Specification and Design of Multi-agent Applications Using Temporal Z

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 proposes a formal approach, based on stepwise refinements, for specifying and designing multi-agent applications. This approach provides a specification language which integrates temporal logic in the Z notation allowing, in this way, to cover static, behavioural, as well as dynamic aspects of multi-agent systems. Moreover, it proposes a methodology giving a set of hints and principles which help and guide the design process. Indeed, this methodology enables the user to develop step by step, in an incremental way, an implementation starting from an abstract requirements (goal) specification. Finally, we illustrate our approach by developing an agent based solution for the *pursuit problem*.

1 Introduction

A multi-agent system is defined as a set of autonomous and distributed entities which cooperate in order to reach a common objective. This cooperation is essentially based on the exchange of data between these entities, on the one hand, and on the coordination of their activities particularly when they access to shared resources, on the other hand. In order to handle these aspects, autonomy, communication and coordination, the development of a multi-agent system should follow two ways diametrically opposite but closely dependent. The first focuses on the internal structure of agents (intra-agent), whereas the second concentrates only on the interactions between them (inter-agent). Considering intra-agent as well as inter-agent aspects makes the development of multi-agent systems an intricate task. Hence, mastering this complexity requires the application of rigourous approaches of software engineering.

In this paper we propose an approach which aims to facilitate the development of multi-agent applications while mastering its complexity. In order to achieve this objective, we follow two complementary principles of formal software design. The first principle stresses the need of a *requirements specification* phase. This phase makes use of formal specifications in order to perform rigourous reasoning. The second principle puts the emphasis on the *formal design*, in order to ensure the correctness of the design specification with respect to the requirements one. Our design process is based on *stepwise refinements* enabling us to

M.W. Barley and N. Kasabov (Eds.): PRIMA 2004, LNAI 3371, pp. 228–242, 2005.

construct a detailed design specification, step by step, starting from an abstract requirements one. Doing so, we will be able to better follow the evolution of the design specification. In this context, refining a specification means enriching it, in the sense that it becomes closer to the implementation. An abstract specification may leave some design decisions open, which will be resolved in later refinement steps [8]. In order to guarantee the correctness of the design specification the refinement steps should preserve the properties of the applications to be developed. Accordingly, it is very important to define a *refinement relation* which states if a specification implements another.

In our approach, we suggest a formal specification language which allows to cover individual (static and behavioural) aspects of agents such as knowledge, goal, and role, as well as collective aspects of a multi-agent application like interaction protocols, organization structure and planning activities. This language is an integration of a first order temporal logic [5] in the framework of the Z notation [7]. 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 [6]. These tools allowed us to perform syntax, type, and domain checking of our specifications as well as to reason about them by proving interesting properties. In the context of Z notation, a specification of a data structure includes two types of schemas in order to describe data and the operation on them. Generally, the refinement of such a specification requires *data* refinement as well as *operation* refinement respectively for data schema and operation schema. Since we do not use operation schemas (called also Δ schemas), we are only interested in refining Z schemas including data as well as behavioural description in terms of temporal properties. Accordingly, we define an appropriate data and behavioural refinement relation which extends the data one presented in [9].

This paper is organized as follows. Section 2 defines the specification language and its semantics. Then, in section 3 we present our development methodology. Thereafter, we illustrate our approach by developing a multi-agent solution for the pursuit problem. Finally, we conclude this paper with some future perspectives.

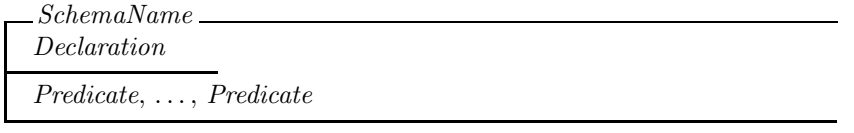
2 The Specification Language

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.

2.1 The Z Notation

The Z notation, as presented in [7], is a model oriented formal specification language which is based on the set theory and the first order predicate logic. This language is used to describe an application in terms of states which may change. A *basic type* is defined using one or several basic types. The definition of a *composed type*, with a collection of objects, needs a schema language. The latter is used to structure and to compose such specifications: collecting objects, encapsulating 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 [5], 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 simple predicate formulas (*formula*), which are closely related to the application to specify, and temporal formulas (*Tempformula*) which connect predicate formulas with temporal operators.

$$\begin{aligned} \mathit{Tempformula} ::= {} & \langle\!\langle \mathit{formula} \rangle\!\rangle \mid \bigcirc \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \mid \Box \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \mid \\ & \Diamond \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \mid \triangledown \langle\!\langle \mathit{Tempformula} \rangle\!\rangle \end{aligned}$$

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

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

2.3 The Time Model

The basic unit of the underlying time model is the agent state. Let [*State*] be the set of possible agent states. We define an entity state (*Entstate*) as a pair of a state and a point of time, where the time is specified as the set of natural numbers ($Time == \{x : \mathbb{N}\}$):

$$\begin{array}{|l} \textit{Entstate} \\ \hline time : Time \\ state : State \\ \hline \end{array}$$

A system state (*Syststate*) is defined as the union of all agents states:

$$\begin{array}{|l} \textit{Syststate} \\ \hline syststate : \mathbb{F}\ Entstate \\ \hline \end{array}$$

A time structure (*StrTime*) is defined as an axiomatic function that associates to each point of time the corresponding system state:

$$\begin{array}{|l} StrTime : Time \rightarrow Syststate \\ \hline \forall\, t : Time \bullet \exists\, s : Syststate \bullet \forall\, e : Entstate \mid e \in s.syststate \bullet \\ \quad StrTime(t) = s \Leftrightarrow e.time = t \end{array}$$

Based on this time structure we will be able to interpret any temporal formula.

2.4 The Semantics of Temporal Formulas

In this section, we define an evaluation function that associates the value true or false to any temporal formula. This step is very significant since it enables to translate temporal formulas into the pure Z notation[2]. In this way, it becomes easy to use the verification tools, such as Z/EVES or Isabelle, which accept the standard syntax of Z. First, we provide an axiomatic function (*Eval*) which evaluates a simple formula in a given system state:

$$\begin{array}{|l} Eval : formula \times Syststate \rightarrow bool \end{array}$$

Next, we define a similar axiomatic function for temporal formulas. This function gives a formal interpretation for the temporal operators $\triangledown$, $\Box$, $\Diamond$, and $\bigcirc$ with respect to a given time structure. For each operator, the recursive function *TempEval* defines a predicate which matches its intuitive meaning. In the following Z schema we present only the predicate defining the "*always*" operator. The predicates defining the other operators are similar.

[2] It should be stressed that temporal operators do not extend the expressive power of the Z notation based on the first order predicate logic.

$$TempEval : Tempformula \times StrTime \rightarrow bool$$

$$\forall f : Tempformula \bullet \forall t : Time \bullet \forall tt : Time \mid tt \geq t \bullet \forall S, SS : Syststate \bullet$$
$$TempEval(\Box\ f, (t, S)) = T \Leftrightarrow SS = StrTime(tt) \wedge$$
$$TempEval(f, (tt, SS)) = T$$

Finally, in order to integrate the temporal operators in their usual notations (i.e. $\Box$ for always) in the Z schema it is necessary to introduce them as axiomatic functions defined with the help of the interpretation function *TempEval*. In this way, we establish a logical equivalence between a temporal operator and the corresponding predicate specified in the above function *TempEval*. This equivalence is described for the $\Box$ operator as follows:

$$[Tempformula]$$
$$\Box : Tempformula \rightarrow bool$$

$$\forall f : Tempformula \bullet \exists present : Time \bullet \forall t : Time \mid t > present \bullet$$
$$\exists S : Syststate \bullet \Box f = T \Leftrightarrow S = StrTime(t) \wedge$$
$$TempEval(\Box f, (t, S)) = T$$

The other temporal operators are introduced with similar axiomatic functions.

3 Formal Design Approach

In order to be useful, a formalism or a set of tools have to be supported with a specification approach. This approach should provide some principles that help and guide the specification process. In this section, some of those principles are clarified.

Indeed, our approach is based on three 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 requirements specification is thus presented by Z schemas which language is extended by temporal logic. The third phase consists of performing verification tasks by formulating and reasoning about the main properties of the multi-agent application to be developed.

3.1 Specification Phase

Generally, in this first phase we specify the requirements which correspond, in the context of multi-agent, to the common objective they have to achieve. It should be stressed that we are not interested, at this stage, in the manner of achieving this goal. In our approach, this stage includes also the description of the environment in which the agents evolve. This environment includes, generally, the work area, the passive objects, and the active entities representing the agents to be employed. The requirements specification is thus presented by a set of Z schema which properties language is extended by temporal logic.

1. *Specification of the Active Entities:* The description of an active entity (agent), at specification phase, consists in presenting, in terms of temporal formula, its static and dynamic original properties. These properties are the information acquired on the agents at the beginning. This description is given by a Z schema which declares the entity attributes, defines its static properties, in term of predicate logic, and its behavioural properties, in term of temporal logic.

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 Environment:* The environment includes active entities (agents) as well as passive entities belonging to the operating field. This specification is given by a set formulas making in relation passive entities with those which are active. Generally, this leads to Z schema of the form:

Environment

$obj_1 : TypeObject_1,\ obj_2 : TypeObject_2, \ldots, obj_k : TypeObject_1$
$Entities$: **set of** $Entity$

$Pr_1, Pr_2, \ldots, Pr_l$

Where obj_i corresponds to a passive entity, *Entities* represents a set of entities where cardinality is unknown at the beginning, and Pr_i represents a temporal formula.

3. *Specification of the Requirements:* This specification describes what we require from the system to develop. In the context of multi-agent, this corresponds to formulate, in term of temporal formula, a future environment state to be reached. According to the Z approach, such a specification is well expressed by a specialization of the schema specifying the environment. Generally, requirements specifications have the following form:

Spec

Environment

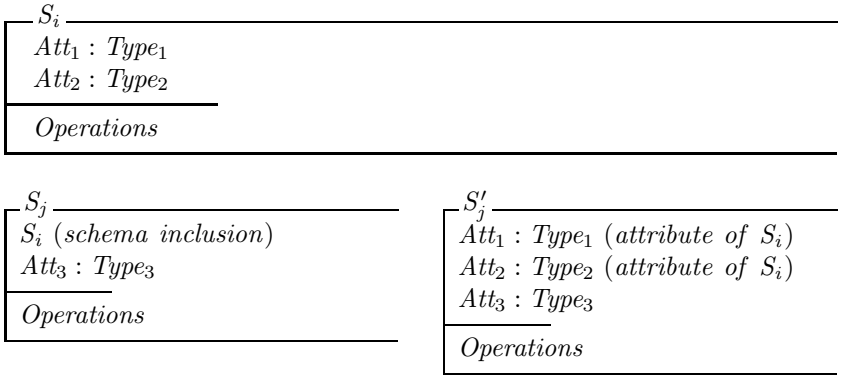
$Tpr_1, Tpr_2, \ldots, Tpr_n$

Where Tpr_1 represents a temporal formula.

3.2 Refinement Phase

The basic idea consists in performing a succession of refinements in terms of specializations of Z schemas (data refinement) and deriving of temporal formulas (behaviour refinement).

Refinement Relation. First of all we define a refinement relation between specifications telling if a specification implements another. In the context of Z notation, we adopt the $\sqsubseteq$ relation defined in [9] with some restrictions. Concerning the data refinement, a schema S_j refines another schema S_i (written $S_i \sqsubseteq S_j$) if and only if the attributes of S_i are included in the declaration part of S_j. This inclusion of attributes can appear following a schema inclusion (S_j schema) or by a new declaration of these attributes (S'_j schema):



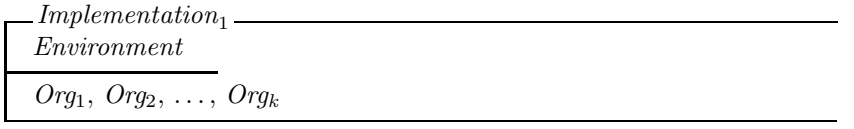
Concerning the behavioural part which is specified in terms of temporal formulas the refinement relation is defined between temporal specifications. For this purpose, we adopt the refinement relation applied in [3] for designing communicating protocols using algebraic and temporal specifications. A temporal specification (set of temporal formulas) $TSpec = \{F_1, F_2, \ldots, F_n\}$ refines another one $TSpec' = \{F'_1, F'_2, \ldots, F'_m\}$, written $TSpec' \sqsubseteq TSpec$, if every formula F'_i in $TSpec'$ could be derived from $TSpec$, denoted $TSpec \vdash F'_i$, that is the validity of F'_i could be inferred from the validity of the formulas in $TSpec$ using the calculus of the first order temporal logic.

Refinement Process. The refinements are carried out at two complementary levels. The first, is the environment 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 on the individual aspects (intra-agent) by extending the specifications of the active entities provided in the first phase. The extensions of individual specifications have to remain consistent with the extensions made at the collective level.

Collective Level: Here we invent a suitable organization structure as well as a communication topology for the system to be developed. An organization structure assigns a role for each active entity belonging to the system. Furthermore, it implicitly defines a control strategy to be respected by these entities. This structure is, generally, defined in terms of temporal formulas referring to several entities at the same time. A sequence of Z schemas will be then generated:

$Implementation_1, Implementation_2, \ldots, Implementation_n$

The first schema corresponds to a direct refinement of the environment specification. It is of the form:

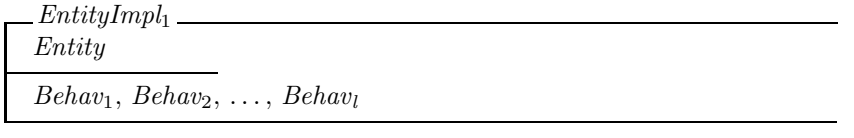


Where Org_i is a temporal formula that assigns roles to one or many entities. These formulas correspond to design decisions describing an organization structure. Each Z schema $implementation_i$ $(i > 1)$ is a refinement (specialization) of the previous one $implementation_{i-1}$. These intermediate schemas may contain design decisions related to the communication topology and actions. It is obvious that the last schema $implementation_n$ is, by transitivity, a refinement of the initial environment schema.

Individual Level: Here we make use of the choices made at the collective level in order to refine, step by step, the internal structure and the behaviour of each agent. For each entity, we develop a succession of refinements as follows:

$$EntityImpl_1,\ EntityImpl_2,\ \ldots,\ EntityImpl_m$$

The first schema is an immediate refinement of the retained one at the specification phase. This schema is of the form:



Where $Behav_i$ is a property describing a design decision related to the behaviour of the considered entity. These properties are given by individual temporal formulas referring to only one entity. Hence, each intermediate schema $Entityimpl_i$ is a refinement of the diagram $Entityimpl_{i-1}$. 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 environment (*Environment*) extended with the union of the properties added at both collective and individual levels.

3.3 Verification Phase

This phase allows to reason, according to the Z notation combined with temporal logic, about the schemas developed during the specification phase. Essentially, we are interested in proving the consistency and the completeness of the global specification resulting from the composition of the agent's individual specifications. On the basis of a set of abstract properties, which are specific to the application to be developed, we try to check the mentioned properties of consistency and the completeness of the global specification. Generally, we develop a schema Z which gathers a set of global and abstract properties:

Spec

Environment

$Tpr_1,\ Tpr_2,\ \ldots,\ Tpr_n$

Where Tpr_1 represents a temporal formula. It is very important to use automatic tools which support the proof of the desired theorems. The temporal logic model that we proposed in the previous section makes possible the use of automatic tools, such as Z-Eves [6], while integrating temporal formulas in our specifications and theorems.

4 Case Study: The Pursuit Problem

We illustrate our approach by specifying a multi-agent solution for the pursuit problem. This application includes one prey and four predators. The prey moves randomly on a grid without perception of its environment. The predators cooperate to capture the prey using their perception and communication abilities.

Initially, each predator moves independently of the others. As soon as one predator perceives the prey, it follows the prey until its capture from the nearest side. This predator, which will play the *supervisor* role, will regularly inform the others about the current prey position. From this moment and based on the received information, each predator tries to capture the prey from the appropriate side, according to its position on the grid.

4.1 Specification Phase

Specification of the Environment: A box on the grid is defined by its abscissa and ordinate.

Box

$abs : \mathbb{N}$

$ord : \mathbb{N}$

$abs \geq 0 \land ord \geq 0$

The basic concept, *Entity*, is characterized by its state (*Entstate*). So, we specify formally the entity state by the position (*pos*) that it occupies on the grid at a given time point.

Entstate

$time : Time$

$pos : Box$

A system state (*Syststate*) is defined as the union of the states of all agents:

Syststate

$$syststate : \mathbb{F}\ Entstate$$

An *Entity* is able to move randomly (*ChangePos*) on the grid. However, it can make at most one step (to the left, the right, the south, or the north) at a moment. This ability is formally specified in the following schema:

Entity

$$state : Entstate$$
$$ChangePos : Box \times Box \rightarrow bool$$

$$\forall s_1, s_2 : Box \mid s_1 = state.pos \bullet \exists i, j : \mathbb{N} \mid i \in \{0, 1, -1\} \land j \in \{0, 1, -1\} \land (i = 0 \lor j = 0) \bullet ChangePos(s_1, s_2) = T \Leftrightarrow s_2.abs = s_1.abs + i \land s_2.ord = s_1.ord + j$$
$$\forall s_1, s_2 : Box \mid s_1 = state.pos \bullet \text{next}\ ChangePos(s_1, s_2) = T$$

As mentioned in the previous section, atomic formulas are, generally, specific for the application to specify. So, we define, in the following, the set of atomic formulas relating to our application. An atomic formula may be a predicate describing an entity state. Formally:

$$formula == \{predicate : bool\} \tag{1}$$

A *Prey* is a simple entity, whereas a *Predator* is able to perceive other entities.

Predator

$$Entity$$
$$Perception : \mathbb{F}\ Entity$$

$$Perception = \{e : Entity \mid (state.pos.abs - 2 \leq e.state.pos.abs \leq state.pos.abs + 2) \land (state.pos.ord - 2 \leq e.state.pos.ord \leq state.pos.ord + 2)\}$$

An environment is composed of a grid, a prey and four predators. It has to meet two conditions. First, two entities cannot occupy the same box on the grid. Second, an entity should not leave the grid. Formally:

Environment

$$X : \mathbb{N}$$
$$Y : \mathbb{N}$$
$$prey : Entity$$
$$pr_1, pr_2, pr_3, pr_4 : Predator$$

$$\forall e : Entity \bullet (e.state.pos.abs \leq X) \land (e.state.pos.ord \leq Y)$$
$$\forall e_1, e_2 : Entity \bullet (e_1.state.pos = e_2.state.pos) \Leftrightarrow (e_1 = e_2)$$

Useful Definitions: In order to simplify our specifications and make them more readable, we provide some useful abbreviations. First, we define the *Side* type including exactly four values corresponding to the sides from which the prey can be captured.

$$Side ::= North \mid South \mid East \mid West$$

Second, we define an axiomatic function *SideCaptured* testing if the prey is captured by a predator from a given side. The following definition presents only the predicate relating to the *South* side. The predicates for the other sides are similar.

$$SideCaptured : Entity \times Predator \times Side \rightarrow bool$$

$$\begin{aligned} &\forall\, a : Entity \bullet \exists\, b : Predator \bullet \\ &\quad SideCaptured(a, b, South) = T \Leftrightarrow b.pos.abs = a.pos.abs \wedge \\ &\qquad b.pos.ord = (a.pos.ord + 1) \end{aligned}$$

The axiomatic function *Captured* abbreviates the fact that the prey is captured from all sides by the four predators.

$$Captured : Entity \times \mathbb{P}\, Predator \rightarrow bool$$

$$\begin{aligned} &\forall\, a : Entity \bullet \exists\, b_1, b_2, b_3, b_4 : Predator \bullet \\ &\quad Captured(a, \{b_1, b_2, b_3, b_4\}) = T \Leftrightarrow SideCaptured(a, b_1, North) = T \wedge \\ &\qquad SideCaptured(a, b_2, South) = T \wedge SideCaptured(a, b_3, East) = T \wedge \\ &\qquad SideCaptured(a, b_4, West) = T \end{aligned}$$

Agent Actions: Before presenting the detailed specifications of the individual and collective aspects of our application, we introduce the set of possible actions which may be performed by a predator. We distinguish two kinds of actions: internal and external. Internal actions are instructions enabling a predator to update its mental state by revising its knowledge base or changing its local goal.

- Updating the knowledge base: *updateBase* $\langle\langle Predator \times Box \rangle\rangle$
  It consists in updating the knowledge base after receiving a new prey position.
- Updating the goal: *updateGoal* $\langle\langle Predator \times Side \rangle\rangle$
  Corresponds to updating the goal after receiving the information telling that the prey has been already captured from the envisaged side.
- Updating the destination: *updateDest* $\langle\langle Predator \times Box \times Side \rangle\rangle$
  It consists in updating the attribute destination following the reception of a captured side.

In our context, the communication actions which are considered as external are very essential. We identified three communication actions: *send*, *receive*, and *broadcast*.

- Send action: $send\ \langle\langle Predator \times Predator \times Message\rangle\rangle$
  This action enables a predator to send some information to another predator.
- Diffusion action: $broadcast\ \langle\langle Predator \times \mathbb{P}\, Predator \times Message\rangle\rangle$
  This action allows to broadcast the same information to a set of predators.
- Receipt action: $receive\ \langle\langle Predator \times Predator \times Message\rangle\rangle$
  This action enables a predator to receive an information sent by another predator.

A message may inform about the prey position (Pos), it may be a request made by a predator to the supervisor ($Request$), or it may be an assignment of a local goal made by the supervisor to a predator ($Assign$).

$$Message ::= Pos\langle\langle Box\rangle\rangle \mid Request\langle\langle Box\rangle\rangle \mid Assign\langle\langle Entity \times Side\rangle\rangle$$

It is very important to note that we consider in our approach synchronous communication between agents. Moreover, we suppose that communication mediums are absolutely reliable. This is formally described by the following equivalences:

$$\begin{array}{l}\forall\, a, b, b_1, ..., b_n : Predator \bullet \forall\, message : Message \bullet \\ \quad broadcast(a, \{b_1, ..., b_n\}, message) = T \Leftrightarrow \\ \qquad \bigwedge_{i \in \{1,...,n\}} receive(b_i, a, message) = T \\ \quad send(a, b, message) = T \Leftrightarrow receive(b, a, message) = T\end{array}$$

4.2 Refinement Phase

In this section, we propose to design, in a first level, the individual aspect of the application and in a second level, the collective aspect. These two levels will be described by axioms that will be added to the schema of the predator.

Individual Aspect. During the game, the possible scenarios as well as the different acts of communication determine the agent mental state (knowledge base) and its future behaviour. This is represented in the different refinements that follow:

- The predator updates its knowledge base as soon as it perceives the prey:

*Predator*0

Predator

$$\begin{array}{l}\exists\, prey : Entity \bullet \\ \quad updateBase(prey.state.pos) = T \Rightarrow posprey = prey.state.pos \\ \forall\, prey : Entity \bullet \exists\, s : Syststate \mid prey.state \in s.syststate \bullet \\ \quad prey \in Perception \Rightarrow \bigcirc\, (updateBase(prey.state.pos), s) = T\end{array}$$

- others updates of the knowledge base, the destination or the goal are done due to:
  - the receipt of an information about the prey position: *Predator1*
  - the receipt of an affectation of capture side: *Predator2*

- Another conceptual decision concerns the speed of predator displacement. The predator speed must be superior to the prey speed: *Predator3*
- Once a predator comes to capture the prey on one side, it is going to remain there always.

*Predator*4

*Predator*3

$$\exists\, prey : Entity \bullet \exists\, side : Side \bullet$$
$$\exists\, s : Syststate \mid prey.state \in s.syststate \bullet$$
$$SideCaptured(prey, side) = T \Rightarrow \Box\,(SideCaptured(prey, side)), s) = T$$

These refinements constitute the description of the individual aspect of the MAS. This aspect concerns the properties referring to one agent whereas the proprieties in the collective aspect, subject of the next section, refer to many agents.

Collective Aspect. Our methodology consists in refining, step by step, the initial specification.
So, we give some modifications to *Environment* where we use the *Predator4* and where we describe the *Captured* predicate and the communication equivalence between *send* and *receive* actions.

- Definition of the organizational structure: In our example, we can distinguish between a supervisor predator (the first that perceives the prey) and the other predators. We define, then, a supervisor schema:

$$Supervisor == \{pr : Predator \mid \exists\, e : Entity \bullet e \in pr.Perception\}$$

The supervisor is charged to capture the prey of the nearest side and to distribute a request of information about the current positions of the other predators in order to affect the remaining sides. Thus, in this stage, a first implementation of the game is:

*GameImpl*0

*Environment*0

$$pr_1 \in Supervisor$$

- In order to describe the communication structure, we will define the communication acts that could take place between the different predators referring to the following property:

$$a, b, b_1, ..., b_n : Predator$$
$$message : Message$$
$$broadcast(a, \{b_1, ..., b_n\}, message) = T \Leftrightarrow$$
$$\bigwedge_{i \in \{1,...,n\}} receive(b_i, a, message) = T$$
$$send(a, b, message) = T \Leftrightarrow receive(b, a, message) = T$$

We can, further, refine the conception of the game : *GameImpl1* and *GameImpl2*.
Finally, the receipt of the capture side, by each predator, guarantees the sides affectation:

GameImpl3

GameImpl2

$$\forall\, pr : Predator4 \mid pr \in \{pr_2, pr_3, pr_4\} \bullet \exists\, side : Side \bullet$$
$$\exists\, S : Syststate \bullet pr_1.receive(pr, Pospr.state.pos) = T \Rightarrow$$
$$\Diamond\, (pr_1.send(pr, Affect(prey, side)), S) = T$$

4.3 Verification

In the verification phase, starting from a requirements specification, we try to prove the set of the theorems that may be generated from it. The requirements specification of the pursuit problem requires that the four predators eventually capture the prey from all sides. This is given by the following schema:

ReqSpec

Environment

$$\Diamond \Box\, Captured(prey, \{pr_1, pr_2, pr_3, pr_4\}) = T$$

To this schema leads the following theorem those we are charged to reduce to *true*:

Theorem 1. *axiomGameSpecFin*
$PreyPredSpec \Rightarrow \Diamond \Box\, Captured(prey, \{pr_1, pr_2, pr_3, pr_4\}) = T$

The proof of these theorems has been accomplished with the Z/EVES tool.

5 Conclusion

In this paper we proposed a formal approach for the development of multi-agent applications. Our contribution concerns, first, the definition of a formal language which covers the static and the behavioural aspects of agents by integrating temporal operators within Z notation. Then, we defined a methodology that permits to develop, step by step, in an incremental way, a design from an abstract specification. The investigation of the pursuit problem allowed a first illustration of our approach. Other case studies are under realization (e.g. the conflicts control in the air-traffic). However, it is necessary to point out that these first results, even original and promising, constitute a modest contribution to the problematic of multi-agent formal development. In a short term, we will proceed to automate the syntactic verification and the semantic validation of the well constructed specifications as well as the formal reasoning on these latest. Further, we plan to implement a tool-kit to deal with the steps of the proposed methodology. It is obvious that such tools must be coupled up with an environment that provides verification and reasoning about specifications such as Z/EVES [6] or Z-Hol [4].

References

1. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, 1998.
2. A. El Fallah. Représentation et manipulation de plans à l'aide des réseaux de Petri. *Actes des 2èmes Journées Francophones IAD-SMA*, 1994.
3. M. Jmaiel and P. Pepper. Development of communication protocols using algebraic and temporal specifications. *Computer Networks Journal*, 42:737–764, 2003.
4. R. Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in isabelle-Hol. In J. von Wright, J. Grundy, and J. Harrison, editors, *9th International Conference on Theorem Proving in Higher Order Logics*, LNCS 1125, pages 283–298. Springer Verlag, 1996.
5. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
6. I. Meisels and M. Saaltink. The Z/EVES 2.0 reference manual. Technical Report TR-99-5493-03e, ORA Canada, Canada, 1999.
7. M. Spivey. The Z notation (second edition). *Prentice Hall International*, 1992.
8. V. Von. *An Integration of Z and Timed CSP for specifying Real-Time Embedded Systems*. PhD thesis, 2002.
9. J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof*. Prentice Hall, 1996.