Formal Specification and Prototyping of Multi-Agent Systems

Vincent Hilaire[1], Abder Koukam[1], Pablo Gruer[1], and Jean-Pierre Müller[2]

[1] Département Génie Informatique-Systèmes Et Transports
Université de Technologie de Belfort Montbéliard
4 rue du château, 90010 Belfort Cedex, FRANCE
(33) 384 845 583 009
vincent.hilaire@utbm.fr
[2] Institut d'Informatique et d'Intelligence Artificielle
Emile-Argand 11, CH-2007 Neuchâtel, SWITZERLAND
Université de Neuchâtel

Abstract. This paper presents a multi agent-oriented prototyping approach. It is a generic approach, applicable to a wide range of multi-agent systems. This approach relies on a few assumptions, the most important is that MAS must be described by an organizational model which semantics is given in term of a formal framework. This model allows for a simple description of both individual and collective multi-agent system aspects. The framework we use to give a formal description of this model is based on a multi-formalism approach. We illustrate this approach through a case study.

Key words: Agent, Specification, Prototyping

1 Introduction

Agent-based systems are new paradigm for modeling and building many computer systems ranging from complex distributed systems to intelligent software applications. It proposes 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. Although many Multi-Agent Systems (MAS for short) have been designed, there is a crucial lack concerning specification and development methodologies.
Process of specification is fundamental to handle the complexity related to build such systems and specifying the desirable behavior of MAS before their implementation phase. The specification process must fulfill two roles. The first is to provide the underlying rationale for the system under development. The second is to guide subsequent design, implementation and verification phases. A variety of specification formalisms are available in the multi-agent field. Such formalisms put the emphasis on the first role and do not provide a basis to fulfill the second.

As stated in [2], they are often abstract and unrelated to concrete computational models. We believe that one way to bridge the gap between the abstract and the concrete level is to build the system specification using a prototyping process [13]. This process provides a support for incremental specification leading to an executable model of the system being built. Indeed, in many areas of software and knowledge engineering, the development process putting emphasis on prototyping and simulation of complex systems before their effective implementation is proven to be a valuable approach.
The purpose of this paper is to present a formal approach to MAS that fits in with prototyping and simulation oriented processes. The first step towards such approach is to choose or to build a formalism for specifying MAS. Harel and Pnueli [10] split systems in two classes : transformational systems which can be described by a functional mapping input to output values and reactive systems which do not compute functions but perform a continuous interaction with their environment. Due to their complexity, MAS have reactive and transformational features. A formalism which : specifies easily and naturally both aspects, enables prototyping and simulation and guides the implementation phase which is to be defined yet. We have thus chosen to use a multi-formalism approach that results of the composition of Object-Z [3] and statecharts [11]. This formalism enables the specification of reactive and transformational aspects of MAS and their prototyping by simulation.
Even though it has enough expressive power to specify MAS aspects, this language does not provide any methodological guidelines. A specification method is essential to manage MAS complexity by decomposition and abstraction. We use an organizational model [5, 18] that consider organizations, interactions and roles as first class citizen. This model allows to go from the requirements to detailed design and helps to decompose a MAS in terms of roles and organizations. Model concepts are specified by a framework which has to be refined to specify a MAS particular application. These concepts are illustrated by the Aphtous Ulcer Fever example which has been presented in [4]. This example consists in cattles disease modeling by using roles and organizations.
The paper is organized as follows. Section 2 introduces the methodological approach proposed which results from an organizational model and formal specification framework. Section 3 gives the specification of the Aphtous Ulcer Fever example. Section 4 is devoted to a summary, and prospects for further research.

2 Specification approach

2.1 Role/Organization model

Our specification approach uses an organizational model which is based on three interrelated concepts: role, interaction and organization. Roles are generic behaviors. These behaviors can interact mutually according to interaction pattern. Such a pattern which groups generic behaviors and their interactions constitutes an organization. Organizations are thus descriptions of coordination structures. Coordination occurs between roles as and when interactions takes place. In this

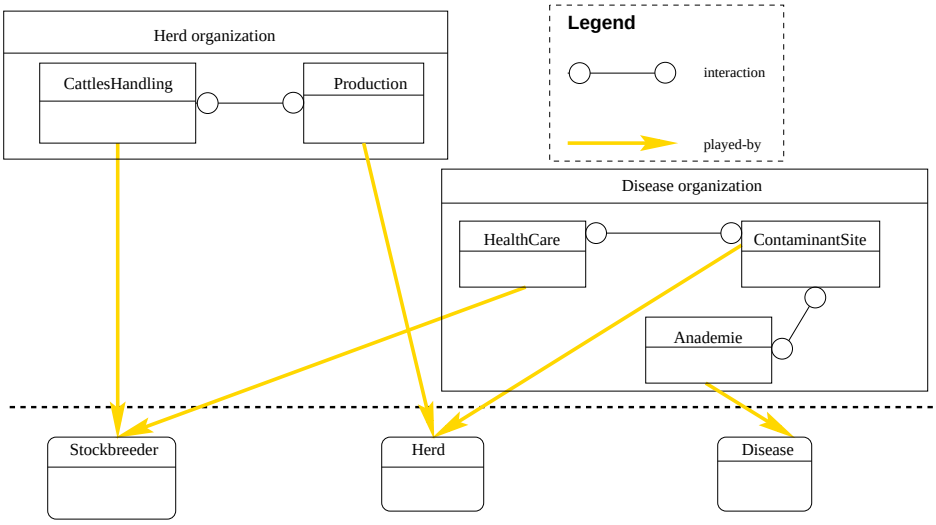


Fig. 1. Ulcer Aphtous Fever semi-formal model

context, an agent is only specified as an active communicative entity which plays roles [5]. 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. An agent may instantiate one or more roles and a role may be instantiated by one or more agents. The role playing relationship between roles and agents is dynamic. We make no assumptions on agent architectures. The generality of the agent definition allows the specification of many agent types. More specific choices can be introduced in more accurate models. This model enables a modular approach by prototyping separate parts of the MAS. The behavior of the MAS as a whole is the result of the role playing by agents. This methodology deals with both the macro (organizations) level and micro (roles) level. Moreover, this MAS decomposition enables the prototyping of only part of the MAS. In fact, the smallest entity one can prototype is a role. The roles, interactions and organizations of the Aphtous Ulcer Fever example are described by figures 1. To describe these organizations we borrow from the OOram notation [16]. A box that represents an organization encloses a set of boxes representing its roles. An interaction is materialized by a line connecting two roles. On each extreme of this line there may be nothing, a circle or a double circle to indicate the interactions arities. In our example, arities are all one to one, i.e concrete interactions always happen between two agents playing the respective roles. We present only a part of the Aphtous Ulcer Fever MAS model of [4]. In the part we present there are two organizations : *Herd* organization which deals with aspects related to cattles and *Disease* organization which models aspects related to disease spread and cure. The former organiza-

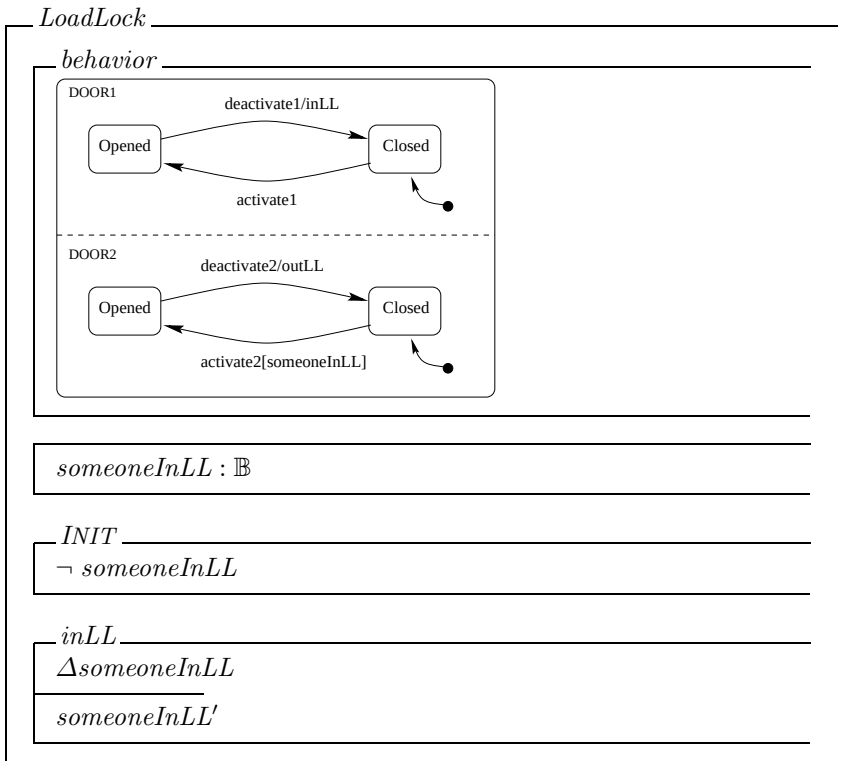
tion is composed of two roles : *CattlesHandling* (placing cattles either in farm or field and selling calfs) and *Production* (simulating calfs birth). The latter organization is composed of three roles : *HealthCare* (searching and cure of infected animals), *ContaminantSite* (simulating of possible contagions) and *Anademie* (simulating self-cures and contaminations). Three agents type play these roles. *STOCKBREEDER* agent plays *CattlesHandling* and *HealthCare* roles. *HERD* agent plays *Production* and *ContaminantSite* roles. Eventually *DISEASE* agent plays *Anademie* role. The roles played by these agents are shown in figure 1 by the arrows from roles to agents.

While this model is represented in a well structured and easy to understand manner, it lacks of formal semantics and means of rigorous analysis such as verification and prototyping. The section 2.3 introduces a formal framework fulfilling those needs.

2.2 Multi-formalism based specification language

Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to modeling all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend one formalism. The multi-formalism approaches [19, 15] 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 and to prove properties with proofs rules and transformation techniques available in a specific formalism.
Our choice is to use Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects. Object-Z extends Z with object-oriented specification support. The basic construct is the class which encapsulates state schema with all the operation schemas which may affect its variables. Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both language have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.
Our method for composition of Object-Z and statecharts relies on the meta-method of Paige [15]. The main step of this method is the definition of an heterogeneous basis which is a set of notations, translations and formalizations that provides a formal semantics to multi-formalism specifications. In our case the main concept of the heterogeneous basis is the integration of statecharts in Object-Z classes. We have extended the expressive capabilities of each formalism by features available in the other. The role of the heterogeneous basis is to provide formal means of expression without translating a formalism in the other. In other words the heterogeneous basis furnishes mean of communication between partial specifications written in either Object-Z or statecharts.
The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes

the possible states of the object and events which may change these states. A statechart included in an Object-Z class can use attributes and operations of the class. The sharing mechanism used is based on name identity. Moreover, we introduce basic types $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes. These types also belong to the heterogeneous basis.
The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between $DOOR1$ and $DOOR2$. The first door reacts to $activate1$ and $deactivate1$ events. When someone enter the *LoadLock* he first activate the first door enter the *LoadLock* and deactivate the first door. The transition triggered by $deactivate1$ event execute the $inLL$ operation which sets the $someoneInLL$ boolean to true. Someone which is between the first and the second door can activate the second door so as to open it. The temporal invariant at the end of the class specifies that the statechart must not be in $DOOR1.opened$ and $DOOR2.opened$ states simultaneously. This invariant uses the predicate $instate(S)$ which is true whenever S state is active.



outLL

$\Delta someoneInLL$

$\neg\ someoneInLL'$

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

The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.
The result of the composition of Object-Z and statecharts seems particularly suited in order to specify MAS. Indeed, each formalism have constructs which enable complex structure specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with. In fact, available constructs enable natural specification of "low" level aspects inherent to MAS. Higher level aspects like coordination are expressed by roles, interactions and organizations classes which we present in the following section.

2.3 Framework

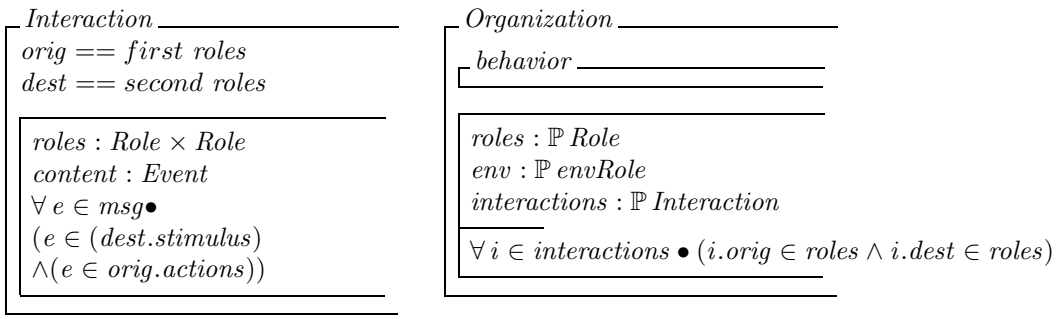
For specifying MAS described with the role/organization meta-model we build a framework based upon Object-Z and statecharts. Our framework is composed of a set of such classes which specify all meta-model concepts: role, interaction and organization.
The structure of the framework as a class hierarchy allows to inherit from defined classes in order to specify more specific model concepts or to extend existing concepts. Every model concept is given a formal semantics by a class of the framework. The framework provides specifiers with a structured approach committed to the organizational model.
We present three classes of the framework: *Role*, *Interaction* and *Organization*. The first concept of the model is role. A role is defined by a behavior schema which has to be refined in order to define the role behavior. Typically it's in a specialized behavior schema where the specifier integrate a statechart. State schema is composed by sets of *Attribute*, *Event* it can react to and *Action* it can execute.

Role

behavior

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

An interaction is composed of two roles and an interaction content, namely *content*, from a role origin of the interaction to a role destination of the interaction. Moreover, one may extend interaction to take into account more than two roles or more complex interactions involving plan exchange.
An organization is composed of a set of roles and their interactions. All interactions must only take places between two roles belonging to the same organization.



In order to specify interaction protocols within an organization one has to refine the *Organization* class by introducing, for instance, temporal logic predicates known as temporal invariants.

3 Specification and prototyping

3.1 Aphtous Ulcer Fever specification

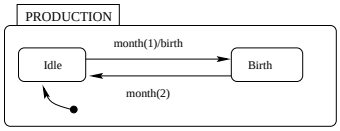
This section gives a formal description about Aphtous Ulcer Fever specification by refinement of the specification framework described previously. This formal description is then used as a basis for prototyping.
Production class inherits from Role. As described in the following it replace behavior sub-schema by PRODUCTION statechart. PRODUCTION behavior consists in waiting in *Idle* state for january to occur and then the *Idle* to *Birth* transition is triggered. This transition execute the *birth* operation which simulate calfs birth. On February the *Birth* to *Idle* transition is triggered. The class state is composed of sets of *Animal*. Each set specify an animal type, cattles for example are all animals, this constraint is specified by the expression $cattles = adult \cup calfs$. These sets belong to the attributes set of the role. The birth operation modifies calfs set. So calfs belongs to the Δ-list of the operation. This operation augments the calfs set size in order to simulate the increase of calfs count.

The same remarks stand for the role CattlesHandling except behavior subschema is replaced by CATTLESHANDLING statechart and *sellCalfs* operation diminishes calfs size. CATTLESHANDLING statechart specify two parallels process (shown as dashed line) : cattles localization and calfs selling. Indeed, calfs can be either in farm or in field and every october calfs are to be sold. The entering of *Farm* and *Field* states trigger respectively $contagion - possible$ and $contagion - impossible$ events.

Production

$Role[PRODUCTION/behavior]$

PRODUCTION



$cattles : \mathbb{P}\ Animal$
$adult : \mathbb{P}\ Adult$
$calfs : \mathbb{P}\ Calf$

$\{cattles, adult, calfs\} \in attributes$
$cattles = adult \cup calfs$

birth

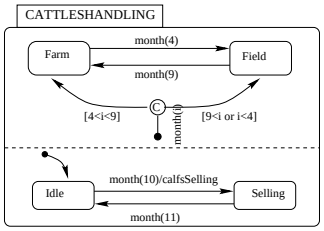
$\Delta calfs$

$\#calfs' \geq \#calfs$

CattlesHandling

$Role[CATTLESHANDLING/behavior]$

CATTLESHANDLING



$cattles : \mathbb{P}\ Animal$
$adult : \mathbb{P}\ Adult$
$calfs : \mathbb{P}\ Calf$

$\{cattles, adult, calfs\} \in attributes$
$cattles = adult \cup calfs$

sellCalfs

$\Delta calfs$

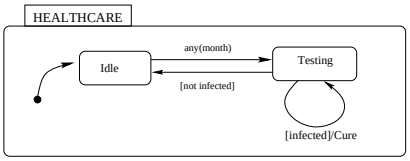
$\#calfs' \leq \#calfs$

HealthCare role consists, each month, in simulating herd scanning. Indeed, the *Idle* to *Testing* transition is triggered every month and if infected animals are detected, a cure is initiated by executing the *Cure* operation. The cure is efficient as soon as it is done and will last for three month. If there are no infected animals the *Testing* to *Idle* transition is triggered.

HealthCare

$Role[HEALTHCARE/behavior]$

HEALTHCARE



$infected : \mathbb{P}\ Animal$
$reminiscence : \mathbb{N}$

Cure

$\Delta reminiscence$

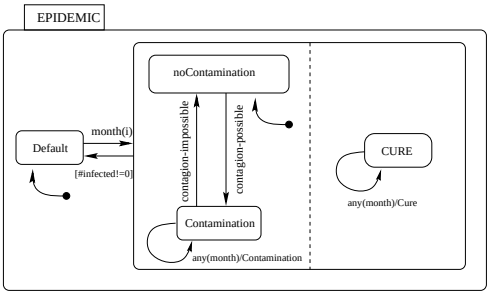
$\#infected \neq 0 \wedge reminiscence' = 3$

Anademie role describes the disease evolution which consists, for any month and if infected animals are detected, of two parallel processes: contamination and cure. The former tests if contagion is possible. If contamination is possible then the *noContamination* to *COntamination* transition is triggered and then, each month, the infected animals number may augment. Whenever there are infected animals the latter process simulates their cure, in other word it may decrease the infected animals number. If there are no infected animals the role is in *Default* state.

Anademie

$Role[EPIDEMIC/behavior]$

EPIDEMIC



$cattles, sanes, infected : \mathbb{P}\, Animal$

$cattles = sanes \cup infected$

Cure

$\Delta(infected, sanes)$

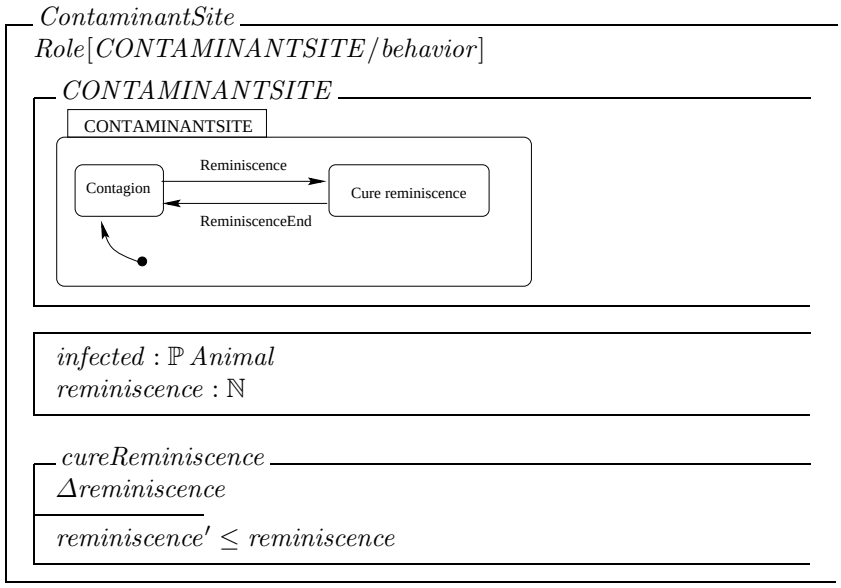
$\#infected - \#infected' = \#sanes' - \#sane]$

Contamination

$\Delta(infected, sanes)$

$\#sanes - \#sanes' = \#infected' - \#infected$

Eventually, *ContaminantSite* role decrement the reminiscence of cure and establish if contagion is possible.



3.2 Specification analysis

The analysis is performed by using STATEMATE [12]; an environment which allows the prototyping and the simulation of the statechart specifications. The specification analysis is based upon execution of the statecharts and can be done using two techniques. The first technique is *simulation* and the second is *animation*. In our case simulation would consist in assigning probabilities to events or actions occurrences. With this technique one can evaluate quantitative parameters of the specified system. For the Aphtous Ulcer Fever example probabilities can be associated to calfs selling or birth so one can evaluate how many animals are involved in each operation. Probabilities can be useful in different disease steps to simulate curing or infections of sane animals. Animation technique consists of testing the specification with predefined interaction scenarios. It enables one to test if the system behavior is consistent with requirements. For example, we have tested that if there is at least one infected animal the Health Care role executes *Cure* operation.

The simulation tool offers an interactive simulation mode and a program controlled mode. In the latter a program written in a high level language replaces the user. One feature of this programming language is the breakpoint construct. Breakpoint stop the specification execution when a condition is verified. Possible uses of breakpoints are, for example, configuration tests with predefined interaction scenarios and output of statistics. For the former one can test if all role reactions correspond to months schedule. For the latter one can associate probabilities for different illness phase so one can simulate illness evolution. The figure 2 is the result of such simulation. On x-axis we have month of the sim-

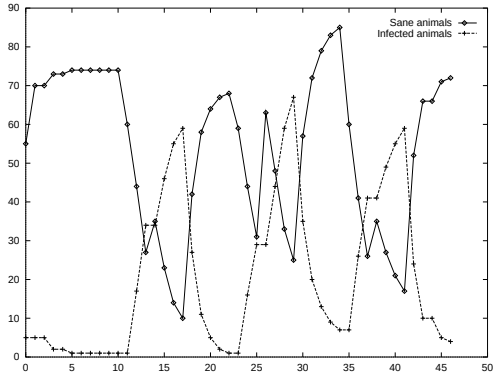


Fig. 2. Statistics of Aphtous Ulcer Fever simulation

ulation and on y-axis animals count. The two curbs are for Sane and Infected animals. One may see that each time animals are at farm the number of infected animals augment and then drop when they return at field. Indeed, contagion is only possible when animals are at farm.

4 Related Work and 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. For the sake of clarity we have only presented roles specification. But there are an agent class in our framework which enables the specification of evolving behaviors. This model is well suited for describing complex interactions which are among MAS main features. Furthermore, each model concepts are given a formal semantic within our specification framework. 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. The used specification language allows prototyping of specification. Prototyping is not the only means of analysis, indeed, in another work [7], 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 [9] which are based upon an organizational model [5].
Formal theories are numerous in the MAS area but they are not all related to concrete computational models [2]. Temporal modal logic, for example, have been widely used [17]. 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 for-

malisms [14]. Our approach is of the latter type. Among these approaches a few [14, 1] provide the specifier with constructs for MAS decomposition. The approach of Luck and d'Inverno has two drawbacks: the use of Z make MAS reactive aspects difficult to specify [6] and these specifications aren't executable so simulation and prototyping are not possible. The formalism used in DESIRE also lack the support regarding simulation or prototyping.
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 MAS development. But we believe that this first step constitutes a specification kernel leading towards a practical approach to formal specification of MAS. Among issues remaining for future work the organizational model used by us needs more work to do ahead. The play-by relationship of our example is of the simplest type. In order to deal with more complex case we have to explore the semantic of this relationship. Moreover, the Object-Z part of the specification is not yet executable. However a preliminary work [8] has shown that it is possible to give an operational semantic to Object-Z but it must be strengthened.

References

1. 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.
2. Mark d'Inverno, Michael Fisher, Alessio Lomuscio, Michael Luck, Maarten de Rijke, Mark Ryan, and Michael Wooldridge. Formalisms for multi-agent systems. In *First UK Workshop on Foundations of Multi-Agent Systems*, 1996.
3. 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.
4. Benoit Durand. *Simulation multi-agents et épidémiologie opérationnelle*. Thèse de doctorat, Université de Caen, 1996.
5. Jacques Ferber and Olivier Gutknecht. Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS'98*, july 1998.
6. 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.
7. Pablo Gruer, Vincent Hilaire, and Abder Koukam. Formal Specification and Verification of Multi-Agent Systems. In *ICMAS*. IEEE Computer Society Press, 2000.
8. Pablo Gruer, Vincent Hilaire, and Abder Koukam. Verification of Object-Z Specifications by using Transition Systems. In *Fundamental Aspects of Software Engineering*, LNCS. Springer Verlag, 2000.
9. Olivier Gutknecht and Jacques Ferber. Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Technical Report 97188, LIRMM, 1997.
10. D. Harel and A. Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*. Springer Verlag, 1985.
11. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

12. David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark B. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.
13. 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 Springer Verlag, editor, *Lecture Note in Artificial Intelligence*, number 1437 in LNAI, 1998.
14. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In AAAI Press/MIT Press, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260, 1995.
15. Richard F. Paige. A meta-method for formal method integration. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 473–494. Springer-Verlag, September 1997. ISBN 3-540-63533-5.
16. Trygve Reenskaug. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1996.
17. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Available by FTP, 1994. Submitted to The Knowledge Engineering Review, 1995.
18. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, pages 69–76, New York, May 1–5 1999. ACM Press.
19. Pamela Zave and Michael Jackson. Conjunction as composition. *acm Transactions of Software Engineering and Methodology*, 2(4):379–411, October 1993.