A Mechanism for Dynamic Role Playing

V. Hilaire* - A. Koukam* - P. Gruer*

* UTBM
Systems and Transports Laboratory
90010 Belfort Cedex
FRANCE
vincent.hilaire@utbm.fr
tel: +33 384 583 009 - fax +33 384 583 342

Abstract The work presented in this article is based upon a methodological approach for building Multi-Agent Systems specifications. The basic idea is to define such systems as a set of entities playing roles which have interactions between them. We present a mechanism for dynamic role-playing specification within a formal framework. The framework use a formalism which can express Multi-Agent Systems aspects. This formalism composes Object-Z and statecharts. The main features of this approach are : enough expressive power to obtain unbiased specifications, tools for specification analysis and refinement mechanisms allowing the refinement of a high level specification into a low level specification which can be easily implemented.

Key words: Multi-Agent Systems, formal specification, methodology

1 Introduction

Software agents and multiagent systems have become an appealing paradigm for the design of computer systems composed of autonomous cooperating software entities. This paradigm consists in new ways for analyzing, designing and implementing such systems based upon the central notions of agents, their interactions and the environment which they perceive and in which they act. There exists several software engineering approaches [22] which try to fill the gap of 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 a methodology. The notation may be formal or not, in this paper we are interested in formal notations which allow formal specification of MAS and which guide implementation phase. Indeed, formal specification approaches may authorize validation and verification of the specification. The process of validation and verification provides a support for incremental specification leading to an executable model of the system being built.
Due to their complexity, MAS have reactive and transformational features. A

formalism which : specifies easily and naturally both aspects, enables validation and verification and guides the implementation phase is to be defined yet. We have thus chosen to use a multi-formalisms approach that results of the composition of Object-Z [10] and statecharts [18]. This formalism enables the specification of reactive and transformational aspects of MAS.
A specification method is essential to manage MAS complexity by decomposition and abstraction. Some approaches use organizational concepts to model MAS [11,29,21]. The use of such primitive concepts enable to go from the requirements to detailed design and helps to decompose a MAS in terms of roles and organizations. In fact, it is a three step approach. The first step views the system as an organization 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 focus on the design of the internal architecture of agents. With these concepts we have defined a framework, named RIO which stands for Role-Interaction-Organization, composed of Object-Z classes which specify each informal concept. In doing so we have methodological guidelines formally grounded for the analysis and design of MAS. We have used the RIO framework on several problems. We have, for example, specified a MAS for solving a radiomobile network field problem [25]. We have also animated a RIO model of foot-and-mouth disease simulation [21].
One drawback of organizational based methodology is that they frequently restrict to static role-playing relationship. The purpose of this paper is to extend our approach to deal with dynamic role-playing relationship.
The idea that we suggest for dynamic role-playing relationship is inspired of viewpoint specification. Viewpoint techniques have been widely used in requirement analysis [1,7]. Indeed, viewpoint specifications consist in separating partial specifications of large and complex systems. There has been some work on combination of specifications. The problem of combining dynamically roles is similar to combining viewpoints. We have to merge several specifications and check their consistency. In this paper, due to lack of place, we deal only with combination of the Object-Z part of the specification.
The rest of the paper is organized as follows. Section 2 introduces the extended RIO framework, section 3 illustrates the dynamic role-playing relationship on an example, section 4 presents related works and eventually section 5 concludes.

2 RIO Framework

This section presents the RIO framework concepts and their specification in Object-Z. Indeed, the RIO framework is composed of two parts. The first part is semi-formal and allows the representation of MAS as systems of agents playing interacting roles in several organizations by the way of graphic diagrams. The second part gives formal meaning and notations for the concepts depicted by the diagrams. In this paper we insist mainly on the latter part.
The first subsection introduces the organizational level which roughly correspond to the analysis level and the second subsection introduces agentification level

which belongs to the design level.
Each concept is specified by an Object-Z class which takes the form of a graphic named box. The specification unit of Object-Z is the schema represented by inner boxes in a class. Each class owns several schemas which are for example state schema and operation schemas. A schema is divided in two parts by a short line. The upper part consists of declaration in a set theory fashion. The part below the short line consists of constraints, in first order predicate logic.

2.1 Organizational level

We base our work on the RIO framework defined in [21]. This framework had three main concepts which was : role, interaction and organization. We have modified these concepts to take into account dynamic role-playing aspects. In fact we have modified relationships between classes of the meta-model. We have generalized attributes, stimulus and actions in a generic entity named *Actor*. *Role* and *Agent* classes inherit from *Actor*. We have also introduced a specific class *RoleContainer* which represents set of roles played simultaneously. The figure 1 describes with UML notation the relationships between these classes. This meta-model is similar to the one used by Yu [30] for his i^* framework.

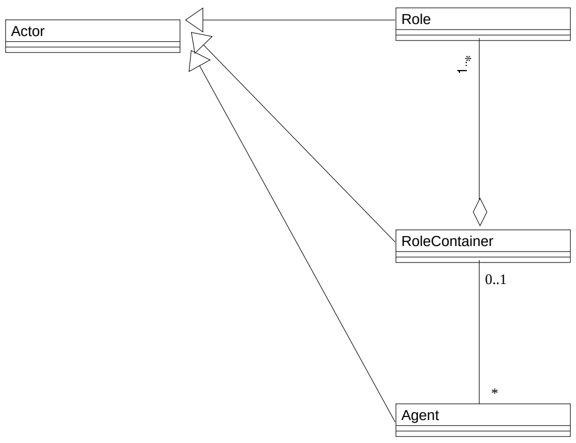

Figure1. RIO meta-model

An actor is an abstraction of a behavior or a status. We have chosen to specify it by the *Actor* class. This class represents characteristic set of attributes whose elements are of [*Attribute*] type. These elements belong to the *attributes* set. An actor is also defined by stimulus it can react to and actions it can execute. They are respectively specified by *stimulus* set and *actions* set. The [*Attribute*], [*Event*] and [*Action*] types are defined as given types which are not defined further.

Actor

$attributes : \mathbb{P}\, Attribute$
$stimulus : \mathbb{P}\, Event$
$actions : \mathbb{P}\, Action$

The *Role* class inherits from *Actor*, This is stated by the first line of the class. The reactive aspect of a role is specified by the sub-schema *behavior* which includes a statechart. It is to say that *behavior* specifies the different states of the role and transitions among these states. It is in the *behavior* sub-schema that we include statecharts. The *obtainConditions* and *leaveConditions* attributes specify conditions requested to obtain and leave the role. These conditions require specific capabilities or features to be present in order to play or leave the role. Stimulus which trigger a reaction in the role behavior must appear in one transition at least. The action belonging to the statechart transitions must belong to the *actions* set. The constraints specified in the *Role* class use the heterogeneous basis concepts in order to ensure coherence between Object-Z and statechart parts.

Role

Actor

behavior

$obtainConditions, leaveConditions : Condition$

$\forall\, s \in stimulus, \exists\, e \in behavior.\rho \bullet$
$\quad (\exists\, t \in e.transitions \bullet t.label.event = s)$
$\forall\, e \in behavior.\rho \bullet$
$\quad (\forall\, t \in e.transitions \bullet t.label.action \subseteq actions)$

An interaction is specified by a couple of role which are the origin and the destination of the interaction. The role *orig* and *dest* interacts by the way of operations op_1 and op_2. These operations are combined by the $\parallel$ operator which equates output of op_1 and input of op_2. In order to extend interaction to take into account more than two roles or more complex interactions involving plan exchange one has to inherit from *Interaction*.

Interaction

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

Organization

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

An organization is specified by a set of roles and their interactions. Interactions happen between roles of the concerned organization. It is to say that for each interaction of the *interactions* set the roles of the interaction must belong to *roles* set of the organization. Moreover, each role must be part of at least one interaction.
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 *RoleContainer* the link between entities which play roles and roles specification is precised. Indeed, three functions f_1, f_2 et f_3 map respectively attributes, stimulus and actions of roles and 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' is said to refine an object o is represented as $o \preccurlyeq o'$.
This mechanism allows the specification of several, possibly overlapping, roles to be merged in one entity. There are three different combination cases. We take the attributes example to illustrate them. The first case is when attributes are completely distincts 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 of the same type. 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 but these domains aren't of the same type. The function must map the two attributes to a subset of the cartesian product of the definition domains.

RoleContainer

$$\begin{array}{l}
Actor
\end{array}$$

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

$\exists f_1$: *Attribute* $\mapsto$ *Attribute* $\bullet$
    $\forall r \in playing \bullet$
    $\forall a \in r.attributes \, \exists a' \in attributes \bullet a' = f_1(a) \wedge a \preccurlyeq a'$
$\exists f_2$: *Action* $\mapsto$ *Action* $\bullet$
    $\forall r \in playing \bullet$
    $\forall a \in r.actions \, \exists a' \in actions \bullet a' = f_2(a) \wedge a \preccurlyeq a'$
$\exists f_3$: *Stimulus* $\mapsto$ *Stimulus* $\bullet$
    $\forall r \in playing \bullet$
    $\forall a \in r.stimulus \, \exists a' \in actions \bullet a' = f_3(a) \wedge a \preccurlyeq a'$

2.2 Agentification level

The *Agent* class inherits from *Actor*. 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 moment in time. These roles define the agent status and behavior in all context it may intervene. The position of an agent may change during it lifetime. This mechanism allows dynamic role-playing. The *addRole* (resp *leaveRole*) operation add (resp substract) a role to an agent. The preconditions of this operation impose that *obtainConditions* (resp *leaveConditions*) must be true. These operations modify the functions which map attributes, stimulus and actions from roles to agent. These functions defined statically in the *RoleContainer* class are modified dynamically whenever an agent change one of his role.
An agent A is also defined by an *acquaintances* set. This set represents the other agents which are currently interacting with A.

Agent
Actor

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

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

$$
\begin{array}{l}
\textit{addRole} \\
\Delta(\textit{position}) \\
r? : \textit{Role} \\
\hline
\textit{position.playing}' = \textit{position.playing} \cup r? \\
r.\textit{obtainConditions} \\
\forall\, a \in r.\textit{attributes} \bullet (\textit{position}.f_1' = \textit{position}.f_1 \oplus \{r.a, a'\}) \\
\qquad \wedge\ ((a' \in \textit{attributes} \wedge a \preccurlyeq a') \\
\qquad\quad \vee\ (\textit{attributes}' = \textit{attributes} \cup \{a'\} \wedge a \preccurlyeq a')) \\
\forall\, a \in r.\textit{actions} \bullet (\textit{position}.f_2' = \textit{position}.f_1 \oplus \{r.a, a'\}) \\
\qquad \wedge\ ((a' \in \textit{actions} \wedge a \preccurlyeq a') \\
\qquad\quad \vee\ (\textit{actions}' = \textit{actions} \cup \{a'\} \wedge a \preccurlyeq a')) \\
\forall\, a \in r.\textit{stimulus} \bullet (\textit{position}.f_3' = \textit{position}.f_3 \oplus \{r.a, a'\}) \\
\qquad \wedge\ ((a' \in \textit{stimulus} \wedge a \preccurlyeq a') \\
\qquad\quad \vee\ (\textit{stimulus}' = \textit{stimulus} \cup \{a'\} \wedge a \preccurlyeq a'))
\end{array}
$$

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

In this context, an agent is only specified as an active communicative entity which plays roles [11]. In fact agents instantiate an organization (roles and interactions) when they exhibit behaviors defined by the organization's roles and when they interact following the organization interactions. The maion reason for this choice is that one can study agent behaviours and agent architecture separately. Indeed, the different roles an agent plays define it behaviour. The architectures used by agents may be different for the same behaviour and so it is sound to study it apart from the core agent behaviour. In [14] we have specified a specific agent architecture by extending the RIO framework classes.

3 Example

In this section we present a simplified example of a virtual university which allows office automation. The aim of this example is to illustrate the dynamic role playing mechanism. As the refinement mechanism for the statechart part is not presented in this paper we present only the Object-Z specification of the roles.
The first role we specify is the *Researcher* role. It is described by a set of *publications*, a set of *contracts*, the *diplomas* it owns and eventually the *institute* it works in.

The team leader is a researcher which leads a team composed of researchers.

$$\begin{array}{l}
\underline{\ Researcher\ } \\
Role \\
\hline
publications : \mathbb{P}\ Publication \\
contracts : \mathbb{P}\ Contract \\
diplomas : \mathbb{P}\{DEA, PhD, Hdr\} \\
institute : \mathbb{P}\ Laboratory \\
\hline
\end{array}$$

$$\begin{array}{l}
\underline{\ TeamLeader\ } \\
Researcher \\
\hline
members : \mathbb{P}\ Researcher \\
name : \mathbb{P}\ Team \\
\hline
(Hdr \in diplomas) \in obtainConditions \\
\hline
\end{array}$$

A teacher may be involved in different courses described by the set *courses*. He is responsible of courses among the one he is involved in. He owns *diplomas* and he is attached to a teaching department.

$$\begin{array}{l}
\underline{\ Teacher\ } \\
\hline
courses : \mathbb{P}\ Courses \\
responsible : \mathbb{P}\ Courses \\
diplomas : \mathbb{P}\{ing, DEA, PhD, Hdr\} \\
institute : \mathbb{P}\ Department \\
\hline
responsible \subseteq courses \\
\hline
\end{array}$$

The *TeacherResearcher* class specifies a *roleContainer* which plays the roles *Teacher* and *Researcher*. The attributes *diplomas* and institute *institute* are refined. In the first case the definition domain is extended by union of the two definition domain. In the second case the definition domain is the cartesian product of the definition domains of role attributes. All other attributes are mapped without change.

Eventually, the *TRVirtualUniversityAgent* specify an agent which can play the roles defined above. Each agent of this type owns several categories which enable, by the way of the *access* function, to determine which role he can play.

$$
\begin{array}{l}
\textit{TeacherResearcher} \\
\hline
\textit{RoleContainer} \\
\hline
\textit{diplomas} : \mathbb{P}(\{\textit{DEA}, \textit{PhD}, \textit{Hdr}\} \\
\qquad\qquad \cup \{\textit{ing}, \textit{DEA}, \textit{PhD}, \textit{Hdr}\}) \\
\textit{institute} : \textit{Department} \times \textit{Laboratory} \\
\hline
\textit{playing} = \{\textit{Teacher}, \textit{Researcher}\} \\
f_1(\textit{Teacher.diplomas}) \\
\quad = f_1(\textit{Researcher.diplomas}) \\
\quad = \textit{diplomas} \\
f_1(\textit{Teacher.institute}) = 1 \upharpoonright \textit{institute} \\
f_1(\textit{Researcher.institute}) = 2 \upharpoonright \textit{institute}
\end{array}
$$

$$
\begin{array}{l}
\textit{TRVirtualUniversityAgent} \\
\hline
\textit{Agent} \\
\hline
\textit{categories} : \mathbb{P}\,\textit{Category} \\
\textit{access} : \textit{Category} \mapsto \mathbb{P}\,\textit{Role} \\
\hline
\textit{INIT} \\
\hline
\forall\, r \in \textit{access}(\textit{categories}) \bullet \textit{addRole}(r) \\
\hline
\{\textit{Teacher}, \textit{Researcher}\} \in \textit{position.playing} \\
\hline
\textit{addCategories} \\
\hline
\Delta(\textit{categories}) \\
\textit{new?} : \textit{Category} \\
\hline
\textit{categories}' = \textit{categories} \cup \textit{new?} \\
\forall\, r \in \textit{access}(\textit{new?}) \bullet \textit{addRole}(r)
\end{array}
$$

4 Related works

This section describes approaches for Agent-Oriented Software Engineering. We have divided these approaches in two parts : the semi-formal approaches and the formal ones.

4.1 Semi-formal approaches

The i^* framework of Yu [30] is based upon a requirements analysis by mean of goals, dependencies, roles, actors, positions and agents. These notions are similar to the one we use. They are graphically presented on a same schema. It is 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 better.
Kendall [23] suggests the use of extended Object Oriented methodologies like design patterns and CRC. CRC are extended to Role Responsabilities and Collaborators. In [24] a seven layered architectural pattern for agent is presented. The seven layers are: mobility, translation, collaboration, actions, reasoning, beliefs and sensory. This architecture is dedicated for mobile agents.
Cassiopeia [5] and Andromeda [9] use role notion and propose a step by step methodology in order to design MAS. Andromeda is an enrichment of Cassiopeia. Indeed, it deals with machine learning techniques for MAS. These methodologies are oriented towards the design of reactive MAS.
The MaSE methodology [6] insist on the necessity of software tools for software engineering, specifically code generation tools. This methodology has seven steps. These steps are structured in sequence. It begins with the identification of overall goals and their structuration. From goals with use case diagrams 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 steps

consist in defining precisely : high level protocols, agent architectures and deployment diagram. For each step a different diagram is introduced. Moreover, MaSE methodology suffers of limited one-to-one agent interactions.
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 specific MAS aspects there are conceptual ontology description, MAS architecture, interaction protocols and agent functionalities.
In [27], authors present an approach extending UML for represent agent relative notions. In particular, authors insist on role concept and suggest the use of modified sequence diagrams to deal with roles.
The problem with the two latter notations is the UML starting point. Indeed, using an object oriented notation in order to describe MAS lead specifiers to use object oriented concepts. We think that a MAS methodology must insist on agent oriented concepts first.

4.2 Formal approaches

Formal approaches for MAS specification are numerous but they are often abstract and unrelated to concrete computational models [8]. Temporal modal logic, for example, have been widely used [28]. 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 [26]. One advantage of using such approaches is that they are more widely used and expertise concerning notations is greater then newer ones and there are tools which help the specification process.
For example, the approach proposed in [19] is based upon the refinement of informal requirement specifications to semi-formal and then formal specifications. The system structuration is based on a hierarchy of components [3]. These components are defined in term of input/ouput and temporal constraints. With this approach it seems difficult to refine down specifications to an implementation language. Moreover, the verification technique is limited to model checking.
Luck and d'Inverno [26] propose a formal framework which use 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. 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, the Luck and d'Inverno framework does not allow to specify temporal and reactive properties of MAS [12]. In our framework these aspects are specified by temporal invariants and statecharts.
Wooldridge, Jennings and Kinny [29] propose the Gaia methodology for agent oriented analysis and design. This methodology is composed of two abstraction levels : agent level and structural organizational level. The role concept exist in Gaia however the relationship between agent and role is static.

5 Conclusion

In this paper, we have presented a formal specification approach for MAS based upon an organizational model. The organizational 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. This model is well suited for describing complex interactions which are among MAS main features. The language used by the specification framework can describe reactive and functional aspects. It is structured as a classes hierarchy so one can inherit from these classes to produce its own specification. As an example we have specified the Virtual University Helper Agent. Several MAS have already been specified with the RIO framework with or without using a specific agent architecture [20,21,4,14]. These MAS have been applied to complex problems like radio-mobile network field [25], foot-and-mouth disease simulation [21] or office automation [13].
The used specification language allows prototyping of specification [21]. Prototyping is not the only means of analysis, indeed, in another work [15], we have introduced a formal verification approach. 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 [17] which is based upon an organizational model [11].
Despite the encouraging results already achieved, we are aware that our approach still has some limitations. Indeed, it doesn't tackle all problems raised by a MAS development methodology. Among issues remaining for future work our organizational model needs more work to do ahead. The Object-Z part of the specification is not yet executable. However a preliminary work [16] 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, like code generation or specifications animation, can be automatized and some software tool must be developed to help the specifier in his task.

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):43–51, 1994.
2. Federico Bergenti and Agostino Poggi. Exploiting uml in the design of multi-agent systems. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2000.
3. F.M.T. Brazier, B. Dunin Keplicz, N. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.
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 Christoph Urban, editor, *Agent Based Simulation*, 2000.

5. Anne Collinot, Alexis Drogoul, and Philippe Benhamou. Agent oriented design of a soccer robot team. In Victor Lesser, editor, *ICMAS*. Springer Verlag, 1995.
6. Scoot DeLoach. Multiagent systems engineering: a methodology and language for designing agent systems. In *Agent Oriented Information Systems '99*, 1999.
7. J. Derrick, H. Bowman, and M. Steen. Viewpoints and objects. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, volume 967 of *Lecture Notes in Computer Science*, pages 449–468, Limerick, September 1995. Springer-Verlag.
8. 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.
9. A. Drogoul and J. Zucker. Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge, 1998.
10. Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Departement of Computer Science, University of Queensland, AUSTRALIA, 1991.
11. Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Y. Demazeau, E. Durfee, and N.R. Jennings, editors, *ICMAS'98*, july 1998.
12. 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.
13. P. Gruer, V. Hilaire, and Abder Koukam. approche multi-formalismes pour la spcification des systmes multi-agents. Technical report, UTBM-SeT, 2001. to appear in "Système Multi-Agents : des Théories Organisationnelles aux Applications Industrielles" Hermés.
14. P. Gruer, V. Hilaire, Abder Koukam, and Krzysztof Cetnarowicz. A formal framework for multi-agent systems analysis and design. *Expert Systems with Applications*, 2002.
15. Pablo Gruer, Vincent Hilaire, and Abder Koukam. an Approach to the Verification of Multi-Agent Systems. In *International Conference on Multi Agent Systems*. IEEE Computer Society Press, 2000.
16. Pablo Gruer, Vincent Hilaire, and Abder Koukam. Verification of Object-Z Specifications by using Transition Systems. In T. S. E. Maibaum, editor, *Fundamental Aspects of Software Engineering*, number 1783 in Lecture Notes in Computer Science. Springer Verlag, 2000.
17. Olivier Gutknecht and Jacques Ferber. The madkit agent platform architecture. In *1st Workshop on Infrastructure for Scalable Multi-Agent Systems*, june 2000.
18. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
19. D. E. Herlea, C. M. Jonker, J. Treur, and N. J. E. Wijngaards. Specification of behavioural requirements within compositional multi-agent system design. *Lecture Notes in Computer Science*, 1647:8–27, 1999.
20. V. Hilaire, T. Lissajoux, and A. Koukam. Towards an executable specification of Multi-Agent Systems. In Joaquim Filipe and José Cordeiro, editors, *International Conference on Enterprise Information Systems'99*. Kluwer Academic Publisher, 1999.
21. Vincent Hilaire, Abder Koukam, Pablo Gruer, and Jean-Pierre Müller. Formal specification and prototyping of multi-agent systems. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents'*

World, number 1972 in Lecture Notes in Artificial Intelligence. Springer Verlag, 2000.

22. Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A survey of agent-oriented methodologies. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 317–330, Berlin, July 04–07 1999. Springer.
23. Elizabeth A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, 2000.
24. Elizabeth A. Kendall, P. V. Murali Krishna, C. B. Suresh, and Chira G. V. Pathak. An application framework for intelligent and mobile agents. *ACM Computing Surveys*, 32(1), 2000.
25. 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, editors, *Lecture Notes in Artificial Intelligence*, number 1437 in LNAI. Springer Verlag, 1998.
26. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press, 1995.
27. J. Odell, H. Parunak, and B. Bauer. Extending uml for agents. In Yves Lesperance E. Y. Gerd Wagner, editor, *Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, 2000.
28. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
29. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.
30. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *3rd IEEE Int. Symp. on Requirements Engineering*, pages 226–235, 1997.