Formal Driven Prototyping Approach for Multi-Agent Systems

Vincent Hilaire and
Pablo Gruer and
Abder Koukam

ScT laboratory,
UTBM,
90010 Belfort cedex FRANCE
E-mail: vincent.hilaire@utbm.fr∗Corresponding author

Olivier Simonin

Maia team,
LORIA
BP 239 - 54506 Vandœuvre-ls-Nancy Cedex FRANCE

Abstract: Even if Multi-Agent Systems 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. Many methodologies exist for engineering MAS. A common point of many of them is to produce different work products or models during the different phases. Each work product or model help to progress towards an implementation of the system under development. Nevertheless, there is very few ways to control 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. 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 [28]. 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 putting emphasis on prototyping and simulation of complex systems before their effective implementation is proven to be a valuable approach. This process is based upon a formal organisational framework. This framework describes the key organisational concepts, such as Role, Interaction and Organisation, with a formal notation, namely OZS, which is the result of the composition of Object-Z and statecharts. The process is illustrated through the specification of a multi-agent architecture, we have used in several applications.

Keywords: Multi-Agent Systems, Formal specification, Validation, Verification
1 Introduction

Even if Multi-Agent Systems (MAS in the sequel) 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 massive number of autonomous components interact it is very difficult to predict that the emergent organisational structure fits the system goals or that the desired functionalities will be fulfilled.

Many methodologies exists for engineering MAS, see [4, 2] 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 to produce different work products or models during the different phases. For example the PASSI methodology [9] uses UML and AUML [3] models. Each work product or model guide subsequent design, implementation and verification phases. Nevertheless, there is very few ways to control 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. 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 [28]. 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 putting emphasis on prototyping and simulation of complex systems before their effective implementation is proven to be a valuable approach. Indeed, prototyping enables to test 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 multi-agent systems. These models are often 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. Thus, formal specification can be used to describe model concepts which can be refined to fulfil a particular system needs. We illustrate this prototyping approach through the formal specification of a multi-agent architecture: the satisfaction-altruism model [8, 42]. 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 MAS. Each concept is given a formal semantics using the OZS formal notation [17]. OZS is a multi-formalisms notation based upon the composition of Object-Z [13] and statecharts [19]. This notation allows to specify 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 [31] or SAL [10]. With this approach we have prototyped several MAS [24, 18, 16, 1, 25, 37].

This paper is organized 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. Eventually section 4 concludes.
2 Concepts

2.1 OZS: multiformalisms 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 multi-formalisms. Such a combination is particularly suited to the specification of complex systems, such as MAS, where both the modelling of processes and states are necessary.

The multi-formalism approaches [47, 35] compose two or more formalisms in order to specify more easily and naturally than with a single formalism. Indeed, the multi-formalism approach deals with complexity by applying formalisms to problem aspects for which they are best suited.

There are two sorts of techniques for multi-formalism integration. The first consists in translating one formalism into another. The second is composition and consists in using in the same specification several formalisms. 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 statecharts constructs.

Object-Z extends Z with object-oriented concepts. The basic construct is the class that 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 communicational aspects [44].

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 STATE-MATE environment [21]. 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 [35]. 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 restraining 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 statecharts actions and Object-Z classes operations. State is the set of states of the included statechart.
Operations are described by Object-Z schemas and can be called in statecharts transition. Attribute is the set of object attributes. The definition of these sets allows the specification in Object-Z of statecharts 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 schema is named, here LoadLock, and contains sub-schemas that specify different aspects of the class. The first sub-schema specifies the state space of the class. In the LoadLock example there is only one boolean attribute called someoneInLL. The following sub-schema 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 sub-schemas 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 sub-schema and a constraints part in the lower predicate part. The ∆-list someoneInLL in the upper part of both sub-schemas 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 (resp outLL) operation states that after the operation someoneInLL becomes true (resp false). The last sub-schema, 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 enter 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 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 in order to specify MAS. 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 [17]. It is an operational semantics which enables automatic verification of specification properties. Available OZS constructs enable natural specification of “low” level aspects inherent to MAS. 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 behaviors. These behaviors can interact mutually according to interaction pattern. Such a pattern which groups generic behaviors and their interactions constitutes an organisation. Organisations are thus descriptions of coordination structures. Coordination occurs between roles as and when interactions takes place. In this context, an agent is only specified as an active communicative entity which plays roles [14]. 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 behaviors defined by the organisation’s roles and when they interact following the organisation 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 [18] 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 runtime. We think this model is a basis for the engineering of societies of agents and what Castelfranchi calls “social order” [7].

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 behavior of the MAS as a whole is the result of the role playing by agents.

The RIO framework is composed of OZS classes [24, 23], one for each concept the RIO metamodel. These classes specify the key organisational concepts needed for specify 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 sub-schema 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 in an heterogeneous basis following the method of Paige[35]. Two constraints (below the short line in the state sub-schema) specified in the Role class use these heterogeneous basis concepts.
An interaction is specified by a couple of roles which are the origin and the destination of the interaction. The role \( \text{orig} \) and \( \text{dest} \) interact using operations \( \text{op}_1 \) and \( \text{op}_2 \). These operations are combined by the \( \parallel \) operator which equates output of \( \text{op}_1 \) and input of \( \text{op}_2 \). 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 one has to inherit 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 [42]. We have made a reverse engineering process to extract from this architecture the different roles and their specifications [25]. 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 s 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 the second subsection.

3.1 The Satisfaction/Altruism architecture

The satisfaction/altruism model proposes a generic mechanism to deal with cooperation and conflict solving in situated multiagent systems [41] [42]. It allows to extend the swarm intelligence approach that 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. Theses 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 is two modules dedicated to individual and cooperative behaviours. The first module computes the Personal Satisfaction (Sat. P box in fig. 1) and the second one measures the agent interaction with its neighbours (Interactive Satisfaction box).

The personal satisfaction gives in real time an evaluation of the agent task progress. This individual satisfaction is expressed by a value $P(t) \in [-P_{\text{max}}, P_{\text{max}}]$. $P_{\text{max}}$ is a limit depending on the application. The Personal satisfaction is computed at each step of the decision-action loop by considering 3 possible task evolution: progress (increase P), regress (decrease P) and blocking (strong decrease of P), see details in [41].

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 (i) to decide if the agent must continue or change its current goal (see details...
We focus on this 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 $I(t) \in [-P_{max}, P_{max}]$ (noted signals $I$ in fig. 1). The semantic of $I$ signals is the following: positive values for attractions and negative ones for repulsions.

A cooperative behaviour consists to react 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

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

This condition expresses that an agent reacts to an external signal if its intensity is greater than the intensity of its current satisfaction (to make 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 the semantic defined above). Note that in order to optimize agents’ navigation this motion vector can be combined to other ones derived from environment constraints, e.g. obstacle avoiding vectors (see [43]).

The satisfaction-altruism model has been validated on different simulated problems such as collective foraging and box-pushing [8], and with real robots to solve navigation conflicts in constrained environments (see details in [29]).

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

In order to specify the Satisfaction-Altruism model we have distinguished two roles: Individual and Altruist. The 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 to 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 not such an action the current action remains unchanged, otherwise the current action is replaced with the new one.

---

**Individual**

<table>
<thead>
<tr>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>current : Action</td>
</tr>
<tr>
<td>initialWeight, weight : Action → [0, 1]</td>
</tr>
<tr>
<td>progressionReward : Action → BMValue</td>
</tr>
<tr>
<td>s : DiscreteSensor</td>
</tr>
<tr>
<td>satisfaction, I : [−P_{max}, P_{max}]</td>
</tr>
</tbody>
</table>

| current ⊆ actions |
| obtainCondition = \{|I_{ext}\() < P() \lor |I_{ext}\() < I\} |
| leaveCondition = \{|I_{ext}\() > P() \land |I_{ext}\() \geq I\} |

---

**I_{ext}**

| ext! : R |
| ext! = s.getMax() |
The Altruist role is also specified by a progressionReward function, a DiscreteSensor and two values, satisfaction and the signal I, measuring satisfaction and external influences.

\[
\text{progressionReward : Action} \rightarrow \text{BMValue} \\
\text{s : DiscreteSensor} \\
\text{satisfaction, I : } [-P_{\text{max}}, P_{\text{max}}] \\
\text{current } \subseteq \text{actions} \\
\text{obtainCondition } = \{ |I_{\text{ext}}| \geq P() \land |I_{\text{ext}}| \geq I \} = \neg \text{leaveCondition} \\
\text{I_{\text{ext}}} \\
\text{ext! : } \mathbb{R} \\
\text{ext! } = \text{s.getMax()}
\]
3.3 Validation of the specification

The prototyping is performed by using STATEMATE [21]; an environment which allows the prototyping and the simulation of the statechart specifications. There exists other software environments for executing statecharts such as, for example, [40] or [20]. STATEMATE is one of the most complete tool regarding statecharts constructs and semantics so we have leaned for 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 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 most dissatisfied 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 find it.

The figures 3b and 3c show an example of such a test. The x axis represents time and y axis represents discretized positions in the corridor for the 3b figure 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 (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 [29], 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 speci-
Figure 4  Hedographs of 2 real robots

ifications 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.

3.4 Proofs of specification properties

In this subsection we establish fundamental properties of the satisfaction/altruism model in corridor environments. OZS semantics [17] is based upon transition systems as defined in [32]. 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 [31, 10] which take as input transition systems and allow to make proofs of properties concerning the specified system. Of course there exists other environments for automatic theorem proving with different input formalism such as, for example, [36, 33, 39]. The semantics defined for our formalism led us to use transition systems tools. Specifically we have chosen the SAL environment. With transition systems and software environment such as SAL [10] 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 a great expertise. A third way proposed in [38] 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 exist a counterexample of length $k$ to invariant $P$ if there is a sequence of states $Init(s_0) \land T(s_0, s_1) \land ... \land T(s_{k-1}, s_k) \land \neg P(s_k)$
The induction is defined as

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

**Step:** \( P(r_1) \land T(r_1, r_2) \Rightarrow P(r_2) \).

This ordinary induction is extended to depth k as:

**Basis:** no counterexample of length k or less

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

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 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.

\[ \Box (\forall i, j \in [1..\sharp \text{Robots}], i \neq j \land \text{left}(r_i) = \text{wall} \land \text{right}(r_i) = \text{robot} \Rightarrow \text{sat}(r_i) < \text{sat}(r_j)) \]

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 other. 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".

\[ \Box (\forall i, j \in [1..\sharp \text{Robots}], i \neq j \land s(i) = \text{altruist} \Rightarrow \text{sat}(i) > \text{sat}(j)) \]

For two different robots in a corridor if one is in altruism state then it 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".

\[ \Box (\forall i, j \in [1..\sharp \text{Robots}], i \neq j \land s(i) = \text{altruist} \Rightarrow \text{direction}(i) = \text{direction}(j)) \]

It means that a robot in altruism state is repulsed and so follow the non altruist 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 [6]. They are graphically presented on a 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 provides only static analysis of goals satisfaction.

Prometheus [2] 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 [26] suggests the use of extended Object Oriented methodologies such as design patterns and CRC. CRC are extended to Role Responsibilities and Collaborators. In [27] a seven layered architectural pattern for agents is presented. These
concepts do not allow the prototyping of MAS.

The Andromeda methodologies [12] use the role notion and propose a step by step methodology in order to design MAS based upon reactive agents integrating machine learning techniques. We have already specified cognitive architecture [18].

The MaSE methodology [11] 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 authorize a dynamic role playing relationship without dropping a hint at how it can be done.

Bergenti and Poggi [3] 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 [34], 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 [22] is based upon the refinement of informal requirement specifications to semi-formal and then formal specifications. The system structuring is based on a hierarchy of components [5]. These components are defined in terms of input/output and temporal constraints. A kind of testing technique based upon model checking is proposed in [22] but it is limited to some logical properties. Luck and d’Inverno [30] 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 a same Object-Z class. Second, Luck and d’Inverno’s framework does not allow one to specify temporal and reactive properties of MAS [15]. 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 [45] and its extension [46] deals with analysis and design of MAS. It is composed of two abstraction levels: agent level and structural organizational level. No tool is proposed for this methodology.

5 Conclusion

In this paper we have 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 multi-formalisms notation is given a semantics in terms of transition systems. This semantics allows the animation of specifications and the verification of properties by using software environments such as STATEMATE [21] and SAL [10]. 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 realized with real robots. In order to validate the specifications we have executed the specifications. In fact, the specifications were animated in order to validate the specifications against real scenarios. Once the specifications were validated so we were sure that they constitute a good model of the architectures we were able to prove pertinent properties using formal proofs. The properties we have proven were linked to the requirements of the architecture. These properties are emerging properties that enables to solve deadlocks during exploration.

This prototyping approach was also used to specify and analyse several MASs from a cognitive architecture [18] to an immune based architecture [1] and holonic MAS [37].

However, we are aware of our approach limitations. 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. Morever, we can’t say that our formalism is adapted for all MAS architectures. Up to now, for the problems we have delt with we encountered no expressive limitations but it is not a 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 ease the use of such techniques as organisation and roles define components that can be verified separately. These techniques will facilitate the proofs of complex properties.

We follow our previous works on the specification of MAS methodologies and architectures. We are currently formalising the RIO semi-formal notation and trying to apply it for holonic MAS modeling. Several methodologies such as PASSI [9] are currently under specification. It will allow to give it a formal and precise meaning. We are also developing a software environment in order to help the specifier using our process.

References and Notes


Jérôme Chapelle, Olivier Simonin, and Jacques Ferber. How situated agents can learn to cooperate by monitoring their neighbors’ satisfaction. In 15th European Conference on Artificial Intelligence, 2002.


M. Fisher. If Z is the answer, what could the question possibly be? In Intelligent Agents III, number 1193 in Lecture Note of Artificial Intelligence, 1997.


John Rushby. Tutorial on mechanized formal methods.


