Towards a Formal Methodology for Designing Multi-agent Applications

Amira Regayeg[1], Ahmed Hadj Kacem[1], and Mohamed Jmaiel[2]

[1] Faculté des Sciences Économiques et de Gestion de Sfax,
B.P. 1088, 3018 Sfax, Tunisia
{Amira.Regayeg, Ahmed}@fsegs.rnu.tn
[2] École Nationale d'Ingénieurs de Sfax,
B.P.W., 3038 Sfax, Tunisia
Mohamed.Jmaiel@enis.rnu.tn

Abstract. This paper has two purposes. First, it defines a formal language for specifying multi-agent systems. This language is expressive enough to cover individual agent aspects (knowledge, goals, roles, ...) as well as collective aspects of in terms of coordination protocols, organization structure and planning activities. Second, it provides a formal design methodology based on stepwise refinements allowing to develop a design specification starting from an abstract requirements one.

1 Introduction

Several researches tried to face the problem of developing software systems using the agent concept. The majority of the suggested approaches are extensions of either object oriented methodologies like, for example, AOAD [20] and MaSE [2], or knowledge based methodologies, such as CoMoMAS [5]. The major problem of these extensions is that they do not provide appropriate tools to model the specific features of agents, such as mental states and social behaviours. Other attempts, like Gaia [19], SODA [13] and Prometheus [14], sought to focus on the social aspect of an agent group or an organization. These approaches, given that they are based on semi-formal notations, they do not enable formal reasoning about specifications. Approaches which make use of formal methods, like for example, the framework suggested by Luck and d'Inverno using the Z language [10], Concurrent-Metatem [4] based on temporal logic, and SLABS (Formal Specification Language for Systems Agent-Based)[23], concentrate their effort only to the specification phase. Recent work, such as ADK [21], although it based on a formal approach and it covers the specification, design and implementation phases, it completely ignores formal reasoning and particularly the verification phases.

In order to overcome these insufficiencies and to master the inherent complexity of multi-agent development, we suggest a formal design approach of multi-agent systems based on stepwise refinements. It is recognized that the formal approach represents an obvious but attractive challenge for Agent Oriented Software Engineering [22]. Here, we try to take advantage of the potential

T. Eymann et al. (Eds.): MATES 2005, LNAI 3550, pp. 153–164, 2005.

of the formal methods in building reliable software. Doing so, we define, on the one hand, a formal specification language which integrates the linear temporal logic in the Z language, and on the other hand, a set of methodological principles and hints which help the user to build in a systematic and incremental way intra and inter agent aspects. This integration is motivated by recent tendencies [15] which are directed towards (1) addressing aspects separated using suitable languages and tools, and (2) integrating various approaches in a unified development process[1]. Indeed, the Z language possesses all ingredients needed to handle static and functional aspects of agents (i.e., the mental state and associated treatments), whereas temporal logic is considered as one of the most eminent formalisms for specifying reactive systems [11]. In addition, approaches based on stepwise refinements [17] proved their impact in developing several software applications [8,3].

In order to provide a formal interpretation for our temporal operators we suggest an operational semantics for multi-agent applications in terms of sequences of system states. The definition of this temporal model within the Z notation enables us to make use of tools supporting pure Z notation, such as Z/EVES [12]. These tools allowed us to perform syntax, type, and domain checking of our specifications and to reason about them by proving desired properties. Our design process is composed of a number of refinement steps where each one provides some methodological guidelines which help the developer to take the suitable design decision as well as rules making it possible to ensure that a refined specification satisfies the initial one.

This paper is organized as follows. Section 2 defines the specification language and its semantics. Then, in section 3 we explain our specification and design approach. Finally, we conclude with drawing some perspectives.

2 The Specification Language

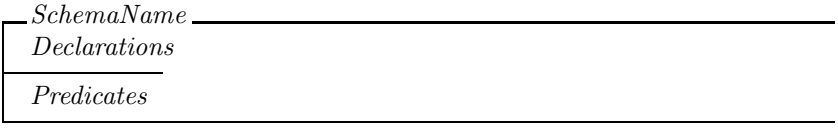
We consider a multi-agent application as a collection of components which evolve in a continuously changing environment containing active agents and passive objects. Accordingly, the specification of a multi-agent application includes descriptions of the environment, the behaviour of individual agents (intra-agent), and the communication primitives as well as the interaction protocols (inter-agent). In addition, we may add to the collective part a description of the organizational structures and planning activities.

For the specification of multi-agent applications, we use an integration of temporal logic in Z schemas as described in our previous work [15]. This integration will enable us to cover all the above mentioned aspects in a unified framework. Indeed, the Z notation allows to describe all components (passive and active) in terms of attributes and related properties. The temporal logic will enrich this description with social behaviour and interaction properties.

2.1 The Z Notation

The Z notation, as presented in [18], is a model oriented formal specification language which is based on set theory and first order predicate logic. This language

is used to describe an application in terms of states and operations on them. In order to structure specifications and to compose them Z uses a *schema language*. The latter enables to collect objects, to encapsulate them, and naming them for reuse. A schema consists of two parts: a declaration part and a predicate part constraining the values of the declared variables. A Z schema has the following form:



2.2 The Temporal Logic

The linear temporal logic, as presented by Manna and Pnueli [11], is suitable for the specification and the verification of concurrent and interactive systems. Actually, there is a variety of temporal operators that can be used to express agents behavioural properties. However, all these operators can be defined in terms of two basic operators. In this paper, we make use only of the necessary operators for development of our multi-agent applications. In the following, we briefly present these operators with an intuitive explanation. Let P be a logical or a temporal formula:

$\triangledown P$ P holds "now"[1] ($\triangledown$ may be omitted);
$\Box P$ "*always*" P, i.e. P holds for the present and for all future points in time;
$\Diamond P$ "*eventually*" P, i.e. P holds at some present or future point in time;
$\bigcirc P$ "*nexttime*" P, i.e. P holds at the next point in time.

In order to integrate these temporal operators in the framework of the Z language, we give the following definition of temporal formulas according to the syntax of Z. We distinguish atomic predicate formulas (*formula*), which are closely related to the application to specify, and temporal formulas (*Tempformula*) which connect predicate formulas with temporal operators.

$$\mathit{Tempformula} ::= \langle\!\langle \mathit{formula} \rangle\!\rangle \mid \bigcirc \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \mid \Box \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \mid \Diamond \langle\!\langle \mathit{Tempformula} \rangle\!\rangle$$

We will show later that these operators are sufficient to express interesting properties of multi-agent applications.

2.3 The Semantics of Temporal Formulas

In this section we provide evaluation functions defining the semantics of our temporal logic. This step is very significant since it enables us to translate temporal formulae into the pure Z notation. Thus, it becomes easy to exploit the

[1] We explain the operators while being based on a concept of "time", but really the fundamental notion is the one of causality.

automatic verification tools, such as Z/EVES or Isabelle, which accept merely the standard syntax of Z.

First, we present the underlying time model. The basic unit of this time model is the agent state. Let $[State]$ be the set of possible agent states. A system state ($SysState$) is defined as the union of the states of the agents belonging to this system:

$$\begin{array}{|l} SysState \\ \hline SysState : \mathbb{F}\ State \\ \hline \end{array}$$

A time model ($Model$) is defined as an axiomatic function that associates to each point of time the corresponding system state, where the time is specified as the set of natural numbers ($Time == \{x : \mathbb{N}\}$):

$$Model == Time \rightarrow SysState$$

Second, we provide an axiomatic function (E) which evaluates a temporal formula in a given model at a given point of time:

$$\begin{array}{|l} E : Temporalformula \times Model \times Time \rightarrow bool \\ \hline \forall f : Formula;\ m : Model;\ t : Time \\ \quad \bullet\ E((\triangledown f), m, t) = T \Leftrightarrow AtomEval(f, m\ t) = T \\ \forall f : Tempformula;\ m : Model;\ t : Time \bullet \\ \quad E((\Diamond f), m, t) = T \Leftrightarrow (\exists\, t1 : Time \mid t1 \geq t \bullet E(f, m, t1) = T) \\ \forall f : Tempformula;\ m : Model;\ t : Time \bullet \\ \quad E((\Box f), m, t) = T \Leftrightarrow (\forall\, t1 : Time \mid t1 \geq t \bullet E(f, m, t1) = T) \\ \forall f : Tempformula;\ m : Model;\ t : Time \bullet \\ \quad E((\bigcirc f), m, t) = T \Leftrightarrow (E(f, m, (t + 1)) = T) \end{array}$$

Next, we generalize the function E by making abstraction of the time parameter. Hence, the function (Eva)below interprets temporal formulas with respect to a given model:

$$\begin{array}{|l} Eva : Tempformula \times Model \rightarrow bool \\ \hline \forall f : Tempformula;\ m : Model \\ \quad \bullet\ Eva(f, m) = T \Leftrightarrow (\forall\, t : Time \bullet (f, m, t) \in \mathrm{dom}\, E \wedge E(f, m, t) = T) \end{array}$$

We can more generalize the function of evaluation by making abstraction of the model. The following function defines a general interpretation of the temporal operators:

$$\begin{array}{|l} Eval : Tempformula \rightarrow bool \\ \hline \forall f : Formula \\ \quad \bullet\ Eval(f) = T \Leftrightarrow (\forall\, m : Model \bullet (f, m) \in \mathrm{dom}\, Eva \wedge Eva(f, m) = T) \end{array}$$

Finally, in order to use the temporal operators in their usual notations (i.e. $\Box$ for always) in the Z schemata it is necessary to introduce them as axiomatic

functions defined with the interpretation function *Eval*. Thus, we could establish a logical equivalence between a temporal operator and the corresponding predicate specified in the above function *Eval*. This equivalence is described for the $\Box$ operator as follows:

$$\Box f \Leftrightarrow Eval(\Box f) = T$$

The other temporal operators are introduced with similar equivalences.

3 Formal Design Approach

In order to be useful, a formalism or a set of tools have to be supported with a design approach. This approach should provide some principles that help and guide the design process. In this section, some of those principles are clarified. Indeed, our approach is based on two principal phases. The first one is a specification phase in which we describe, in an abstract way, the user requirements. The second one is a design phase based on a succession of refinements in terms of collective behaviours (inter-agents) as well as individual behaviours (intra-agent). The verification that the developed design specification satisfies the requirements one is considered as essential tasks which is progressively performed during the refinement steps.

3.1 Specification Phase

In this first phase, we specify the requirements which correspond, in the context of multi-agent, to a common objective to be achieved by the agents. In our approach, this stage provides also a description of the environment in which the agents evolve and which includes, generally, the working space, the passive objects, and the active entities representing the agents to be deployed.

1. *Specification of the active entities:* The description of an active entity (agent) consists in presenting, in terms of temporal formulae, its static and dynamic properties. This description is given by a Z schema of the following form:

   Entity

   $atr_1 : Type_1,\ atr_2 : Type_2\ \ \ldots atr_m : Type_m$

   $Spr_1, \ldots, Spr_n,$
   $Cpr_1, \ldots, Cpr_{n'}$

   Where atr_i corresponds to an attribute, Spr_i represents a static property and Cpr_i represents a behavioural property.
2. *Specification of the system*: The system includes active entities (agents) and passive entities belonging to the working space. This specification is given by a set of formulas relating passive entities with active ones. Generally, this leads to a Z schema of the form:

System

$obj_1 : TypeObject_1, \ldots, obj_k : TypeObject_k$

Entities : **set of** *Entity*

$Pr_1, Pr_2, \ldots, Pr_l$

Where obj_i corresponds to a passive entity, *Entities* represents a set of entities, and Pr_i represents a temporal formula.

3. *Requirements Specification*: This specification describes what we require from the system to develop. In the context of multi-agent application, this corresponds to a set of temporal formulas specifying *the Common Objective* (*CO*) in terms of the desired future state.
   According to the Z approach, such a specification is well expressed with a specialization of the *System* schema:

ReqSpec

System

$CO_1, CO_2, \ldots, CO_n$

Where CO_i represents a temporal formula.

3.2 Design Phase

The basic idea consists in performing a sequence of refinements made by specializations of Z schemas for data refinement, and derivation of temporal formulas for behavioural refinement. The refinement steps are supported by a set of rules which help the transitions between specifications. The refinements are carried out at two complementary levels. The first, is the collective level which will be augmented by properties referring, primarily, to collective aspects (inter-agent) characterizing, in particular, organization and communication structures. The second level rather stresses the individual aspects (intra-agent) by extending the specifications of the active entities provided in the first phase.

Collective Level. Designing the collective aspects of a multi-agent application is made, in our approach, within three aspects : (1) cooperation strategy, (2) organization structure and (3) interaction protocol.

1. Cooperation Strategy
   Step 1: Cooperation Strategy definition
   Starting from *CO* and using a hierarchical representation, we iterate the composition, based on logical connections, of the requirements specification (*ReqSpec* schema) until we obtain elementary temporal formulas in a way that each one corresponds to a subgoal.
   That is, for each formula CO_i which composed *CO*, we generate by decomposition and transformation a finite set of temporal formulas connected by the logical connectors $(\wedge, \vee)$. The conjonction $\wedge$ indicates a sequence

of subgoals whereas the disjonction $\vee$ presents different subgoals for the achievement of the goal.
Finally, we obtain a list of scenarios for CO_i where each one is described by a sequence of elementary temporal formulas. These formulas represent the different local goals: $\{bl_{i1}, \ldots, bl_{in}\}$.
At the end of this step, we generate a graph and/or which summaries the various decomposition levels.

Each scenario leads to the following specification which corresponds to a refinement of the *ReqSpec* specification:

Implementation$_0$

System

$bl_{i1}, bl_{i2}, \ldots, bl_{in}$

This refinement step requires the proof of the following theorem for each scenario of CO_i present in the requirements specification:

theorem CoopScenario
$bl_{i1}, bl_{i2}, \ldots, bl_{in} \vdash CO_i$

2. Organization Structure
The organization structure implicitly defines a control strategy to be respected by these entities. It is, generally, defined in terms of temporal formulas referring to several entities at the same time.
Here, we invent a suitable organization structure for the system to be developed. We, first, identify the necessary roles, then we assign a role for each active entity belonging to the system.
(a) *Social Level*
   – *Step 2: Identification of roles*
   In our approach, an agent role is formally represented by a set of temporal formulas corresponding to local goals.
   Hence, starting form the above defined local goals, we can regroup them according to predicates (actions) present in the various formulas describing the different local goals bl_{ij}. Thus, we will have as many roles as actions describing the various local goals.

$$role_i == \{bl_{i1}, \ldots, bl_{ik}\}$$

   Where $bl_{i1}, \ldots, bl_{ik}$ present the same action.
   This step leads to a refined specification:

Implementation$_1$

Implementation$_0$
$R : \mathbb{P}_1\ Role$

$\forall\, bl : BL \bullet \exists\, r : Role \bullet bl \in r$

This regroupment must respect the following completeness theorem which guarantees that every local goal is associated to a role:

theorem completeness
$\bigcup_{i=1}^{m} role_i = \{bl_{i1}, bl_{i2}, \ldots, bl_{in}\}$

– *Step 3: Definition of organization relationships*
At this level, we refer to the model [9] where a Generic organization structure *OrgStructure* is defined by a finite set of relations between the various necessary roles [*OrgRelationship*] for the achievement of a common objective.
Our goal, in this step, is to express how starting from the set of roles defined in the previous step, we find the various possible organization relationships.
We propose, for this definition, to search the common arguments present in the predicates describing the local goals of different roles. A common argument for two or several local goals of different roles proves the existence of an organizational relation between these roles. After a succession of iterations, we obtain a set of organization relationships connecting the various roles.
This step leads to a refined specification:

Implementation$_2$
Implementation$_1$
$Rorg : \mathbb{P}_1\ OrgRelationship$

$\forall r : Role \bullet \exists rorg : OrgRelationship \mid rorg \in Rorg \bullet r \in \mathrm{dom}\ rorg$

In this context, a constraint to check is that each role must have at least a relationship to one or more other roles. This is guaranteed by the proof of the following theorem:

theorem RoleRelation
$\forall r \in Role \bullet r \in \mathrm{dom}\ Sorg$

(b) *Agent Level*
This level consists on defining the agents for which each role will be associated. Then, we instantiate the different organization relationships, defined in the previous level, in order to find the eventually organization links between these agents.
– *Step 4: The role assignation*
This step consists on defining, given a set of roles, the agents which will be charged with each one of these roles.
To find the number of the necessary agents for each role, we need to define the precedence order between the various local goals of the retained roles. This order relation is based on the different temporal operators describing the list of local goals. Thus, we can define a precedence graph for the common objective.

The basic idea consists on referring to precedence order of the different local goals in order to avoid assigning simultaneous local goals to the same agent. Also, one agent can have more than one role provided that the local goals of these various roles are not in contradiction. Thus, several scenarios can arise.
At this level, we can define a first refinement of *Entity* schema where we add the concept of *role*:

$$
\begin{array}{|l}
\textit{Entity}_1 \\
\hline
\textit{Entity} \\
\textit{roles} : \mathbb{P}_1\ \textit{Role} \\
\hline
\end{array}
$$

Then, we can refine the system specification as follows:

$$
\begin{array}{|l}
\textit{Implementation}_3 \\
\hline
\textit{Implementation}_2 \\
\hline
\forall\, r : \textit{Role} \mid r \in R \bullet \exists\, e : \textit{Entity}_1 \mid e \in \textit{entities} \bullet r \in e.\textit{roles} \\
\hline
\end{array}
$$

– *Step 5: The acquaintances definition (organisation links)*
The distribution of the roles induces an instantiation of the generic organisation structure, called concrete organisation structure *ConcreteOrg* [9].
An organization link is defined as follows:

$$
\begin{array}{|l}
\textit{OrganisationLink} \\
\hline
E : \mathbb{P}\ \textit{Entity} \\
\hline
\#E \geq 2 \\
\hline
\end{array}
$$

An organization link *OrganisationLink* makes it possible to associate two or several agents of different roles. Thus, each relation between two roles, described in the previous level (Step 2.1.b), will be expressed in term of organization links referring to a set of agents.

$$
\begin{array}{|l}
\textit{Implementation}_4 \\
\hline
\textit{Implementation}_3 \\
\textit{organisationlinks} : \mathbb{P}_1\ \textit{OrganisationLink} \\
\hline
\forall\, \textit{Or} : \textit{organisationlinks} \bullet \forall\, e : \textit{Entity} \bullet \\
\qquad e \in \textit{entities} \Leftrightarrow e \in \textit{Or}.E \\
\hline
\end{array}
$$

At this stage, the following theorem must be proofed indication that every organization link instantiates an organization relationship:

theorem Instantiation
$\forall\, \textit{OR} : \textit{OrgRelationship} \bullet r : \mathbb{P}_1\ \textit{Role} \mid r \in \operatorname{dom} \textit{OR} \bullet$
$\exists\, \textit{OL} : \textit{OrganisationLink} \bullet \textit{OL}.E.R \in r$

3. Interaction Protocol:
   Step 6: Interaction Protocol definitions
   Having his role, each entity have some communication acts whose will be achieved ($CommAct_i$).
   Thus we can describe the $Implementation_i$ schema:

$$\begin{array}{|l} \textit{Implementation}_i \\ \hline \textit{Implementation}_{i-1} \\ \hline \textit{CommAct}_{i1},\ \textit{CommAct}_{i2},\ \ldots,\ \textit{CommAct}_{in} \\ \hline \end{array}$$

   Each formula ($CommAct_i$) is a temporal formula that describes a communication act.
   These formulas are found by deriving each $role_i$ formula using the *Achiev* function which associate for each *role* the necessary communication acts *commacts*:

theorem RoleAchiev
$\forall\, role : Role \bullet \exists\, commacts : \mathbb{P}_1\, Action \bullet$
$Achiev(role) = commacts$

Individual Level. The specification of the individual futures is generated by the definition of the different entity capabilities allowing the realization of the actions defined in the collective level as well as cooperative, organizational and interactive actions.
Step 7: Individual Capabilities definitions
In order to execute each communication act, an entity must be equipped by some capabilities as well as send and receive capabilities.
Also, due to the execution of each communication act, an entity have some internal actions whose must be executed as well as the knowledge updates.
We denote by $Behav_i$ these different actions describing the behaviour of each entity.
We obtain the specifications describing the individual properties of each agent which will be regarded as an entity to implement separately.
For each entity, we can found a refined schema as:

$$\begin{array}{|l} \textit{EntityImpl} \\ \hline \textit{Entity}1 \\ \hline \textit{Behav}_1,\ \textit{Behav}_2,\ \ldots,\ \textit{Behav}_l \\ \hline \end{array}$$

Each $Behav_i$ can be derived from one or more communication acts using the *Execute* function which associate to a set of communication acts a behavioural action describing the internal updates due to the execution of these acts :

theorem BehavDerivation
$\exists\, Commacts : \mathbb{P}_1\, CommAct \bullet \exists\, BehavAct : Action \bullet$
$Execute(Commacts) = BehavAct$

The design phase leads to a detailed specification of the environment and detailed behaviours of the active entities. The refinement specification corresponds to the schema for the system (*System*) extended with the union of the properties added at both collective and individual levels.

4 Conclusion

In this paper, we proposed a formal approach for specifying and verifying multi-agent applications. Our main contribution consists in defining a methodology that permits to develop, step by step, in an incremental way, a design from an abstract specification. Some case studies are under realization (e.g. the conflicts control in the air-traffic). The introduction of a temporal model for multi-agent applications in the Z framework enabled us to exploit a Z supporting tool (Z/EVES) for syntax and type checking as well as theorem proving.

However, it is necessary to point out that these first results, even original and promising, constitute a modest contribution to the definition of a formal methodology for the design process of multi-agent applications.

Finally, some future works deserve to be undertaken. Indeed, each proposed step should be supported by verification tools. Also, we intend to provide tools which help to generate more concrete specifications, using process algebra, CSP [6] for example, instead of temporal logic. These more concrete specifications can be implemented using the system SPIN [7]. This latter enables us (1) to simulate the system behaviour, and (2) to verify the desired temporal properties.

References

1. D. Bjorner. New results and trends in formal techniques for the development of software for transportation systems. In G. Tarnai and E. Schnieder, editors, *Proceedings of the FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*, pages 69–76, Braunschweig, Germany, 1999.
2. S. Deloach and M. Wood. Analysis and design using mase and agenttool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference MAICS 2001*, Miami University, Oxford, Ohio, 2001.
3. F. Erasmy and E. Sekerinski. Stepwise refinement of control software-a case study using raise. In M. Naftalin, T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, pages 547–566. Springer, Berlin, Heidelberg, 1994.
4. M. Fisher. A survey of concurrent METATEM – the language and its applications. In *Proc. 1st InternationalConf. Temporal Logic*, Lecture Notes in Computer Science, pages 480–505. Springer-Verlag, 1994.
5. N. Glaser. The comomas methodology and enironment for multi-agent system development. In *DAI*, pages 1–16, 1996.
6. C.A. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.
7. G.J. Holzmann. *The Spin Model Checker: Primer and Reference Manual.* Addison-Wesley, 2003.
8. M. Jmaiel and P. Pepper. Development of communication protocols using algebraic and temporal specifications. *Computer Networks Journal*, 42:737–764, 2003.

9. M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Formalization of cooperation in MAS: Towards a generic conceptual model. In *The IX Ibero-American Conference on Artificial Intelligence (IBERAMIA 2004)*, volume 3315 of *Lecture Notes in Artificial Intelligence*, pages 43–52. Springer-Verlag, 2004.
10. M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *Proceedings of the first international conference on Multi-Agent Systems*, pages 254–260. AAAI Press/MIT Press, 1995.
11. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.
12. I. Meisels and M. Saaltink. The Z/EVES 2.0 reference manual. Technical Report TR-99-5493-03e, ORA Canada, Canada, 1999.
13. A. Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *AOSE*, pages 185–193, 2000.
14. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *AOSE*, pages 174–185, 2002.
15. A. Regayeg, A. Hadj-Kacem, and M. Jmaiel. Specification and Verification of Multi-Agent Applications using Temporal Z. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, pages 260–266, Beijing, China, September 2004.
16. A. Regayeg, A. Hadj-Kacem, and M. Jmaiel. Towards a formal methodology for developing multi-agent applications using temporal Z. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05)*, Cairo, Egypt, January 2005.
17. D. Sannella. Algebraic specification and program development by stepwise refinement. In *Proc. 9th Intl. Workshop on Logic-based Program Synthesis and Transformation, LOPSTR'99*, volume 1817 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2000.
18. M. Spivey. The Z notation (second edition). *Prentice Hall International*, 1992.
19. M. Wooldridge, N. Jenning, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents*, 3, 2000.
20. M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.
21. H. Xu and S. M. Shatz. A framework for model-based design of agent-oriented software. *IEEE Transactions on Software Engineering*, 29(1):15–30, January 2003.
22. F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.
23. H. Zhu. A formal specification language for agent-oriented software engineering. In *Proceedings of AAMAS'2003*, pages 1174–1175, Melbourne, Australia, 2003.