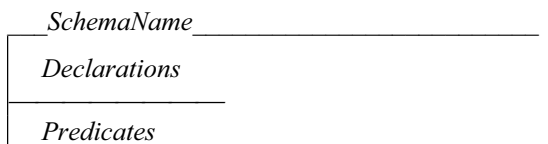




above mentioned aspects in a unified framework. Indeed, the Z notation allows to describe all components (passive and active) in terms of attributes and related properties. The temporal logic will enrich this description with social behavior and interaction properties.

**2.1 The Z notation**

The Z notation, as presented in [15], 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 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:



**3. Design Method**

In order to be useful, a formalism has to be supported with a design method providing some principles which help and guide the design process. In this section, some of those principles are clarified. Indeed, our method called ForMAAD 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 of collective (inter-agents) and individual (intra-agent) behaviors. The adopted refinement relation is described in one previous work [13]. The verification that the resulted design specification satisfies the requirements one is considered as an essential task which is progressively performed during the refinement steps (Fig. 1).

**3.1 Specification Phase**

In this phase, we specify the requirements which correspond, in a society of agents, to a common objective which has to be achieved by these agents. In our approach, this step includes also an abstract specification of the environment in which the agents evolve and which is, generally, composed of a working space and passive entities (objects). The specification of the environment may include some active entities (agents), if they are known in advance.

- (i) Entity Specification: An entity is defined, within a Z schema, in terms of a set of attributes and logic formulas describing its structural part. In the case of an active entity, we enhance the Z schema with temporal formulas specifying the behavioral part. The specification of an active entity is usually refined during the design process.
- (ii) System Specification: This specification describes the environment in which the agents will be executed. In this specification, we introduce formulas relating passive entities with active ones.
- (iii) Requirement Specification: In ForMAAD, the requirements specification corresponds to a temporal

formula  $C$  (presented as a conjunction of temporal formulas  $c_i$ ) specifying a Common Objective to be achieved by a group of agents evolving in the specified environment. A common objective describes a desired future system state. According to the Z approach, this specification corresponds to a specialization of the System schema.

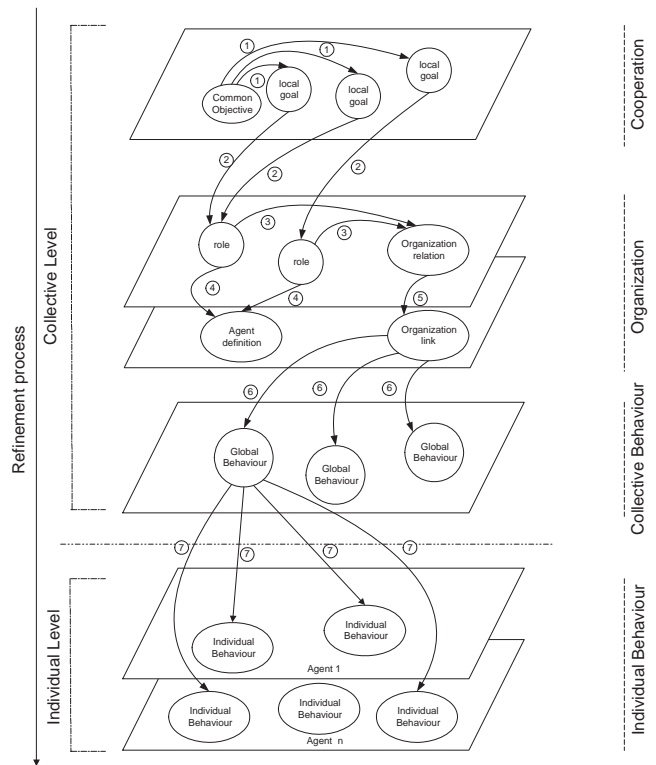


Fig. 1. Design process based on seven refinement steps.

**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 behavioral 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 augment the System specification by properties referring, essentially, to collective aspects (inter-agent) characterizing, in particular, organization and interaction structures. The second level rather stresses individual aspects (intra-agent) by refining the specifications of the active entities provided in the first phase. In addition, we, possibly, add and refine new agents if they are needed.

The suggested methodology provides seven refinement steps as shown in Fig. 1.

The first step defines a cooperation strategy for achieving the common objective. It tries to decompose it in a set of sub-goals, called local goals. The definition of an organization structure is performed into two steps. First, we identify the roles by regrouping local goals. Then, we try to relate them with suitable links. Simultaneously, we try to assign roles to agents. Usually, an agent may play different roles, and different agents may have the same role. The relationships between roles are translated at the agent level into organization links. Based on this links, we identify the needed



the global properties have been proven based only on the individual properties. ■

Each refinement step is accompanied with a proof obligation. Because we don't have space enough to describe all the theorems, we propose to present, as an example, that corresponding at Step 4.

**Theorem 1.** RoleAssignment.

$$Implementation_3 \Rightarrow \bigcup_{a \in Ag} a.roles = R$$

where Ag is the set of agents present in the system.

## 4. A Case Study: Air Traffic Control Application

In the literature, many projects have been developed in order to ensure the air conflict resolution. Among them, some works are interested in ameliorating the existent centralized control [6]; whereas, some others propose many decentralized solutions based on different approaches, methodologies and principles such as the neuronal methods using the genetic algorithms [5], the repelling powers methods [16] or sequential methods [1].

Since this kind of application is critical, a formal verification is an obligation in order to ensure the reliability of the final solution. Thus, we proceed to present a specification using the previously defined method ForMAAD. Each refinement is proved using the Z/EVES tools.

The main concept is an air route. In many parts of the airspace, planes are required to follow established routes, rather like highways in the sky. Another concept is an air traffic control sector. Sectors are blocks of airspace assigned to different air traffic controllers, and the "rule" in this case is which controller or team of controllers is responsible for keeping airplanes from getting too close to each other. To keep the workload of controllers reasonable, sectors have to be sized so that not too many airplanes will be in them at the same time (so sectors are smaller in busy areas than in areas where traffic is light).

### 4.1 Specification Phase

In this case study, our domain is an airspace sectors. Each air sector is composed of several routes connected by points called waypoints. Between two waypoints, the air route consists of a sequence of parallel corridors which are described by their level of flight (altitude), a starting point and an arrival point. A corridor can cross with another one (of another route) having the same altitude; the intersection point is called the waypoint.

$$Pos \hat{=} [x, y: \mathbb{N} \mid x > 0 \wedge y > 0]$$

$$WayPoint \hat{=} [name: String; pos: Pos; R: \mathbb{N}]$$

#### 4.1.1 Entity Specification

The plane is specified by a current sector, a position, a speed and an altitude.

$$Corridor \hat{=} [alt: \mathbb{N}; free: bool; wp_1, wp_2: WayPoint \mid wp_1.pos \neq wp_2.pos]$$

A sequence of parallel corridors forms a route having a number Num.

$$Route \hat{=} [Num: \mathbb{N}; SeqCorr: seq Corridor]$$

Each plane has a field of perception making it possible to

detect another plane, from where a conflict situation, if exists, must be solved as quickly as possible.

Plane
sector: seqWayPoints pos: Pos; speed, alt: \mathbb{N}; Corr: Corridor route: Route; way: seqWayPoint Perception: PPos; PlSect: PPos
Perception = { p: Pos \mid pos.x \le p.x \le pos.x + PerMinMax \wedge pos.y \le p.y \le pos.y + PerMinMax }

#### 4.1.2 System Specification

For our case study, after specifying the basic entities, we define a System as a set of planes called PL. This set contains at least two planes.

$$System \hat{=} [PL: \mathbb{F} Plane \mid \#PL \geq 2]$$

#### 4.1.3 Requirement Specification

The common objective consists in solving each conflict situation which may occurs between two planes. If these planes have the same altitude and they are located on two different corridors reaching the same waypoint, then they are in a conflict situation. Also, a plane can participate in one or more conflicts.

Thus, the common objective of such system is presented in the predicative part of the following requirement specification:

$$ReqSpec \hat{=} [System \mid \forall pl_1, pl_2 \in PL \bullet Conf(pl_1, pl_2) \Rightarrow \diamond SolveConf(pl_1, pl_2)]$$

where the predicate Conf describes a conflict situation.

### 4.2 Design Phase

#### 4.2.1 Collective Level

Cooperation strategy.

Step 1. Cooperation Strategy definition. Each plane participating in the conflict situation must be able to detect this conflict and to solve it after negotiation. Thus, we decompose SolveConf as follows.

SolveConf: Plane \times Plane
$\forall pl_1, pl_2: Plane \bullet SolveConf(pl_1, pl_2) \Leftrightarrow$ $DetectionConf(pl_1, pl_2) \wedge$ $\diamond (Negotiate(pl_1, pl_2) \wedge \diamond SolConf(pl_1, pl_2))$

This decomposition leads to the refined specification:

Implementation <sub>0</sub>
System
$\forall pl_1, pl_2 \in PL \bullet$ $Conf(pl_1, pl_2) \Rightarrow DetectionConf(pl_1, pl_2) \wedge$ $\diamond (NegotiateWith(pl_1, pl_2) \wedge$ $NegotiateWith(pl_2, pl_1) \wedge$



$$\begin{aligned} & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \exists solution: Solution \bullet \\ & \quad NegotiateWith(pl_1, pl_2) \Rightarrow \\ & \quad Send(pl_1, pl_2, Propose(solution)) \vee \\ & \quad Receive(pl_1, pl_2, Propose(solution)) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \\ & \quad Send(pl_1, pl_2, Propose(solution)) \Rightarrow \\ & \quad \diamond Evaluate(pl_2, pl_1, solution) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \\ & \quad Evaluate(pl_2, pl_1, solution) \Rightarrow \\ & \quad \diamond Send(pl_1, pl_2, Accept(solution)) \vee \\ & \quad \diamond Send(pl_1, pl_2, Reject(solution)) \end{aligned}$$

#### 4.2.2 Individual Level

**Step 7. Individual behavior definition.** In this step, we describe all the individual behaviors of the planes by proving the global ones referring to negotiation carried out. An individual behavior is presented by a temporal formula where the envisaged predicates concern only one plane. In order to deduce these individual behaviors from the global ones, we define an equivalence relation between Send and Receive stating that each information sent by a plane is received by the other one:

$$S/REquiv: Send(pl_i, pl_j, inf) \Leftrightarrow Receive(pl_i, pl_j, inf)$$

We can deduce a list of the individual behaviors for the planes: In the following, we proceed, as an example, to prove one of the three global behaviors described in *Implementation<sub>5</sub>*; let be:

$$DetectionConf(pl_1, pl_2) \text{ before } SolConf(pl_1, pl_2)$$

In order to prove this causality relation, we have to add a temporal formula indicating that when a plane detects a conflict, it informs the other one:

$$\begin{aligned} & DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Send(pl_1, pl_2, Propose(solution)) \quad [IndBehav1] \\ & \Leftrightarrow DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Receive(pl_2, pl_1, Propose(solution)) \quad [S/REquiv] \\ & \Leftrightarrow DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Evaluate(pl_2, pl_1, solution) \quad [IndBehav2] \\ & \Leftrightarrow DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Send(pl_2, pl_1, Accept(solution)) \quad [IndBehav3] \\ & \Leftrightarrow DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Receive(pl_1, pl_2, Accept(solution)) \quad [S/REquiv] \\ & \Leftrightarrow DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad SolConf(pl_1, pl_2) \quad [IndBehav4] \end{aligned}$$

Thus the list of formulas describing the needed individual behaviors is:

$$\begin{aligned} & IndBehav_1: DetectionConf(pl_1, pl_2) \text{ before} \\ & \quad Send(pl_1, pl_2, Propose(solution)) \\ & IndBehav_2: Receive(pl_2, pl_1, Propose(solution)) \text{ before} \\ & \quad Evaluate(pl_2, pl_1, solution) \\ & IndBehav_3: Evaluate(pl_2, pl_1, solution) \text{ before} \\ & \quad Send(pl_2, pl_1, Accept(solution)) \vee Send(pl_2, pl_1, \\ & \quad \quad Reject(solution)) \\ & IndBehav_4: Receive(pl_1, pl_2, Accept(solution)) \text{ before} \\ & \quad SolConf(pl_1, pl_2) \end{aligned}$$

Given the following decomposition of the local goals:

$$DetectionConf: Plane \times Plane$$

$$\begin{aligned} & \forall pl_1, pl_2: Plane \bullet \exists solution: Solution \bullet \\ & \quad DetectionConf(pl_1, pl_2) \Leftrightarrow \\ & \quad Perceive(pl_1, pl_2) \wedge \\ & \quad \diamond Send(pl_1, pl_2, Propose(solution)) \end{aligned}$$

$$SolConf: Plane \times Plane$$

$$\begin{aligned} & \forall pl_1, pl_2: Plane \bullet \\ & \quad SolConf(pl_1, pl_2) \Leftrightarrow \\ & \quad ChangeAlt(pl_1, pl_1.pos) \vee \\ & \quad ChangeSpeed(pl_1, pl_1.speed) \end{aligned}$$

We can deduce a list of the individual behaviors for the planes:

$$Evaluate: Plane \times Plane \times Solution$$

$$\begin{aligned} & \forall pl_1, pl_2: Plane \bullet \forall pos: Pos \bullet \\ & \quad Evaluate(pl_1, pl_2, ChangAlt(pos)) \Leftrightarrow \\ & \quad ChangeAlt(pl_2, pos) \\ & \forall pl_1, pl_2: Plane \bullet \forall wayp: WayPoint \bullet \\ & \quad Evaluate(pl_1, pl_2, ChangSpeed(pl_2, speed)) \\ & \quad \Leftrightarrow ChangeSpeed(pl_2, wayp, pl1) \end{aligned}$$

$$Implementation_7$$

$$\begin{aligned} & System R: \mathbb{F} Role; Rorg: \mathbb{F} OrgRelationship \\ & organizationlinks: \mathbb{F} OrganizationLink \end{aligned}$$

$$\begin{aligned} & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \exists solution: Solution \bullet Perceive(pl_1, pl_2) \Rightarrow \\ & \quad \diamond Send(pl_1, pl_2, Propose(solution)) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \exists solution: Solution \bullet \\ & \quad Receive(pl_1, pl_2, Propose(solution)) \Rightarrow \\ & \quad pl_2.pos \in pl_1.PIsect \wedge \\ & \quad \diamond Evaluate(pl_1, pl_2, solution) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \\ & \quad Evaluate(pl_1, pl_2, solution) \Rightarrow \\ & \quad \diamond Send(pl_1, pl_2, Accept(solution)) \vee \\ & \quad \diamond Send(pl_1, pl_2, Reject(solution)) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \exists solution2: Solution \bullet \\ & \quad Send(pl_1, pl_2, Reject(solution)) \Rightarrow \\ & \quad \diamond Send(pl_1, pl_2, CounterPropose(solution2)) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \\ & \quad Receive(pl_1, pl_2, Accept(solution)) \Rightarrow \\ & \quad \diamond ChangeAlt(pl_1, pl_1.pos) \vee \\ & \quad \diamond ChangeSpeed(pl_1, pl_1.speed) \\ & \forall pl_1, pl_2: Plane \mid pl_1 \in PL \wedge pl_2 \in PL \bullet \\ & \quad \forall solution: Solution \bullet \\ & \quad Receive(pl_1, pl_2, CounterPropose(solution)) \\ & \quad \Rightarrow \diamond Evaluate(pl_1, pl_2, solution) \end{aligned}$$

Each refinement step is accompanied with a proof obligation that guarantees the refinement relation between



## Author Bios

**Amira Regayeg** was born in 1978 in Sfax, Tunisia. She received the DEA (Master) degree in computer science from the Faculty of Economic Sciences and Management, University of Sfax, Tunisia. She is now a Ph.D. student in computer science. Her current research interests include Multi-agent systems, Agent oriented software engineering, Formal Language and Specification.

**Slim Kallel** was born in 1980 in Sfax (Tunisia). He obtained his Mater degree in computer science in 2005 from the National School of Engineers in Sfax, (University of Sfax). He is now a Ph.D. student in computer science. He started his doctoral works at the National School of Engineers in Sfax (ENIS, Tunisia) and Darmstadt University of Technology (TUD, Germany) in 2006. The current research was focused on the control of Distributed Collaborative Applications, Formal Methods and Aspect Oriented Programming.

**Ahmed Hadj Kacem** was born in 1967 in Sfax, Tunisia.

He obtained his diploma of Ph.D. from the University of Paul Sabatier, Toulouse III (France) in 1995. He joined the University of Sfax as an associated professor in 1996. He participated to the initiation of many graduate courses. His current research areas include Multi-agent Systems, Agent Oriented Software Engineering, Component Oriented Software Engineering.

**Mohamed Jmaiel** was born in 1966 in Sfax, Tunisia. He obtained his diploma of engineer in Computer Science from Kiel (Germany) University in 1992 and Ph.D. from the Technical University of Berlin in 1996. He joined the National School of Engineers in Sfax as an Associate Professor in 1997. He participated to the initiation of many graduate courses at the University of Sfax. He has published more than 40 papers in refereed Journals and Conference Proceedings. His current research areas include Software Engineering, Multi-agent Systems, Component Oriented Development, Formal methods and Communication Protocols.