ForMAAD Method: An Experimental Design for Air Traffic Control

Amira Regayeg 1  Slim Kallel 1  Ahmed Hadj Kacem 2  Mohamed Jmaiel 1

1 École Nationale d’Ingénieurs de Sfax Research Unit ReDCAD
B.P.W., 3038 Sfax, Tunisia
2 Faculté des Sciences Économiques et de Gestion de Sfax Laboratory MIRACL
B.P.1088, 3018 Sfax, Tunisia

Abstract: This paper proposes a formal methodology for designing multi-agent systems based on stepwise refinements. The main contribution consists in providing a set of methodological principles and hints which help the user to build, in a systematic and incremental way, a correct design specification starting from an abstract requirements one. This method will be illustrated by developing a multi-agent solution for an air traffic control application.

Keywords: agent oriented software engineering, formal methodoloy, stepwise refinement.

1. Introduction

Many research works try to face the complexity of developing agent-based systems. The majority of the suggested solutions are extensions of either object oriented methodologies using UML, like MaSE [3], or knowledge based methodologies, like MAS-CommonKADS [7]. These approaches, which are based on semi-formal notations, do not enable formal reasoning about developed specifications.

On the other hand, some other methodologies make use of formal techniques. For instance, a very eminent work in the formalization of agents and multi-agent systems is that of Luck and d’Inverno [10]. They provide formal definitions of a four-tiered hierarchy of entities which compose an agent-based system. These mainly consider static properties of agents and entirely ignore behavioral properties. These latter are dealt with in DESIRE [2] using temporal logic. DESIRE is a specification and design framework which supports agents based on recursive composition of interconnected tasks. The interaction and coordination among agents is specified as an interchange of information and control dependencies.

Most of provided formal approaches, although they contribute to the formal description and rigorous reasoning about multi-agent applications, suffer from the lack of practical assistance during the design phase. Some approaches provide methodological guidelines for developing design models. Among others, we cite the approach of Jiao [8]. He defines design steps for building related organizational models and interaction patterns starting from the goals. Other methodologies adopt techniques such as refinement [10], composition [2], or graph transformation [4] for guiding the design process.

In order to take advantage of the potential of adopting a rigorous design process, ForMAAD provides a methodology based on stepwise refinements allowing developing a design specification starting from an abstract requirements specification. These refinements are based on methodological principles and hints which help the user to build, in a systematic and incremental way, collective and individual aspects. The foundation of our methodology is the conceptual model presented in [9] using the Z notations [15]. Based on this model, we provide guidelines which help to reach the defined concepts starting from the system requirements.

The weak point in the majority of methodologies is the absence of a verification phase. This verification is completely omitted in the approaches, like Graph Processes [4]. However, in the Fallah approach [11], which is based on protocol diagrams, a set of rules are defined in order to translate these diagrams into colored Petri nets allowing, thus, their validation and verification.

In ForMAAD, we propose a compositional verification method which follows the refinement process. For each refinement step we define a proof obligation which ensures that the resulted specification is a refinement of the initial one. Technically, we derive the formulas of the initial specification from those of the resulted one. Hence, by transitivity of the refinement relation, we guarantee that final design specification refines the requirements specification of the system. In order to automatically perform the proof obligation we make use of the Z/EVES [12] environment which provides syntactic and type checking as well as proving of theorems.

This paper is organized as follows. In section 2, we define the specification language and its semantics. Then, in section 3 we present in details our design approach. We present in section 4 a multi-agent solution for the air traffic control. Finally, we conclude and draw perspectives for our work.

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 behavior 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 which we have detail in [14]. 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 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:

\[
\begin{array}{c}
\text{SchemaName} \\
\text{Declarations} \\
\text{Predicates}
\end{array}
\]

### 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.](image)

#### 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
global behavior in terms of precedence order between the local goals. Next, we have to define appropriate individual behavior of each agent.

All these steps will be given in details in the following. Our primary specification is the System schema. We will refine it by adding implementation details according to the proposed rules in each step.

3.2.1 Collective Level

The design of collective aspects accentuates three points: cooperation strategy, organization structure, and global behavior.

Cooperation strategy.

Step 1. Cooperation strategy definition. The definition of a cooperative strategy consists in defining a set of interrelated elementary (local) goals. In order to determine these goals, we iterate rewriting the common goal until reaching its conjunctive normal form. This latter is a conjunction of a set of clauses, where each one is a disjunction of elementary temporal formulas. Hence, we generate an AND/OR graph which summarizes the decomposition of the common objective. Its root node represents the common objective, whereas, the end-nodes represent local goals. We denote by $L$ the set of these local goals. However, it should be stressed that it is not always possible to generate a conjunctive normal form of a temporal formula. This is due to the fact that one cannot distribute the operators $\lor$ and $\land$. A solution for this problem is to consider the conjunction following these operators as a new common objective ($C_i$) for which we must find the conjunctive normal form. Then, we can describe the initial common objective by substituting $C_i$ with its conjunctive normal form.

We deduce, then, the set of local goals.

Organization structure.

The organization structure implicitly defines a control strategy to be respected by the active entities. This organization is, generally, defined in terms of temporal formulas. Here, we invent a suitable organization structure for the system to be developed by identifying the necessary roles. Then, we assign each role for one or many active entities (Agent) belonging to the system.

Step 2. Role identification. This step aims at defining the main roles needed to achieve the local goals identified in the previous step. In our method, a role is formally represented by a set of temporal formulas. Accordingly, we perform this step by grouping predicates together, which have the same name. Each group represents a role.

Step 3. Definition of organization relationships. Our purpose, in this step, is to establish organization relationships between the defined roles. In practice, we try to identify common parameters of the predicates describing the local goals of different roles. A common parameter between two roles indicates an organizational relation between them. Finally, we obtain a set of organization relationships $\text{[OrgRelationship]}$ connecting the retained roles. Here, we refer to the conceptual model presented in [9] where a generic organization structure $\text{OrgStructure}$ is defined as a dependency graph of roles.

Step 4. Role assignment. This step consists in identifying

generic agents and assigning the retained roles to them. At this stage, we do not consider any implementation details for agents. In addition, many roles may be assigned to the same agent, and a role may be assigned to several agents.

In order to allocate roles to agents, we need to specify a precedence order between the local goals composing the considered roles. This order depends on the temporal operators used for describing these local goals.

In practice, we develop the Disjunctive Normal Form of $C$ which generates a list of exclusive scenarios for its execution starting from its Conjunctive Normal Form. Each scenario is described with a sequence of atoms ($\phi$). Based on this Disjunctive Normal Form, we establish a precedence graph for each scenario. A precedence graph is a directed, acyclic graph where nodes represent activities and arcs represent causal relations between them.

Step 5. Acquaintance definition (organization links). This step is devoted to instantiate the organization relationships in order to set organization links between agents according to the role assignment. An organization link between two agents corresponds to an organization relationship between the related roles. Hence, each relation between two roles $\text{OrgRelationship}$, fixed in the previous level, leads to an organization link between the agents having these roles. An organization link is defined as follows:

$$\text{OrganizationLink}$$

$$E: \& \text{Agent}; Lg: \& \text{LocalGoal}$$

$$\#E > 2$$

Collective behavior.

Step 6. Collective behavior definition. The aim of this step is to define the collective behavior of the agents according to their roles and their organization links. This behavior is expressed in terms of global (temporal) formulas where each formula refers to more than one agent. In our method, these formulas can be deduced (invented) based on the local goals and the organization links established between agents. In general, an organization link leads to a global formula describing a causality relation between the local goals included in this link. Note that thanks to the use of temporal operators, the determination of global properties, from causal relation between goals and organization links, becomes a simple task.

3.2.2 Individual Level

The main interest of this step is to invent an individual behavior for each agent. In practice, this consists in replacing, in the specification, each global formula by individual formulas where each formula refers to only one agent. It is very important to perform this step, since each agent will be implemented separately.

Step 7. Individual behavior definition. In order to determine design decisions about the agent’s individual behaviors, we proceed as follows. We start to prove the global properties established in the previous step. In doing so, we introduce additional properties that make the proof go through. These new properties lead to a refined specification with additional structures and/or properties, to which the same derivation process needs to be applied in turn. The process ends, when

\[^1\] marks the end of the description of a step where the end is not obvious.
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.

\[
\text{Implementation} \Rightarrow \bigcup_{a \in \text{Ag}} a.\text{roles} = R
\]

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 different corridors reaching the same waypoint, then they are in a 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:

\[\text{ReqSpec} \equiv \{\text{System} \mid \forall p_{l1}, p_{l2} \in \text{PL} \bullet
\]

\[\text{Conf}(p_{l1}, p_{l2}) \Rightarrow \phi \text{SolveConf}(p_{l1}, p_{l2})\]

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.

\[\text{SolveConf}: \text{Plane} \times \text{Plane}\]

\[\forall p_{l1}, p_{l2} \in \text{PL} \bullet \phi \text{SolveConf}(p_{l1}, p_{l2}) \wedge
\]

\[\phi (\text{DetectionConf}(p_{l1}, p_{l2}) \wedge \text{SolveConf}(p_{l1}, p_{l2}))\]

This decomposition leads to the refined specification:

\[\text{Implementation}_0\]

**System**

\[\forall p_{l1}, p_{l2} \in \text{PL} \bullet
\]

\[\phi (\text{Conf}(p_{l1}, p_{l2}) \Rightarrow \text{DetectionConf}(p_{l1}, p_{l2}) \wedge
\]

\[\phi (\text{Negotiate}(p_{l1}, p_{l2}) \wedge \text{SolveConf}(p_{l1}, p_{l2}))\]

\[\text{NegotiateWith}(p_{l1}, p_{l2}) \wedge
\]

\[\text{NegotiateWith}(p_{l2}, p_{l1})\]
Organization structure.

Step 2. Role identification. We denote three roles: Detector, Negotiator and Solver, respectively, for the predicate names DetectionConf, NegotiateWith and SolConf.

Step 3. Definition of organization relationships. A first organization relationship rorg1 concerns Detector and Negotiator due to the presence of two common arguments (pl1 and pl2) between DetectionConf and NegotiateWith. In the same way, we define another relationship rorg2 between Detector and Solver and rorg3 relating Negotiator and Solver.

Step 4. Role assignment. DetectionConf is the first local goal that must be executed by one plane: it is the first one which detects the conflict. This plane can be pl1 or pl2. Thus, Detector can be assigned to pl1 or to pl2. Many assignment scenarios could be present. In the following, we will consider a scenario which consists in assigning Detector, Negotiator and Solver to pl1; whereas only Negotiator is assigned to pl2.

Step 5. Acquaintance definition (organization links). Each organization relationship relying two roles must be instantiated by at least one organization link between the planes having these roles. Thus, rorg1, is instantiated by ol1, rorg2 is instantiated by ol2; and rorg3 is instantiated by ol3. rorg1 is a relationship between Detector and Negotiator. So, ol1, instantiating rorg1 must rely upon a Detector plane and a Negotiator one. These two planes are pl1 and pl2. Thus, the participants in the link ol1 are pl1 and pl2. Similarly, we can find the participants in ol2 and ol3. This leads to the following refinement:

Collective behavior.

Step 6. Collective behavior definition. For ol1 connecting pl1 (the Detector) and pl2 (the Negotiator), we identify a causality relation between the local goals of the Detector role and those of the Negotiator. Hence, we present a refined schema Implementation4, regrouping the list of relations between the local goals. Each one represents a global behavior:
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:

\[ S/\text{REquiv}: \text{Send}(pl_i, pl_j, \text{inf}) \Leftrightarrow \text{Receive}(pl_i, pl_j, \text{inf}) \]

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 implementations, let be:

DetectionConf\( (pl_1, pl_2) \) 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:

DetectionConf\( (pl_1, pl_2) \) before SolConf\( (pl_1, pl_2) \)

\[ \forall pl_1, pl_2: \text{Plane} \mid pl_1 \in PL \land pl_2 \in PL \]
\[ \exists \text{solution}: \text{Solution} \]
\[ \Rightarrow \]
\[ \text{NegotiateWith}(pl_1, pl_2) \Rightarrow \forall \text{Receive}(pl_1, pl_2, \text{Propose(solution)}) \lor \forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \]
\[ \forall \text{solution}: \text{Solution} \]
\[ \Rightarrow \forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_2, pl_1, \text{solution}) \]
\[ \forall \text{solution}: \text{Solution} \]
\[ \Rightarrow \forall \text{Evaluate}(pl_2, pl_1, \text{solution}) \Rightarrow \forall \text{Send}(pl_1, pl_2, \text{Accept(solution)}) \lor \forall \text{Send}(pl_1, pl_2, \text{Reject(solution)}) \]

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

IndBehav1: DetectionConf\( (pl_1, pl_2) \) before Send\( (pl_1, pl_2, \text{Propose(solution)}) \)

IndBehav2: Receive\( (pl_1, pl_2, \text{Propose(solution)}) \) before Evaluate\( (pl_2, pl_1, \text{solution}) \)

IndBehav3: Evaluate\( (pl_2, pl_1, \text{solution}) \) before Send\( (pl_1, pl_2, \text{Accept(solution)}) \lor \forall \text{Send}(pl_1, pl_2, \text{Reject(solution)}) \)

IndBehav4: Receive\( (pl_1, pl_2, \text{Accept(solution)}) \) before SolConf\( (pl_1, pl_2) \)

Given the following decomposition of the local goals:

\[
\text{DetectionConf: Plane} \times \text{Plane}
\]
\[
\forall pl_1, pl_2: \text{Plane} \bullet \exists \text{solution}: \text{Solution} \bullet
\]
\[
\text{NegotiateWith}(pl_1, pl_2) \Rightarrow
\]
\[
\forall \text{Receive}(pl_1, pl_2, \text{Propose(solution)}) \lor \forall \text{Send}(pl_1, pl_2, \text{Propose(solution)})
\]
\[
\forall \text{solution}: \text{Solution} \bullet
\]
\[
\forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_2, pl_1, \text{solution})
\]
\[
\forall \text{solution}: \text{Solution} \bullet
\]
\[
\forall \text{Evaluate}(pl_2, pl_1, \text{solution}) \Rightarrow \forall \text{Send}(pl_1, pl_2, \text{Accept(solution)}) \lor \forall \text{Send}(pl_1, pl_2, \text{Reject(solution)})
\]

\[
\text{SolConf: Plane} \times \text{Plane}
\]
\[
\forall pl_1, pl_2: \text{Plane} \bullet \exists \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{SolConf}(pl_1, pl_2) \Rightarrow \forall \text{ChangeAlt}(pl_1, pl_1, \text{pos}) \lor \forall \text{ChangeSpeed}(pl_1, pl_1, \text{speed})
\]

\[
\text{Evaluate: Plane} \times \text{Plane} \times \text{Solution}
\]
\[
\forall pl_1, pl_2: \text{Plane} \bullet \forall \text{pos}: \text{Pos} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{Evaluate}(pl_1, pl_2, \text{ChangSpeed}(pl_1, \text{speed})) \Rightarrow \forall \text{ChangSpeed}(pl_1, \text{speed})
\]

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

\[
\text{Implementation}\]

\[
\text{System R: } \forall \text{Role; Rorg: } \forall \text{OrgRelationship organizationlinks: } \forall \text{OrganizationLink}
\]

\[
\forall pl_1, pl_2: \text{Plane} \mid pl_1 \in PL \land pl_2 \in PL \bullet \exists \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_1, pl_2, \text{solution})
\]
\[
\forall pl_1, pl_2: \text{Plane} \mid pl_1 \in PL \land pl_2 \in PL \bullet \exists \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_1, pl_2, \text{solution})
\]
\[
\forall pl_1, pl_2: \text{Plane} \mid pl_1 \in PL \land pl_2 \in PL \bullet \exists \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_1, pl_2, \text{solution})
\]
\[
\forall pl_1, pl_2: \text{Plane} \mid pl_1 \in PL \land pl_2 \in PL \bullet \exists \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution} \bullet \forall \text{solution}: \text{Solution}
\]
\[
\forall \text{Send}(pl_1, pl_2, \text{Propose(solution)}) \Rightarrow \forall \text{Evaluate}(pl_1, pl_2, \text{solution})
\]

Each refinement step is accompanied with a proof obligation that guarantees the refinement relation between
Implementation$_i$ and Implementation$_{i+1}$ as shown in [13]: (Implementation$_{i+1}$ $\subseteq$ Implementation$_i$). Finally, in order to verify the common objective achievement, we are interested to prove, by transitivity, the following theorem: it consists on solving the air conflict between two planes endowed by the behaviors described in Implementation$_i$:

**Theorem 2. VerifSpec.**

\[
\text{Implementation}_i \Rightarrow \\
\text{(Conf}(p_1, p_2) \Rightarrow \Diamond \text{SolveConf}(p_1, p_2))
\]

These theorems are proved using the Z/EVES tools.

![Fig. 2. The proof of VerifSpec theorem.](image)

### 5. Conclusion

In this paper, we proposed a formal approach (ForMAAD) for designing multi-agent applications. Our main contribution consists in providing a set of methodological hints which guide the design process. We also stressed the correctness of the obtained design with respect to the requirements specification. The introduction of a temporal model for multi-agent applications in the Z framework enabled us to exploit Z supporting tools, like Z/EVES [12], for syntax and type checking of specifications, as well as for reasoning about correctness of refinement steps. The investigation of the air traffic control allowed an illustration of the proposed approach.

It is necessary to point out that these results, even original and promising, constitute a contribution to the definition of a useful formal methodology for designing verified multi-agent applications. However, some issues, concerning the suggested approach, remain to be investigated. First, it is interesting to inspect the approach with real applications. We plan, for this matter, to develop an agent-based design for the problem of network management. Such an application requires a rather significant number of agents. Second, and concerning the refinement steps, although their verification is totally supported using the Z/EVES tool, we feel that it is necessary to set up a tool that guides the user during these steps and possibly helps him to make appropriate design decisions. This tool consists of a set of user-interfaces that will be coupled with Z/EVES.

Third, we project to complete the development process by translating a design specification towards another more concrete language, like Z-CSP.

### References


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.