International Journal of Software Engineering
and Knowledge Engineering
Vol. 17, No. 5 (2007) 615–641
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

FORMAL SPECIFICATION APPROACH OF ROLE DYNAMICS IN AGENT ORGANISATIONS: APPLICATION TO THE SATISFACTION-ALTRUISM MODEL

VINCENT HILAIRE*, PABLO GRUER†, ABDER KOUKAM‡ and OLIVIER SIMONIN§

Systems and Transportation Laboratory,
University of Technology of Belfort Montbéliard, Belfort, 90010, France
* *vincent.hilaire@utbm.fr*
† *pablo.gruer@utbm.fr*
‡ *abder.koukam@utbm.fr*
§ *olivier.simonin@utbm.fr*
http://set.utbm.fr

Received 6 March 2006
Revised 3 April 2006
Accepted 25 April 2006

This article deals with the problem of dynamic role-playing in Multi-Agent organisations. The approach presented uses a formal specification notation and is based upon a formal framework which defines the concepts of role, interaction and organisation. Within this framework the problem of dynamic role-playing specification is related to the merging of specifications. The formal notation used composes Object-Z and Statecharts. The main features of this approach are: enough expressive power to represent Multi-Agents dynamic aspects, tools for specification analysis and mechanisms allowing the refinement of a high level specification into a low level specification which can be easily implemented. The last part of this paper presents an application with the specification of a reactive and cooperative MAS model named Satisfaction Altruism. An analysis of the specification validates the agents' behaviours.

Keywords: Multi-agent systems; dynamic role-playing; formal specification; organisational methodology.

1. Introduction

Software agents and Multi-Agent Systems (MAS in the sequel) have become an appealing paradigm for the design of computer systems composed of autonomous cooperating software entities [45]. This paradigm provides a new approach to analyse, design and implement such systems based upon the central notions of agents, their interactions and the environment which they perceive and within which they act. Nevertheless it is still difficult to engineer MAS. Indeed, when massive number of autonomous components interact it is very difficult to predict the behaviour of the system and ensure that the desired functionalities will be fulfilled.

This article presents a methodological approach for building Multi-Agent Systems specifications. The basic idea is to define such systems as a set of entities playing roles that have interactions between them. Specifically, we present a mechanism for dynamic role-playing specification within a formal framework.

There exists several software engineering approaches [28] which try to fill the gap between MAS analysis and development life-cycle. These approaches differ in many ways like, for example, the notation employed, the basic concepts used and the presence or absence of methodological guidelines. The notation may be formal or not.

Due to their complexity, MAS have reactive and transformational features. A formalism is yet to be defined that specifies easily and naturally both aspects, enables validation and verification, and guides the implementation phase. We have thus chosen to use a multi-formalisms approach that results in the composition of Object-Z [11] and Statecharts [20] which are presented later on. The advantages of this combination are threefold. First, this formalism enables the specification of reactive and transformational aspects of MAS. Second, both formalisms relate strongly to the construction of actual software. Indeed, in Object-Z there are refinement mechanisms [44] and statecharts which allow code generation [21]. Third, we have defined an operational semantics in terms of transition systems for this combination [16]. The latter point enables the use of theorem prover such as STeP [36] or SAL [6]. We have called this formalism OZS (which stands for Object-Z and Statecharts).

A specification method is essential to manage MAS complexity using decomposition and abstraction. Some approaches use organisational concepts to model MAS [12, 47, 26]. The use of such primitive concepts enables one to go from the requirements to detailed design and helps to decompose a MAS in terms of roles and organisations. In fact, it is a three steps approach. The first step views the system as an organisation or a society defined by a set of roles and their interactions. The second introduces the agents and assigns roles to them according to some design criteria. The third focuses on the design of the internal architecture of agents. With the concepts we have defined the Role Interaction Organisation (RIO) Framework which specifies each concept using OZS classes. By doing so we obtain methodological guidelines formally grounded for the analysis and design of MAS. Moreover, the semantics we have defined for the OZS notation allows the animation and verification of properties. The latter is very important in order to ensure that desired properties or required functionalities will be satisfied in the system. We have, for example, specified a MAS for solving a radio-mobile network field problem [32, 18]. For this problem we have proved safety and liveness properties with the STeP environment [36]. We have also animated a RIO model of foot-and-mouth disease simulation [26].

One drawback of organisational-based methodologies is that they are frequently restricted to static role-playing relationships. The purpose of this article is to extend our approach to deal with dynamic role-playing relationships. Thus we can specify models such as the satisfaction-altruism one that has been applied to the mobile robotic field [34, 41].

The idea that we suggest for dynamic role-playing relationship is inspired from viewpoints specification. Viewpoints approaches have been widely used in requirement analysis [1, 8, 14, 31]. They consist in separating partial specifications of large and complex systems. There has been some work undertaken on a combination of these specifications [1, 8]. The problem of dynamically combining roles specifications is similar to combining viewpoints. We have to merge several specifications dynamically and check their consistency. We propose an approach to merge role specifications based upon a refinement relationship. The refinement maps a set of roles to the result of the merging.

The rest of this paper is organised as follows: Section 2 introduces the OZS formalism; Sec. 3 presents the dynamic role-playing relationship on an example; Sec. 4 illustrates the dynamic roleplaying relationship on the satisfaction/altruism example; Sec. 5 presents related works and eventually Sec. 6 concludes.

2. Background

2.1. *OZS = Object-Z + statecharts*

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 [49, 39] 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-formalisms 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 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 [43].

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 [22]. 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 [39]. 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 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 Δ-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.

LoadLock

$someoneInLL : \mathbb{B}$

INIT

$\neg\ someoneInLL$

inLL

$\Delta someoneInLL$

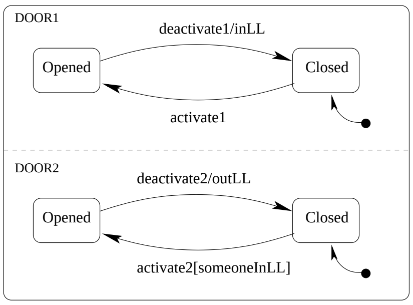
$someoneInLL'$

outLL

$\Delta someoneInLL$

$\neg\ someoneInLL'$

behaviour



$\neg \Diamond(instate(DOOR1.opened) \land instate(DOOR2.opened))$

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 specifications. Moreover, aspects such as reactivity and

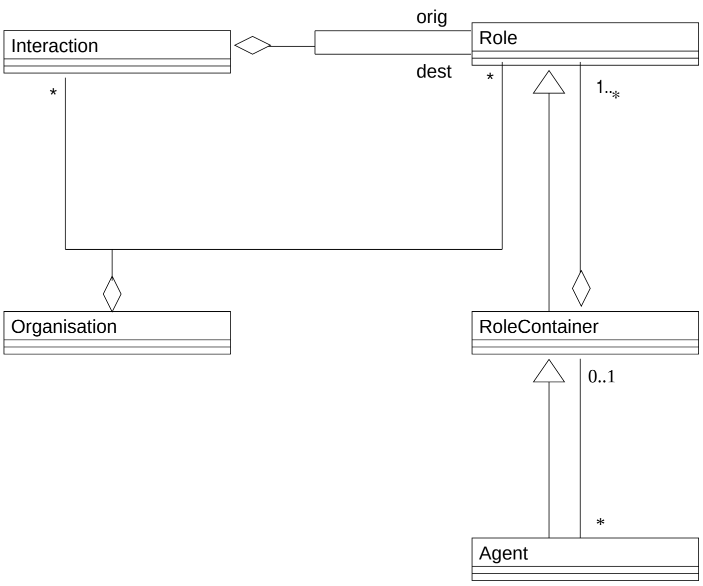


Fig. 1. RIO meta-model.

concurrency can be easily dealt with. The semantics of the OZS notation is defined by means of transition systems [16]. 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.

3. Dynamic Role-Playing Mechanism

3.1. *RIO overview*

The RIO framework is composed of OZS classes [26, 24]. These classes specify the following abstract concepts: Role, Interaction, Organisation, RoleContainer and Agent. Figure 1 describes the relationships between the RIO framework classes using the UML notation.

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. *RoleContainer* classes specify social positions i.e. roles that are played simultaneously. For example, the roles Lecturer and Researcher can be played simultaneously and they define a specific position in the university. Each agent is related to several roles it is playing. In fact an agent occupies a specific position in an organisation specified by a *RoleContainer*. The roles an agent plays at a specific moment are specified by the position it is occupying. When an agent occupies another position it plays another set of 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 and under which constraints. These are specified by *RoleContainer* classes. *RoleContainer* groups a set of coherent roles played simultaneously.

Playing a role means for an agent that it exhibits the behaviour and characteristics specified by the role. One can see roles as specific viewpoints. Indeed, each role is a partial specification of an agent. There has been several attempts to combine viewpoints specifications [1, 8].

The idea we adopt to merge several role specifications and check their consistency is to define a relationship between the roles and agents' specifications. This relationship is a sort of refinement that defines the attributes, actions, stimulus and behaviours of the agent by refining those of the roles it plays.

3.2. *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 [39]. Two constraints specified in the *Role* class use these heterogeneous basis concepts.

Role

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

behaviour

An interaction is specified by a couple of roles which are the origin and the destination of the interaction. The role *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 one has to inherit from the *Interaction* class.

Interaction

$$\begin{array}{l}
orig, dest : Role \\
op_1, op_2 : Action \\
\hline
op_1 \in orig.action \\
op_2 \in dest.action \\
\hline
\diamond(orig.op_1 \| dest.op_2)
\end{array}$$

Organisation

$$\begin{array}{l}
roles : \mathbb{P}\, Role \\
interactions : \mathbb{P}\, Interaction \\
\hline
\forall i \in interactions \bullet \\
\quad (i.orig \in roles \wedge i.dest \in roles)
\end{array}$$

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

A *RoleContainer* specifies an entity which plays a set of roles. The role-playing relationship is static. In other words, the set of played roles does not change. In the *RoleContainer* the link between entities which play roles and roles specification is specified. Indeed, three functions $Retrieve_{att}$, $Retrieve_{stim}$ and $Retrieve_{op}$ map respectively attributes, stimulus and actions of roles with attributes, stimulus and actions of the entity playing the roles. These functions impose that for each attribute, stimulus or action of a role there exists an attribute, stimulus or action which refines it in the *RoleContainer*. The refinement relationship is denoted by the $\preccurlyeq$ symbol. An object o' refining an object o is represented as $o \preccurlyeq o'$.

RoleContainer

Role

$playing : \mathbb{P}\, Role$

$\exists\, Retrieve_{att}: Attribute \rightarrow Attribute \bullet$
$\quad \forall\, r \in playing \bullet$
$\quad \forall\, a \in r.attributes, \exists\, a' \in attributes \bullet a' = Retrieve_{att}(a) \wedge a \preccurlyeq a'$
$\exists\, Retrieve_{op}: Action \rightarrow Action \bullet$
$\quad \forall\, r \in playing \bullet$
$\quad \forall\, a \in r.actions, \exists\, a' \in actions \bullet a' = Retrieve_{op}(a) \wedge a \preccurlyeq a'$
$\exists\, Retrieve_{stim}: Stimulus \rightarrow Stimulus \bullet$
$\quad \forall\, r \in playing \bullet$
$\quad \forall\, a \in r.stimulus, \exists\, a' \in actions \bullet a' = Retrieve_{stim}(a) \wedge a \preccurlyeq a'$
$\exists\, Retrieve_{beh}: Statechart \rightarrow Statechart \bullet$
$\quad \forall\, r \in playing \bullet$
$\quad \forall\, Retrieve_{beh}(r.behaviour) \preccurlyeq behaviour$

The *Agent* class inherits from *RoleContainer*. This class is defined by a *position* which is an instance of *RoleContainer*. The agent position is the set of roles it plays at a specific moment in time. These roles define the agent's status and behaviour in all contexts which it may encounter. The position of an agent may change during its lifetime. This mechanism allows dynamic role-playing. The *addRole* (resp. *leaveRole*) operation adds (resp. substracts) a role to (from) an agent. The preconditions of this operation impose that *obtainConditions* (resp. *leaveConditions*) must be satisfied. These operations modify the functions which map attributes, stimulus and actions from roles to agents. These functions defined statically in the *RoleContainer* class are modified dynamically whenever an agent changes one of its roles.

An agent A is also defined by an *acquaintances* set. This set represents the other agents which are currently interacting with A.

Agent

RoleContainer

$acquaintances : \mathbb{P}\, Agent$
$position : RoleContainer$

$\forall\, a \in acquaintances, \exists\, r_1, r_2 : Role,$
$\quad \exists\, i : Interaction \bullet$
$\quad r_1 \in playing$
$\quad \wedge\, r_2 \in a.playing$
$\quad \wedge\, (r_1, r_2) \in i.roles$
$\forall\, r \in position \bullet r.behaviour \subseteq behaviour$

addRole

$$\begin{array}{l}
\Delta position \\
r? : Role \\
\hline
r.obtainConditions \\
playing' = playing \cup r? \\
\forall\, a \in r.attributes \bullet (Retrieve'_{att} = Retrieve_{att} \oplus \{r.a, a'\}) \\
\qquad \wedge ((a' \in attributes \wedge a \preccurlyeq a') \\
\qquad\quad \vee (attributes' = attributes \cup \{a'\} \wedge a \preccurlyeq a')) \\
\forall\, a \in r.actions \bullet (Retrieve'_{op} = Retrieve_{op} \oplus \{r.a, a'\}) \\
\qquad \wedge ((a' \in actions \wedge a \preccurlyeq a') \\
\qquad\quad \vee (actions' = actions \cup \{a'\} \wedge a \preccurlyeq a')) \\
\forall\, a \in r.stimulus \bullet (Retrieve'_{stim} = Retrieve_{stim} \oplus \{r.a, a'\}) \\
\qquad \wedge ((a' \in stimulus \wedge a \preccurlyeq a') \\
\qquad\quad \vee (stimulus' = stimulus \cup \{a'\} \wedge a \preccurlyeq a'))
\end{array}$$

leaveRole

$$\begin{array}{l}
\Delta(position) \\
r? : Role \\
\hline
r.leaveConditions \\
playing' = playing \setminus r?
\end{array}$$

$$\begin{array}{l}
\Box(\forall\, r \in playing, \forall\, e \in r.behaviour.\varrho \bullet \\
\qquad\qquad e.instate \\
\qquad\qquad \wedge\, (\exists\, t : Transition \bullet t.source = e) \Rightarrow \\
\qquad\qquad\quad (t.event \\
\qquad\qquad\quad\; \wedge\, t.conditions \Rightarrow \\
\qquad\qquad\quad\; (\forall\, f \in t.destinations \bullet \bigcirc f.instate)) \\
\Box(\forall\, r : Role \bullet (r \notin playing) \wedge \bigcirc(r \in playing) \Rightarrow r.obtainConditions) \\
\Box(\forall\, r : Role \bullet (r \in playing) \wedge \bigcirc(r \notin playing) \Rightarrow r.leaveConditions)
\end{array}$$

In this context, an agent is only specified as an active communicative entity which plays roles [12]. 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 organisation's 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 [17] we have specified a specific cognitive agent architecture by extending the RIO framework classes.

3.3. *Refinement relationship*

The dynamic role-playing mechanism is based on the refinement of the roles contained in the playing set of a *RoleContainer*. The attributes, actions, stimulus and behaviours are refined to define attributes, actions, stimulus and behaviour of a *RoleContainer* which can be substituted to any of the roles. If such a refinement exists, we can construct it, given a relation between attributes, actions, stimulus and behaviours of roles and those of a *RoleContainer*. The definition process of this relation is a four step process. First, the attributes are mapped. In the second step the actions are mapped. The two last steps are stimulus and behaviour mapping.

If r_1 and r_2 are two roles and $\preccurlyeq$ the refinement relationship we want to construct a *RoleContainer* rc such that $r_1 \wedge r_2 \preccurlyeq rc$. Let the relations $Retrieve_{beh}$, $Retrieve_{stim}$, $Retrieve_{op}$ and $Retrieve_{att}$ be the relations between roles and *RoleContainer* behaviour, stimulus, operations and attributes, respectively. We must verify

$$\begin{aligned}
&rc.attributes = Retrieve_{att}(r_1.attributes) \cup Retrieve_{att}(r_2.attributes)\\
&\wedge\ rc.actions = Retrieve_{op}(r_1.actions) \cup Retrieve_{op}(r_2.actions)\\
&\wedge\ rc.stimulus = Retrieve_{stim}(r_1.stimulus) \cup Retrieve_{stim}(r_2.stimulus)\\
&\wedge\ rc.behaviour = Retrieve_{beh}(r_1.stimulus) \cup Retrieve_{beh}(r_2.stimulus)
\end{aligned}$$

The first two steps are the definition of $Retrieve_{att}$ and $Retrieve_{op}$ that concern the Object-Z part. The third and fourth steps are the definition of $Retrieve_{stim}$ and $Retrieve_{beh}$ that concern the statechart parts of the multi-formalism notation. The following subsections deal with each part separately.

3.4. *Object-Z part*

This mechanism allows the specification of several, possibly overlapping, roles to be merged into one entity. There are three different combination cases which can concern the attributes and the operations.

The first case is when attributes specifying the same *RoleContainer* attribute are defined on the same definition domain. The mapping is then the identity function.

The second case is when two attributes specifying the same *RoleContainer* attribute are defined with different definition domains. The function must map these attributes to a *RoleContainer* attribute which is defined over the union of the two definition domains.

The third case is when two attributes specifying the same *RoleContainer* attribute are defined with different definition domains. The function must map the two attributes to a subset of the Cartesian product of the definition domains.

Let $\perp$ be the undefined value for all definition domains, a_1 and a_2 two attributes of two roles. The notation $a \upharpoonright i$ stands for the ith component of the tuple a. The three cases can be summarized as follows:

$$\begin{cases} a = \mathit{Retrieve}_{att}(a_1) = \mathit{Retrieve}_{att}(a_2) \\ (a_1 = \bot \Rightarrow a = \mathit{Retrieve}_{att}(a_2)) \vee (a_2 = \bot \Rightarrow a = \mathit{Retrieve}_{att}(a_1)) \\ a = (a_1, a_2) \wedge \mathit{Retrieve}_{att}(a_1) = a \upharpoonright 1 \wedge \mathit{Retrieve}_{att}(a_2) = a \upharpoonright 2 \end{cases}$$

For the operations, the first and simplest case is when operations are equivalent. The resulting mapping is the identity function. The other cases occur when at least one mapped attribute belongs to the parameters of several operations. If the mapped attribute is the result of the union of several definition domains then the operation applied is the one which is concerned by the restricted domain. If the mapped attribute is defined as a Cartesian product then each operation is sequentially applied modifying each component of the Cartesian product. The $\widehat{=}$ notation used below stands for schema (operation schema in this case) definition and the notation $\fatsemi$ stands for schema composition.

$$\begin{cases} op \mathrel{\widehat{=}} \mathit{Retrieve}_{op}(op_1) \mathrel{\widehat{=}} \mathit{Retrieve}_{op}(op_2) \\ (\mathit{pre}\ op_1 \Rightarrow op \mathrel{\widehat{=}} \mathit{Retrieve}_{op}(op_1)) \vee (\mathit{pre}\ op_2 \Rightarrow op \mathrel{\widehat{=}} \mathit{Retrieve}_{op}(op_2)) \\ op \mathrel{\widehat{=}} op_1(a \upharpoonright 1) \fatsemi op_2(a \upharpoonright 2) \end{cases}$$

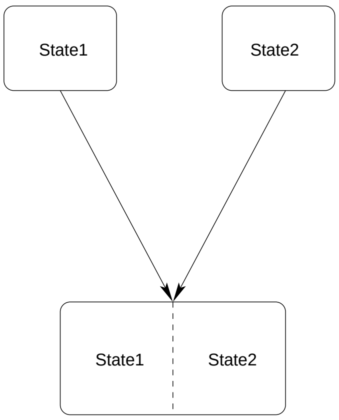
The predicate *pre op* is true whenever the preconditions of op_1 are true.

3.5. *Statechart part*

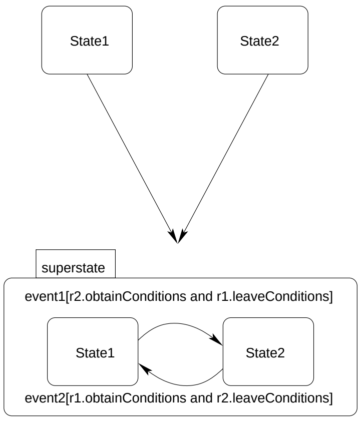
Since we consider only simple events $\mathit{Retrieve}_{stim}$ is just a renaming. There are three different cases for the statecharts refinement. The first case is when two statecharts specifying two role behaviours are independent. In this case, the result of the refinement of the two roles consists in behaving like both roles at the same time. The resulting statechart is the AND composition of the statecharts specifying the two role behaviours. If State_1 is the statechart corresponding to the first role and State_2 is the statechart corresponding to the second role the resulting statechart is a super-state which encapsulates State_1 and State_2 composing them by an AND composition as illustrated in Fig. 2(a). The semantic of the AND composition is that State_1 and State_2 are both active simultaneously.

The second case occurs when two statecharts specifying two role behaviours are not independent but exclusive. This is the case when one role must not be played in conjunction with the other. In terms of statecharts the resulting statechart is the XOR composition of the statecharts specifying the two roles behaviours. The XOR composition means that the two statecharts are part of a same super-state and that there may exist a transition from one state to another as illustrated in Fig. 2(b). This transition is guarded by the *obtainCondition* of the destination role and *leaveCondition* of the origin role.

The third case occurs when one statechart, say State_1, specifying a role behaviour is part of another statechart, say State_2, specifying the behaviour of another role. In this case the role specified by State_1 must be included in the hierarchy of



(a) and refinement


State1
State2
superstate
event1[r2.obtainConditions and r1.leaveConditions]
State1
State2
event2[r1.obtainConditions and r2.leaveConditions]

(b) xor refinement

Fig. 2. Statecharts refinement.

roles which constitutes the behaviour of the role specified by $State_2$. This case results from a modelling problem. Indeed playing the role specified by $State_2$ implies playing the role specified by $State_1$.

$$\begin{cases} stc = (AND(Retrieve_{beh}(stc_1), Retrieve_{beh}(stc_2)) \\ stc = (XOR(Retrieve_{beh}(stc_1), Retrieve_{beh}(stc_2)), \\ (stc_1, stc_2, [r2.obtainConditions \wedge r1.leaveConditions], \\ (stc_2, stc_1, [r1.obtainConditions \wedge r2.leaveConditions]))) \\ (stc_1 \subseteq stc_2 \Rightarrow stc = Retrieve_{beh}(stc_2)) \vee (stc_2 \subseteq stc_1 \Rightarrow stc = Retrieve_{beh}(stc_1)) \end{cases}$$

In the following section, we present a specification of a concrete multi-agent model called the satisfaction altruism model. The aim of this example is to illustrate the proposed formal specification framework.

4. Example: the Satisfaction-Altruism Model

4.1. *Introduction to the satisfaction-altruism model*

This model aims at providing a means of cooperation and conflict solving for behaviour-based agents working in the same environment [41, 42]. In order to introduce local intelligent interactions into the collective approach an extension of the artificial potential fields (APF) approach is proposed. The satisfaction-altruism model introduces *new artificial fields* in the environment [42]. These fields are *dynamically* and intentionally generated by agents thanks to the emission of attractive and repulsive *signals*. Agents broadcast such signals in order to influence their close neighbours. These artificial fields augment the information present in the environment to improve cooperation between agents. At the heart of the model there are

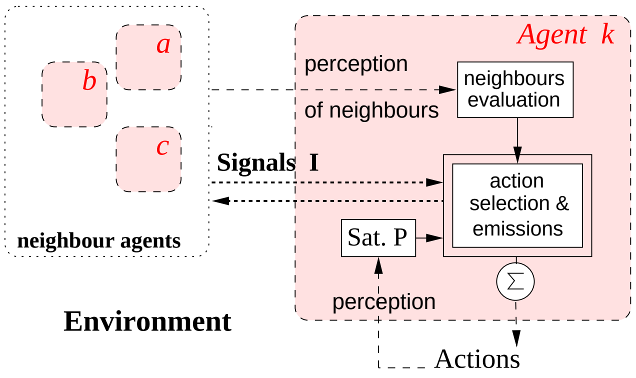


Fig. 3. Scheme of agent-agents and agent-environment interactions in the Satisfaction-Altruism model.

two modules dedicated to individual and cooperative behaviours. The first one measures the agent satisfaction, i.e. gives in real time an evaluation of the agent task progress. This *individual satisfaction* is defined by a value $P(t) \in [-P_{\max}, P_{\max}]$ (represented by Sat P box in Fig. 3). The second module evaluates a complementary satisfaction which concerns the interactions of the agent with its neighbours (noted perception of neighbours in Fig. 3). These satisfactions are used in the action-selection module to decide if the current behaviour must be continued or changed [5]. These measures of satisfactions are also used to enable cooperative interactions. Agents can communicate their satisfactions in order to influence their neighbours. They can locally broadcast attractive or repulsive signals, defined as numerical values: $I(t) \in [-P_{\max}, P_{\max}]$ (noted signals I in Fig. 3). The semantic is the following: positive values for attractions and negative ones for repulsions. A *cooperative behaviour* consists in moving according to the perceived signals: go towards the source if attraction or go away if repulsion. This reaction, named the altruistic behaviour, is performed only when

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

where I_{ext} is the strongest perceived signal. This condition expresses that an agent reacts to a signal if the signal intensity is greater than the intensity of its own satisfaction. The altruistic reaction is defined as a repulsive or attractive vector (which depends directly on the sign and intensity of the value received). Note that this vector can be combined to environment constraints, e.g. obstacles, to optimize agents' navigation.

Eventually, a process of signal-passing ensures the propagation of the signals. The signal-passing defines quick recruitment processes and chains of repulsive influences to free blocked agents [42].

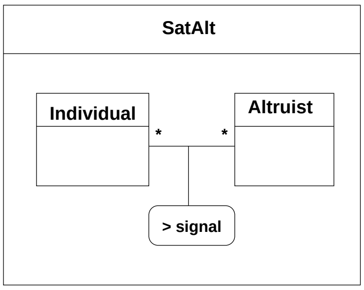


Fig. 4. Satisfaction Altruism organisation.

This model has been validated on different simulated problems such as foraging, navigation in constrained environments, box-pushing [5] and with real robots (see details in [34]). Figure 4 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.

4.2. *Roles*

In order to specify the Satisfaction-Altruism model we have distinguished two roles: *Individual* and *Altruist*.

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 evolution of the system. The initial values of the weights are defined by *initialWeight*. The *progressionReward* function maps each action to a 3-uplet giving the bonus 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

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

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

I_{ext}

$ext! : \mathbb{R}$

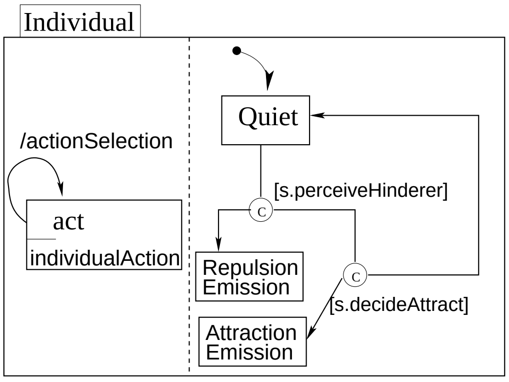
$ext! = s.getMax()$

actionSelection

$\Delta(current, weight)$

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

behaviour



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

Altruist

Role

$progressionReward : Action \rightarrow BMValue$
$s : DiscreteSensor$
$satisfaction, I : [-P_{\max}, P_{\max}]$

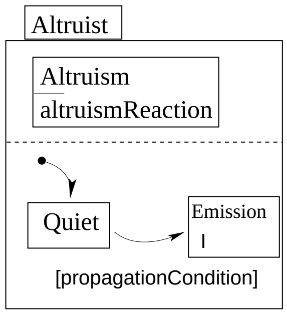
$current \subseteq actions$
$obtainCondition = \{|I_{\text{ext}}()| \geq P() \wedge |I_{\text{ext}}()| \geq I\} = \neg\ leaveCondition$

I_{ext}

$ext! : \mathbb{R}$

$ext! = s.getMax()$

behaviour



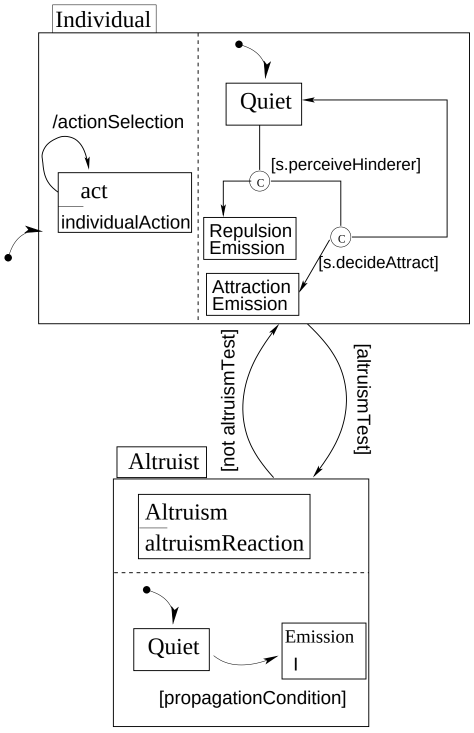
The *RoleContainer SatAlt* groups the role *Individual* and *Altruist.* All attributes of the two roles are mapped by the identity function as they share the same semantics and definition domains. The I_{ext} and *actionSelection* operations are mapped to the operations of *SatAlt.* The behaviour of *SatAlt* is defined by an exclusive mapping of the behaviours of *Individual* and *Altruist.* Indeed, an agent can either play the *Individual* role or the *Altruist* role but not both simultaneously. Satisfied transitions between each role are fired when the *leaveCondition* of the active role is true.

SatAlt

RoleContainer

$playing = \{Individual, Altruist\}$

behaviour



4.3. *Agent*

The class SAAgent inherits from Agent and its position is *SatAlt*. It means that initially the agent plays the role *Individual* and if the *altruismTest* is true then it leaves this role to play the *Altruist* role. If the *altruismTest* becomes false then it leaves the *Altruist* role to play the *Individual* role. The *altruismTest* is defined using the *obtainCondition* of the *Altruist* role.

SAAgent

Agent

$position = SatAlt$

4.4. *Specification analysis*

The specification analysis is performed by using STATEMATE [22]; an environment which allows the prototyping and simulation of statechart's specifications. The specification analysis is based on the execution of the statechart and can be carried out 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.

Animation technique consists of testing the specification with predefined interaction scenarios. It enables one to test if the system's behaviour is consistent with requirements. For example, we have tested that simulated autonomous robots blocked in a narrow corridor can coordinate their behaviours to explore it by using the satisfaction-altruism model [27]. Figure 5 shows an example of such a test. The x axes represents time-points and the y axes represents discretized positions in the corridor for the upper figure, and the level of satisfaction of each robot for the lower figure. The individual behaviour consists of exploring the corridor with closed extremities. One can see that levels of satisfaction and trajectories are correlated. Indeed, each time the two robots are locked the satisfaction levels decrease. Here the satisfaction variation is defined to depend on environmental constraints: the more a robot is surrounded by walls, the faster its satisfaction falls. Thus the robot locked against a wall is more quickly dissatisfied than the other. Agents emit their dissatisfactions as local signals, then the altruism test becomes true for the less constrained robot. This robot plays the altruist role and changes its direction (it is the case around times 109, 155 and 235 in Fig. 5). If a robot is not 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. The parameters can be easily modified in order to check the specification against pertinent test cases. This approach enables the validation of the specification.

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 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.

4.5. *Property proof*

OZS semantics [16] is based upon transition systems as defined in [37]. 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.

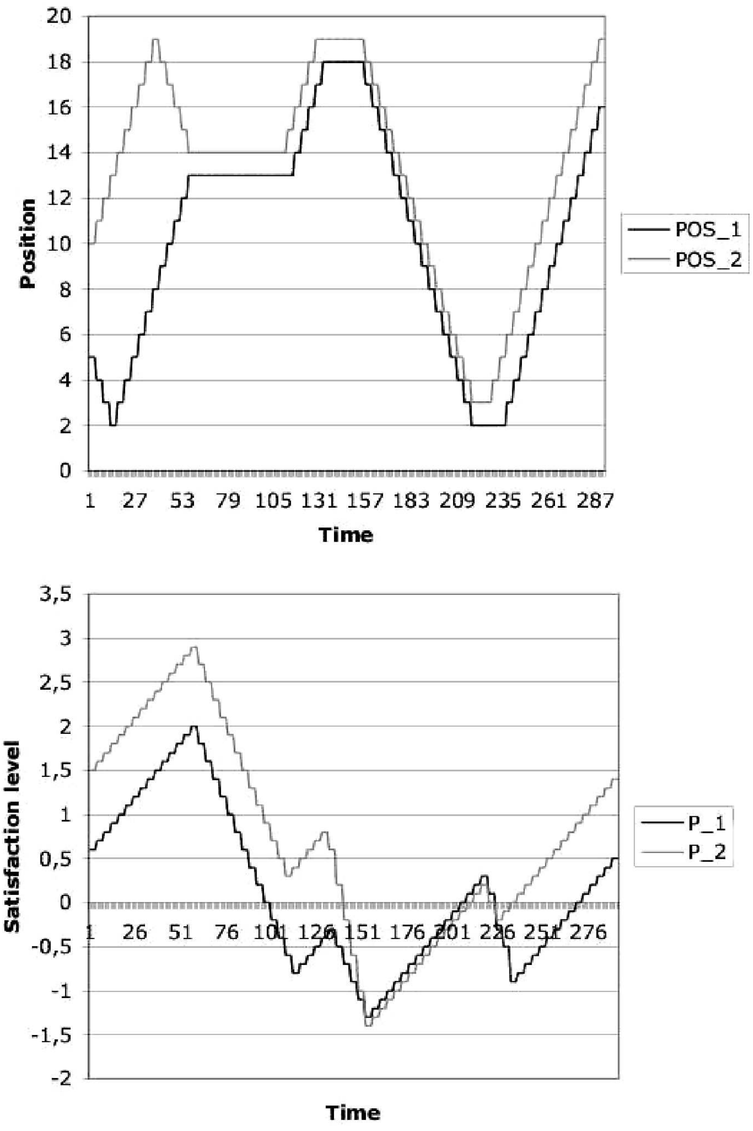


Fig. 5. Robots' trajectories and satisfactions.

With such transition systems and software tools like SAL [6] one can verify specification properties.

Among the tools proposed by SAL we have chosen the SAL model checker which enables the verification of the satisfiability of a property. The SAL model-checker proves or refutes validity of Linear Temporal Logic (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. We have also used first the SAL path finder. This tool is a random trace generator based on SAT solving. It enables us to be sure that the transition system specification we have produced from the specification has the desired semantics.

In order to prove properties we have generated the Transition System considering the classes presented in this section.

The property we have proven using the model checker is specified as follows:

$$\mathit{left}(r_1) = \mathit{wall} \wedge \mathit{right}(r_1) = \mathit{robot} \Rightarrow \Diamond \mathit{right}(r_1) = \mathit{empty}$$

The functions *left* and *right* specify the perception of a robot inside the corridor. These functions can take value in the $\{\mathit{empty}, \mathit{robot}, \mathit{wall}\}$ set. The formula states that if a robot r_1 perceives a wall on its left side and a robot on its right side then eventually the robot at its right will move.

The model checker with this property and the transition system produced generate no counter-example. It means that whenever a robot is locked between a wall and another robot it will be freed.

5. Related Works

This section describes approaches for Agent-Oriented Software Engineering. We have divided these approaches into two parts: the semi-formal approaches and the formal ones. Our approach uses the semi-formal diagrams to represent the analysis in terms of roles, interactions and organisations. Each concept has a formal notation and semantics. This mapping from semi-formal to formal representation is not present in the following semi-formal approaches.

5.1. *Semi-formal approaches*

The i^* framework of Yu [48] is based upon requirements analysis by means of goals, dependencies, roles, actors, positions and agents. These notions are similar to those we use and enables us to deal with dynamic role playing. They are graphically presented on the same schema. It is sometimes difficult to read such schemas where all concepts are on the same level. We think that a schema which separates analysis and design level is needed.

Kendall [29] suggests the use of extended Object Oriented methodologies like design patterns and CRC. CRC are extended to Role Responsibilities and Collaborators. In [30] a seven layered architectural pattern for agents is presented. The seven layers are: mobility, translation, collaboration, actions, reasoning, beliefs and sensory. This architecture is dedicated to mobile agents. Dynamic role playing is enabled by the use of aspects oriented programming and so considered only at implementation level. With our approach we consider dynamic role playing at the analysis level.

The Andromeda methodologies [10] use the role notion and propose a step by step methodology in order to design MAS. Andromeda deals with machine learning techniques for MAS. These methodologies are oriented towards the design of reactive MAS. We have already specified cognitive architecture [17].

The MaSE methodology [7] insists upon the necessity of software tools for software engineering, specifically code generation tools. This methodology has seven steps. These steps are structured sequentially. It begins with the identification of overall goals and their structuring. From goals with use case and sequence diagrams one can identify roles and define them as a set of tasks. Agents are then introduced with class diagrams with a specific semantics. The last step consists in defining precisely: high level protocols, agent architectures and deployment diagrams. For each step a different diagram is introduced. MaSE methodology suffers from limited one-to-one agent interactions. The authors authorize a dynamic role playing relationship without dropping a hint of how it can be done.

Bergenti and Poggi [2] 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 [38], 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.

The problem with latter notations is the UML starting point. Indeed, using an object oriented notation in order to describe MAS leads specifiers to use object oriented concepts. We think that a MAS methodology must insist on agent oriented concepts first as it is the case in our approach.

5.2. *Formal approaches*

Formal approaches for MAS specification are numerous but they are often abstract and unrelated to concrete computational models [9]. Temporal and modal logics, for example, have been widely used [46]. Despite the important contribution of these works to a solid underlying foundation for MAS, no methodological guidelines are provided concerning the specification process and how an implementation can be derived.

Another type of approach consists in using traditional software engineering or knowledge based formalisms [35]. One advantage of using such approaches is that they are widely used. The expertise concerning such notations is greater than newer ones and there are tools which help the specification process.

For example, the approach proposed by [23] 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 [3]. These components are defined in terms of input/ouput and temporal constraints. With this approach it seems difficult to refine specifications into an implementation language. Moreover, the verification technique is limited to model checking. It seems difficult to introduce a kind of dynamic role playing relationship.

Luck and d'Inverno [35] 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 two 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 [13]. In our framework these aspects are specified by temporal invariants and statecharts. Concerning the specification of organisational aspects the authors adopt a norm approach [33]. The norms allows the definition of organisational rules that regulate societies of agents. Agents can adapt their goals according to these norms. We have chosen a more constructivist approach in order to be able to deduce the agent implementation. Nevertheless, this framework has a significant contribution to clarify several aspects of agent oriented approaches.

Wooldridge, Jennings and Kinny [47] propose the Gaia methodology for agent oriented analysis and design. This methodology is composed of two abstraction levels: agent level and structural organisational level. The role concept exists in Gaia; however the relationship between agent and role is a static one.

6. Conclusion

In this paper, we have presented a formal specification approach for MAS based upon an organisational model. The organisational model describes interaction patterns which are composed of roles. When playing these roles, agents instantiate interaction patterns. An agent can play several roles and can change the roles it plays. Specifically, we have built a formal specification of role dynamics. Among organisational approaches, to our knowledge, there are no such semantics. This model is thus well suited for describing complex interactions which are among MAS main features. As an example we have presented a part of the specification of the satisfaction-altruism model used for autonomous robots [42, 34]. The language used by the specification framework can describe reactive and functional aspects. It is structured as a class hierarchy. A new specification is produced by inheriting from these classes. Several MAS have already been specified with the RIO framework with or without using a specific agent architecture [25, 26, 4, 17]. These MAS have been applied to complex problems like radio-mobile network field [32], foot-and-mouth disease simulation [26] and office automation [15].

The specification language used allows the prototyping of specification [26]. In this paper we have pointed out some advantages and examples of the use of this technique. Prototyping is not the only means of analysis. Indeed, we are working on automatic verification of the specification presented in this paper. For example, we have used the SAL model checker to verify a property which may be interpretated as "When a robot is locked by another robot it's eventually freed". Moreover, the specification structure enables incremental and modular validation and verification through its decomposition. Eventually, such a specification can be refined to an implementation with multi-agent development platform like MadKit [19] which is based upon an organisational model [12].

Despite the encouraging results already achieved, we are aware that our approach still has some limitations. Indeed, it does not tackle all problems raised by a MAS development methodology. Among issues remaining for future work our organisational model needs to be improved. The Object-Z part of the specification is not yet executable. However preliminary work [18] has shown that it is possible to give an operational semantic to Object-Z but it must be strengthened. We are also working on the automation of the methodology steps. Indeed, many steps, such as code generation and animation of specifications, can be automatised and a software tool must be developed to help the specifier in his tasks. Up to now, we have not specified all possible families of MAS. We have tried to cover the widest range from reactive MAS [25, 27] to cognitive MAS [17] and holonic MAS [40]. In order to be sure that OZS has no limitations concerning MAS features we need to experiment it on more case studies.

References

1. M. Ainsworth, A. H. Cruickshank, P. J. L. Wallis, and L. J. Groves, Viewpoint specification and Z, *Information and Software Technology* **36**(1) (1994) 43–51.
2. F. Bergenti and A. Poggi, 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, 2000.
3. F. M. T. Brazier, B. D. Keplicz, N. Jennings, and J. Treur, Desire: Modelling multi-agent systems in a compositional formal framework, *Int. J. Cooperative Information Systems* **6** (1997) 67–94.
4. R. Campero, P. Gruer, V. Hilaire, and P. Rovarini, Modeling and simulation of agent-oriented systems: an approach based on object-z and the statecharts, in C. Urban (ed.), *Agent Based Simulation*, 2000.
5. 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*, 2002.
6. L. de Moura, S. Owre, Ha. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, SAL 2, in R. Alur and D. Peled (eds.), *Computer-Aided Verification, CAV 2004*, LNAI, Vol. 3114, Springer-Verlag, pp. 496–500.
7. S. DeLoach, Multiagent systems engineering: a methodology and language for designing agent systems, in *Agent Oriented Information Systems '99*, 1999.
8. J. Derrick, H. Bowman, and M. Steen, Viewpoints and objects, in J. P. Bowen and M. G. Hinchey (eds.), *Ninth Annual Z User Workshop*, LNCS, Vol. 967, Springer-Verlag, 1995, pp. 449–468.
9. M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge, Formalisms for multi-agent systems, *Knowledge Engineering Review* **12**(3) (1997).
10. A. Drogoul and J. Zucker, Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge, 1998.
11. R. Duke, P. King, G. Rose, and G. Smith, The Object-Z specification language,

Technical report, Software Verification Research Center, Department of Computer Science, University of Queensland, Australia, 1991.

12. 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. R. Jennings (eds.), *ICMAS'98*, July 1998.
13. M. Fisher, If Z is the answer, what could the question possibly be?, in *Intelligent Agents III*, LNAI, Vol. 1193, 1997.
14. P. Gruer, V. Hilaire, A. Koukam, and S. Hayat, A methodology based on multiple views for multi-agent systems in simulation, application to the transportation domain in *13th European Simulation Symposium*, 2001.
15. P. Gruer, V. Hilaire, and A. Koukam, Approche multi-formalismes pour la spécification des systèmes multi-agents, *Organisation et Applications des SMA*, Hermès, 2002, Chapter 1.
16. P. Gruer, V. Hilaire, and A. Koukam, Heterogeneous formal specification based on object-z and state charts: semantics and verification, *J. Systems and Software* **70**(1–2) (2004) 95–105.
17. P. Gruer, V. Hilaire, A. Koukam, and K. Cetnarowicz, A formal framework for multi-agent systems analysis and design, *Expert Systems with Applications* **23**(4) (2002) 349–355.
18. P. Gruer, V. Hilaire, and A. Koukam, Verification of Object-Z Specifications by using Transition Systems, in T. S. E. Maibaum (ed.), *Fundamental Aspects of Software Engineering*, LNCS, Vol. 1783, Springer Verlag, 2000.
19. O. Gutknecht and J. Ferber, The madkit agent platform architecture, in *1st Workshop on Infrastructure for Scalable Multi-Agent Systems*, June 2000.
20. D. Harel, Statecharts: A visual formalism for complex systems, *Science of Computer Programming* **8**(3) (1987) 231–274.
21. D. Harel and E. Gery, Executable object modeling with statecharts, *IEEE Computer* **30**(7) (1997) 31–42.
22. 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 Trans. on Software Engineering* **16**(4) (1990) 403–414.
23. D. E. Herlea, C. M. Jonker, J. Treur, and N. J. E. Wijngaards, Specification of behavioural requirements within compositional multi-agent system design, LNCS, Vol. 1647, 1999, pp. 8–27.
24. V. Hilaire, A. Koukam, and P. Gruer, A mechanism for dynamic role playing, in *Agent Technologies, Infrastructures, Tools and Applications for E-Services*, LNAI, Vol. 2592, Springer Verlag, 2002.
25. V. Hilaire, T. Lissajoux, and A. Koukam, Towards an executable specification of multi-agent systems, in J. Filipe and J. Cordeiro (eds.), *Int. Conf. on Enterprise Information Systems'99*, Kluwer Academic Publishers, 1999.
26. 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 (eds.), *Engineering Societies in the Agents' World*, LNAI, Vol. 1972, Springer Verlag, 2001.
27. V. Hilaire, O. Simonin, A. Koukam, and J. Ferber, A formal framework to design and reuse agent and multiagent models, in J. Odell, P. Giorgini, and J. Müller (eds.), *Agent Oriented Software Engineering*, LNCS, Vol. 3382, Springer, 2005.
28. C. Iglesias, M. Garrijo, and J. Gonzalez, A survey of agent-oriented methodologies, in J. Müller, M. P. Singh, and A. S. Rao (eds.), *Proc. 5th Int. Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)*, LNAI, Vol. 1555,

Springer, Berlin, 1999, pp. 317–330.
29. E. A. Kendall, Role modeling for agent system analysis, design, and implementation, *IEEE Concurrency* **8**(2) (2000) 34–41.
30. E. A. Kendall, P. V. Murali Krishna, C. B. Suresh, and C. G. V. Pathak, An application framework for intelligent and mobile agents, *ACM Computing Surveys* **32**(1) (2000).
31. A. Koukam, B. Mazigh, P. Gruer, and V. Hilaire, A multiview approach to modeling and analysis of discrete event systems, *System Analysis — Modelling — Simulation* **43**(6) (2003) 721–740.
32. T. Lissajoux, V. Hilaire, A. Koukam, and A. Caminada, Genetic Algorithms as Prototyping Tools for Multi-Agent Systems: Application to the Antenna Parameter Setting Problem, in S. Albayrak and F. J. Garijo (eds.), LNAI, Vol. 1437, Springer Verlag, 1998.
33. F. Lopez, Y. Lopez, and M. Luck, Modelling norms for autonomous agents, in *Proc. Fourth Mexican Int. Conf. on Computer Science*, IEEE Computer Society Press, 2003.
34. P. Lucidarme, O. Simonin, and A. Liégeois, Implementation and evaluation of a satisfaction/altruism based architecture for multi-robot systems, in *IEEE Int. Conf. on Robotics and Automation*, 2002.
35. M. Luck and M. d'Inverno, A formal framework for agency and autonomy, in V. Lesser and L. Gasser (eds.), *Proc. First Int. Conf. on Multi-Agent Systems*, AAAI Press, 1995, pp. 254–260.
36. Z. Manna, N. Bjoerner, A. Browne, and E. Chang, *STeP: The Stanford Temporal Prover*, LNCS, Vol. 915, 1995, pp. 793–794.
37. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer, 1991.
38. J. Odell, H. Parunak, and B. Bauer, Extending uml for agents, in Y. Lesperance and E. Y. Gerd Wagner (eds.), *Information Systems Workshop at the 17th National Conf. on Artificial Intelligence*, 2000, pp. 3–17.
39. R. F. Paige, 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*, Graz, Austria, September), LNCS, Vol. 1313, Springer-Verlag, 1997, pp. 473–494.
40. S. Rodriguez, V. Hilaire, and A. Koukam, Towards a methodological framework for holonic multi-agent systems, in *Proc. ESAW'03 Workshop*, 2003, pp. 179–185.
41. O. Simonin, *Le modèle satisfaction-altruisme: coopération et résolution de conflits entre agents situés réactifs, application à la robotique*, PhD thesis, USTL, 2001.
42. O. Simonin and J. Ferber, Modeling self satisfaction and altruism to handle action selection and reactive cooperation, in *Sixth Int. Conf. on the Simulation of Adaptative Behavior FROM ANIMALS TO ANIMATS 6*, 2000, pp. 314–323.
43. G. Smith, 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*, Graz, Austria, September, 1997), Vol. 1313, Springer-Verlag, 1997, pp. 62–81.
44. G. Smith, *The Object Z Specification Language* (Kluwer Academic Publishers, 1999).
45. G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (MIT Press, Cambridge, MA, 1999).
46. M. Wooldridge and N. R. Jennings, Intelligent agents: Theory and practice, *The Knowledge Engineering Review* **10**(2) (1995) 115–152.

47. M. Wooldridge, N. R. Jennings, and D. Kinny, A methodology for agent-oriented analysis and design, in *Proc. Third Int. Conf. on Autonomous Agents* (*Agents'99*), ACM Press, Seattle, WA, 1999, pp. 69–76.
48. E. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in *3rd IEEE Int. Symp. on Requirements Engineering*, 1997, pp. 226–235.
49. P. Zave and M. Jackson, Conjunction as composition, *ACM Trans. Software Engineering and Methodology*, **2**(4) (1993) 379–411.