and AMAS Theory

Let us suppose that agents of a system are endowed with the ability to autonomously and locally modify their interactions in order to react to changes in their environment. These alterations also transform their collective function i.e. the function performed by the multi-agent system they belong to. This latter is a self-organising system and becomes then able to adapt to changes in its environment. This assumption is the foundation of the AMAS theory that considers that "cooperation" is the criterion for self-organisation [4]. Actually, every agent pursues a local objective and interacts with others while respecting cooperative rules that make it changing its interactions in order to avoid and possibly remove situations that are judged as non cooperative from its local point of view (these situations are called non cooperative situations or NCS).

Roughly, applying the AMAS theory consists then on the one hand in trying to anticipate NCS and to avoid them and on the other hand, when NCS cannot be avoided, to process them to come back to a cooperative state. Therefore a designer has to enumerate, according to the current problem to solve, all the cooperation failures that can appear during a MAS functioning and then to define actions cooperative agents have to apply to stay in or to come back to a cooperative state. However, building self-organising systems this way is not so easy and tools are required to help designers enforcing this theory.

2.2 ADELFE Methodology

The ADELFE agent-oriented methodology [5, 6] aims at guiding AMAS designers through a development process based on the RUP (Rational Unified Process) [10]. This process has been modified to take into account specific features of AMAS, especially those concerning cooperation. In particular, during the final requirement phase, designers are invited to think about the situations that can be "unexpected" or "harmful" for the system because they can lead to NCS at the agent level. These situations can be then expressed in use case diagrams. Modifications in the design phase essentially concern the design of cooperative

agents. The most difficult point here is to find local cooperation rules for guiding the behaviour of agents in order to have the best consistent behaviour at the system level. A fast prototyping activity is then provided for testing the behaviour of agents mainly during interaction protocols. However, according to the AMAS theory, designers have to find all non cooperative situations an agent may encounter and provide, for every detected NCS, all actions this agent has to do to stay cooperative toward others and itself.

A first additional tool was then added during the implementation phase to help designers finding NCS not taken into account yet for an agent. This simulation tool [7] based on SeSAm platform [11] enables designers to observe a prototype of the targeted MAS while running and automatically detects some kinds of NCS. Designers are then able to change accordingly the behaviour of the concerned agents.

These modifications are still done manually and further help would be required to allow an agent to always choose the action which can be qualified as the best cooperative one. Our final objective is to enhance ADELFE with such a tool and the work presented in this paper is the first step in this direction.

3 PROBLEM STATEMENT

Most of the existing agent-oriented methodologies [2] use interaction protocols and roles in order to design the decision process of agents. How to find the algorithm applied by an agent is generally not well described and most of the task falls on designers. The aim of the work presented in this paper is to alleviate this task by providing designers with a means for automatically designing an agent behaviour considering the following context:

- This behaviour can be expressed with a set of behavioural rules which follow this pattern:

  ```
  if premise then consequent
  ```

  where `premise` is a logical predicate made up of elements coming from agent perceptions or characteristics and consequent activates one of the possible actions this agent may perform. All the rules needed to design the decision process are given by a designer that is the agent does not learn new rules during the process and the set of given rules is complete and correctly written.
- The system interacts with a dynamic environment.

Therefore, an agent behaviour consists in executing some rules of a set of executable rules at a given time in the decision process. Modelling this kind of behaviour was inspired by the subsumption architecture where an agent executes the rule with the highest priority if several rules are executable. A subsumption architecture [12] is a way to express the behaviour of an agent by using rules represented by (conditions, action) tuples. Such a tuple represents the action to do when the related conditions are true. In a general way, conditions are linked

to environmental perceptions of the agents. These tuples are ordered depending on the level of priority of an action relatively to another one; the action on the hierarchy top-level has then priority over the actions of the lower levels. Generally in the subsumption architecture from Brooks, agents have no memory and only one rule is executed at a time. These assumptions were modified in our model in the following way: an agent may have a memory and several rules, i.e. a set of rules, can be executed at the same time. However the priority between the sets of rules, which gives the algorithm executed by an agent, was retained from the subsumption concept. This subsumption-inspired hierarchy is used in our model as follows.

An agent for which the behaviour has to be enhanced is simulated under SeSAm by reusing the cooperative agent model described in [7] to implement a prototype of the targeted MAS this agent is part of. This agent runs, interacts with its environment and gives its conclusions (drawn from interpreting and reasoning about its perceptions) to its Behaviour Self-Design Module (BSDM) Considering what this agent knows, its BSDM gives it back a list of actions to be executed which corresponds to those of the top-level set of rules of the current hierarchy.

A Behaviour Self-Design Module inside an agent is implemented as an adaptive MAS, behavioural rules forming it have to collectively adapt to the agent's environment and are then considered as agents. They are called rule-agents for not confusing them with the agent which contains the module, this latter agent is called a simulated agent.

Our aim is to enable the rules inside the BSDM to self-organise in order to find the best hierarchy of rules that is to say the most efficient behaviour for the simulated agent it belongs to. At the beginning of the implementation, the designer supplies the BSDM with all the behavioural rules put in a random order. By forming sets of rules which have to be executed simultaneously and then by prioritizing these different sets according to what its simulated agent perceives, the BSDM provides this agent with the ability to self-design its behaviour.

The next section describes the adopted approach to implement a BSDM.

4 SELF-DESIGNING AGENT BEHAVIOURS USING SIMULATION

The aim of this section is to describe the AMAS that enables a simulated agent to design its behaviour by itself. The right collective function this AMAS has to perform is to produce the hierarchy that gives the simulated agent its most efficient behaviour. According to the AMAS theory, an AMAS carries out the right functionality if agents composing it cooperate in the right manner. Therefore, this section describes first the rule-agents that make up the self-design module and then how these rules cooperate in order to find their right place in the subsumption-inspired hierarchy presented above.

4.1 Structure of Behavioural Rules and their Relationships

As said before, the pattern followed by the behavioural rules of a simulated agent can be summarized by: if conditions then actions. [7] considered conditions written with propositional logic formulae, this work was reused and generalised in order to consider conditions and actions as logical predicates that may be parameterised. A rule is still triggered if all its conditions are satisfied and actions are still described using a set of preconditions (i.e. conditions that trigger them), a set of additional effects (predicates added when these actions are executed) and a set of removal effects (predicates removed when these actions are executed). A rule is then structured in this way:

$$\begin{aligned} Cond1([args]) \wedge \ldots \wedge Condn(args]) \rightarrow Action([args]) : \\ +\{Add1([args]), \ldots, Addm([args])\} \\ -\{Rem1([args]), \ldots, Remp([args])\} \end{aligned}$$

This structure allows us to identify two types of relations between agents representing these rules:

- A rule-agent $R1$ interferes with another rule-agent $R2$ because (1) $R1$ performs an action which contains effects that are opposite to those of $R2$ or (2) by executing an action, $R1$ removes at least one of the conditions of $R2$. This relation is called inhibition of effects in the first case and is written: $InhibitsE(R1, R2)$; inhibition of activation in the second case which is written: $InhibitsA(R1, R2)$.
- A rule-agent $R1$ helps another rule-agent $R2$ because it makes at least one of the conditions of $R2$ appear. This relation is called permission and is written: $Permits(R1, R2)$.

As a simulated agent executes, it gives its conclusions to the BSDM expressed as logical predicates. If conditions of a rule-agent R can be instantiated by these predicates, R may be triggered and this state is written: $MayBeTriggered(R)$. R is actually triggered (and possibly executed by the simulated agent) if when R has to act, depending on its level in the hierarchy, its conditions are still satisfied, this is written: $IsTriggered(R)$.

We also consider that a rule-agent $R1$ knows whether it is situated above another one $R2$ in the hierarchy, this is written: $Above(R1, R2)$.

For performance reasons and because the solving process in AMAS is a local one, cooperative agents have only a local view of their environment. Consequently, a rule-agent only knows some other rule-agents, called neighbours: those which may be triggered, those which are actually triggered and the rule-agent which is situated just immediately above it.

Finding the right hierarchy is to find the right organisation between the rule-agents composing the self-design module. By definition this organisation has to emerge from the cooperative interactions between rule-agents. Cooperation rules that must be obeyed by rule-agents have then to be defined considering the above relationships.

4.2 Non Cooperative Situations Encountered by Rule-Agents

Hints for finding the cooperation rules that govern rule-agents can be discovered considering the assumptions that make a hierarchy being the right one:

- Rules that are triggered have to always be consistent because a simulated agent cannot execute some actions which inhibit each other,
- Rules that have little probability to be executed have to be given priority because it was assumed that every rule is useful,
- Considering that this last point remains satisfied, a maximum number of rules has to be executed making the simulated agent the most efficient possible.

If one of these criteria is not satisfied, the order of the hierarchy is not optimal i.e. at least two rule-agents are in a NCS.

All the kinds of NCS that can theoretically appear in an AMAS [4] were studied with respect to these previous assumptions and three potential NCS were identified between rule-agents:

- Theoretically a conflict may appear when two agents have antinomic effects. A conflict occurs then if two rule-agents $R1$ and $R2$ are triggered but $R1$ prevents $R2$ from being expressed i.e. $InhibitsE(R1, R2) \wedge IsTriggered(R1) \wedge IsTriggered(R2)$.
- Another conflict is possible if $R1$ is above $R2$, $R2$ could be activated but is not triggered and $R1$ inhibits $R2$ because it is actually triggered i.e. $InhibitsA(R1, R2) \wedge IsTriggered(R1) \wedge MayBeTriggered(R2) \wedge \neg IsTriggered(R2) \wedge Above(R1, R2)$.
- An NCS of uselessness may occur when a cooperative agent believes what it is going to do is not beneficial for others or itself. A uselessness situation will then appear if a rule-agent $R1$ is triggered, another one $R2$ situated above $R1$ is not and $R1$ considers that it could have helped $R2$ to be triggered i.e. $IsTriggered(R1) \wedge Above(R2, R1) \wedge \neg IsTriggered(R2) \wedge Permits(R1, R2) \wedge \neg InhibitsA(R1, R2) \wedge \neg InhibitsE(R1, R2)$.

In order to detect possible NCS, a rule-agent learns by observing the relationships it has with its neighbouring rule-agents. To always stay in a cooperative state, this agent has then to remove the NCS it has detected.

4.3 Solving NCS by Moving in the Hierarchy

Rule-agents autonomously change their interactions by going up or down the hierarchy in order to avoid or remove NCS. For simplifying the behaviour of these agents, they can only move from one position to the adjacent one at the same time. Furthermore, considering that when "$R2$ is above $R1$", "$R1$ moves up just above $R2$" is equivalent to "$R2$ moves down just below $R1$", giving the ability to move in only one direction is enough (for instance, upwards). A rule-agent that succeeds in climbing the hierarchy by only local ascents had then

all the legitimate reasons to be there. Therefore a rule-agent is going to move upwards for solving NCS.

If several NCS are encountered by a cooperative agent, this latter is not always able to remove them all and a priority to process them has to be given: a rule-agent tries first to avoid interfering with others and then tries to help them. A rule-agent is then going to try to avoid conflicts before solving situations of uselessness.

Let us suppose now that a rule-agent $R1$ is above another rule-agent $R2$. If $R2$ moves above $R1$, it prevents it from going up. In order to stay the most cooperative possible, a rule-agent can move up only if it has at least as many reasons to go up than the rule-agent just above. Another criterion is then taken into account for allowing rules to move: the priority they have to do this. This priority is inversely proportional to the number of times a rule-agent R verified $MayBeTriggered(R)$.

To sum up, a rule-agent $R1$ decides to go up considering the rule $R2$ just below it as follows:

- If $R1$ wants to move up because of a conflict and $R2$ wants also to go up because of a situation of uselessness, then $R1$ has priority over $R2$; solving a conflict is more important than removing a uselessness situation.
- If $R1$ and $R2$ are motivated to move up by the same kind of NCS, then the rule with the highest priority does.
- If $R1$ wants to go up for solving an NCS and $R2$ has detected no NCS, then $R1$ is free to move up.

4.4 Mechanism of Self-Organisation

The mechanism that makes rule-agents reorganise their interactions in an autonomous way is a loop made up of two steps:

1. In the first one, each rule-agent of the hierarchy which is possibly and/or actually triggered learns the new relationships it has with its neighbours considering the execution of the current hierarchy.
2. In the second step, each rule-agent (from the one on top of the hierarchy to the one at the bottom) which is possibly and/or actually triggered, reasons and decides to move (up) in the hierarchy or to stay where it is depending on the relationships it has previously learned and the NCS it has therefore may be detected.

Because rule-agents have only a local view, they do not have a way for knowing whether the current hierarchy they belong to is the right one i.e. the behaviour they have collectively found is the best one for the simulated agent they are working for. To be stopped, this mechanism needs therefore an external intervention, coming from a designer who observes the simulation and the dynamic structuring of the hierarchy and decides that the obtained hierarchy is steady and efficient enough for forming the behaviour he/she was seeking.

This module has been implemented as a plug-in for the SeSAm simulation platform [11]. In order to validate the approach proposed above, experiments were carried out on a MAS aimed at simulating a society of foraging ants.

5 EXPERIMENTS ON SELF-DESIGN OF ANTS BEHAVIOUR

This MAS was implemented using SeSAm and the ant agents composing it use this module to automatically find their behaviour. The set of rules, in which the order of rules is completely arbitrary, initially given to the simulated ant-agents is the following:

R1 $AvailableTime() \wedge FullLoaded() \wedge Nest(X) \wedge On(X) \rightarrow DropFood(X)$
R2 $AvailableTime() \wedge FullLoaded() \wedge NoCollision() \wedge Nest(X) \rightarrow MoveTowards(X)$
R3 $AvailableTime() \wedge Foraging() \wedge Food(X) \wedge On(X) \rightarrow PickUpFoop(X)$
R4 $AvailableTime() \wedge Foraging() \wedge NoCollision() \wedge Food(X) \wedge Near(X) \rightarrow MoveTowards(X)$
R5 $AvailableTime() \wedge Foraging() \wedge NoCollision() \wedge Pheromone(X) \wedge Near(X) \rightarrow MoveTowards(X)$
R6 $AvailableTime() \wedge Foraging() \wedge NoCollision() \rightarrow FreeMove()$
R7 $AvailableTime() \wedge Collision() \rightarrow DodgeObstacle()$
R8 $FullLoaded() \rightarrow DropPheromone()$

Where the meaning of the conditions are the following:

- $AvailableTime()$: means that the ant has time to execute an action
- $Collision()$: the ant collides with an obstacle (another ant is also considered as an obstacle)
- $Foraging()$: the ant forages (searches randomly after food)
- $Food(X)$: X is an item of food
- $FullLoaded()$: the ant has picked up some food and cannot pick up more
- $Near(X)$: the ant has detected X which is near it
- $Nest(X)$: X is a nest of a colony of foraging ants
- $NoCollision()$: the ant has no risk of collision with an obstacle
- $On(X)$: the ant is on X
- $Pheromone(X)$: X is some pheromone item

Effects of actions a simulated ant-agent can take are described as follows:

A1 $DropFood(X) : +\{Foraging()\}; -\{AvailableTime(), FullLoaded()\}$ This means that the effect of dropping food enables the ant to forage again ($+Foraging()$) and implies the ant has no time to do another action during this step ($-AvailableTime()$ and it is no more full loaded ($-FullLoaded()$).
A2 $MoveTowards(X) : +\{\}; -\{AvailableTime()\}$ Effect of moving towards another location implies the ant has no time to do another action during this step ($-AvailableTime()$).

A3 $PickUpFood(X) : +\{FullLoaded()\}$;
$-\{AvailableTime(), Foraging(), Food(X), On(X)\}$ When an ant picks up food this implies that this ant is full loaded ($+FullLoaded()$), it has no time to do another action during this step ($-AvailableTime()$), it cannot continue to forage ($-Foraging()$), the food X exists no more ($-Food(X)$) and the ant is no more on X ($-On(X)$).

A4 $FreeMove(X) : +\{\}; -\{AvailableTime()\}$ The effect of moving anywhere for an ant implies this ant has no time to do another action during this step ($-AvailableTime()$).

A5 $DodgeObstacle(X) : +\{NoCollision()\}$;
$-\{AvailableTime(), Collision()\}$ This means that the effect of avoiding an obstacle implies there is no more collision ($+NoCollision()$), the ant has no time to do another action during this step ($-AvailableTime()$) and the collision is avoided ($-Collision()$).

A6 $DropPheromone(X) : +\{\}; -\{\}$ This means that dropping pheromone has no effect.

For example, according to the rule $R3$ if an ant is foraging (randomly moving) and is situated on X where X is recognised as food, then this ant acts to pick up the food it has found. Acting for picking up the food ($A3$) makes this ant full loaded, and removes the time allocated to the ant for this step, its foraging state, the food from the environment and the fact that the ant is on a piece of food. Almost all the actions of the given rules consume the time allocated to an ant-agent for the current step, almost all the rules are therefore going to inhibit each other. Some rules are helping each other, for instance, $R3$ permits $R8$.

Among the different experiments that were carried out to evaluate and validate the proposed approach to implement the module, only the ordered set of rules obtained and the convergence study are presented here. In these experiments, ant-agents are using the self-design module in order to automatically reorganise their set of rules and improve their behaviour.

5.1 Final Order of Rules

The first experiments done enable us to observe the evolution of the self-organising mechanism and to ensure that the ordered set of rules obtained is stable and functionally adequate with the simulation environment. Because the rule-agents have no knowledge of the global system and process, an observer has to decide that the solution has reached a stable and proper state. This observer can estimate that the system is steady enough when the order between rules does not change for around 100 cycles. During a cycle, every rule-agent of the BSDM behaves by detecting NCS and deciding to change its location. The functional adequacy is judged by the observer by analysing the behaviour of one ant in its environment (see figure 1).

The experiment with the same initial unordered list of rules (section 5) was done 50 times. At the end of each series of experiments, it can be seen that rules are ordered in the same way with the exception of the last two rules ($R2$, $R7$)

(see below). This can be explained because rules $R2$ and $R7$ have no inhibition link and their order is not significant. We can assume that the solution obtained by the BSDM is quasi-unique.

The ordered rules obtained at the end of the self-organising process with $R2$ above $R7$ is the following:

SET 0

/* rules which can be executed with all other sets if their conditions are satisfied*/

R8 $FullLoaded() \rightarrow DropPheromone()$

SET 1

R3 $AvailableTime() \wedge Foraging() \wedge Food(X) \wedge On(X) \rightarrow PickUpFoop(X)$

SET 2

R4 $AvailableTime() \wedge Foraging() \wedge NoCollision() \wedge Food(X) \wedge Near(X) \rightarrow MoveTowards(X)$

SET 3

R5 $AvailableTime() \wedge Foraging() \wedge NoCollision() \wedge Pheromone(X) \wedge Near(X) \rightarrow MoveTowards(X)$

SET 4

R6 $AvailableTime() \wedge Foraging() \wedge NoCollision() \rightarrow FreeMove()$

R1 $AvailableTime() \wedge FullLoaded() \wedge Nest(X) \wedge On(X) \rightarrow DropFood(X)$

SET 5

R2 $AvailableTime() \wedge FullLoaded() \wedge NoCollision() \wedge Nest(X) \rightarrow MoveTowards(X)$

R7 $AvailableTime() \wedge Collision() \rightarrow DodgeObstacle()$

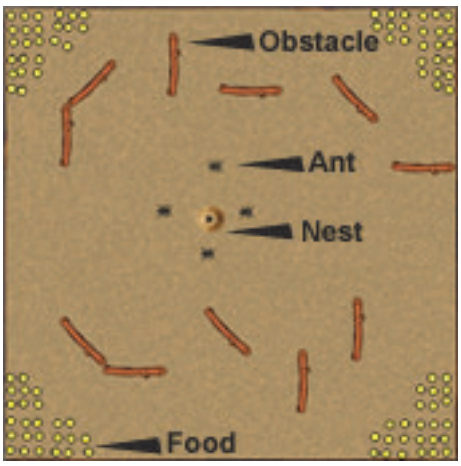

Fig. 1. Environment in the considered experiments (with 4 ants in this case)

In this list, during a cycle, the set i-1 subsumes the set i (except set 0 for which rules can be executed with all the others) and all the rules in a same set are triggered during the same cycle.

5.2 Study of Convergence

The following experiment studies how fast an accurate solution is found. The number of cycles required to have a steady hierarchy is measured for four different situations: when 1, 2, 4 or 8 ants are evolving in the simulated MAS and are simultaneously using the BSDM to learn their own behaviour (see table 1).

Number of cycles	With 1 ant	With 2 ants	With 4 ants	With 8 ants
Minimum number	401	410	384	389
Maximum number	2561	1997	1843	1285
Mean number	803	643	584	508

Table 1. Number of simulation cycles before reaching a steady herarchy

These results show that the learning time (number of needed cycles) is relatively short compared to the number of cycles of a simulation, in the best case, 400 cycles are enough to make ant-agents have a steady behaviour. We can observe that the mean number of cycles diminishes when the number of ants increases, this can be explained by the fact that the more ants there are and the more they can be aware of new situations and the faster they can "learn".

6 SIMULATION AND SOFTWARE ENGINEERING

Other works that use simulation for designing agent-based systems or multi-agent systems exist and this section places our work with regard to them.

Gardelli et al. assess that simulation could provide a substantial added-value when applied to support the development process of self-organising systems [13]. Simulation is used to detect abnormal agents' behaviours in a system in order to improve security. Authors took inspiration from the human immune system and exploit Pi-Calculus in the TucSoN infrastructure for simulating the security system. Unlike the aim of the work proposed here, the use of formal simulation is intended to let designers detect abnormal behaviours at the early stages of design before implementing any prototype. In our work, simulation is used at a different level of the development process, not in the early stage but during the implementation stage.

Rhl and Uhrmacher propose a modelling and simulation framework called JAMES, based on the discrete event formalism DYNDEVS for supporting the development process of multi-agent systems [14]. Agents' behaviour is validated on a model in a virtual environment and the real implementation starts when

the model is mature. Sierra et al. develop an Integrated Development Environment to design e-institutions [15]. Within this environment, a simulation tool SIMDEI is used to dynamically verify the specifications and the protocols to be implemented. Fortino et al. propose a simulation-driven development process [16, 17] and enrich the PASSI (Process for Agent Societies Specification and Implementation) methodology with the simulation tool called MASSIMO (Multi-Agent System SIMulation framewOrk), a Java-based discrete event simulation framework [18]. Authors propose a semi-automatic translation of the agent implementation model provided by PASSI in the distilled statecharts specification of a multi-agent system needed by MASSIMO. At present, the simulation is used for validating the requirements of the system and evaluating some performances.

In these last three works, the simulation is more used to analyse the running system and to verify the behaviours of agents and system. In our work, the simulation is also used to verify the global behaviour but also for taking a part to the design process itself. The main objective is to help designers to adjust and to build the behaviour of agents.

De Wolf et al. combine agent-based simulations with scientific numerical algorithms for dynamical systems design [19]. In this approach, designers have to define the results that are expected from the analysis, the parameters of the simulation are then initialised and simulations are launched. Finally, results of the simulation are analysed and depending on the outcomes, the next initial values are determined. The analysis approach is integrated into the engineering process in order to achieve a systematic approach for building self-organising emergent systems. In this way this work is probably the most close to ours.

7 CONCLUSION AND PERSPECTIVES

This article has studied how simulation could be used to alleviate designers' task when building complex multi-agent systems that have to operate in evolving and open environments. The approach adopted here proposes to implement a prototype of the targeted MAS under the SeSAm platform in which each (simulated) agent uses a Behaviour Self-Design Module to find the most efficient behaviour.

The BSDM is designed, following the AMAS theory we presented, as an adaptive MAS made up of agents that represent the behavioural rules of the simulated agent. An organisation of this AMAS represents a certain behavioural hierarchy and therefore a certain behaviour for the simulated agent that uses the BSDM. The objective of a rule-agent within the BSDM is to find its right place in the behavioural hierarchy i.e. to establish the relationships with other rules that make it stay the most cooperative possible. We studied and enumerated the possible relationships between rule-agents to determine the cooperative attitude of a rule-agent and the actions a rule-agent could do to remove the situations that prevented it from staying cooperative toward others and itself. This has enabled us to describe the self-organisation mechanism used by the BSDM to establish the subsumption-like hierarchy of behavioural rules used by a simulated agent.

The BSDM was implemented as a plug-in for SeSAm and experiments were carried out to validate the proposed approach. Even if more experiments are still needed, preliminary results show that this approach is feasible and can help a designer in three different ways:

- Simulation may make him/her saving time. Behavioural rules can be given in any order enabling him/her to observe their impact on the individual behaviours (agent level) or on the collective one (system level).
- Simulation may assist him/her by proposing a behavioural strategy that is efficient and adapted to the simulated environment.
- Simulation can be used for any application based on such a kind of reactive behavioural rules. The self-organisation process of rules is not based on their semantics but on their relationships that are automatically detected by the provided plug-in.

However a great number of improvements have to be made yet. For instance, this study was limited to a complete set of consistent rules. Even if this limitation does not prevent the module from providing a designer with a consistent hierarchy, always supplying such a complete set may represent a difficult task for this designer. The mechanism of self-organisation used by the BSDM could also be improved by giving it the ability to judge the impact of a triggered rule on the environment of the simulated agent that uses this module. NCS encountered by this latter agent should then be taken into account and used by the BSDM to question the place of the involved rule. The work presented in [7] could be used to detect NCS, however integrating this detection in the current module seems to be a difficult but interesting perspective at the present time.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence - from Natural to Artificial Systems. Oxford University Press (1999)
2. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies. Idea Group Pub (2005)
3. Bergenti, F., Gleizes, M.P., Zambonelli, F.: Methodologies and Software Engineering for Agent Systems. Kluwer Publishing (2004)
4. Capera, D., Georgé, J.P., Gleizes, M.P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: 12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Linz, Australia, IEEE Computer Society 2003 (2003) 383–388
5. Picard, G., Gleizes, M.P.: The ADELFE Methodology - Designing Adaptive Cooperative Multi-Agent Systems. In Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems. Kluwer Publishing (2004) 157–176
6. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Engineering Adaptive Multi-agent Systems: the ADELFE Methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Pub (2005) 172–202
7. Bernon, C., Gleizes, M.P., Picard, G.: Enhancing Self-Organising Emergent Systems Design with Simulation. In 2006, E., ed.: 6th International Workshop Engineering Societies in the Agent World, Dublin, Ireland (2006) 284–299

8. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-Organization and Emergence in Muli-Agent Systems. The Knowledge Engineering Review **20**(2) (2005) 165–189
9. Heyligen, F.: The Science of Self-organization and Adaptivity – Knowledge Management, organizational Intelligence and Learning, and Complexity. In: The Encyclopedia of Life Support Systems. Publishers (2003)
10. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)
11. Klügl, F., Herrler, R., Oechslein, O.: From Simulated to Real Environments: How to use SeSAm for Software Development. In Schillo, M., Klusch, M., Müller, J.P., Tianfield, H., eds.: Multiagent System Technologies - 1st German Conferences MATES. Volume 2831 of LNCS., Springer-Verlag (2003) 13–24
12. Brooks, R.: A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation **2**(1) (1986) 14–23
13. Gardelli, L., Viroli, M., Omicini, A.: On the Role of Simulations in the Engineering of Self-Organising MAS: the case of an intrusion detection system in TuCSoN. In: 3rd International Workshop Engineering Self-Organising Applications. (2005) 161–175
14. Röhl, M., Uhrmacher, A.: Controlled Experimentation with Agents - Models and Implementations. In Gleizes, M.P., Omicini, A., Zambonelli, F., eds.: 5th International Workshop Engineering Societies in the Agent World. Volume 3541 of LNAI., Springer-Verlag (2005)
15. Sierra, C., Rodriguez-Aguilar, J., Noriega, P., Esteva, M., Arcos, J.: Engineering Multi-Agent Systems as Electronic Institutions. Novatica **170** (2004)
16. Fortino, G., Garro, A., Russo, W.: A Discrete-Event Simulation Framework for the Validation of Agent-based and Multi-Agent Systems. In: Workshop on Objects and Agents, Camerino, Italia (2005)
17. Fortino, G., Garro, A., Russo, W., Caico, R., Cossentino, M., Termine, F.: Simulation-Driven Development of Multi-Agent Systems. In: Workshop on Multi-Agent Systems and Simulation, Paermo, Italia (2006)
18. Cossentino, M.: From Requirements to Code with the PASSI Methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Pub (2005) 79–106
19. De Wolf, T., Samaey, G., Holvoet, T.: Engineering Self-Organising Emergent Systems With Simulation-Based Scientific Analysis. In Brueckner, S., Di Marzo Serugendo, G., Hales, D., Zambonelli, F., eds.: the Third International Workshop on Engineering Self-Organising Applications, Utrech, Netherlands (2005) 146–160