Formal driven prototyping approach for multiagent systems

Vincent Hilaire*, Pablo Gruer and Abderrafiâa Koukam

SeT Laboratory
UTBM
90010 Belfort Cedex, France
E-mail: vincent.hilaire@utbm.fr
E-mail: pablo.gruer@utbm.fr
E-mail: abder.koukam@utbm.fr
*Corresponding author

Olivier Simonin

Maia Team
LORIA
BP 239 – 54506 Vandoeuvre-ls-Nancy Cedex, France
E-mail: olivier.simonin@loria.fr

Abstract: Even if Multiagent Systems (MASs) are recognised as an appealing paradigm for designing many computer systems ranging from complex distributed systems to intelligent software applications, they are still difficult to engineer. A common point of MAS engineering methodologies is the production of different work products or models during the different phases. Each work product or model helps to progress towards an implementation of the system under development. Nevertheless, there are very few ways to ensure that the work products and models produced are validated against the requirements. The aim of this paper is to present a formal driven prototyping approach for MAS. This process provides a support for incremental specification leading to an executable model of the system being built. Indeed, this process is based upon a formal organisational framework which uses a formal notation, namely OZS. The process is illustrated through the specification of a multiagent architecture.

Keywords: Multiagent Systems; MASs; formal specification; validation; verification.

Reference to this paper should be made as follows: Hilaire, V., Gruer, P., Koukam, A. and Simonin, O. (2008) 'Formal driven prototyping approach for multiagent systems', *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No. 2, pp.246–266.

Biographical notes: Associate Prof. Vincent Hilaire obtained his PhD in Computer Science and his position at the University of Technology of Belfort Montbéliard in 2000. The main concern of the PhD was formal specification and methodologies for engineering of Multiagent Systems (MASs). Since then

his researches follow the axes MAS organisational theories and holonic systems, languages for formal specification prototyping and proofs of MASs, agent architectures and agent-mediated knowledge management.

Dr. Juan Pablo Gruer holds an Engineering degree in Computer Science (1982) from ENSIMAG, Grenoble, France. After five years in industry as a designer of real-time systems and applications, Dr. Gruer obtained in 1989 a Doctoral degree in Computer Science from the 'Universite de Haute Alsace', Mulhouse, France. Presently, Dr. Gruer is an Associate Professor in Computer Science at the 'Université de Technologie de Belfort-Montbeliard', and the Systems and Transportation Laboratory, where he works on software engineering issues for the design of MAS applications and on MAS-based approaches to the modelling and simulation of transportation systems.

Prof. Abderrafiâa Koukam received his PhD in Computer Science from the University of Nancy I (France) in 1990, where he served as an Assistant Professor from 1986 to 1989 and as Researcher at 'Centre de Recherche en Informatique de Nancy' from 1985 to 1990. In 1999, He received the enabling to direct researches in computer science from the University of Bourgogne. Presently, he is Professor of Computer Science at 'Université de Technologie de Belfort-Montbeliard'. He heads research activities at the Systems and Transportation Laboratory on modelling and analysis of complex systems, including software engineering, MASs and optimisation.

Olivier Simonin received his BSc, MSc and PhD in Computer Science in 1995, 1997 and 2001 from the University of Montpellier II, France. Since 2001 he has been an Associate Professor in Computer Science, specialised on reactive multiagent models and bio-inspired algorithms. His main focuses are distributed problem solving, simulation and design of collective robotics systems. From 2001 to 2006, he worked for the University of Technology Belfort-Montbéliard, France, as a member of the SeT laboratory ICAP team. Then he moved to INRIA Maia project, as a member of the LORIA laboratory and of the University Henri Poincaré – Nancy 1, France.

1 Introduction

Even though Multiagent Systems (MASs) are recognised as an appealing paradigm for designing many computer systems ranging from complex distributed systems to intelligent software applications, they are still difficult to engineer. When a massive number of autonomous components interact, it is very difficult to predict that the emergent organisational structure will fit the system goals or that the desired functionalities will be fulfilled.

Many methodologies exist for engineering MASs (see Bernon *et al*., 2005; Bergenti *et al*., 2004 for a survey). The goal of these methodologies is to help the developer during the analysis and design phase of a MAS. A common point of many of them is the production of different work products or models during the different phases. For example, the PASSI methodology (Chella *et al*., 2004) uses UML and AUML (Bergenti and Poggi, 2000) models. Each work product or model guides subsequent design, implementation and verification phases. Nevertheless, there are very few ways to ensure that the work products and models produced are validated against the requirements.

The aim of this paper is to present a formal driven prototyping approach for MASs. We believe that one way to bridge the gap between the abstract and the concrete level is to build the system specification using a prototyping process (Lissajoux *et al*., 1998). This process provides a support for incremental specification, leading to an executable model of the system being built. Indeed, in many areas of software and knowledge engineering, the development process, which puts emphasis on prototyping and simulation of complex systems before their effective implementation, is proven to be a valuable approach. Indeed, prototyping enables the testing of MASs before their actual deployment and execution. Another important application of this approach is the specification and prototyping of specific models or architectures of agent and multiagent systems. These models are often defined in an informal way and applied in an *ad hoc* fashion. Consequently, MAS designers have been unable to fully exploit these models' commonalities and to specialise or reuse them for specific problems. Thus, formal specification can be used to describe model concepts which can be refined to fulfil particular system needs. We illustrate this prototyping approach through the formal specification of a multiagent architecture: the satisfaction/altruism model (Chapelle *et al*., 2002; Simonin and Ferber, 2000). This model is based on a behaviour-based architecture and introduces a cooperation mechanism for situated agents.

Our approach is based on an organisational formal framework. This framework uses organisational concepts such as role, interaction and organisation to describe MASs. Each concept is given a formal semantics using the OZS formal notation (Gruer *et al*., 2004). OZS is a multiformalism notation based upon the composition of Object-Z (Duke *et al*., 1991) and statecharts (Harel, 1987). This notation allows the specifying of reactive and transformational aspects of agent roles. We have defined a formal semantics which allows the execution of the specifications and the use of theorem provers such as STeP (Manna *et al*., 1995) or SAL (de Moura *et al*., 2004). With this approach we have prototyped several MASs (Hilaire *et al*., 2001; Gruer *et al*., 2002; 2001; Bakhouya *et al*., 2003; Hilaire *et al*., 2005; Rodriguez *et al*., 2007). This paper is organised as follows: Section 2 presents the OZS formal notation and the organisational framework defined with this notation. Section 3 presents some examples of the use of our approach. Section 4 presents related works. Section 5 concludes.

2 Concepts

2.1 OZS: multiformalism specification

Few specification languages, if any, are well suited to model all aspects of a system. This has led to the development of new specification languages which combine features of two or more existing formalisms. These languages are called multiformalisms. Such a combination is particularly suited to the specification of complex systems, such as MASs, where the modelling of both processes and states is necessary.

The multiformalism approaches (Zave and Jackson, 1993; Paige, 1997) compose two or more formalisms in order to produce system specifications more easily and naturally than with a single formalism. Indeed, the multiformalism approach deals with complexity by applying formalisms to problem aspects for which they are best suited.

There are two kinds of techniques for multiformalism integration. The first consists in translating one formalism into another. The second is composition and consists in using several formalisms in the same specification. We have chosen the latter type of approach.

The main principle of our approach is to integrate within an Object-Z class a specific schema called behaviour that specifies the behaviour of the class using a statechart. It enables specifiers to use all Object-Z and statechart constructs.

Object-Z extends Z with object-oriented concepts. The basic construct is the class, which encapsulates a state schema with all the operation schemas that may affect it. Object-Z is well suited for specifying the state space and the methods of a class in a predicative way. It is, however, weak at describing dynamic and communication aspects (Smith, 1997).

Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication. Both languages have constructs which enable the refinement of the specifications. Moreover, statecharts have an operational semantic which allows the execution of a specification owing to the STATEMATE environment (Harel *et al*., 1990). However, statecharts have little support for modelling the structural and functional aspects of a complex system.

Our method for composition relies on precisely combining the two notations. We define a heterogeneous basis consisting of the notations of interest and we resolve syntactic differences among the notations as presented in Paige (1997). The role of the heterogeneous basis is twofold. First, it provides relationships between Object-Z and statecharts without translating a formalism into another. Second, it extends expressive capabilities of each formalism using features available in the other. In other words, the heterogeneous basis enables the use of both specification styles without being restrained to a subset of any of the formalisms.

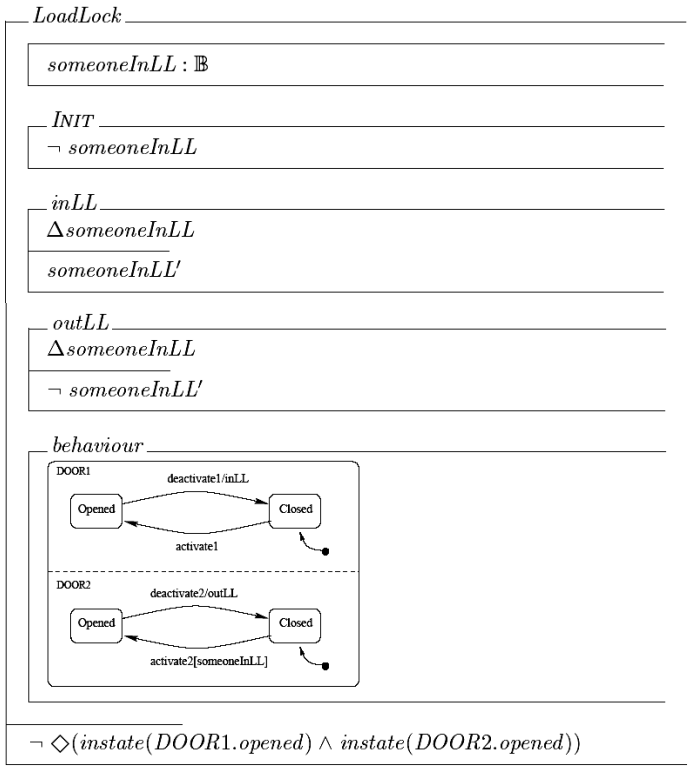
The class describes the attributes and operations of the objects. This description is based upon set theory and first-order predicate calculus. The statechart specifies the possible states of the object and how events may change these states. The statechart included in an Object-Z class can use attributes and operations of the class. The sharing mechanism used is based on name identity. Moreover, we introduce basic types [*Event, Action, Attribute, State*]. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statechart actions and Object-Z class operations. *State* is the set of states of the included statechart.

Operations are described by Object-Z schemas and can be referenced in statecharts transition. *Attribute* is the set of object attributes. The definition of these sets allows the specification in Object-Z of statechart features, such as events and states, and the use of Object-Z features such as attributes and operations within the included statechart.

2.2 Example

The *LoadLock* class, presented below, illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors whose states evolve concurrently. The key syntactic element is the class schema. Each class has a name, (*LoadLock* in the example), and contains subschemas that specify different aspects of the class. The first subschema specifies the state space of the class. In the *LoadLock* example there is only one boolean attribute called *someoneInLL.* The following subschema defines the initial state, for instance, of the class, and is called *Init.* The constraint placed here states that initially, the boolean *someoneInLL* is false. The next two subschemas are operations that modify the state of the class. The *inLL* and *outLL* operations are divided into two parts: a declarative part in the upper part of the subschema and a constraints part in the lower predicate part.

The Δ-list *someoneInLL* in the upper part of both subschemas is an abbreviation for *someoneInLL* and *someoneInLL'* and, as such, includes the state of *someoneInLL* before and after the operation. The predicate part of the *inLL* (or *outLL*) operation states that after the operation *someoneInLL* becomes true (or false). The last subschema, called behaviour, includes a statechart and specifies the behaviour of the class. Parallelism between the two doors is expressed by the dashed line between *DOOR1* and *DOOR2*. The first door reacts to *activate1* and *deactivate1* events. The sequence, in order to go through the loadlock, is the following: someone activates the first door and enters the loadlock. It may then enter the loadlock and deactivate the first door from inside. This done, he can activate the second door and go out of the loadlock. The transition triggered by the *deactivate1* event executes the *inLL* operation, which sets the *someoneInLL* boolean to true. The temporal invariant at the end of the class specifies that a *LoadLock* must not be in *DOOR1.opened* and *DOOR2.opened* states simultaneously. This invariant uses the predicate *instate(S)*, which is true whenever the *S* state of the statechart is active.



The result of the composition of Object-Z and statecharts seems particularly suited for specifying MASs. Indeed, each formalism has constructs which enable complex specifications. Moreover, aspects such as reactivity and concurrency can be easily dealt with. The semantics of the OZS notation is defined by means of transition systems (Gruer *et al*., 2004). It is an operational semantics which enables automatic verification of specification properties.

Available OZS constructs enable the natural specification of 'low'-level aspects inherent to MASs. Higher-level aspects like coordination are expressed by roles, interactions and organisation classes, which are presented in the following section.

2.3 RIO

Our specification approach uses an organisational metamodel which is based on three interrelated concepts: Role, Interaction and Organisation. Roles are generic behaviours. These behaviours can interact mutually according to interaction patterns. Such a pattern, which groups generic behaviours and their interactions, constitutes an organisation. Organisations are thus descriptions of coordination structures. Coordination occurs between roles as and when interactions take place.

In this context, an agent is only specified as an active communicative entity which plays roles (Ferber and Gutknecht, 1998). This model places no constraints on the internal architecture of agents and does not assume any formalism for individual agents. In fact, agents instantiate an organisation (roles and interactions) when they exhibit behaviours defined by the organisation's roles and when they interact following the organisational interactions. The main reason for this choice is that one can study agent behaviours and agent architectures separately. Indeed, the different roles an agent plays define its behaviour. The architectures used by agents may be different for the same behaviour and so it is sound to study them apart from the core agent behaviour. For example, in Gruer *et al*. (2002) we have specified a specific cognitive agent architecture by extending the RIO framework classes.

An agent may instantiate one or more roles and a role may be instantiated by one or more agents. The role-playing relationship between roles and agents is dynamic. It means that agents can change the roles they are playing at run-time. We think this model is a basis for the engineering of societies of agents and what Castelfranchi (2000) calls 'social order'.

We make no assumptions on agent architectures. The generality of the agent definition allows the specification of many agent types. More specific choices can be introduced in more accurate models. This model enables a modular approach by prototyping separate parts of the MAS. The behaviour of the MAS as a whole is the result of the role playing by agents.

The RIO framework is composed of OZS classes (Hilaire *et al*., 2001; 2002), one for each concept in the RIO metamodel. These classes specify the key organisational concepts needed to specify a MAS. In this paper we will describe the following concepts: Role, Interaction and Organisation. The *Role* class is a superclass for all acting entities of the system. A role is a specific behaviour; for example, Lecturer, Researcher and Student are roles. An interaction occurs when two roles communicate; for example, the Lecturer and Student roles interact during a course. Interactions are then defined by the origin and destination roles involved. Organisations are sets of interacting roles. The University organisation may group Lecturer, Researcher and Student roles. The RIO framework is associated with a step-by-step process to guide the development from analysis to design. This process is a refinement-based process where each step is used to build the next step. In the analysis stage, roles in the system are identified and their interactions are specified. Coherent patterns of interacting roles are grouped so as to form organisations. Once

pertinent organisations and their components are specified, the next stage consists in identifying which roles can be played simultaneously by one or several agents and under which constraints.

2.4 Formal specifications

A role is an abstraction of an acting entity. We have chosen to specify it by the *Role* class. This class represents the characteristic set of attributes whose elements are of [*Attribute*] type. These elements belong to the *attributes* set. A role is also defined by stimuli it can react to and actions it can execute. These are specified by *stimulus* set and *actions* set respectively. The [*Attribute*], [*Event*] and [*Action*] types are defined as given types and are not defined further.

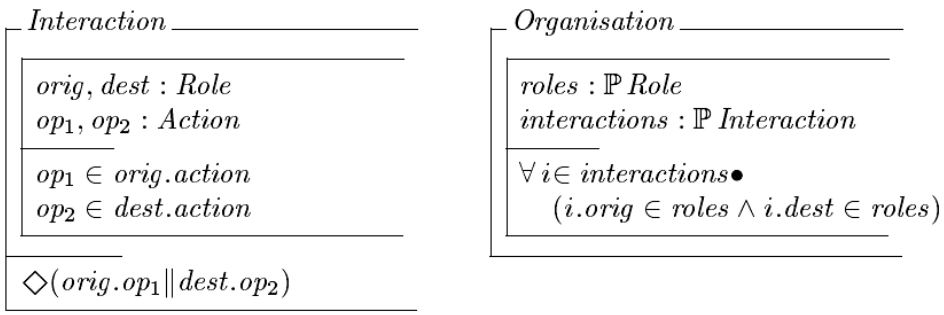
The reactive aspect of a role is specified by the subschema *behaviour*, which includes a statechart. The *behaviour* schema specifies the different states of the role and transitions among these states. The *obtainConditions* and *leaveConditions* attributes specify conditions required to obtain and leave the role. These conditions require specific capabilities or features to be present in the agent in order to play or leave the role. Stimuli which trigger a reaction in the role's behaviour must appear in at least one transition. The action belonging to the statechart transitions must belong to the *actions* set. In order to ensure coherence between Object-Z and statechart parts, we have specified common concepts grouped on a heterogeneous basis following the method of Paige (1997). Two constraints (below the short line in the state subschema) specified in the *Role* class use these heterogeneous basis concepts.

Role

$$
\begin{array}{l}
attributes : \mathbb{P}\, Attribute \\
stimulus : \mathbb{P}\, Event \\
actions : \mathbb{P}\, Action \\
obtainConditions, leaveConditions : Condition \\
\hline
\forall\, s \in stimulus, \exists\, e \in behaviour \bullet \\
\qquad (\exists\, t \in e.transitions \bullet t.label.event = s) \\
\forall\, e \in behaviour \bullet \\
\qquad (\forall\, t \in e.transitions \bullet t.label.action \subseteq actions)
\end{array}
$$

behaviour

An interaction is specified by a couple of roles, which are the origin and the destination of the interaction. The roles *orig* and *dest* interact using operations op_1 and op_2. These operations are combined by the $\|$ operator, which equates output of op_1 and input of op_2. The $\Diamond$ symbol is a temporal logic operator which states that, eventually, the expression following the symbol will be true. In order to extend interactions to take into account more than two roles or more complex interactions, has to create a new class which inherits from the *Interaction* class.



An organisation is specified by a set of roles and their interactions. Interactions happen between roles of the concerned organisation. It means that for each interaction of the *interactions* set, the roles of the interaction must belong to the *roles* set of the organisation.

3 Validation and verification

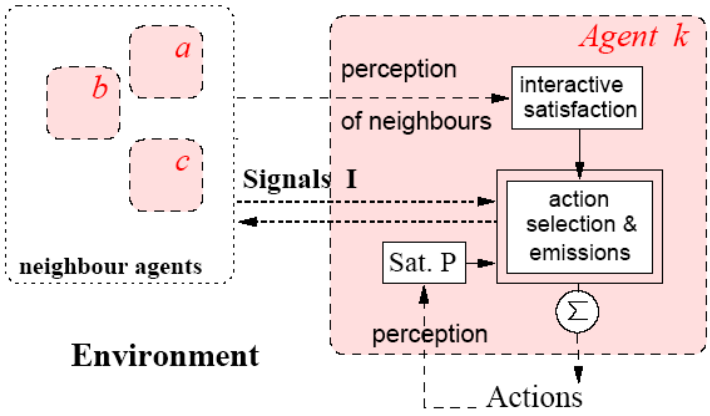
The example described here illustrates two different aspects of the prototyping process. The first aspect concerns the validation of the specification. We have worked on an already existing MAS architecture, named satisfaction/altruism model, dedicated to collective mobile robots (Simonin and Ferber, 2000). We have made a reverse engineering process to extract from this architecture the different roles and their specifications (Hilaire *et al*., 2005). The validation consists in asserting that the specification is a good model of the real system. Our approach for establishing this assertion consists in executing the specification and observing the behaviour of the system. The execution of the specification is allowed by the operational semantics of the OZS formalism. This approach is very useful but not complete in the sense that an execution is not a formal proof. It depends on the context of the execution and on the chosen scenarios. To overcome this problem we propose to formally prove some desired properties of the specifications made using our approach. This aspect is detailed in Section 3.2.

3.1 The satisfaction/altruism architecture

The satisfaction/altruism model proposes a generic mechanism to deal with cooperation and conflict solving in situated MASs (Simonin, 2001; Simonin and Ferber, 2000). It allows us to extend the swarm intelligence approach, which requires a lot of agents and time as it relies on indirect communications. So the satisfaction/altruism model introduces direct interactions between agents through the emission of attractive and repulsive signals. These signals act as *new artificial fields* which are *dynamically* generated by agents. They augment the information present in the environment to enable direct cooperation between agents.

At the heart of the agent architecture there are two modules dedicated to individual and cooperative behaviours. The first module computes the Personal Satisfaction (Sat. P box in Figure 1) and the second one measures the agent interaction with its neighbours (Interactive Satisfaction box).

Figure 1 Satisfaction/Altruism model: agent-agents and agent-environment interactions



The personal satisfaction gives an evaluation in real time of the agent task progress. This *individual satisfaction* is expressed by a value $P(t) \in [-P_{max}, P_{max}]$, where P_{max} is a limit depending on the application. The personal satisfaction is computed at each step of the decision-action loop by considering three possible task evolutions: progress (increase in P), regress (decrease in P) and blocking (strong decrease in P) (see details in Simonin, 2001).

At the opposite, the second module evaluates interactions of the agent with its neighbours. This interactive satisfaction can be negative (if the other agent hinders), positive (if the other can potentially help) or neutral.

Both satisfactions are used in the action-selection module to compute two decisions: (1) to decide if the agent must continue or change its current goal (see details in Chapelle *et al*., 2002) and (2) to emit signals in order to influence other agents' behaviours.

We focus on the latter, as direct cooperation relies on signals emitted by agents. Agents can broadcast locally attractive and repulsive signals, named $I(t)$, defined as numerical values with $I(t) \in [-P_{max}, P_{max}]$ (denoted 'Signals I' in Figure 1). The semantic of Signals I is the following: positive values for attractions and negative ones for repulsions.

A *cooperative behaviour* consists in reacting to the perception of such a signal. This reaction is defined as a simple coherent displacement according to the signal semantic: go towards the signal origin when the agent perceives an attraction and go away when it perceives a repulsion. If several signals are simultaneously received, the agent selects the strongest one, which is noted $I_{ext}(t)$, in order to react to the greatest and nearest requests.

This kind of cooperative reaction is called altruistic behaviour in the model, and it is performed only when:

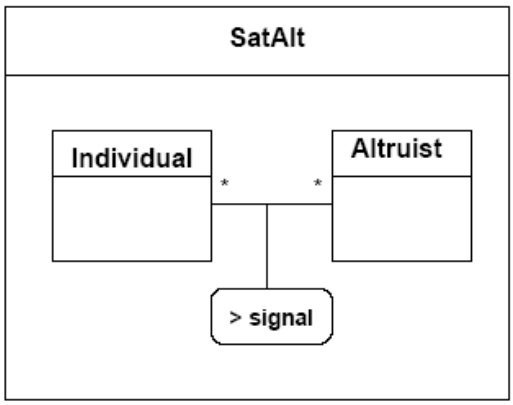
$$| I_{ext}(t) | \geq P(t) \wedge | I_{ext}(t) | \geq I(t).$$

This condition expresses that an agent reacts to an external signal if its intensity is greater than the intensity of its current satisfaction (to give priority to altruism when the current task cannot be achieved easily). The altruistic reaction is defined as a displacement, which is expressed by a vector computed simply from the sign and the intensity of the received signal (according to the semantic defined above). Note that in order to optimise agents' navigation, this motion vector can be combined with others derived from environmental constraints, *e.g.*, obstacle-avoiding vectors (see Simonin *et al*., 2000).

The satifaction/altruism model has been validated in different simulated problems such as collective foraging and box-pushing (Chapelle *et al*., 2002), and with real robots to solve navigation conflicts in constrained environments (see details in Lucidarme *et al*., 2002).

Figure 2 represents the Satisfaction/Altruism organisation. It is specified by two roles: Individual and Altruist. Each role may interact with other Individual and Altruist role-players. The interactions mentioned earlier are signals. The next section presents the formal specification of these roles.

Figure 2 Satisfaction/Altruism RIO diagram



3.2 Specification

In order to specify the Satisfaction/Altruism model we have distinguished two roles: *Individual* and *Altruist.* Figure 2 represents the roles and their interactions.

The first attribute of the *Individual* role, *current*, is the action the role-player is executing. This role is also described by weights associated with the actions it can carry out. These different weights, representing the task's priority, can be modified by the experience of the agent. The initial values of the weights are defined by *initialWeight*. The *progressionReward* function maps each action to a 3-uplet, giving the reward or penalty values when the agent is respectively in progression, regression or locked. The element of *DiscreteSensor* type specifies a sensor which enables the perception of the environment. This part of the specification is out of the scope of this paper so it will not be discussed here. Note that in general, agents emit a value (*I*) equal to their *satisfaction* level. Therefore, *satisfaction* and *I* are numbers which allow the comparison between the level of satisfaction of the role and external influences (perceived signals). The I_{ext} operation outputs the maximum signal perceived.

The *actionSelection* operation chooses the best action according to the personal satisfaction of the current action and the individual weights of the others. There are two different cases detailed in the constraints. The two cases depend on whether there is an action with true preconditions and a greater weight than the current action or not. If there is no such action, the current action remains unchanged; otherwise the current action is replaced with the new one.

Individual
Role

$current : Action$
$initialWeight, weight : Action \rightarrow [0,1]$
$progressionReward : Action \rightarrow BMValue$
$s : DiscreteSensor$
$satisfaction, I : [-P_{max}, P_{max}]$

$current \subseteq actions$
$obtainCondition = \{|I_{ext}()| < P() \vee |I_{ext}()| < I\}$
$leaveCondition = \{|I_{ext}()| \geq P() \wedge |I_{ext}()| \geq I\}$

I_{ext}

$ext! : \mathbb{R}$

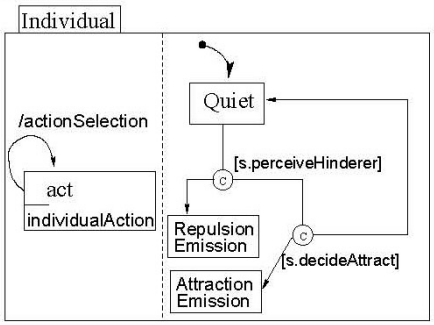
$ext! = s.getMax()$

actionSelection

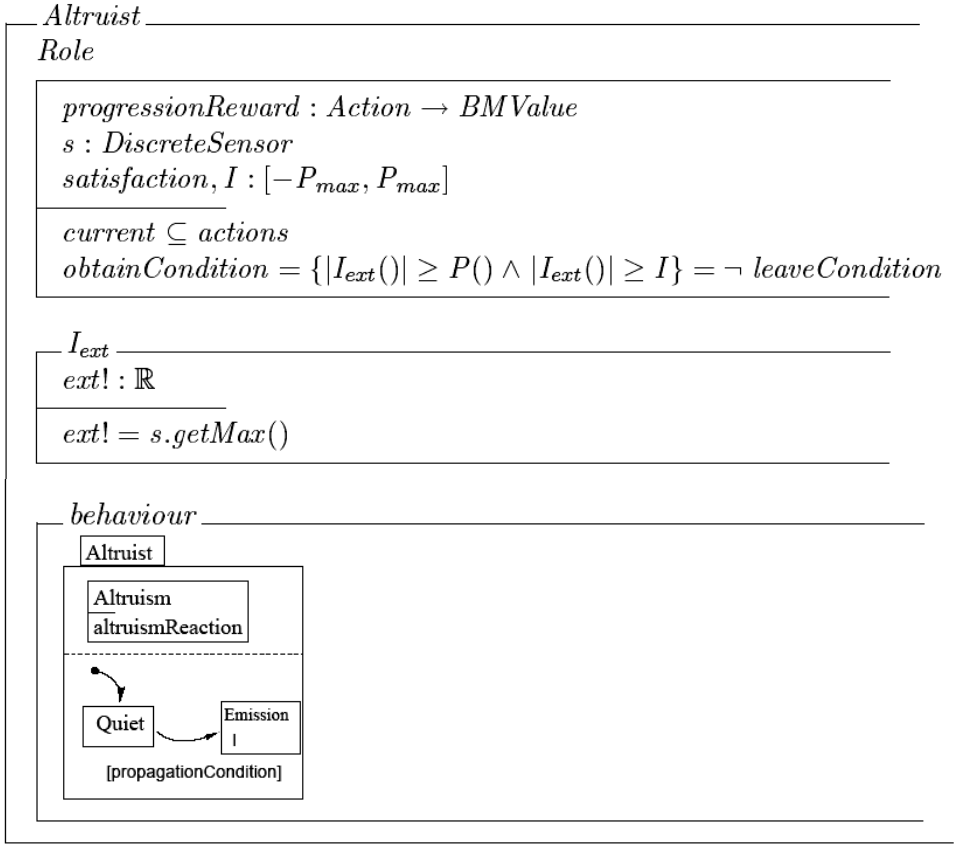
$\Delta(current, weight)$

$$\begin{aligned}(current' = current \wedge (\nexists a : Action \bullet\ & a \in actions \\ & \wedge\ pre\ a \\ & \wedge\ weight(a) - \gamma \geq P())) \\ \vee\ (current' = b \bullet\ & b \in actions \\ & \wedge\ pre\ b \\ & \wedge\ (\nexists a : Action \bullet\ a \in actions \\ & \qquad \wedge\ pre\ a \\ & \qquad \wedge\ weight(a) \geq weight(b)) \\ & \wedge\ \ (P() \leq 0 \Rightarrow weight'(current) = 0))\end{aligned}$$

behaviour



The *Altruist* role is also specified by a *progressionReward* function, a *Discrete-Sensor* and two values – *satisfaction* and Signal I – measuring satisfaction and external influences.



3.3 Validation of the specification

The prototyping is performed by using STATEMATE (Harel *et al.*, 1990), an environment which allows the prototyping and the simulation of the statechart specifications. There exist other software environments for executing statecharts, such as Selic (1996) or Harel and Gery (1997). STATEMATE is one of the most complete tools regarding statechart constructs and semantics, and so we have relied on it. The specification analysis is based upon execution of the statecharts and can be done using two techniques. The first technique is *simulation* and the second is *animation.* In our case simulation would consist in assigning probabilities to events or action occurrences. With this technique one can evaluate quantitative parameters of the specified system. As an example, in the satisfaction/altruism model, probabilities can be assigned to an agent in order to simulate exploration of various environments.

The animation technique consists of testing the specification with predefined interaction scenarios. It enables one to test if the system behaviour is consistent with requirements.

In order to evaluate our specification of the architecture, we simulated the behaviour of two robots evolving in a particular environment. We defined a closed narrow corridor where it is impossible for two agents to intercross, as shown in Figure 3(a). The goal of each agent is to find an exit by exploring the whole corridor. With such an environment, exploration conflicts are unavoidable and lead to the emission of repulsive signals and altruistic reactions. In particular, when the agents meet around the centre they both try to push the other back, which causes a quick fall in their satisfactions. The most dissatisfied agent repulses the other to the end of the corridor. As the ends are closed, the agents will be blocked again. The first agent to arrive at one end of the corridor will be surrounded by three walls. Thus it will be more constrained than the other agent and its satisfaction will decrease faster. The model ensures that it will then repulse the other agent and thus both will continuously explore the environment. If an exit for the corridor is artificially created, the robots will find it.

Figures 3(b) and 3(c) show an example of such a test. The x-axis represents time; and the y-axis represents discretised positions in the corridor for Figure 3(b) and level of satisfaction for each robot for Figure 3(c). One can see that levels of satisfaction and trajectories are correlated. Indeed, each time the two robots are locked in, the satisfaction levels decrease. They decrease faster when a robot is locked against a wall. As soon as the altruism test becomes true the concerned robot plays the altruist role and changes its direction (this is the case around time 109, 155 and 235). If a robot is not locked in and can explore the corridor following its initial direction, its satisfaction level increases.

This animation shows an example of the execution of the specification for a specific environment (the corridor) and a specific number of agents (limited by computer capacity). These parameters can be easily modified in order to check the specification against pertinent test cases. It is important to note that the validation of the specification by simulation gives similar results as the real-world experiments (Lucidarme *et al*., 2002). In Figure 4 a hedograph, from the Greek *hedos*, which means satisfaction, shows the satisfaction levels of two real robots.

The simulation is performed by executing the behaviour part of the obtained specifications without developing a specific simulator. The simulation tool offers an interactive simulation mode and a program-controlled mode. In the latter a program written in a high-level language replaces the user. One feature of this programming language is the breakpoint construct. The breakpoint stops the specification execution when a condition is verified. Possible uses of breakpoints are, for example, configuration tests with predefined interaction scenarios and output of statistics.

Figure 3 Corridor environment (a), agents location (b) and satisfaction (c)

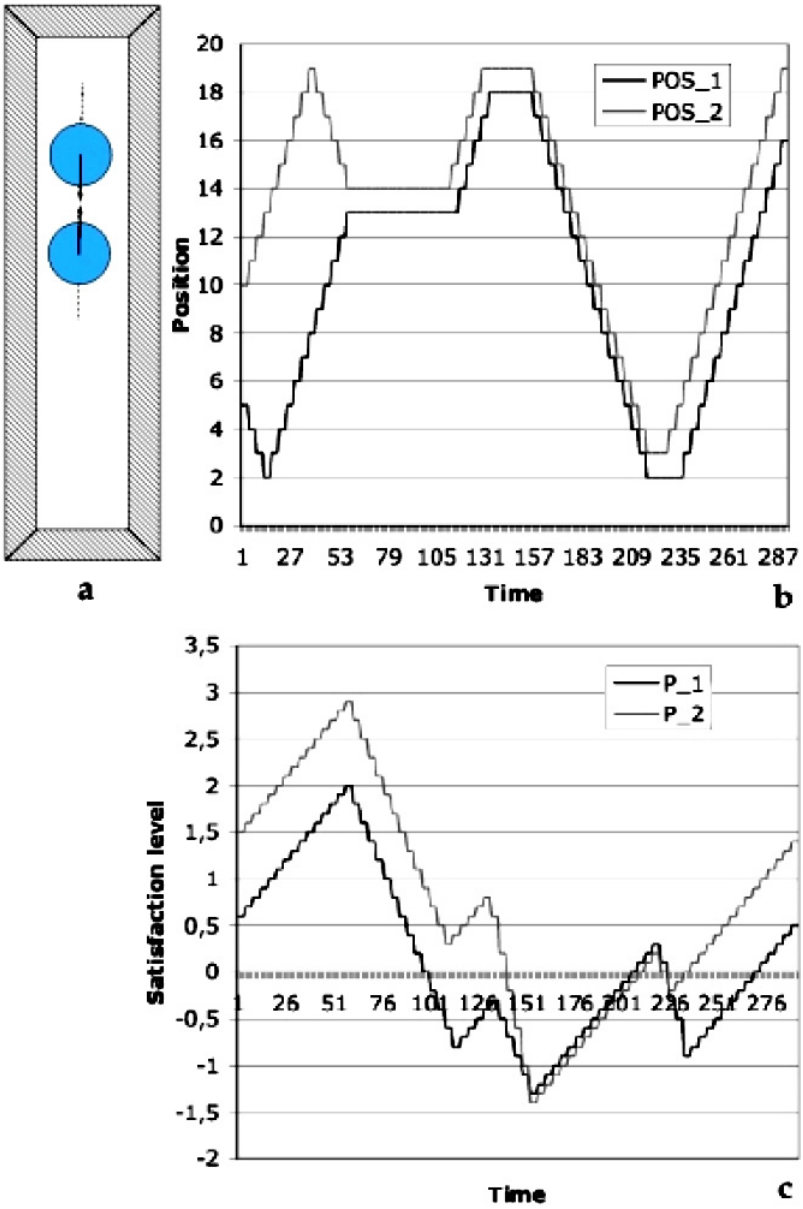
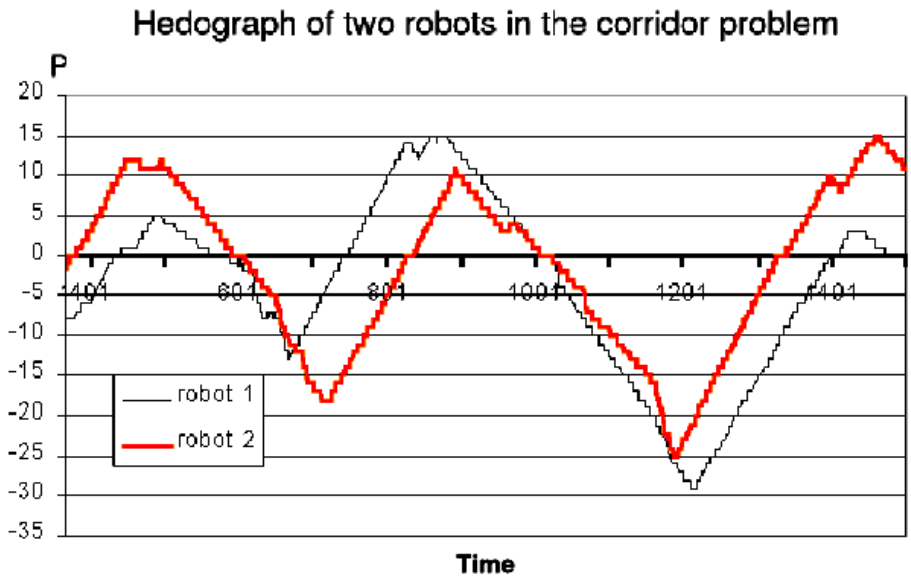


Figure 4 Hedographs of two real robots



3.4 Proofs of specification properties

In this subsection we establish fundamental properties of the satisfaction/altruism model in corridor environments. OZS semantics (Gruer *et al*., 2004) is based upon transition systems as defined in Manna and Pnueli (1995). It means that for each OZS specification there is an associated transition system. This transition system represents the set of possible computations the specification can produce. There are software environments, such as Manna *et al*. (1995) and de Moura *et al*. (2004), which take as input transition systems and allow the making of proofs of properties concerning the specified system. Of course there exist other environments for automatic theorem proving with different input formalisms such as Parent (1994), Nipkow *et al*. (2002) and Saaltink (1997). The semantics defined for our formalism has led us to use transition systems tools. Specifically we have chosen the SAL enviroment. With transition systems and a software environment such as SAL (de Moura *et al*., 2004), one can use several model checkers or theorem provers.

The model checkers are easy to use but suffer from combinatorial explosion. A solution to avoid this is to use theorem provers but their use requires great expertise. A third way proposed in Rushby (2006) is to use bounded model checking and induction to prove theorems. Given a system S specified by an initial predicate *Init* and a transition relation T, there exists a counterexample of length k to invariant P if there is a sequence of states $Init(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \neg P(s_k)$.

The induction is defined as:

Basis: $P(s_0) \Rightarrow I(s_0)$ and

Step: $P(r_1) \wedge T(r_1, r_2) \Rightarrow P(r_2)$.

This ordinary induction is extended to depth k as:

Basis: no counterexample of length k or less

Step: $P(r_1) \wedge T(r_1, r_2) \wedge P(r_2) \wedge \ldots \wedge P(r_{k-1}) \wedge T(r_{k-1}, r_k) \Rightarrow P(r_k)$.

The bounded model checker is used to establish the initial predicate. The induction theorem prover is used to prove the general induction property.

With this technique we have proven two lemmas, which have led us to a third interesting property. The first lemma may be interpreted as 'the most constrained robot is the most dissatisfied'. It is specified by the following formula:

$$\Diamond(\forall i, j \in [1..\#Robots], i \neq j \wedge left(r_i) = wall \wedge right(r_i) = robot \Rightarrow sat(r_i) < sat(r_j)).$$

It is eventually true that, for two different robots, if one is locked between a wall and another robot, its satisfaction will be lesser than the others. This lemma was proved by the technique presented above.

The second lemma may be interpreted as 'it's the less constrained robot which becomes altruist'.

$$\Box(\forall i, j \in [1..\#Robots], i \neq j \wedge s(i) = altruist \Rightarrow sat(i) > sat(j)).$$

For two different robots in a corridor, if one is in the altruism state then its satisfaction is lesser than the other's satisfaction. This lemma was also proved using the SAL environment. With these two lemmas taken as axioms, the SAL prover has established the following theorem: 'When a robot is in altruism state it unlocks the conflict.'

$$\Box(\forall i, j \in [1..\#Robots], i \neq j \wedge s(i) = altruist \Rightarrow direction(i) = direction(j)).$$

It means that a robot in the altruism state is repulsed and so follows the nonaltruist robot's direction.

4 Related works

This section describes some approaches for Agent-Oriented Software Engineering. The TROPOS methodology is based on a requirements analysis performed by means of goals, dependencies, roles, actors, positions and agents (Bresciani *et al*., 2004). They are graphically presented on the same schema. It is sometimes difficult to read such schemas, where all concepts are on the same level. Moreover, the tools associated with this methodology provide only static analysis of goals satisfaction.

Prometheus (Bergenti *et al*., 2004) is a methodology which defines a process associated with a set of deliverables. It covers phases from specification to architectural and detailed designs. The available supporting tool verifies features at the architectural and detailed design levels and it is mainly concerned with interfaces compatibility and plans consistency.

Kendall (2000) suggests the use of extended Object Oriented methodologies such as design patterns and CRC. CRC is extended to Role Responsibilities and Collaborators. In Kendall *et al*. (2000), a seven-layered architectural pattern for agents is presented. These concepts do not allow the prototyping of MASs.

The Andromeda methodologies (Drogoul and Zucker, 1998) use the role notion and propose a step-by-step methodology in order to design a MAS based upon reactive agents integrating machine-learning techniques. We have already specified the cognitive architecture (Gruer *et al*., 2002).

The MaSE methodology (DeLoach, 1999) insists upon the necessity of software tools for software engineering, specifically code generation tools. MaSE methodology suffers from limited one-to-one agent interactions. The authors authorise a dynamic role-playing relationship without suggesting how it can be done.

Bergenti and Poggi (2000) suggest the use of four UML-like diagrams. These diagrams are modified in order to take into account MAS-specific aspects. Among these MAS-specific aspects there are conceptual ontology description, MAS architecture, interaction protocols and agent functionalities.

In Odell *et al*. (2000), the authors present an approach extending UML for representing agent relative notions. In particular, the authors insist upon role concept and suggest the use of modified sequence diagrams to deal with roles. To the authors' knowledge no specific tool is proposed to deal with these diagrams.

The approach proposed by Herlea *et al*. (1999) is based upon the refinement of informal requirement specifications to semiformal and then formal specifications. The system structuring is based on a hierarchy of components (Brazier *et al*., 1997). These components are defined in terms of input/ouput and temporal constraints. A kind of testing technique based upon model checking is proposed in Herlea *et al*. (1999) but it is limited to some logical properties. Luck and d'Inverno (1995) propose a formal framework which uses the Z language. This framework is the starting point of any specification. It is composed of concepts to be refined in order to obtain a MAS specification. However, this approach has three main drawbacks. First, the specifications unit is the schema. Therefore, state spaces and operations of agents are separated. This drawback is avoided in our approach as we specify structure, properties and operations of an entity in the same Object-Z class. Second, Luck and d'Inverno's framework does not allow one to specify temporal and reactive properties of MASs (Fisher, 1997). In our framework these aspects are specified by temporal invariants and statecharts. Third, no operational semantics is given for Z specifications. So it is very difficult to use these formal specifications for prototyping or code generation.

The Gaia methodology (Wooldridge *et al*., 1999) and its extension (Zambonelli *et al*., 2003) deals with the analysis and design of MASs. It is composed of two abstraction levels: agent level and structural organisational level. No tool is proposed for this methodology.

5 Conclusion

In this paper we presented a formal-driven prototyping approach for MAS based upon a formal framework using organisational concepts such as role, interaction and organisation. The formal notation used, OZS, is based on the composition of Object-Z and statecharts. This multiformalism notation is given semantics in terms of transition systems. These semantics allow the animation of specifications and the verification of properties by using software environments such as STATEMATE (Harel *et al*., 1990) and SAL (de Moura *et al*., 2004). We have illustrated this approach by specifying an existing MAS architecture. The first step consists in defining a RIO model for this

architecture as a result of an organisational reverse engineering. The second step consists in validating the formal specification against existing experiments realised with real robots. In order to validate the specifications we have executed them. In fact, the specifications were animated in order to validate them against real scenarios. Once the specifications were validated so we were sure that they constituted a good model of the architectures, we were able to prove pertinent properties using formal proof. The properties we proved were linked to the requirements of the architecture. These properties are emerging properties that enable us to solve deadlocks during exploration.

This prototyping approach was also used to specify and analyse several MASs from a cognitive architecture (Gruer *et al*., 2002) to an immune-based architecture (Bakhouya *et al*., 2003) and holonic MAS (Rodriguez *et al*., 2007).

However, we are aware of our approach's limitations. In particular we plan to ease the specification process by associating it with a methodology. A CASE tool could be helpful to support the specification. Morever, we cannot say that our formalism is adapted to all MAS architectures. Up to now, for the problems we have dealt with we encountered no expressive limitations, but this is not proof.

Future works may take several directions. We are working on techniques to reduce combinatorial explosion of theorem proving and model checking, such as compositional verification techniques. The RIO organisational framework eases the use of such techniques as organisations and roles define components that can be verified separately. These techniques will facilitate the proving of complex properties.

We follow our previous works on the specification of MAS methodologies and architectures. We are currently formalising the RIO semiformal notation and trying to apply it to holonic MAS modelling. Several methodologies such as PASSI (Chella *et al*., 2004) are currently under specification. The specifications of these methodologies will contribute to give a formal and precise meaning to each one of them. We are also developing a software environment in order to help the specifier using our process.

References

Bakhouya, M., Rodriguez, S., Hilaire, V., Koukam, A. and Gaber, J. (2003) 'Intelligent immune-based system for autonomous soccer robots', *FIRA Robot World Congress Austria*.

Bergenti, F., Gleizes, M-P. and Zambonelli, F. (Eds.) (2004) *Methodologies and Software Engineering for Agent Systems*, Kluwer Academic Press.

Bergenti, F. and Poggi, A. (2000) 'Exploiting UML in the design of multi-agent systems', in A. Omicini, R. Tolksdorf and F. Zambonelli (Eds.) *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence, Springer Verlag.

Bernon, C., Cossentino, M. and Pavón, J. (2005) 'An overview of current trends in European AOSE research', *Informatica (Slovenia)*, Vol. 29, No. 4, pp.379–390.

Brazier, F.M.T., Dunin Keplicz, B., Jennings, N. and Treur, J. (1997) 'Desire: modelling multi-agent systems in a compositional formal framework', *International Journal of Cooperative Information Systems*, Vol. 6, pp.67–94.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. (2004) 'Tropos: an agent-oriented software development methodology', *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 3, pp.203–236.

Castelfranchi, C. (2000) 'Engineering social order', *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence, Springer Verlag.

Chapelle, J., Simonin, O. and Ferber, J. (2002) 'How situated agents can learn to cooperate by monitoring their neighbors' satisfaction', *15th European Conference on Artificial Intelligence*.

Chella, A., Cossentino, M., Sabatucci, L. and Seidita, V. (2004) 'From PASSI to agile PASSI: tailoring a design process to meet new needs', *IAT*, IEEE Computer Society, pp.471–474.

De Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M. and Tiwari, A. (2004) 'SAL 2', in R. Alur and D. Peled (Eds.) *Computer-Aided Verification, CAV 2004, Lecture Notes in Computer Science*, Boston, MA: Springer-Verlag, Vol. 3114, July, pp.496–500.

DeLoach, S. (1999) 'Multiagent systems engineering: a methodology and language for designing agent systems', *Agent Oriented Information Systems '99*.

Drogoul, A. and Zucker, J. (1998) 'Methodological issues for designing multi-agent systems with machine learning techniques: capitalizing experiences from the robocup challenge', Technical Report of LIP6 No. 041, ftp://ftp.lip6.fr/lip6/reports/1998/lip6.1998.041.ps.gz.

Duke, R., King, P., Rose, G. and Smith, G. (1991) 'The object-Z specification language', Technical report, Software Verification Research Center, Department of Computer Science, University of Queensland, Australia.

Ferber, J. and Gutknecht, O. (1998) 'A meta-model for the analysis and design of organizations in multi-agent systems', in Y. Demazeau, E. Durfee and N.R. Jennings (Eds.) *ICMAS'98*, July.

Fisher, M. (1997) 'If Z is the answer, what could the question possibly be?', *Intelligent Agents III*, Lecture Notes in Artificial Intelligence, No. 1193.

Gruer, P., Hilaire, V. and Koukam, A. (2001) 'Multi-agent approach to modeling and simulation of urban transportation systems', *IEEE Systems, Man, and Cybernetics Conference*.

Gruer, P., Hilaire, V. and Koukam, A. (2004) 'Heterogeneous formal specification based on object-z and state charts: semantics and verification', *Journal of Systems and Software*, Vol. 70, Nos. 1–2, pp.95–105.

Gruer, P., Hilaire, V., Koukam, A. and Cetnarowicz, K. (2002) 'A formal framework for multi-agent systems analysis and design', *Expert Systems with Applications*, Vol. 23, December.

Harel, D. (1987) 'Statecharts: a visual formalism for complex systems', *Science of Computer Programming*, June, Vol. 8, No. 3, pp.231–274.

Harel, D. and Gery, E. (1997) 'Executable object modeling with statecharts', *IEEE Computer*, Vol. 30, No. 7, pp.31–42.

Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A. and Trakhtenbrot, M.B. (1990) 'Statemate: a working environment for the development of complex reactive systems', *IEEE Transactions on Software Engineering*, April, Vol. 16, No. 4, pp.403–414.

Herlea, D.E., Jonker, C.M., Treur, J. and Wijngaards, N.J.E. (1999) 'Specification of behavioural requirements within compositional multi-agent system design', *Lecture Notes in Computer Science*, Vol. 1647, pp.8–27.

Hilaire, V., Koukam, A. and Gruer, P. (2002) 'A mechanism for dynamic role playing', *Agent Technologies, Infrastructures, Tools and Applications for E-Services*, Lecture Notes in Artificial Intelligence, Springer Verlag, No. 2592.

Hilaire, V., Koukam, A., Gruer, P. and Müller, J-P. (2001) 'Formal specification and prototyping of multi-agent systems', in A. Omicini, R. Tolksdorf and F. Zambonelli (Eds.) *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence, Springer Verlag, No. 1972.

Hilaire, V., Simonin, O., Koukam, A. and Ferber, J. (2005) 'A formal framework to design and reuse agent and multiagent models', in J. Odell, P. Giorgini and J. Muller (Eds.) *Agent Oriented Software Engineering*, LNCS 382, Springer, No. 3382.

Kendall, E.A. (2000) 'Role modeling for agent system analysis, design, and implementation', *IEEE Concurrency*, Vol. 8, No. 2, pp.34–41.

Kendall, E.A., Murali Krishna, P.V., Suresh, C.B. and Pathak, C.G.V. (2000) 'An application framework for intelligent and mobile agents', *ACM Computing Surveys*, Vol. 32, No. 1.

Lissajoux, T., Hilaire, V., Koukam, A. and Caminada, A. (1998) 'Genetic algorithms as prototyping tools for multi-agent systems: application to the antenna parameter setting problem', in S. Albayrak and F.J. Garijo (Eds.) *Lecture Notes in Artificial Intelligence, LNAI*, Springer Verlag, No. 1437.

Lucidarme, P., Simonin, O. and Liégeois, A. (2002) 'Implementation and evaluation of a satisfaction/altruism based architecture for multi-robot systems', *IEEE Int. Conf. on Robotics and Automation*.

Luck, M. and d'Inverno, M. (1995) 'A formal framework for agency and autonomy', in V. Lesser and L. Gasser (Eds.) *Proceedings of the First International Conference on Multi-Agent Systems*, AAAI Press, pp.254–260.

Manna, Z., Bjoerner, N., Browne, A. and Chang, E. (1995) 'STeP: the Stanford Temporal Prover', *Lecture Notes in Computer Science*, Vol. 915, pp.793–795.

Manna, Z. and Pnueli, A. (1995) *Temporal Verification of Reactive Systems: Safety*, Springer.

Nipkow, T., Paulson, L.C. and Wenzel, M. (2002) 'Isabelle/HOL – a proof assistant for higher-order logic, *LNCS*, Springer-Verlag, Vol. 2283.

Odell, J., Parunak, H. and Bauer, B. (2000) 'Extending UML for agents', in E.Y. Gerd Wagner and Y. Lesperance (Eds.) *Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, pp.3–17.

Paige, R.F. (1997) 'A meta-method for formal method integration', in J. Fitzgerald, C.B. Jones and P. Lucas (Eds.) *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, *Lecture Notes in Computer Science*, Springer-Verlag, ISBN 3-540-63533-5, Vol. 1313, September, pp.473–494.

Parent, C. (1994) 'Developing certified programs in the system Coq – the program tactic', in H. Barendregt and T. Nipkow (Eds.) *Proceedings of Types'93*, Galement rapport LIP-ENS Lyon RR 93-29 and by anonymous ftp from lip.ens-lyon.fr file pub/Rapports/RR/RR93/RR93-29.ps.Z.

Rodriguez, S., Hilaire, V., Gruer, P. and Koukam, A. (2007) 'A formal holonic framework with proved self-organizing capabilities', *International Journal of Cooperative Information Systems*, March, Vol. 16, No. 1, pp.7–25.

Rushby, J. (2006) *Tutorial on Mechanized Formal Methods*.

Saaltink, M. (1997) 'The Z/EVES system', *ZUM '97: Z Formal Specification Notation. 11th International Conference of Z Users. Proceedings*, Berlin, Germany: Springer-Verlag, 3–4 April, pp.72–85.

Selic, B. (1996) 'Real-time Object-Oriented Modeling (ROOM)', *IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, Washington, Brussels, Tokyo: IEEE Computer Society Press, June, pp.214–219.

Simonin, O. (2001) 'Le modele satisfaction-altruisme: cooperation et resolution de conflits entre agents situes reactifs, application a la robotique', PhD thesis, USTL.

Simonin, O. and Ferber, J. (2000) 'Modeling self satisfaction and altruism to handle action selection and reactive cooperation', *The Sixth International Conference on the Simulation of Adaptative Behavior FROM ANIMALS TO ANIMATS 6*, pp.314–323.

Simonin, O., Liégeois, A. and Rongier, P. (2000) 'An architecture for reactive cooperation of mobile distributed robots', *5th International Symposium on Distributed Autonomous Robotic Systems*, Knoxville, Tennessee.

Smith, G. (1997) 'A semantic integration of object-Z and CSP for the specification of concurrent systems', in J. Fitzgerald, C.B. Jones and P. Lucas (Eds.) *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, Springer-Verlag, Vol. 1313, pp.62–81.

Wooldridge, M., Jennings, N.R. and Kinny, D. (1999) 'A methodology for agent-oriented analysis and design', *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, Seattle, WA: ACM Press, pp.69–76.

Zambonelli, F., Jennings, N. and Wooldridge, M. (2003) 'Developing multiagent systems: the gaia methodology', *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No. 3.

Zave, P. and Jackson, M. (1993) 'Conjunction as composition', *ACM Transactions on Software Engineering and Methodology*, October, Vol. 2, No. 4, pp.379–411.