A Formal Approach to Design and Reuse Agent and Multiagent Models

Vincent Hilaire[1], Olivier Simonin[1], Abder Koukam[1], and Jacques Ferber[2]

[1] Université de Technologie de Belfort Montbéliard., 90010 Belfort Cedex, France
vincent.hilaire@utbm.fr
(33) 384 583 009
[2] LIRMM Université Montpellier II - CNRS, 161 rue Ada
34392 Montpellier Cedex 5 - France

Abstract. While there are many useful models of agents and multi-agent systems, they are typically defined in an informal way and applied in an ad-hoc fashion. Consequently, multi-agent system designers have been unable to fully exploit these models commonalities and specialise or reuse them for specific problems. In order to fully exploit these models and facilitate their reuse we propose a formal approach based upon organisational concepts. The formal notation is the result of the composition of Object-Z and statecharts. The semantics of this multi-formalisms is defined by transition systems. This operational semantics enables validation and verification of specifications. We present this approach through the specification of the satisfaction-altruism model which has been used to design situated multi-agent systems. We put the emphasis on the specification of a mobile robot architecture based on the refinement of this model. The availability of such generic models is a fundamental basis for reuse. We also show how to analyse the specification by validation and verification.

1 Introduction

While there are many useful models of agents and multi-agent systems, they are typically defined in an informal way and applied in an ad-hoc fashion. Consequently, multi-agent system designers have been unable to fully exploit these models commonalities and specialise or reuse them for specific problems. We believe, and the experience bear this out, that formal specification can be used to describe model concepts which can be refined to fulfil a particular system needs.

A whole range of methodological efforts relating to MAS have been undertaken, and can be divided into those that are based upon semi-formal models [2, 4, 22] on the one hand, and those that are based on formal models [17, 11, 24]. A drawback of semi-formal methods is that they do not allow validation or verification of MAS. Among formal models many impose specific agent architecture like the BDI one and are not well fitted to enable reuse of concepts. The aim of this paper is to present a formal approach for the specification of MAS models and their reuse. This formal approach allows validation and verification and is presented through a case study. First a formal specification of a multi-agent model is given then it is refined for a specific application.

The specification approach is based on a formal notation OZS [7] and an organisational framework [12]. The organisational framework gives methodological rules in order to specify a system in terms of an organisational meta-model named RIO which stands for Role, Interaction and Organisation [13]. This point of view is the basis of several other specification approaches such as [24, 6, 23]. The OZS notation is the result of the composition of Object-Z [5] and statecharts [9]. This multi-formalisms notation allows the modelling of systems with both reactive and functional aspects. Indeed, the basic construct of this notation is an Object-Z class which encapsulates a statechart. The Object-Z class specify attributes and operations in a set-theoretic fashion and the statechart specifies how the class react to external and internal events. This notation allows the prototyping and the verification of produced specifications. The prototyping and verification processes enable the production of correct specifications which can be refined downto executable code as shown in the figure 1. Each concept of the RIO meta-model is specified by an OZS class which may encapsulates a statechart in order to specify behaviours.

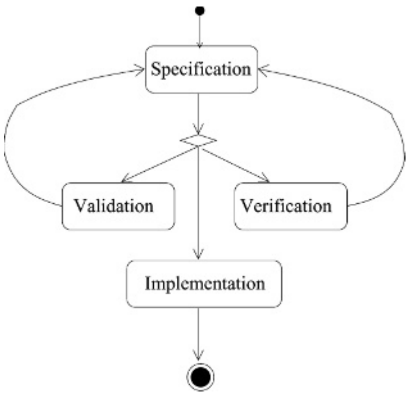


Fig. 1. Specification refinement process

We present this approach through the formal specification of a multi-agent architecture: the satisfaction-altruism model [20, 21]. This model is based on a behaviour-based architecture as [1] and introduces a cooperation mechanism. This one is defined as reactions to the perception of simple signals broadcasted by agents. We illustrate the refinement of the resulting formal model in order to specify a mobile robot application. However, the specification approach isn't limited to behaviour-based architecture. Indeed, in [8] we have specified a mentalistic based architecture.

The paper is organised as follows : section 2 introduces the OZS notation. Section 3 abstracts the satisfaction-altruism model. Section 4 presents the formal specification of the satisfaction-altruism model with the OZS notation and section 5 the mobile robots extension. Section 6 illustrates prototyping and verification on this case study. Eventually, section 7 concludes.

2 OZS Notation

Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to model all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend one formalism.

Our choice is to use Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects. Object-Z extends Z [17] with object-oriented specification support. The basic construct is the class which encapsulates state schema and operation schemas which may affect its variables.

Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both language have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.

We introduce a multi-formalisms notation that consists in integrating statecharts in Object-Z classes. The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes the possible states of the object and events which may change these states. A 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 $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes.

The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between $DOOR1$ and $DOOR2$. The first door reacts to $activate1$ and $deactivate1$ events. When someone enter the *LoadLock* he first activate the first door enter the *LoadLock* and deactivate the first door. The transition triggered by $deactivate1$ event execute the *inLL* operation which sets the *someoneInLL* boolean to true. Someone which is between the first and the second door can activate the second door so as to open it.

LoadLock

$someoneInLL : \mathbb{B}$

INIT

$\neg\ someoneInLL$

inLL

$\Delta someoneInLL$

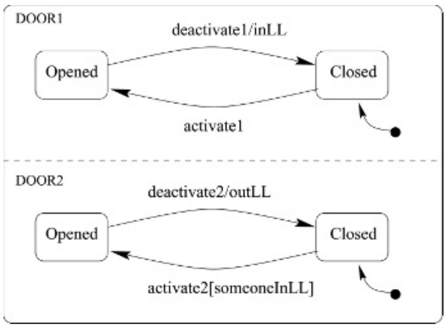
$someoneInLL'$

outLL

$\Delta someoneInLL$

$\neg\ someoneInLL'$

behavior



The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.

The result of the composition of Object-Z and statecharts seems particularly suited in order to specify MAS. Indeed, each formalism has constructs which enable complex structure specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with.

3 Overview of Satisfaction-Altruism Model

The satisfaction-altruism model has been developed to integrate intentional cooperative behaviours into the collective solving problem approach. Assume the system is composed of simple self-organised entities (reactive agents) working for a common goal. Intentional cooperation is integrated thanks to three concepts.

The first concept is personal satisfaction. It is a value computed continuously by each agent (noted P), representing the evolution of the current task. At each step of the decision cycle, the agent computes a variation v of the value P, which can be a positive value if it progresses or a negative value if it regresses or stagnates (see details in [20]). The cycle action-perception-update of P is shown in figure 2 for one agent.

The second concept is dynamical influences. They are simple attractive or repulsive signals locally emitted by the agents towards their neighbours (represented by Signal arrows in figure 2). An influence is released when an agent perceives conflicts or possibilities to cooperate with its neighbours (represented by the constraints evaluation box in figure 2). The influences are coded by values defined in the same interval as P, with positive values for attraction and negative ones for repulsion.

The last concept is altruistic behavior. Each agent computes in its action-selection module a comparative test between the values of the perceived influences and its personal satisfaction (P). If the intensity of an external influence is

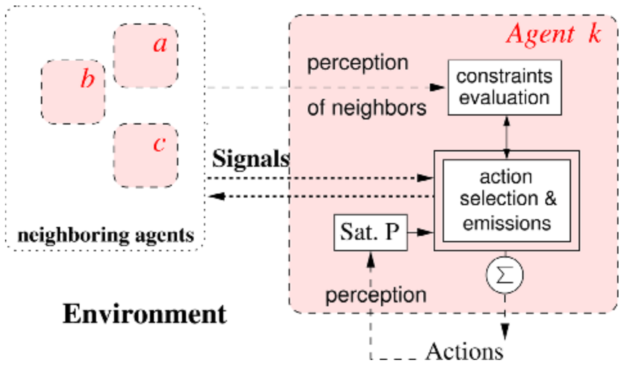


Fig. 2. Satisfaction Altruism principle

greater than P then the agent stops its current task and executes a predetermined altruistic behavior. It is a response to the strongest influence: a displacement to go towards the emitter or to move away. Note that agents executing an altruistic behavior can propagate the signal they perceive. It is useful to solve conflicts involving several close agents [21].

This model constitutes an extension of the Artificial Potential Field (APF) approach in the sense that agents emit dynamical signals which are perceived by others as environmental influences. These dynamical influences can be then combined to other classical perceptions, for instance the repulsions from the close obstacles. This combination is represented by the sum operation in figure 2.

Different distributed problems have been efficiently solved by this model. On the one hand with simulation tools: foraging tasks [20], navigation conflicts solving [21], cooperation for box-pushing and box-cutting in heterogeneous multi-robot systems [3]. On the other hand with real autonomous mobile robots: navigation conflicts solving [16].

4 Satisfaction-Altruism Kernel Specification

The satisfaction-altruism model is now detailed through its formal specification, using the previously introduced approach. The analysis of the satisfaction-altruism model using RIO is out of the scope of this paper. We only present the classes we obtained.

The class *DiscreteSensor* specifies a discrete sensor device. The first line of the class introduces the *Intensity* abbreviation by the == symbol. *Intensity* denotes $[-P_{max}, P_{max}]$ interval where P_{max} is a real constant. The next unnamed sub-schema specifies the state space of the class. Such a device takes as input a discrete number of signals which are represented by the *stimulus* sequence. Each different sensor, indexed by a number i in $\mathbb{N}$, gives a different signal, $stimulus(i)$. One can select the greater signal received by the *getMax* operation. An operation is enclosed in named sub-schema and is divided in two parts. The first part, above a short line is a declaration part and the second part below the short line is a predicate part. To include output (resp input) parameters the name of the variable should end with an exclamation (resp question) mark. In the *getMax* operation there is an output variable s. The predicate part specifies the s domain. It belongs to the *stimulus* sequence range. It also states that s is the greater stimulus perceived in terms of greater absolute value of signal intensity.

DiscreteSensor

$Intensity == [-P_{max}, P_{max}]$

$stimulus : \mathrm{seq}\ Signal$

getMax

$s! : Intensity$

$s \in ran\ stimulus$

$\nexists s' \in ran\ stimulus \bullet |s'| \geq |s|$

With the *SAAgent* class we introduce an agent based upon the satisfaction-altruism kernel. Its attributes, operations and behaviour are defined according to the satisfaction-altruism model. The class *SAAgent* inherits from Agent defined in the RIO framework [12] and specifies the Satisfaction/Altruism behavior model. The *altruismTest* boolean decides if the agent must be in individual state or altruism state. It is defined as a λ-expression which evaluates the condition included in the predicate part.

The agent attributes are first an action it currently undergo : *current*. Then the agent is described by weights attached to the actions it can do *weight*. These different weights can be modified by the agents evolution and their initial values are defined by *initialWeight*.

The *progressionReward* function maps each action to a 3-uplet giving the bonus or penaltypenalty values for agents satisfaction when it is respectively in progression, in regression or locked.

The statechart included in the behavior sub-schema specifies the behavior of *SAAgent*. It consists of two exclusive-or states. These states specify the individual and altruistic behaviours. By default an agent is in individual state and if the altruism test becomes true then it is in altruistic state. Each superstate is divided in two parts. First an action part which is the reaction to events and second a communication part which emits a I valued signals.

The operations of the class are P which computes the personal satisfaction of the agent according to the Satisfaction Altruism model and I_{ext} which selects the perceived signal with the maximum intensity. Eventually, *actionSelection* decides for the individual state which action is to be achieved. This operation modifies the state space of the class. Specifically it modifies the *current* and *weight* variables. These variables are listed in the Δ-list of the operation. The mechanism of action selection isn't described further here since we restrict actions to movements in the sequel.

SAAgent

Agent

$altruismTest : \mathbb{B}$

$altruismTest \Leftrightarrow |I_{ext}()| \geq P() \wedge |I_{ext}()| \geq |I|$

$BMValue == [-\Delta s, \Delta s] \times [-\Delta s, \Delta s] \times [-\Delta s, \Delta s]$

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

$current \subseteq actions$
$dom\ initialWeight = dom\ weight \subseteq actions$
$\forall\, v \in ran\ progressionReward \bullet v =$
$(n, m, f) \wedge \Delta s \leq f \leq n \leq 0 \leq m \leq \Delta s$

P

$satisfaction! : [-P_{max}, P_{max}]$

$progression(current) \Rightarrow satisfaction' = satisfaction + v.m$
$regression(current) \Rightarrow satisfaction' = satisfaction + v.n$
$locked(current) \Rightarrow satisfaction' = satisfaction + v.f$

I_{ext}

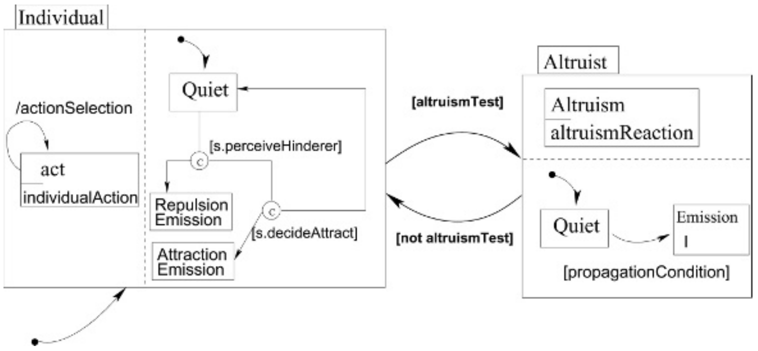
$ext! : \mathbb{R}$

$ext! = s.getMax()$

actionSelection

$\Delta(current, weight)$

behavior



5 Mobile Robots Extension

5.1 Principle of Robots Behavior

In the previous section we have presented the kernel of the satisfaction-altruism model. To apply it on concrete problems it is necessary to refine the agent perceptions and to clarify the satisfaction computation. Here we aim to define a mobile robot architecture using the model to allow agents cooperation for navigation. In order to treat navigational conflicts the satisfaction-altruism kernel is refined by adding the key concepts of : displacement, perception and satisfaction.

When an agent tries to move towards a direction its displacement is defined by a vector (V). This vector represents the current task of the agent. When the agent gets the altruist state this vector becomes equal to the predetermined reaction to signals. In any case the vector V integrates obstacles avoidance.

An agent blocked by the fault of others emit a repulsive signal when its satisfaction becomes negative (the value emitted is then equals to P). When an agent perceives such a signal and the intensity of the signal is higher than its personal satisfaction the agent tries to move away from the transmitting agent. It is the altruistic reaction.

Each agent is equipped with sensors allowing the detection of walls and robots, towards different directions (into [0°,360°]). When the agent can progress towards its goal the variation of satisfaction v is positive (equal to $m > 0$). When it moves away v is a negative value (equals to $n \leq 0$). If the agent is locked (paralysed) the variation of P is equal to f, with $f < n \leq 0$. The value f is computed in function of the elements surrounding the agent. Each element has a negative weight: value θ for robots, θ' for walls as $\theta' < \theta < 0$ because walls are stronger constraints than mobile robots. v is then computed as the sum of these weights over all the directions. In other words, the more an agent is surrounded by walls the faster its satisfaction will decrease. As a consequence, there is an induced emergent phenomenon that moves agents from the more constrained regions to the less constrained ones, cf [21].

5.2 Classes for the Mobile Robot Extension

The *DiscreteSituatedSensor* class inherits from *DiscreteSensor*. It specifies a sensor which devices are located at given angles. The signal type introduced in the beginning of the class is defined as a free type by the ::= symbol and we enumerate the possible value for this type. The different possible values are: ∅ (which means no signal), obstacle or agent (another robot). For each specific sensor one can know the signal type with *getSignalType* operation and the number of sensors activated with the *getActivatedSensor* operation.

$$
\begin{array}{|l}
\hline
\textit{DiscreteSituatedSensor} \\
\textit{DiscreteSensor} \\
\textit{SignalType} ::= \varnothing \mid \textit{obstacle} \mid \textit{agent} \\
\quad \begin{array}{|l}
\hline
\textit{angularDistance} : [0, 360] \\
\textit{numberOfSensors} : \mathbb{N} \\
\hline
\textit{dom stimulus} \subseteq [1, \textit{numberOfSensors}] \\
\textit{angularDistance} = 360 / \textit{numberOfSensors} \\
\hline
\end{array} \\
\quad \begin{array}{|l}
\hline
\textit{getSignalType} \\
n? : \mathbb{N} \\
t! : \textit{SignalType} \\
\hline
\end{array} \\
\quad \begin{array}{|l}
\hline
\textit{getActivatedSensor} \\
n! : \mathbb{N} \\
\hline
n! \leq \textit{numberOfSensors} \\
\hline
\end{array} \\
\hline
\end{array}
$$

Eventually the *MobileRobot* class specifies a robot which behavior is based on the satisfaction altruism model. This class inherits from SAAgent. In this class we precise many things. The propagation condition of the agent is true whenever the external signal (I_{ext}) is less than zero (i.e. repulsive signal) and when there is a robot in its way. One agent perceives a hinderer when there is one agent just in front of him. An agent progresses if and only if its move is non zero. An agent is locked whenever its moving vector is zero and it can't be in regression. In order to simulate random we use a non described operation *random* which outputs a random number in the set $\{-30, -15, 0, 15, 30\}$.

$$
\begin{array}{|l}
\hline
\textit{MobileRobot} \\
\textit{SAAgent} \\
\quad \begin{array}{|l}
\textit{propagationCondition} : \mathbb{B} \\
\hline
\textit{propagationCondition} \Leftrightarrow (I_{ext} \leq 0) \\
\wedge\ (s.\textit{getSignalType}(s.\textit{getActivatedSensor}() + 180) = \textit{agent})
\end{array} \\
\quad \begin{array}{|l}
\textit{perceiveHinderer}! : \mathbb{B} \\
\hline
\textit{perceiveHinderer}! \Leftrightarrow s.\textit{getSignalType}(0) = \textit{agent}
\end{array} \\
\quad \begin{array}{|l}
\textit{progression} : \mathbb{B} \\
\hline
\textit{progression} \Leftrightarrow \overrightarrow{V}!_{goal} \neq \overrightarrow{0}
\end{array} \\
\quad \begin{array}{|l}
\textit{locked} : \mathbb{B} \\
\hline
\textit{locked} \Leftrightarrow \overrightarrow{V}!_{goal} = \overrightarrow{0}
\end{array}
\qquad
\begin{array}{|l}
\textit{regression} : \mathbb{B} \\
\hline
\neg\ \textit{regression}
\end{array}
\end{array}
$$

$$calculateV_f! : \mathbb{R}$$

$$calculateV_f! = \Sigma_{n=0}^{n=s.numberOfSensors} valObs(n)$$

$$valObs! : \mathbb{R}$$
$$n? : \mathbb{N}$$

$$s.getSignalType(n?) = obstacle \Rightarrow valObs(n)! = -0.75$$
$$s.getSignalType(n?) = agent \Rightarrow valObs(n)! = -0.25$$
$$s.getSignalType(n?) = \varnothing \Rightarrow valObs(n)! = 0.5$$

$$\operatorname{dom} progressionReward = \{altruismReaction, individualReaction\}$$
$$\operatorname{ran} progressionReward = (2, 0, calculateV_f)$$
$$s \in DiscreteSituatedSensor$$

calculateSlide

$$\overrightarrow{V_{sli}} = \sum_{i \in \{s.getSignalType(i) = obstacle\}} ((i \times angularDistance) + 90$$

altruismReaction

$$\overrightarrow{V}!_{goal} : Vector$$

$$\overrightarrow{V}_{slide} = calculateSlide()$$
$$\overrightarrow{V_{goal}} = \overrightarrow{V_{altruism}} + \overrightarrow{V_{slide}}$$
$$\overrightarrow{V_{altruism}} = k \times s.getMax().(s.getActivatedSensor() \times s.angularDistance)$$

individualReaction

$$\overrightarrow{V}! : Vector$$

$$s.getSignalType(0) = obstacle \Rightarrow \overrightarrow{V_{goal}} = 0$$
$$s.getSignalType(0) \neq obstacle \Rightarrow \overrightarrow{V_{goal}} = random$$
$$\overrightarrow{V_{slide}} = calculateSlide()$$
$$\overrightarrow{V}! = \overrightarrow{V_{goal}} + \overrightarrow{V_{slide}}$$

getDirection

$$\overrightarrow{V}! : Vector$$

$$instate(Indivual) \Rightarrow \overrightarrow{V}! = individualReaction()$$
$$instate(Altruism) \Rightarrow \overrightarrow{V}! = altruismReaction()$$

The mobile robots environment is specified by the coordinate of the mobile robots and the position of obstacles. It refines the *Environment* class of the RIO framework.

$$
\begin{array}{|l}
\hline
\textit{RobotsEnvironment} \\
\textit{Environment} \\
\textit{Coordinate} == \mathbb{R} \times \mathbb{R} \\
\begin{array}{|l}
\hline
\textit{situation} : \textit{MobileRobot} \rightarrow \textit{Coordinate} \\
\textit{obstacles} : \mathbb{P}\, \textit{Coordinate} \\
\hline
\forall\, a \in \textit{agents} \bullet \textit{situation}(a) \notin \textit{obstacles} \\
\forall\, a, b \in \textit{agents} \bullet a \neq b \Rightarrow \textit{situation}(a) \neq \textit{situation}(b) \\
\hline
\end{array} \\
\hline
\Box(\forall\, a \in \textit{agents} \bullet \\
\quad (x, y) = \textit{situation}(a) \\
\quad \wedge \bigcirc(\textit{situation}(a) = (x, y) + a.\textit{getDirection}())) \\
\hline
\end{array}
$$

6 Specification Analysis

6.1 Prototyping

The prototyping is performed by using STATEMATE [10] ; an environment which allows the prototyping and the simulation of the statechart specifications. 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 actions 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 agent in order to simulate exploration of various environments.

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

In order to evaluate our specification of the architecture we simulated the behavior of two robots evolving in a particular environment. We defined a closed narrow corridor where it is impossible for two agents to inter-cross, as showed in figure 3a. 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 back the other, this causes a quick fall of their satisfactions. The more unsatisfied agent repulses the other to an extremity 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 take it.

The figures 3b and 3c shows an example of such a test. The x axis represents time and y axis represents discretized positions in the corridor for the 3b figure

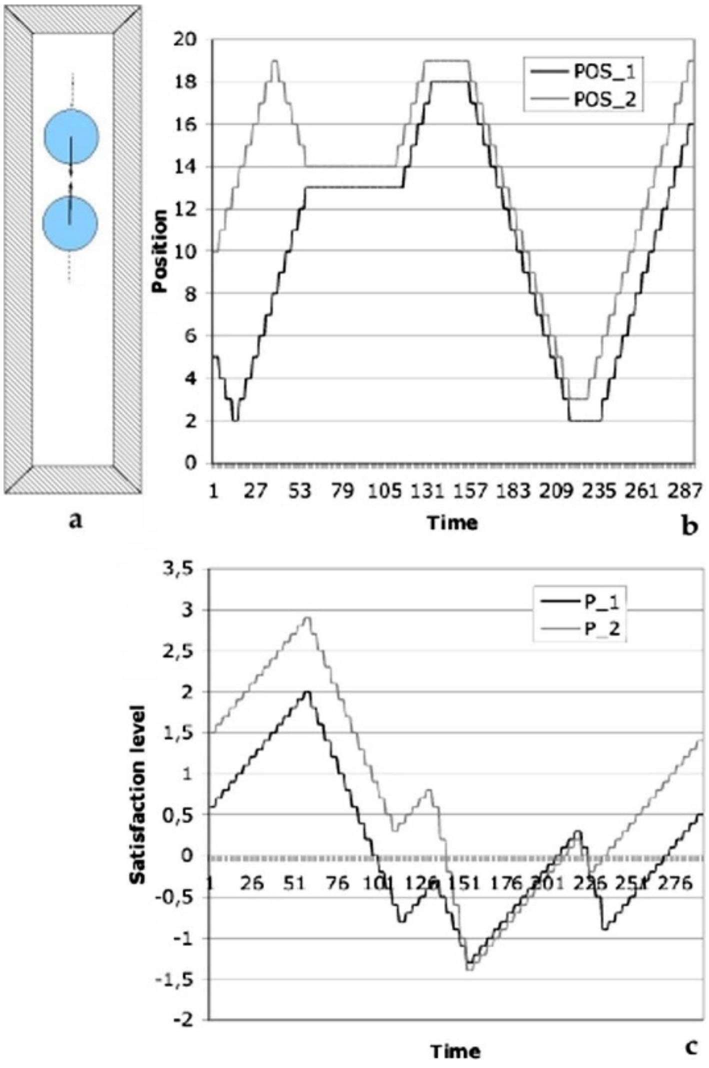


Fig. 3. Corridor environment and curves

and level of satisfaction for each robot for the 3c figure. One can see that levels of satisfaction and trajectories are correlated. Indeed, each time the two robots are locked 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 (it is the case around times 109, 155 and 235). If a robot isn't locked 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. These

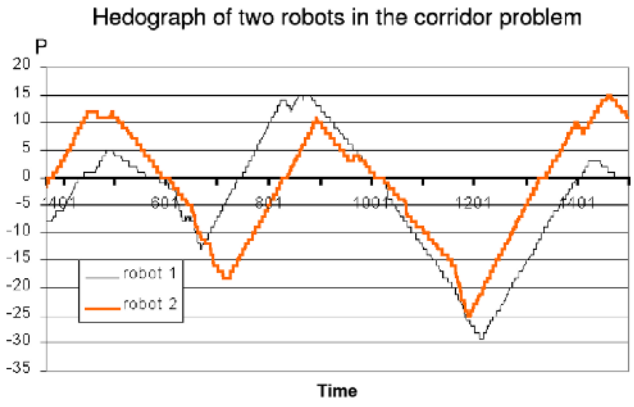


Fig. 4. Hedographs of 2 real robots

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 [16], see figure 4. This figure, called 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. Breakpoint stop 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.

6.2 Verification

OZS semantics [7] is based upon transition systems as defined in [19]. 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.

With such transition systems and software tools like STeP [18] one can verify specification properties. With STeP specification properties are expressed in Linear-time Temporal Logic formulas and the verification may be done by using two techniques.

The first technique is model-checking which enables the verification of the satisfiability of a property. The STeP model-checker proves or refutes validity of LTL formulas relatively to a transition system. To establish the satisfiability of history invariant H one must actually establish that $\neg H$ is not valid. This technique is the simplest to use but is limited by the specification state space.

The second technique is semi-automatic proof. It is based on deductive method. The deductive methods of STEP verify temporal properties of systems by means of verification rules, e.g. induction and verification diagrams. Verification rules are used to reduce temporal properties of systems to first-order verification conditions. Verification diagrams provide a visual language for guiding, organising and displaying proofs. Verification diagrams allows the user to construct proofs hierarchically, starting from a high-level, intuitive proof sketch and proceeding incrementally, as necessary, through layers of greater detail. The specifier can help the proover if it can not proove a property by introducing axioms.

For our case study one has to refine the *RobotsEnvironment* class to specify the corridor example and try to verify with the STeP model-checker the satisfiability of the following informal property "When a robot is locked by another robot it's eventually be freed". This property is specified by the following LTL formula :

$$\begin{aligned}
&(\exists\, a \in agents \bullet \\
&\qquad \forall\, i \in a.s.numberOfSensors \bullet \\
&\qquad a.s.getSignalType(i) = agent \\
&\qquad \vee\; a.s.getSignalType(i) = obstacle) \Rightarrow \\
&\Diamond(\exists\, j \in a.s.numberOfSensors \bullet \\
&\qquad a.s.getSignalType(i) = \varnothing)
\end{aligned}$$

It means that if there exists one agent that perceives an obstacle or other agents from all its sensors (the agent is locked), eventually this agent will be freed ans thus at least one perception will be void. To be verified by a model checker the corresponding transition system should include only finite and bounded sets. All part of the specification that describe continuous or infinite types must be discretized and bounded.

7 Conclusion

In this paper we have presented an approach which allows specification, validation and verification of MAS. Moreover, we have specified a particular MAS model, the satisfaction-altruism one. This reactive-based model is useful for cooperation between situated agents (or robots). We have shown validation and verification examples for the satisfaction-altruism specification. This formalisation shows that our approach can be applied to behavior-based architectures, that were not formally analysed at the beginning. We have thus applied a reverse-engineering approach to specify formally this model. The result is composed of OZS [7] classes which specify each component of the satisfaction-altruism model. The first level of the specification specifies the kernel of the satisfaction-altruism model. The second level refines the first and specifies an extension applied to mobile robots. The advantages of our approach are first that the satisfaction-altruism model is presented in formal and non ambiguous terms and second that the specification decomposes the model in formal concepts which can be reused in specific applications. The validation phase enables specifications test with predefined interaction scenarios such as the corridor application. The verification phase allows the proof of logical properties. The reusability of such a specification is enhanced by the results of the validation and the verification. Indeed, all the roles and agents, validated and verified, constitute reliable components which can be used in other applications.

At this time, our approach can be improved. In particular we plan to ease the specification process by associating a methodology. A methodology must be associated to ease the specification process. A CASE tool could be helpful to support the specification. We plan to explore the verification of properties for such specifications by using semi-automatic proofs. We also plan the specification of others multi-agent models following the process described in this paper. This will constitute a library of reusable generic agent and multi-agent models a sort of design patterns for agents.

References

1. R. Arkin. *Behavior Based Robotics.* The MIT Press. 1998
2. Bergenti, F. and A. Poggi: 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.*
3. J. Chapelle, O. Simonin, and J. Ferber. How situated agents can learn to cooperate by monitoring their neighbors' satisfaction. In *15th European Conference on Artificial Intelligence*, pages 68–72, Lyon, France, 2002.
4. DeLoach, S.: 1999, 'Multiagent Systems Engineering: a Methodology and Language for Designing Agent Systems'. In: *Agent Oriented Information Systems '99.*
5. R. Duke, P. King, G. Rose, and G. Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Departement of Computer Science, University of Queensland, AUSTRALIA, 1991.
6. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Y. Demazeau, E. Durfee, and N. Jennings, editors, *ICMAS'98*, july 1998.
7. P. Gruer, V. Hilaire, and A. Koukam. Heterogeneous formal specification based on object-z and state charts: semantics and verification. *Journal of Systems and Software*, 70(1), 2004.
8. P. Gruer, V. Hilaire, A. Koukam, and K. Cetnarowicz. A formal framework for multi-agent systems analysis and design. *Expert Systems with Applications*, 23. 2002.
9. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
10. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. B. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, Apr. 1990.
11. Herlea, D. E., C. M. Jonker, J. Treur, and N. J. E. Wijngaards: 1999, 'Specification of Behavioural Requirements within Compositional Multi-agent System Design'. *Lecture Notes in Computer Science* **1647**, 8–27.

12. V. Hilaire, A. Koukam, and P. Gruer. A mechanism for dynamic role playing. In *Agent Technologies, Infrastructures, Tools and Applications for E-Services*, number 2592 in Lecture Notes in Artificial Intelligence. Springer Verlag, 2002.
13. V. Hilaire, A. Koukam, P. Gruer, and J.-P. Müller. Formal specification and prototyping of multi-agent systems. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents' World*, number 1972 in Lecture Notes in Artificial Intelligence. Springer Verlag, 2000.
14. V. Hilaire, A. Koukam, O. Simonin, and P. Gruer. Formal specification of role dynamics in agent organizations: Applications to the satisfaction-altruism model. *Autonomous Agents and Multi-Agent Systems*, 2003. submitted.
15. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press, 1995.
16. P. Lucidarme, O. Simonin, and A. Ligeois. Implementation and evaluation of a satisfaction/altruism based architecture for multi-robot systems. In *International Conference of Robotics and Automation (ICRA'2002)*, pages 1007–1012, Washington, USA, 2002.
17. Luck, M. and M. d'Inverno: 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*. pp. 254–260.
18. Z. Manna, N. Bjoerner, A. Browne, and E. Chang. STeP: The Stanford Temporal Prover. *Lecture Notes in Computer Science*, 915:793–??, 1995.
19. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.
20. O. Simonin and J. Ferber. Modeling self satisfaction and altruism to handle action selection and reactive cooperation. *6th International Conference On the Simulation Of Adaptive Behavior (SAB 2000 volume 2)*, pages 314–323, 2000.
21. O. Simonin, A. Liegois, and P. Rongier. An architecture for reactive cooperation of mobile distributed robots. In *DARS 4 Distributed Autonomous Robotic Systems 4*, pages 35–44, Knoxville, TN, 2000. Springer.
22. J. Odell, M. Nodine, and R. Levy. A metamodel for agents, roles and groups. In J. Odell, P. Giorgini, and J. P. Müller, editors, *The Fifth International Workshop on Agent-Oriented Software Systems*, volume in this book. Springer-Verlag, 2004.
23. E. Steegmans, D. Weyns, T. Holvoet, and Y. Berbers. Designing roles for situated agents. In J. Odell, P. Giorgini, and J. P. Müller, editors, *The Fifth International Workshop on Agent-Oriented Software Systems*, volume in this book. Springer-Verlag, 2004.
24. M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.