A Stepwise Refinement based Development of Self-Organizing Multi-Agent Systems: Application to the Foraging Ants

Zeineb Graja, Frédéric Migeon, Christine Maurel, Marie-Pierre Gleizes, Ahmed Hadj Kacem

A Stepwise Refinement based Development of Self-Organizing Multi-Agent Systems: Application to the Foraging Ants

Zeineb Graja*

Research Laboratory on Development and Control of Distributed Applications (ReDCAD)
Faculty of Economics and Management University of Sfax, Tunisia
E-mail: zeineb.graja@redcad.org

Frédéric Migeon*, Christine Maurel
Marie-Pierre Gleizes

Toulouse Institute of Computer Science Research (IRIT)
Paul Sabatier University, Toulouse, France
E-mail: migeon@irit.fr
E-mail: maurel@irit.fr
E-mail: gleizes@irit.fr

Ahmed Hadj Kacem

Research Laboratory on Development and Control of Distributed Applications (ReDCAD)
Faculty of Economics and Management University of Sfax, Tunisia
E-mail: ahmed.hadjkacem@fsegs.rnu.tn
*Corresponding author

Abstract: This paper proposes a formal modelling for Self-Organizing Multi-Agent Systems (*SOMAS*) based on stepwise refinements, with the Event-B language and the Temporal Logic of Actions (*TLA*). This modelling allows to develop this kind of systems in a more structured manner. In addition, it enables to reason, in a rigorous way, about the correctness of the derived models both at the local level and the global level. Our work is illustrated by the foraging ants case study.

Keywords: Self-organizing multi-agent systems, foraging ants, formal verification, convergence, resilience, refinement, Event-B, Temporal Logic of Actions.

Biographical notes: Zeineb Graja received her PhD in Artificial Intelligence from the Paul Sabatier University (Toulouse) and the University of Sfax in September 2015. Her research focuses on the formal verification of self-organising multi-agent systems, the Event-B language and formal proof.

Frédéric Migeon is a Lecturer of computer science at the Paul Sabatier University of Toulouse and a researcher at the Cooperative Multi-Agent Systems team . His

research interests are related to the development of Self-Adaptive Multi-Agent Systems, formal modelling and Model Driven Architecture.
Christine Maurel is a Lecturer of computer science at the Paul Sabatier University of Toulouse and a researcher at the Cooperative Multi-Agent Systems team. Her research interests are related to formal modelling languages and Self-Adaptive Multi-Agent systems.
Marie-Pierre Gleizes is a Full Professor of computer science at the Paul Sabatier University of Toulouse and the leader of the Cooperative Multi-Agent Systems team. Her research interests are related to the design of complex systems with emergent functionality, in particular to ambient systems and their design with Self-Adaptive Multi-Agent Systems.
Ahmed Hadj Kacem is a Full Professor of computer science at the University of Sfax and a researcher of the Research Laboratory on Development and Control of Distributed Applications. His research activities are comcentrated on distributed adaptive software, in particular to the development of models, methods, tools and mechanisms for designing and validating via formal techniques these systems.

1 Introduction

Self-Organizing Multi-Agent Systems (*SOMAS*) are made of a set of autonomous entities (called agents) interacting together and situated in an environment. Each agent has a limited knowledge about the environment and possesses its own goals. The global function of the overall system emerges from the interactions between the individual entities composing the system as well as interactions between the entities and the environment ([1]).

When designing this kind of systems, two levels of observation are generally distinguished: the micro level which corresponds to the agents local behaviour and the macro level which describes the global behaviour.

Thanks to their self-organizing mechanisms, *SOMAS* are able to adjust their behaviour and cope with the environment changes, i.e. to self-adapt. *SOMAS* are generally conceived following bottom-up approaches. Thus, the focus is on the local behaviour of the agents. The global function of the system emerges based on some heuristics independent from the overall function like cooperation for example (the case of the *AMAS* theory [2]).

One of the main challenges when engineering a *SOMAS* is about giving formal assurances and guarantees particularly related to its correctness, robustness and resilience. Correctness refers to fulfilment of the different constraints on to the agents activities. Robustness ensures that the system is able to converge to its global objective [3]. Whereas resilience informs about the capability of the system to adapt when robustness fails or a better performance is possible [4].

In order to promote the acceptance of *SOMAS*, it is essential to have effective tools and methods to give such assurances. Some works propose using test and simulation techniques [5], others define metrics for evaluating the resulting behaviour of the system [6]. These techniques offer an experimental way to verify *SOMAS* but don't give any formal guarantee. Thus, our proposal is to take advantage of formal methods. This paper is a first contribution in an ambitious work having as objective to verify formally (by means of theorem proving) *SOMAS*. For this primary work, we are situated in a particular case where the emergent function of the system is known and observed via simulation. Our aim is to prove these observed properties.

We propose a formal modelling for the local behaviour of the agents based on stepwise refinement steps and the *Event-B* formalism [7]. Our refinement strategy guarantees the correctness of the system. In order to prove the desired global properties related to robustness and resilience, we use the Lamport's Temporal Logic of Actions (*TLA*) and its fairness-based proof rules. The use of *TLA* was recently proposed in [8] in the context of population protocols and fits well with *SOMAS*. Our work is illustrated by the foraging ants case study.

This paper is organized as follows. Section 2 presents a background related to the *Event-B* language, the main principles on which it is based as well as the *TLA* logic. In section 3, our refinement strategy of *SOMAS* is presented. An illustration of this strategy by the foraging ants is given in section 4. Section 5 presents a summary of related works dealing with verification of *SOMAS*. Section 6 concludes and draws future perspectives.

2 Background

2.1 Event-B

The *Event-B* formalism was proposed by J.R. Abrial [7] as an evolution of the *B* language. It allows a correct by construction development for distributed and reactive systems. *Event-B* uses set theory as a modelling notation which enables, contrary to process algebra approaches, to support scalable solutions for system modelling. In order to make formal verification, *Event-B* is based on theorem proving. This technique avoids the problem of states explosion encountered with the model checking technique.

The concept used to make a formal development with *Event-B* is that of a $model$. A model is formed of components which can be of two types: $machine$ and $context$. A context describes the static part of the model and may include sets and constants defined by the user with their corresponding axioms. A machine is the dynamic part of the model and allows to describe the behaviour of the designed system. It is composed by a collection of variables v and a set of events ev_i.

The variables are constrained by conditions called $invariants$. The execution of the events must preserve these invariants. A machine may see one or more contexts, this will allow it to use all the elements defined in the seen context(s). The structures of a machine

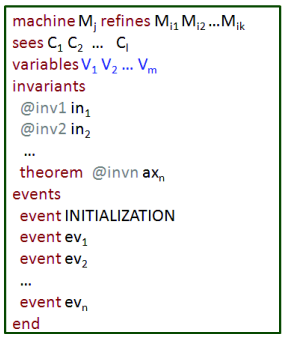

Figure 1 The machine structure in *Event-B*

and an event in *Event-B* are described as presented respectively in the figures 1 and 2. An

event is defined by a set of parameters p, the guard $G\_evi(p, v)$ which gives the necessary conditions for its activation and the action $A\_evi(p, v, v')$ which describes how variables v are substituted in terms of their old values and the parameters values. The action may

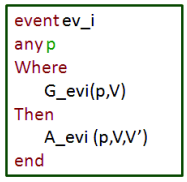

Figure 2 The event structure in *Event-B*

consist in several assignments which can be either deterministic or non-deterministic. A deterministic assignment, having the form $x := E(p, v)$, replaces values of variables x with the result obtained from the expression $E(p, v)$. A non-deterministic assignment can be of two forms: 1) $x :\in E(p, v)$ which arbitrarily chooses a value from the set $E(p, v)$ to assign to x and 2) $x :| Q(p, v, v')$ which arbitrarily chooses to assign to x a value that satisfies the predicate Q. Q is called a *before-after predicate* and expresses a relation between the previous values v (before the event execution) and the new ones v' (after the event execution).

An *Event-B* machine can be considered as an automaton which states are described by the values of the set of its variables and transitions between states are captured by the events. Based on this interpretation, T.S. Hoang and J.-R. Abrial ([9]) defined a framework for reasoning about liveness properties over an *Event-B* machine.

This framework focuses on three types of liveness properties: *existence* (some good property will always eventually happen), *progress* (the machine will always evolve from a state $P1$ to a state $P2$) and *persistence* (a property will eventually always hold). In our work, we only use the persistence property presented in the next paragraph.

The persistence property is expressed for a machine M by the formula $M \vdash \Diamond\Box P$ and is proved according to the rule $LIVE_{\Diamond\Box}$ given below. The symbols $\Box$ and $\Diamond$ are temporal operators. $\Box P$ called $always P$ means that P is always true in a given sequence of states. $\Diamond P$ called $eventually\ P$ means that P will hold in some state in the future.

$$\frac{M \vdash \nearrow P \quad M \vdash \circlearrowleft \neg P}{M \vdash \Diamond\Box P}\ LIVE_{\Diamond\Box}$$

In this rule, the first antecedent ensures that any infinite trace of the machine M will terminate on an infinite sequence of states verifying property P. The second antecedent guarantees that any finite trace of the machine M will not end with a state satisfying $\neg P$.

Proof obligations

Proof Obligations are associated with *Event-B* machines in order to prove that they satisfy certain properties. As an example, we mention the *Preservation Invariant* INV and the *Feasibility* FIS proof obligations. INV proof obligation is necessary to prove that invariants hold after the execution of each event. Proving (or discharging) FIS proof obligation means that when an event guard holds, every action can be executed. This proof

obligation is generated when actions are non-deterministic.

Refinement

This technique, allowing a *correct by construction* design, consists in adding details gradually while preserving the original properties of the system. The refinement relates two machines, an *abstract* machine and a *concrete* one. Data refinement consists in replacing the abstract variables by the concrete ones. In this case, the refinement relation is defined by a particular invariant called *gluing invariant*. The refinement of an abstract event is performed by strengthening its guard and reducing non determinism in its action. The abstract parameters can also be refined. In this case, we need to use *witnesses* describing the relation between the abstract and the concrete parameters. The correctness of the refinement is guaranteed essentially by discharging GRD (GuaRD) and SIM (SIMulation) proof obligations. GRD states that the concrete guard is stronger than the abstract one. SIM states that the abstract event can simulate the concrete one and preserves the corresponding gluing invariants. An abstract event can be refined by more than one event. In this case, we say that the concrete event is *split*. In the refinement process, new events can be introduced. In order to preserve the correctness of the model, we must prove that these new introduced events do not take the control for ever; i.e. they will *terminate* at a certain point or are *convergent*. This is ensured by the means of a *variant* –a numerical expression or a finite set– that should be decreased by each execution of the convergent events.

B-event is supported by the *Rodin* platform[1] which provides considerable assistance to developers by automating the generation and verification of all necessary proof obligations.

2.2 Temporal Logic of Actions (TLA)

TLA combines temporal logic and logic of actions for specifying and reasoning about concurrent and reactive discrete systems [10]. Its syntax is based on four elements [8].

1. constants, and constant formulas, i.e. functions and predicates over these,
2. state formulas for reasoning about states, expressed over variables as well as constants,
3. transition or action formulas for reasoning about (before-after) pairs of states, and
4. temporal predicates for reasoning about traces of states. These are constructed from the other elements and certain temporal operators.

In the remainder of this section, we give some concepts that will be used further in sections 3 and 4.

Stuttering step

A stuttering step on an action A under the vector variables f occurs when either the action A occurs or the variables in f are unchanged. We define the stuttering operator $[A]$ as: $[A]_f \mathrel{\widehat{=}} A \vee (f' = f)$. In a dual way, $\langle A \rangle$ asserts that A occurs and at least one variable in f changes. $\langle A \rangle_f \mathrel{\widehat{=}} A \wedge (f' \neq f)$.

$$\frac{F \wedge (c \in S) \Rightarrow (H_c \rightsquigarrow (G \vee \exists d \in S.(c \succ d) \wedge H_d))}{F \Rightarrow ((\exists c \in S.H_c) \rightsquigarrow G)} \; LATTICE$$

Figure 3 $LATTICE$ rule of *TLA*

Fairness

Fairness asserts that if a certain action is enabled, then it will eventually be executed. Two types of fairness can be distinguished.

1. Weak Fairness for action A denoted $WF_f(A)$ asserts that if an action A is constantly activated, then it will be eventually executed.

$$\boxed{WF_f(A) \mathrel{\widehat{=}} \Diamond\Box Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f}$$

2. Strong Fairness for action A denoted $SF_f(A)$ asserts that if an action A is often activated, it will be eventually executed.

$$\boxed{SF_f(A) \mathrel{\widehat{=}} \Box\Diamond Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f}$$

$Enabled\langle A\rangle_f$ asserts that it is possible to execute the action $\langle A\rangle_f$.

In addition, we define the $leads\ to$ operator: $P \rightsquigarrow Q \mathrel{\widehat{=}} \Box(P \Rightarrow \Diamond Q)$, meaning that whenever P is true, Q will eventually become true.

Proof rules for simple TLA

We consider the three proof rules $LATTICE$ (figure 3), $WF1$ and $SF1$ (figure 4) proposed in [10]. $LATTICE$ is an inductive proof rule in which F, G, H_c and H_d denote *TLA* formulas, S represents a given set and $\succ$ is a partial order relation defined on the set S. Informally, this rule means that provided that it is possible to move from a state satisfying the formula H_c to a state satisfying the formula G or to move to a state wherein the formula H_d is true for a value d ($d < c$), it is guaranteed by induction that formula G will be reached.

$WF1$ gives the conditions under which weak fairness assumption of action A is sufficient to prove $P \rightsquigarrow Q$. Condition $WF1.1$ describes a progress step where either state P or Q can be produced. Condition $WF1.2$ describes the inductive step where $\langle A\rangle_f$ produces state Q. Condition $WF1.3$ ensures that $\langle A\rangle_f$ is always enabled.

$SF1$ gives the necessary conditions to prove $P \rightsquigarrow Q$ under strong fairness assumption. The two first conditions are similar to $WF1$. The third condition ensures that $\langle A\rangle_f$ is eventually, rather than always, enabled. In these two rules, N represents a disjunction of actions.

3 Formal modelling of *SOMAS*

The formal modelling is based on two levels of abstraction; i.e. the micro level which corresponds to the local behaviour of the agents and the macro level which describes the

$$\frac{\begin{array}{ll} WF1.1 & P \wedge [N]_f \Rightarrow (P' \vee Q') \\ WF1.2 & P \wedge \langle N \wedge A \rangle_f \Rightarrow Q' \\ WF1.3 & P \Rightarrow Enabled\langle A \rangle_f \end{array}}{\Box[N]_f \wedge WF_f(A) \Rightarrow P \rightsquigarrow Q} \; WF1$$

$$\frac{\begin{array}{ll} SF1.1 & P \wedge [N]_f \Rightarrow (P' \vee Q') \\ SF1.2 & P \wedge \langle N \wedge A \rangle_f \Rightarrow Q' \\ SF1.3 & \Box P \wedge \Box[N]_f \Rightarrow \Diamond Enabled\langle A \rangle_f \end{array}}{\Box[N]_f \wedge SF_f(A) \Rightarrow P \rightsquigarrow Q} \; SF1$$

Figure 4 Proof rules $WF1$ and $SF1$ for *TLA*

global behaviour of the system. In this section, we identify the main properties that must be ensured when designing a *SOMAS* according to these levels. We give also a refinement strategy allowing to ensure the proof of these properties.

3.1 Formal modelling of the micro level

The main concern at this level is the design of the behaviour of the agents and their interactions. We consider that the agents interact via their environment. Thus, we give first the environment definition and then the agent and the *SOMAS* definitions.

3.1.1 Formal modelling of the environment

We suppose that the environment is composed by a set of m elements noted $l_1, ...l_m$. The environment state is described by the states of these different elements. We denote by E_{change} the environment actions changing these elements. Formally, the environment is described by the automaton $E = (SE, SE_{init}, TE, \delta E)$ where:

- SE the set the environment states.
  $SE = \prod_{i:1..m} Sl_i$ where Sl_i is the state of the l_i element.
- $SE_{init} \in SE$ denotes the environment initial state.
- TE a labels set formalising the environment actions.
  $TE = E_{change}$
- δE the set of all the possible transitions between the environment states.
  $\delta E \subseteq SE \times TE \times SE$

In *Event-B*, the environment dynamic (E_{change}) is formalised by a set of events for which the action is described by the *before-after predicate* $EnvironmentChange(l, l')$.

3.1.2 Formal modelling of the agents local behaviour

In a very abstract way, the behaviour of each agent is composed by three steps: the agent senses information from the environment (perception step), makes a decision according to these perceptions (decision step) and finally performs the chosen action (action step). We refer to these steps as the $perceive - decide - act$ cycle. Thus, an agent is characterized

by the representations of the environment that it possesses (A_{rep}), the set of decision rules telling it which decisions to make (A_{decide}), the set of actions it can perform ($A_{perform}$) and the set of operations allowing it to update its representations of the environment ($A_{perceive}$).

Moreover, an agent is identified by its intrinsic characteristics such as the representations it has on itself (A_{prop}), the state of its sensors (A_{sens}) and the state of its actuators (A_{act}).

More formally, an agent is described by an automaton $A = (SA, SA_{init}, TA, \delta A)$ where,

- SA is the set of the agent states,
  $SA = A_{rep} \times A_{prop} \times A_{sens} \times A_{act}$
- $SA_{init} \in SA$ denotes the agent initial state,
- TA is a set of labels representing the transitions between the agent states. Each transition represents a step of the agent life cycle,
  $TA = A_{perceive} \cup A_{decide} \cup A_{perform}$
- δA is the set of all the possible agent state transitions.
  $\delta A \subseteq SA \times TA \times SA$

3.1.3 Formal modelling of a SOMAS

A *SOMAS* composed by n agents $A_1, A_2, ...A_n$ and situated in an environment is modelled by means of the automaton $SYSTEM = (S, S_{init}, T, \delta)$ where,

- S denotes the set of the system state. It is derived from the agents states and the environment state. $S = \prod_{i:1..n} SA_i \times SE$.
- S_{init} is the initial system state. $S_{init} = \prod_{i:1..n} SA_{i,init} \times SE_{init}$ with $S_{init} \in S$.
- T is the set of the system states transitions. These transitions are obtained from the agents and the environment transitions.
  $T = \bigcup_{i:1..n} TA_i \cup TE$
- δ is the set of the possible transitions between the states of the system. $\delta \subseteq S \times T \times S$

In Event-B, the characteristics of the agents, their representations of the environment, the states of their sensors and actuators are modelled by means of variables. Whereas their decisions, actions and update operations are formalized by events. Hence, a *before-after-predicate* can be associated with each one of them. As a consequence, the decisions of each agent a, can be formalised by a set of *before-after-predicates* allowing changing the properties of the agent denoted $DecideAct\_i(a, A_{prop}, A'_{prop})$.

Moreover, the actions of each agent a can be considered as a set of *before-after predicates* denoted $PerformAct\_i(a, A_{prop} \cup A_{sens} \cup E, A'_{prop} \cup A'_{sens} \cup E')$. An action event is responsible for moving the agent to the perception step, thus an action event allows to activate the agent sensors. In addition, the actions of an agent can affect its properties (A_{prop}) as well as a part of its environment. Finally, the event enabling an agent to update its perceptions is described by the *before-after predicate*: $PerceiveEnvironment(a, A_{rep}, A'_{rep})$.

The automaton $SYSTEM$ modelling the *SOMAS* at the micro level is described by means of the machine $MicroLevel$ given by the figure 5. The local agents behaviour described earlier is said "correct", if the following properties are satisfied.

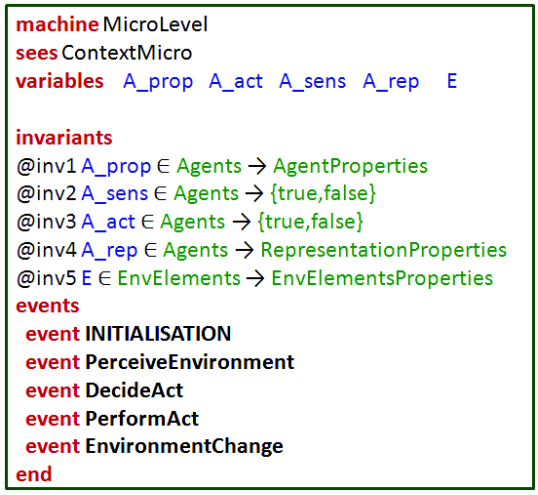

Figure 5 The *SOMAS* modelling at the micro level: the machine $MicroLevel$

- LocProp1: the behaviour of each agent is complied with the *perceive-decide-act* cycle.
- LocProp2: the agent must be not blocked in the decision step, i.e. the made decision must enable the agent to perform an action.

$$LocProp2 \mathrel{\widehat{=}} \forall a \cdot a \in Agents \wedge DecideAct\_i(a, A_{prop}, A'_{prop}) \Rightarrow \exists PerformAct\_i \cdot G\_PerformAct\_i(PerformAct\_i(a, A_{prop} \cup A_{sens} \cup E, A'_{prop} \cup A'_{sens} \cup E'))$$

- LocProp3: the agent must not be blocked in the perception step; i.e. the updated representations should allow it to make a decision.

$$LocProp3 \mathrel{\widehat{=}} \forall a \cdot a \in Agents \wedge PerceiveEnvironment\_i(a, A_{rep}, A'_{rep}) \Rightarrow \exists DecideAct\_i \cdot G\_DecideAct\_i(DecideAct\_i(a, A_{prop}, A'_{prop}))$$

In the two above formulas, $G\_PerformAct\_i()$ and $G\_DecideAct\_i()$ denote respectively the guards of the events $PerformAct\_i$ and $DecideAct\_i$.

3.2 Formal modelling of the macro level

At the macro level, the main concern is to model the observation of the macro state evolution of the system according to the agents local actions, their interactions and the environment changes. The system macro state is described in terms of an aggregation of its elements (the agents and the environment). Thus, the number of agents being in a particular state or the number of agents performing a particular behaviour can describe the system macro state.

The system can be either in a functionally adequate state; i.e. a state where the system ensures its global function or in a state where the system employs its self-organising mechanisms to cope with perturbations and to come back to a functionally adequate state. Figure 6 summarises the *SOMAS* states observed by an external observer.

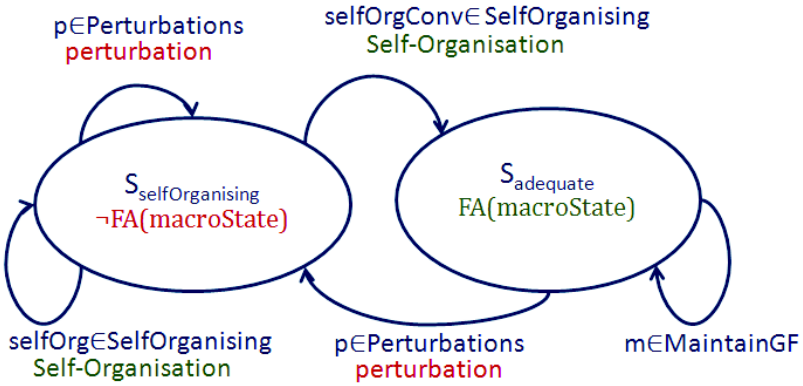

Figure 6 The abstract formalization of the system macro behaviour

$S_{adequate}$ denotes a functionally adequate state and $FA(macroState)$ the predicate defined in terms of the system macro state and describing a functionally adequate state. When the system global function is disturbed, the system switches to the $S_{selfOrganising}$ state where it applies its self-organising mechanisms to ensure again its global function.

The observed transitions between these different states are due to changes classified as follows:

- changes having no effects on the system functional adequacy represented by the set $MaintainGF$.
- changes disrupting the system operation and preventing it to ensure its function, denoted by the set $Perturbations$. In the foraging ants case study, adding food or putting obstacles in the environment constitute perturbations for the ants.
- system actions allowing it to come back to a functionality adequate state denoted by the set $SelfOrganising$. For example, the ants should ensure an efficient environment exploration in order to discover new food sources. Thus, each ant need to be able to go into new directions even if they contains less food.

Formally, a *SOMAS* observed by an external observer is defined as given in the definition 3.1

Definition 3.1: A *SOMAS* observed at the macro state is an automaton $MacroLevel = (GS, GS_{init}, GT, G\delta)$ where,

- $GS = S_{adequate} \cup S_{selfOrganising}$
  where $S_{adequate} \cap S_{selfOrganising} = \varnothing$
- GS_{init} is the system initial state.
  $GS_{init} \in GS$
- $GT = MaintainGF \cup Perturbations \cup SelfOrganising$
  where $MaintainGF$, $Perturbations$ and $SelfOrganising$ are pairwise disjoint.
- $G\delta \subseteq GS \times GT \times GS$

The observed system behaviour is defined as a sequence (which can be infinite) alternating states and actions $gs_0 \xrightarrow{gt_1} gs_1 \xrightarrow{gt_2} gs_2...$ where for all $i > 0$, $gt_i \in GT$ such as $(gs_{i-1}, \xrightarrow{gt_i}, gs_i) \in G\delta$. We denote by $\varepsilon(GS)$ the set of all the observable traces of the system. $state(\epsilon, i)$ denotes the observable sate of the system at time i in the trace $\epsilon \in \varepsilon(GS)$.

The observed macro behaviour is modelled by means of an Event-B machine in which the variables describes the global system state and the events belongs to the set $MaintainGF, SelfOrganising, Perturbations$. This machine called $MacroLevel$ is given by the figure 7.

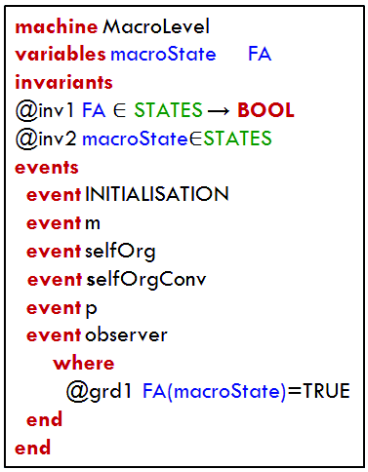

Figure 7 The *SOMAS* modelling at the macro level: the machine $MacroLevel$

Based on these definitions, the convergence and resilience properties of the macro level are defined in the following paragraphs.

3.2.1 SOMAS convergence formalization

The convergence ([3]) indicates the capacity of the system to reach it objective in the absence of disturbances. The system converges either upon initialisation, it is in a functionally adequate state or when it is guaranteed that the system will be able to achieve a functionally adequate state after a finite number of transitions.

Formally, convergence of the $MacroLevel$ system is expressed by the formula:

$$GS_{init} \in S_{adequate} \vee (\forall \epsilon \in \varepsilon(MacroLevel) \cdot \exists i \cdot state(\epsilon, i) \in S_{adequate}).$$

This definition is formulated by the use of temporal operators as follows.

$$MacroLevel \vdash \Diamond\Box\, FA(macroState)$$

The rule $LIVE_{\Diamond\Box}$ given above allows us to prove convergence property.

$$\frac{MacroLevel \vdash \nearrow FA(macroState) \quad MacroLevel \vdash \circlearrowleft \neg FA(macroState)}{MacroLevel \vdash \Diamond\Box\, FA(macroState)}\ LIVE_{\Diamond\Box}$$

The first antecedent in this formula ($MacroLevel \vdash\nearrow FA(macroState)$) ensures that every execution of the machine $MacroLevel$ terminates with an infinite sequence of states satisfying the predicate $FA(macroState)$. To prove this antecedent, three conditions need to be guaranteed:

- Define the variant $convProg$ in the machine $MacroLevel$. This variant is defined in terms of the system macro state and models the system progression towards the functional adequacy. The figure 8 supposes that the variant $convProg$ is a natural number, but it can be also a non empty finite set.

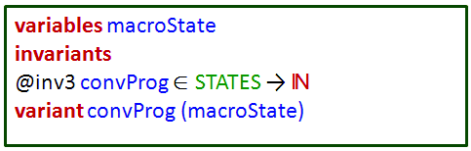

Figure 8 The variant $convProg$ definition in the machine $MacroLevel$.

- Prove that the event $selfOrgConv$ (Figure 9) decrements, in each execution, the variant $convProg$. The variable $agentStep$ is used for defining the $perceive - decide - act$ cycle. The role of the variable $agentMode$ is to synchronise the agents execution and the environment one. So that he execution of the environment occurs when each agent has accomplished one cycle. Thus these variables are used for the synchronization of the system elements.

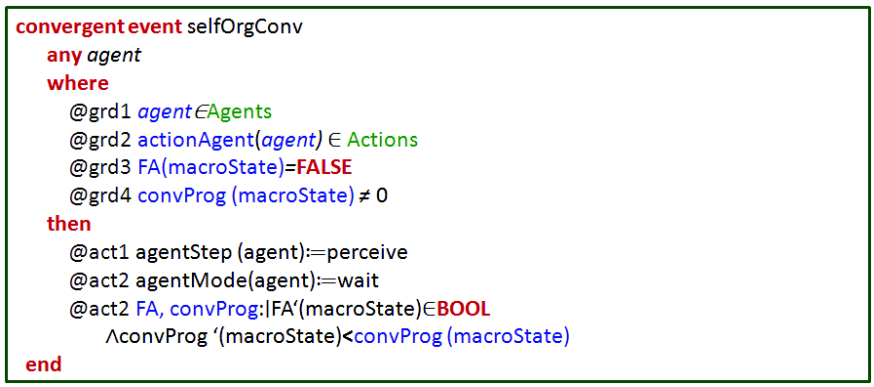

Figure 9 The event $selfOrgConv$ in the machine $MacroLevel$

- Prove that the events m and $observer$ (Figure 10) do not increment the variant $convProg$ in each execution.

The second antecedent ($MacroLevel \vdash\circlearrowleft \neg FA(macroState)$) guarantees that the machine $MacroLevel$ is deadlock-free in a state not satisfying $FA(macroState)$. This proof is formulated by the theorem given by figure 11.

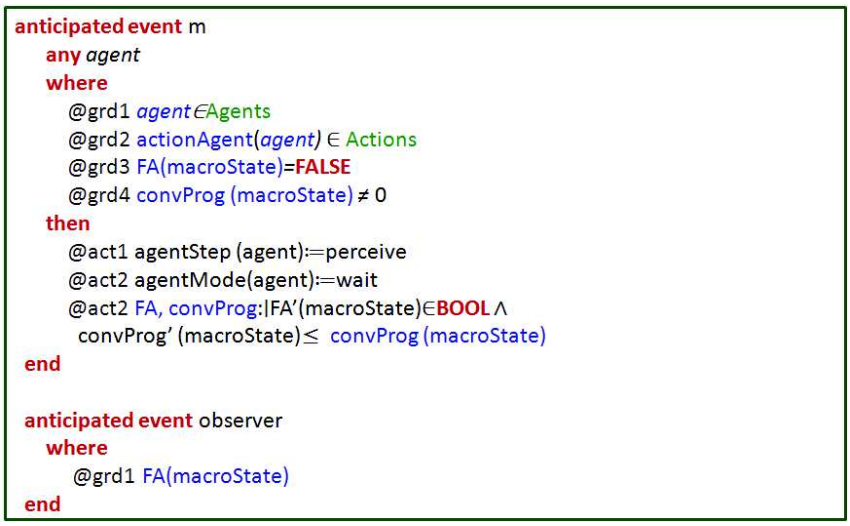

Figure 10 The events m and $observer$ in the machine $MacroLevel$.

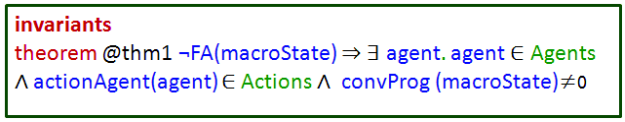

Figure 11 The deadlock-free theorem of the machine $MacroLevel$ in the state $\neg FA(macroState)$

3.2.2 SOMAS resilience formalization

The resilience ([3]) describes the ability of the system to adapt to changes and disruptions that may occur. The resilience analysis evaluates the ability of self-organization mechanisms to restore the system state after disturbances without explicitly detect an error.

We note by p a distribution coming from the agents or the environment and causing a set of disturbances noted ψ ($\psi \subseteq GS \times p \times GS$). This disruption moves the system from a functionally adequate state to a state of self-organization. The system is said resilient to the p disruption if it is able to find a functionally adequate state after this disruption.

Formally, the system $MacroLevel = (GS, GS_{init}, GT, G\delta)$ is resilient to p disruption if for any trace $\epsilon \in \varepsilon(GS)$ in which exists $i \geqslant 0$ verifying $state(\epsilon, i) = gs$ and for any self-organising state gs' verifying $(gs, p, gs') \in \psi$, the following formula is true.

$$\forall \epsilon' \in \varepsilon(MacroLevel, gs') \cdot \exists j > 0 \cdot state(\epsilon', j) \in S_{adequate}.$$

This formula is expressed by means of first order logic and temporal operators as follows:

$$MAS \vdash \Box(\neg FA(macroState) \Rightarrow \Diamond FA(macroState)). \quad (1)$$

The formula 1 can be rewritten by means of the $leadsto$ operator as follows:

$$MAS \vdash \neg FA(macroState) \rightsquigarrow FA(macroState). \quad (2)$$

In order to prove this formula, we use the *TLA* proof rule $SF1$ offering a more explicit means for expressing scheduling execution of events.

Consider the notations:

- $N = selfOrg \vee selfOrgConv \vee p \vee observer \vee m$,
- $f = convProg(macroState)$,

The rule $SF1$ is rewritten as:

$$\frac{\begin{array}{ll} SF1.1 & \neg FA(macroState) \wedge [N]_f \Rightarrow (\neg FA'(macroState) \vee FA'(macroState)) \\ SF1.2 & \neg FA(macroState) \wedge \langle N \wedge selfOrgConv \rangle_f \Rightarrow FA'(macroState) \\ SF1.3 & \Box\neg FA(macroState) \wedge \Box[N]_f \Rightarrow \Diamond Enabled\langle selfOrgConv \rangle_f \end{array}}{\Box[N]_f \wedge SF_f(selfOrgConv) \Rightarrow P \rightsquigarrow FA'(macroState)} SF1$$

The condition $SF_f(selfOrgConv)$ is a strong fairness assumption on the action of the $selfOrgConv$ event. It is expressed using temporal operators as follows:

$$SF_f(selfOrgConv) \widehat{=} \Box\Diamond Enabled\langle selfOrgConv \rangle_f \Rightarrow \Box\Diamond\langle selfOrgConv \rangle_f \quad (3)$$

Thanks to the formula 3, we prove that the action of the event $selfOrgConv$ will eventually be executed when this event is infinitely often activated (it is not constantly activated).

3.3 The refinement strategy

The formal development of *SOMAS* begins by a very abstract model representing the system as a set of agents operating according to the *Perceive-Decide-Act* cycle. These agents are situated in an environment which behaviour is modelled by the event $EnvironmentChange$. The context $ContextInit$ describes the modes $wait$ et $work$ (the set $Modes$) and the steps (the set $Steps$) in which an agent could be. This abstract model guarantees $LocProp1$. An overview of this machine is given in figure 12.

Each of the events $Perceive$, $Decide$ and $Perform$ (figure 13) corresponds to one step in the agent cycle and allows to move the considered agent from one step to the next. In this abstract machine, the $EnvironmentChange$ models the environment execution. It will be refined in the next steps for modelling the changes that can occur in the environment like perturbations for example.

This abstract model guarantees $LocProp1$ (Each agent functions according to the cycle *perceive-decide-act*). It will be subject of a three steps refinement series.

The first refinement consists in identifying the different actions performed by the agents. Thus, the refinement of the machine $InitialModel$ by $Agents1$ is achieved by splitting the $Perform$ event into the different actions an agent can perform. This refinement should ensure $LocProp2$. Figure 14 is an excerpt from the $Agents1$ machine modelling the actions of an agent.

In the second refinement step, we refine the machine $Agents1$ by the machine $Agents2$. We specify the events corresponding to the decisions that an agent can make. In addition, we describe the rules allowing the agent to decide. We also introduce the actuators of the agents. By using witness, we connect the actions introduced in the previous refinement with the corresponding decisions defined in this stage of refinement. Figure 15 describes how the decision and action events are refined.

```
machine InitialModel
sees ContextInit
variables agentStep  agentMode
Invariants
@inv1 agentStep ∈ Agents → Steps
@inv2 agentMode ∈ Agents → Modes
events
 event INITIALISATION
  then
    @act1 agentMode, agentStep :| agentMode'=(Agents×{work}) ∧ agentStep'=Agents×{perceive}
 end
 event Perceive
 event Decide
 event Perform
 event EnvironmentChange
  where
    @grd1 ∀agent agent∈Agents⇒agentMode(agent)=wait
  then
    @act1 agentMode :| agentMode'=(Agents×{work})
end
end
```

Figure 12 The initial model $InitialModel$

```
event Perceive
  any agent
  where
    @grd1 agent ∈Agents
    @grd2 agentMode (agent)=work
    @grd3 agentStep(agent)=perceive
  then
    @act1 agentStep (agent):=decide
  end
```

```
event Perform
  any agent
  where
    @grd1 agent ∈Agents
    @grd2 agentMode (agent)=work
    @grd3 agentStep(agent)=perform
  then
    @act1 agentStep (agent):=perceive
    @act2 agentMode(agent):=wait
  end
```

```
event Decide
  any agent
  where
    @grd1 agent ∈Agents
    @grd2 agentMode (agent)=work
    @grd3 agentStep(agent)=decide
  then
    @act1 agentStep (agent):=perform
end
```

Figure 13 Events $Perceive$, $Decide$ and $Perform$ of the initial machine $InitialModel$

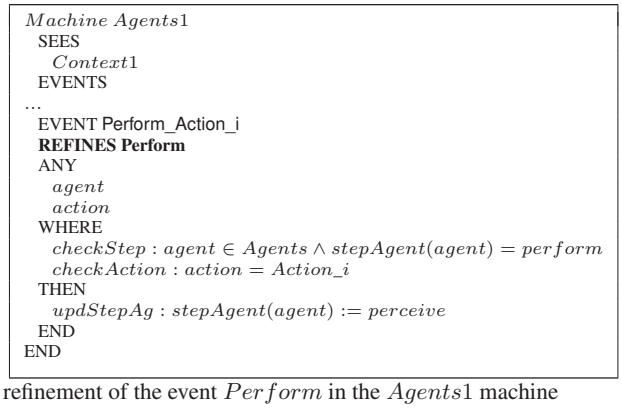

Figure 14 The refinement of the event $Perform$ in the $Agents1$ machine

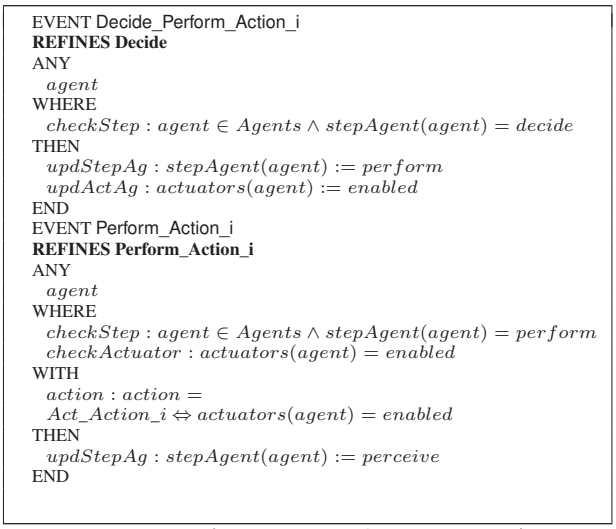

Figure 15 The refinement of the $Perform$ and $Decide$ events in the $Agents2$ machine

In the third refinement, the perceptions of the agents and the necessary events to update them are identified. As a consequence the different events related to the decisions and actions are refined and property $LocProp3$ should be satisfied.

Figure 16 shows an excerpt from the $Agents3$ machine that refines the $Agents2$ machine. The $gluInvSensorsPercept$ invariant is a gluing invariant making connection between the perception and the activation of the agent's sensors. In the context $Context3$, we define the ability $AbilityToPerceive$ (used in the $Perceive$ event in the figure 16) allowing the agent to determine the state of its local environment based on the global system state.

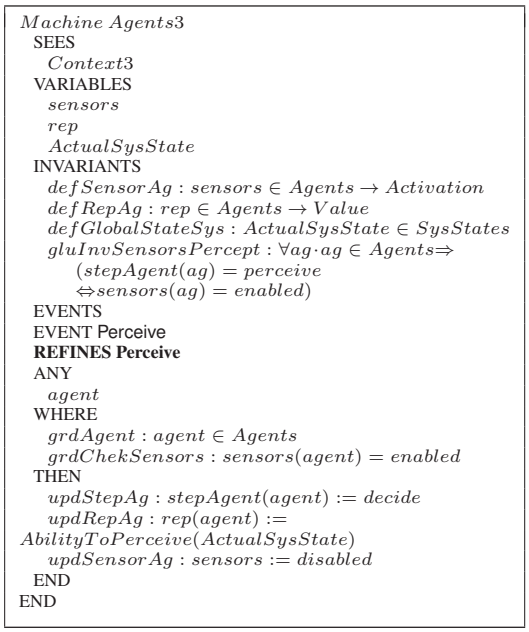

Figure 16 Refinement of the $Perceive$ event in the machine $Agents3$

4 Application to the foraging ants

The case study is a formalization of the behaviour of a foraging ants colony. The system is composed of several ants moving and searching for food in an environment. Their main goal is to bring all the food placed in the environment to their nest. Ants do not have any information about the locations of the sources of food, but they are able to smell the food which is inside their perception field. The ants interact with one another via the environment by dropping a chemical substance called *pheromone*. In fact, when an ant discovers a source of food, it takes a part of it and comes back to the nest by depositing pheromone for marking food paths. The behaviour of the system at the micro-level is described as follows. Initially, all ants are in the nest. When exploring the environment, the ant updates its representations in its perception field and decides to which location to move. When moving, the ant must avoid obstacles. According to its smells, three cases are possible:

1. the ant smells food: it decides to take the direction in which the smell of food is stronger (even if it smells some pheromone).

2. the ant smells only pheromone: it decides to move towards the direction in which the smell of pheromone is stronger.

3. the ant doesn't smell anything: it chooses its next location randomly.

When an ant reaches a source of food in one location, it collects it and comes back to the nest. If some food remains in this location, the ant drops pheromone when coming back. Arriving at the nest, the ant deposits the harvested food and begins another exploration.

In addition to the properties $LocProp1$, $LocProp2$ and $LocProp3$ (described in section 3), the following properties should be verified at the micro-level.

- $LocInv1$: the ant should avoid obstacles
- $LocInv2$: a given location cannot contain both obstacle and food.

The main global properties associated with the foraging ants system are:

- convergence property: the ants are able to bring all the food to the nest
- resilience property: when a source of food is added, the ants are able to detect it

4.1 Formalization of the ants local behaviour

Abstract model: the initial machine $Ants0$ describes an agent (each agent is an ant) operating according to the *Perceive-Decide-Act* cycle. It contains three events $Perceive$, $Decide$ and $Perform$ describing the agent behavioural rules in each step. At this very abstract level, these events are just responsible for switching an agent from one step to another. The current cycle step of each agent is depicted by the variable $stepAgent$ defined as follows.

$$inv1 : stepAgent \in Ants \rightarrow Steps$$

where $Ants$ defines the set of the agents and $Steps$ is defined by the axiom $axm1$.

The $partition$ operator allows the enumeration of the different steps of an ant.

$$axm1 : partition(Steps, \{perceive\}, \{decide\}, \{perform\})$$

As an example, we give below the event $Perform$ modelling the action step. The only action specified at this level is to switch the ant from the action step to the perception one.

EVENT Perform
  ANY
    ant
  WHERE
    $grd12 : ant \in Ants \wedge stepAgent(ant) = perform$
  THEN
    $act1 : stepAgent(ant) := perceive$
END

The proof obligations related to this machine concern essentially preservation of the invariant $inv1$ by the three events. All of them are generated and proved automatically under the Rodin platform.

First refinement: in the first refinement $Ants1$, we add the variables $QuantityFood$, $Obstacles$ modelling respectively the food and the obstacles distribution in the environment, $currentLoc$ and $load$ which give respectively the current location and the quantity of food loaded of each ant. Invariants $inv5$ and $inv3$ guarantee the properties $LocInv1$ and $LocInv2$ respectively. The notation dom is the domain of a function. The symbol $\rhd$ denotes a range subtraction. Thus, $QuantityFood \rhd \{0\}$ is a subset of the relation $QuantityFood$ that contains all pairs whose second element is not equal to zero.

$inv1 : QuantityFood \in Locations \rightarrow \mathbb{N}$
$inv2 : Obstacles \subseteq Locations \setminus \{Nest\}$
$inv3 : Obstacles \cap dom(QuantityFood \rhd \{0\}) = \varnothing$
$inv4 : currentLoc \in Ants \rightarrow Locations$
$inv5 : \forall ant \cdot ant \in Ants \Rightarrow currentLoc(ant) \notin Obstacles$
$inv6 : load \in Ants \rightarrow \mathbb{N}$

Moreover, the $Perform$ event is refined by the four following events:

1. $PerformAntsMove$: the ant moves in the environment
2. $PerformAntsMoveDropPheromone$: the ant moves and drops pheromone when coming back to the nest
3. $PerformAntsHarvestFood$: the ant picks up food
4. $PerformAntsDropFood$: the ant drops food at the nest

In the following, the event $PerformAntsMove$ is presented as an action event example.

EVENT PerformAntsMove
REFINES Perform
ANY
$ant, loc, decideAct$
WHERE
$grd12 : ant \in Ants \wedge stepAgent(ant) = perform$
$grd34 : loc \in Next(currentLoc(ant)) \wedge decideAct = move$
THEN
$act12 : stepAgent(ant) := perceive || currentLoc(ant) := loc$
END

The parameter loc is the next location to which the ant will move. It is the result of the decision process. This decision process will be modelled in the next refinement. The parameter $decideAct$ is also an abstract parameter that will be refined in the next step. It indicates what type of decision can lead to the execution of the $PerformAntsMove$ event.

The majority of the generated proof obligations are related to proving the refinement correctness (the *SIM* proof obligation) and the preservation of invariants. With the presented version of the $PerformAntsMove$ event, it is impossible to discharge the $inv5$ preservation proof obligation ($inv5$ states that an ant cannot be in a location containing obstacles). In fact, if loc belongs to the set $Obstacles$, $PerformAntsMove$ will enable ant to move to a location containing an obstacle, which is forbidden by $inv5$. In order to discharge the $inv5$ preservation proof obligation, we need to add the guard $grd5 : loc \notin Obstacles$ to $PerformAntsMove$ event. Finally, in order to

guarantee the property $LocProp2$ for the $PerformAntsMove$ event, it is necessary to add another event $PerformAntsMoveImpossible$ that refines $Perform$ and allows to take into account the situation where the move to loc is not possible because of obstacles. $PerformAntsMoveImpossible$ will just allow ant to return to the perception step. The same reasoning is applied for $PerformAntsMoveDropPheromone$. For $PerformAntsHarvestFood$, we should consider the case where the food disappears before that the ant takes it.

The Rodin tool generates 35 proof obligations for the correctness of the refinement. 85% of them are proved automatically and the rest has been proven using the interactive proof environment.

Second refinement: the second refinement $Ants2$ serves to create the links between the made decision and the corresponding action. We add the actuators of an ant: paw, $exocrinGland$, $mandible$ as well as the ant's characteristic $nextLocation$ which is updated when taking a decision. The $Decide$ event is split into five events:

1. $DecideAntsMoveExplore$: decide to move for exploring the environment
2. $DecideAntsMoveBack$: decide to come back to the nest
3. $DecAntsMoveBackDrop$: decide to come back while dropping pheromone
4. $DecideAntsHarvestFood$: decide to take the food
5. $DecideAntsDropFood$: decide to drop food at the nest

As an example, we give the event $DecideAntsMoveExplore$ below.

EVENT DecideAntsMoveExplore
REFINES Decide
ANY
ant, loc
WHERE
$grd12 : ant \in Ants \land stepAgent(ant) = decide$
$grd3 : loc \in Next(currentLoc(ant)) \land loc \neq Nest$
THEN
$act123 : stepAgent(ant) := perform||nextLocation(ant) := loc||paw(ant) := activate$
END

As a result of the event $DecideAntsMoveExplore$ execution, the ant chooses its next location and activates its paws. What is necessary now, is to link the activation of the paws with the triggering of the move action. Thus, we need to refine the event $PerformAntsMove$ by adding a $Witness$ relating the parameter $decideAct$ in the event $PerformAntsMove$ with the variable paw.

EVENT PerformAntsMove
REFINES PerformAntsMove
ANY
ant
WHERE
$grd123 : ant \in Ants \land stepAgent(ant) = perceive \land loc \in Next(currentLoc(ant))$
$grd4 : paw(ant) = activate$
WITNESSES
$decideAct : decideAct = Move \Leftrightarrow paw(ant) = activate$
$loc : loc = nextLocation(ant)$
THEN
$act12 : stepAgentCycle(ant) := perceive||currentLoc(ant) := nextLocation(ant)$
$act3 : paw(ant) := disabled$
END

The Rodin tool generates 62 proof obligations for the correctness of the refinement. 79% of them are proved automatically and the rest has been proven using the interactive proof environment.

Third refinement: at this level of refinement ($Ants3$), the ants representations about the environment are introduced. Every ant can sense food smell ($food$) as well as pheromone scent ($pheromone$). We introduce also the variable $DePhero$ modelling the distribution of pheromone in the environment.

The event $Perceive$ (here below) is refined by adding the necessary event actions for updating the perceptions of an ant.

EVENT Perceive
REFINES Perceive
ANY
ant, loc, fp, php
WHERE
$grd123 : ant \in Ants \wedge stepAgent(ant) = perceive \wedge loc = currentLoc(ant)$
$grd45 : fp \in Locations \times Locations \pfun \mathbb{N} \wedge fp = FPerc(QuantityFood)$
$grd67 : php \in Locations \times Locations \pfun \mathbb{N} \wedge php = PhPerc(DePhero)$
THEN
$act1 : stepAgentCycle(ant) := decide$
$act2 : food(ant) := \{loc \mapsto fp(loc \mapsto dir) | dir \in Next(loc)\}$
$act3 : pheromone(ant) := \{loc \mapsto php(loc \mapsto dir) | dir \in Next(loc)\}$
END

$FPerc$ (guard $grd45$) and $PhPerc$ (guard $grd67$) models the ability of an ant to smell respectively the food and the pheromone situated in its perception field. They are defined in the accompanying context of $Ants3$. After execution of the event $Perceive$, the ant acquires a knowledge about the food smell and pheromone scent for each direction from its current location.

Moreover, we split the event $DecideAntsMoveExplore$ into three events:

1. $DecideAntsMoveRandom$: decide to move to a location chosen randomly because no scent is smelt

2. $DecideAntsMoveFollowFood$: decide to move towards the direction where the food smell is maximum

3. $DecideAntsMoveFollowPheromone$: decide to move towards the direction in which the pheromone smell is maximum

This split guarantees the $LocProp3$ property for the decision concerning the move. The event $PerformAntsMove$ is also refined in order to take into account these different decisions.

The Rodin tool generates 59 proof obligations for the correctness of the refinement. 40% of them are proved automatically.

4.2 Formalization of the ant global properties

The three refinement steps described in the last section have enabled us to specify a correct individual behaviour for the ants. Let us now focus on the ability of the modelled behaviour to reach the desired global properties.

4.2.1 Proving the ants convergence

The ants convergence concerns their ability to collect and bring all food to the nest. The proof of this property needs the specification of the observer event and the termination proof of all the events representing ants actions.

The observer event

The observer event is responsible to detect if the system has achieved its overall functionality. This is a particular event with no action whose guard describes the state of the system when it reaches its goal.

For the foraging ants, we considered that the overall functionality system is to bring the food, initially dispersed in the environment, to the nest. Thus, the guard of the observer event called $AllFoodAtNest$ is described by the following expression.

$$\forall loc \cdot loc \in Locations \setminus \{Nest\} \Rightarrow QuantityFood(loc) = 0 \wedge \\ TotalFood(InitFoodDist \mapsto Locations) = QuantityFood(Nest)$$

In this expression, $QuantityFood$ is a total function giving for each location in the grid (environment) the amount of food it contains. The function $TotalFood$ returns the sum of the amount of food in the environment.

Termination of the action events

As mentioned in the sectin 1.1, the proof of events termination requires the definition of a numerical expression or a finite set, called variant. It is possible to define a single variant by machine, so this termination proof requires several refinement steps during each of which a single action event will be considered. Table 1 describes for each event, the variant needed to prove its termination.

Once the variants defined, the next step is to show that in each execution of these events, the corresponding variants will be decreased. In some cases, this proof is trivial and requires a slight modification of the guards and actions of the event responsible of the variant decrease (the case of events $PerformAntsDropFood$, $PerformAntsHarvestFood$ and $PerformAntsDropPheromone$).

In other cases, the termination proof requires the addition of new axioms ($PerformAntsMoveBack$, $PerformAntsMoveExploreFollowFood$ and $PerformAntsMoveExploreFollowPheromone$). Finally, it is necessary to suppose strong fairness assumption to prove that the event $PerformAntsMoveExploreRandom$ decrements its variant. This proof is done therefore by means of the *TLA* logic.

In the following paragraphs we show the proof termination of one event among the three groups defined above. We are interested particularly to the events $PerformAntsDropFood$, $PerformAntsMoveBack$ and $PerformAntsMoveExploreRandom$.

The event $PerformAntsDropFood$ proof termination.

To prove that the event $PerformAntsDropFood$ converges, we define the variant $AntsDroppingFoodAtNest$ as the set of ants dumping food at nest. The events responsible for changing this variant are $DecideAntsDropFood$ and $PerformAntsDropFood$. They are thus refined as shown in the following.

Event	Variant
PerformAntsDropFood	V1: the set of ants dumping food at nest
PerformAntsHarvestFood	V2: the total quantity of food in the environment except the nest
PerformAntsDropPheromone	V3: the set of ants putting pheromone
PerformAntsMoveBack	V4: the sum of the distances between the locations of ants returning to the nest and the nest
PerformAntsMoveExploreFollowFood	V5: the sum of the distances between the locations of ants moving towards a particular source of food and the location of the corresponding source of food
PerformAntsMoveExploreFollowPheromone	V6: the sum of the distances between the locations of ants following a particular pheromone smell and the location containing this pheromone
PerformAntsMoveExploreRandom	V7: the set of ants moving aleatory

Table 1 The necessary variants for proving the termination of the action events.

Event $DecideAntsDropFood$ **refinement.** This refinement models the set $AntsDroppingFoodAtNest$ updating. The update is done by adding a fully loaded ant arriving at nest in this set. This refinement consists in adding the action $addAntDrpping$ (described below) among the actions of the event $DecideAntsDropFood$.

$$addAntDrpping : AntsDroppingFoodAtNest := AntsDroppingFoodAtNest \cup \{ant\}$$

Event $PerformAntsDropFood$ **refinement.** In order to prove that the event $PerformAntsDropFood$ decreases in each execution the variant $AntsDroppingFoodAtNest$, it is necessary to add the action $removeAntsDropFood$ defined below.

$$removeAntsDropFood : AntsDroppingFoodAtNest := AntsDroppingFoodAtNest \setminus \{ant\}$$

The event $PerformAntsMoveBack$ proof termination

To prove the convergence of the event $PerformAntsMoveBack$, we define the varying $V4$ formalized by the following expression.

$$SumDistances(\{a \cdot a \in AntsApproachingNest | a \mapsto Dist(currentLoc(a) \mapsto Nest)\})$$

Informally, the variant $V4$ is the sum of the distances between the nest and the locations of all ants coming back to the nest (ants of the set $AntsApproachingNest$). In this expression, the function $SumDistances$ returns the sum of distances. $Dist$ is a function measuring the distance between two locations in the environment. When coming back to

the nest, the ant chooses the next location where it feels more the smell of the nest, i.e. the closest location to the nest. Thus, in order to allow the $PerformAntsMoveBack$ event convergence proof, we add necessary axioms stating that when an ant fully loaded and coming back to the nest chooses its next location, the distance between its current location and the nest decreases.

The event $PerformAntsMoveExploreRandom$ *proof termination*

To prove the termination of the $PerformAntsMoveExploreRandom$ event, we add the variable $AntsMovingRandom$ representing all the ants exploring at random the environment. Moreover, we refine the event $PerformAntsMoveExploreRandom$ by splitting it in two events: $PerformAntsMoveExploreRandomRef$ and $PerformAntsMoveExploreRandomConv$.

The event $PerformAntsMoveExploreRandomRef$ models an ant random movement that keeps the ant concerned throughout $AntsMovingRandom$ i.e. the next move of this ant will also be at random. The event $PerformAntsMoveExploreRandomConv$ describes a random movement allowing the ant who made it to leave the set $AntsMovingRandom$ and so allowing it to carry on exploring the environment by following food or pheromone. Proving the convergence of theses two events needs to prove the formula $P \leadsto Q$ where P denotes a state describing the current cardinality of the set $AntsMovingRandon$ and Q denotes a state where this cardinality is decreased.

Using the strong fairness rule $SF1$ of the *TLA* logic, it is possible to prove that formula.

We consider:
$N \widehat{=} PerformAntsMoveExploreRandomRef \vee$
$PerformAntsMoveExploreRandomConv$
$A_{MovRandConv} \widehat{=} PerformAntsMoveExploreRandomConv$
$P \widehat{=} card(AntsMovingRandom) = n + 1$
$Q \widehat{=} card(AntsMovingRandom) = n$
where $card(AntsMovingRandom)$ denotes the cardinality of the set $AntsMovingRandom$,
the rule $SF1$ can be rewritten as

$SF1.1$ $\quad P \wedge [N]_{AntsMovingRandom} \Rightarrow (P' \vee Q')$
$SF1.2$ $\quad P \wedge \langle N \wedge A_{MovRandConv} \rangle_{AntsMovingRandom} \Rightarrow Q'$
$SF1.3$ $\quad \Box P \wedge \Box[N]_{AntsMovingRandom} \Rightarrow \Diamond Enabled \langle A_{MovRandConv} \rangle_{AntsMovingRandom}$

$SF1$ $\quad \Box[N]_{AntsMovingRandom} \wedge SF_{AntsMovingRandom}(A_{MovRandConv}) \Rightarrow P \leadsto Q$

The formula $SF1.1$ states that the execution of one event among the events $PerformAntsMoveExploreRandomRef$ and $PerformAntsMoveExploreRandomConv$ from the P state, can either move the system to a state satisfying $P' = card(AntsMovingRandom) = n + 1$, i.e. the cardinality of the set $AntsMovingRandom$ does not change, or move the system to the state $Q' = card(AntsMovingRandom) = n$, i.e. the cardinality of the set $AntsMovingRandom$ is decreased. In the formula $SF1.2$, the action $A_{MovRandConv}$ allows to achieve the state $Q' = card(AntsMovingRandom) = n$. The formula $SF1.3$ indicates that the action $A_{MovRandConv}$ is eventually enabled.

Strong fairness assumption of the action $A_{MovRandConv}$ formulated by $SF_{AntsMovingRandom}(A_{MovRandConv})$, allows to prove the formula $P \leadsto Q$.

4.2.2 *Proving the ants resilience to new food introduction*

The models obtained so far describe the ants behaviour. They guarantee deadlock freeness at the local level and the system convergence to a state where all the food is harvested at the global level. This convergence has been proved by assuming that the environment remains unchanged. Thus, the events related to the emergence of new food sources or the appearance of new obstacles were not taken into account.

So, in this section, we are interested by the reaction of the ants when new food sources are added to the environment. We want to prove that ants are able to detect it. We express this property by using temporal logic by the formula 4.

$$ResNewFood \mathrel{\widehat{=}} \Box(\forall loc.(loc \in NewFoodLocations \wedge \exists ant.comeBackAnt(ant) = FALSE \wedge nextLocation(ant) = loc) \Rightarrow \Diamond(loc \in DetectedFoodLocations)) \quad (4)$$

In the formula 4, the variable $NewFoodLocations$ denotes the set of locations in which the food is added. The variable $DetectedFoodLocations$ denotes the set of food sources detected.

Informally, this formula means that any food source (denoted loc) added in the environment ($loc \in NewFoodLocations$) will eventually be detected. The condition ($\exists ant.comeBackAnt(ant) = FALSE \wedge nextLocation(ant) = loc$) specifies that there is an ant in exploration mode ($comeBackAnt(ant) = FALSE$) which has detected a source of food by making a move into its direction.

The introduction of a new food in the environment is modelled by the event $EnvironmentChangeAddFood$ which refines $EnvironmentChange$ and given by the figure 17.

EVENT EnvironmentChangeAddFood
REFINES EnvironmentChange
ANY
  $newFood$
WHERE
  $grd1 : \forall agent \cdot agent \in Ants \Rightarrow antMode(agent) = wait$
  $grd2 : newFood \subseteq (\{loc \cdot loc \in Locations \wedge QuantityFood(loc) = 0 | loc\} \setminus \{Nest\})$
THEN
  $act1 : antMode : | antMode' = Ants \times \{work\}$
  $act2 : NewFoodLocations := NewFoodLocations \cup newFood$
  $act3 : QuantityFood : | QuantityFood'(newFood) \in 1..QuantityFoodMax \wedge$
    $\forall loc \in Locations \setminus \{newFood\} \Rightarrow QuantityFood'(loc) = QuantityFood(loc)$
END

Figure 17 The event $EnvironmentChangeAddFood$

The action $act3$ changes the amount of food in one location by adding food in the relevant one and keeping the amount of food unchanged in the other locations.

Moreover, we model the self-organization mechanisms allowing the ants to explore more efficiently the environment by going into directions not yet visited and thus discover new source of food. Therefore, we introduce the event $PerformAntsMoveExploreAvoidCompetition$ enabling the ants to avoid going into directions containing a lot of ants even if they contain food.

In addition, it is necessary to model the events allowing the system to progress towards the desired state, i.e. the introduced source of food is detected. Thus, the event $PerformAntsMoveExploreFollowFood$ is refined by the event $PerformAntsMoveExploreFollowFoodFirstTime$ which models the detection of a source of food for the first time. A source of food is detected for the first time if it is chosen by an ant to be its next location. Thus the guard of the concrete event $PerformAntsMoveExploreFollowFoodFirstTime$ is enriched by the expression $nextLocation(ant) \in NewFoodLocations$.

Additionally, the location containing the detected source of food should be removed from the set of the locations containing the newly introduced food and added to the set of detected food locations. These actions are formalized as described in the figure 18.

$actionDetectedUpdate : DetectedFoodLocations := DetectedFoodLocations \cup \{nextLocation(ant)\}$
$actionNewFooddUpdate : NewFoodLocations := NewFoodLocations \setminus \{nextLocation(ant)\}$

Figure 18 Actions added to the actions list of the event $PerformAntsMoveExploreFollowFoodFirstTime$

Finally, the observer event $ObservResilience$ is introduced. This event contains no action and its guard describes the state on which any added food is detected, i.e. $NewFoodLocations = \varnothing$.

Once the model elements are defined, we proceed to prove the resilience property $ResNewFood$. We consider the two predicates P and $Q_{Detected}$ defined according to the cardinality of the set $NewFoodLocations$.

$P \widehat{=} card(NewFoodLocations) = n + 1$,
$Q_{Detected} \widehat{=} card(NewFoodLocations) = n$
and we want to prove the formula $P \rightsquigarrow Q_{Detected}$.
We define N and $A_{Detected}$ as follows:
$N \widehat{=} PerformAntsMoveExploreFollowFoodFirstTime \vee$
$PerformAntsMoveExploreFollowFood \vee$
$PerformAntsMoveExploreAvoidCompetition \vee$
$PerformAntsMoveExploreFollowPheromone \vee$
$PerformAntsMoveExploreRandom \vee$
$EnvironmentChangeAddFood \vee$
$ResilienceOberver$
and
$A_{Detected} \widehat{=} PerformAntsMoveExploreFollowFoodFirstTime$.
By applying $SF1$, it is possible to prove $P \rightsquigarrow Q_{Detected}$:

$SF1.1$ $P \wedge [N]_{NewFoodLocations} \Rightarrow (P' \vee Q'_{Detected})$
$SF1.2$ $P \wedge \langle N \wedge A_{Detected} \rangle_{NewFoodLocations} \Rightarrow Q'_{Detected})$
$SF1.3$ $\Box P \wedge \Box[N]_{NewFoodLocations} \Rightarrow \Diamond Enabled \langle A_{Detected} \rangle_{NewFoodLocations}$

$SF1$ $\Box[N]_{NewFoodLocations} \wedge SF_{NewFoodLocations}(A_{Detected}) \Rightarrow P \rightsquigarrow Q_{Detected}$

The condition $SF1.1$ describes a stage in which the system progresses to a state satisfying the predicate P or to a state verifying $Q_{Detected}$. The condition $SF1.2$ is an induction step during which the action $\langle A_{Detected} \rangle_{NewFoodLocations}$ (detection of food added) leads to a state satisfying $Q_{Detected}$. The condition $SF1.3$ ensures that $\langle A_{Detected} \rangle_{NewFoodLocations}$ will eventually be activated.

Assuming that the operations of adding food in the environment will eventually be arrested and by applying the rule $LATTICE$, we can prove that the set $NewFoodLocations$ will eventually be void. The achievement of this state activates the $ResilienceObserver$ event marking the end of the resilience process.

5 Related work

Related work cited in this section deals in the first part, with the formal modelling and verification of self-organization. The second part is dedicated to the presentation of works using *Event-B* for the development of adaptive systems.

5.1 Formal modelling of self-organizing systems

The verification of self-organizing systems was the subject of several research works. The majority of the proposed approaches is based on simulation and the stochastic model checking to measure the impact of various parameters on the system behaviour.

In [11], $Gardelli$ uses *stochastic Pi-Calculus* for modelling *SOMAS* for intrusion detection. This formalization was used to perform simulations using the $SPIM$ tool to evaluate the impact of certain parameters, such as the number of agents and frequency of inspections, on the system behaviour.

In [12], a hybrid approach for modelling and verifying self-organizing systems has been proposed. This approach uses stochastic simulations to model the system described as Markov chains and the technique of probabilistic model checking for verification. To avoid the state explosion problem, encountered with model-checkers, the authors propose to use approximate model-checking based on simulations. The approach was tested for the problem of collective sorting using the $PRISM$ tool.

Konur and colleagues ([13]) use also the $PRISM$ tool and probabilistic model checking to verify the behaviour of robot swarm and particularly foraging robots. The authors verify properties expressed by $PCTL$ logic (Probabilistic Computation Tree Logic) for several scenarios. These properties provide information, in particular, on the probability that the swarm acquires a certain amount of energy for a certain number of agents and in a certain amount of time. Simulations were also used to show the correlation between the density of foraging robots in the arena and the amount of energy gained.

Most of the works exposed above use the model checking technique to evaluate the behaviour of the system and adjust its parameters. Although they were able to overcome the state explosion problem and prove the effectiveness of their approaches, these works do not offer any guidance to help the designer to find the source of error in case of problems and to correct the local behaviour at the micro level. For the purpose of giving more guidance for the designer, we find that the use of *Event-B* language and its principle of refinement are very useful.

5.2 Formal modelling using the Event-B language

In [14], the authors propose a formal modelling framework for critical MAS, through a series of refinement step to derive a secure system implementation. Security is guaranteed by satisfying three properties: 1) an agent recovering from a failure cannot participate in a cooperative activity with others, 2) interactions can take place only between interconnected

agents and 3) initiated cooperative activities should complete successfully. This framework is applied to model critical activities of an emergency.

An *Event-B* modelling for fault tolerant MAS was proposed in [15]. The authors propose a refinement strategy that starts by specifying the main purpose of the system, defines the necessary agents to accomplish it, then introduces the various failures of agents and ends by introducing the communication model and error recovery mechanisms. The refinement process ensures a set of properties, mainly 1) reachability of the main purpose of the system, 2) the integrity between agents local information and global information and 3) efficiency of cooperative activities for error recovery.

The work of $Hoang$ and $Abrial$ in [16] was interested in checking liveness properties in the context of the nodes topology discovery in a network. The proposed refinement strategy allows to prove the stability property, indicating that the system will reach a stable state when the environment remains inactive. The system is called stable if the local information about the topology in each node are consistent with the actual network topology.

These works based on the $correct\ by\ construction$ approach, often providing a top-down formalization approach, have the particularity of being exempt from the combinatorial explosion problem found with the model checking technique. They have the advantage of allowing the designer to discover the restrictions to be imposed to ensure the desired properties. We share the same goals as the works presented i.e. ensuring liveness properties and simplifying the development by the use of stepwise refinements. Our refinement strategy was used to guide the modelling of individual behaviours of agents, unlike the proposed refinement strategies that use a top-down development of the entire system. We made this choice to be closer as possible to the bottom-up nature of self-organizing systems.

In this paper, we use the theorem proving technique. Our aim is double. First, to allow the designer to specify formally by stepwise refinements the local behaviour of the agents. Second, to prove the properties at the local and the global levels. Contrary to [15] and [14], we propose a bottom-up refinement strategy that we consider more appropriate and more natural to model self-organizing MAS. In fact, this strategy allows us to model the local behaviour of the agents and then to reason about the global behaviour of the system.

There are other works using $ObjectZ$ or B like [17] and [18]. In [17], the authors propose the formalization of an organisational meta-model for *SOMAS* by the use of $ObjectZ$ and the statechart diagrams. This meta-model specify formally the concepts of *role*, *interaction* and *organisation* but does not allow to reason about convergence and resilience of the system. The work presented in [18] proposes design patterns based on the B language for modelling situated agents according to the model *Influence/Reaction* which is functioning similarly to the $perceive - decide - act$ cycle adopted in our work. The objective of this formalization was to guarantee security properties but not properties related to convergence and resilience as the case of our work.

6 Conclusion

We have presented in this paper a formal modelling for *SOMAS* by means of *Event-B*. In our formalization, we consider the system in two abstraction levels: the micro and macro levels. This abstraction allows to focus the development efforts on a particular aspect of the system. We propose a stepwise refinement strategy to build a correct individual behaviour. This refinement strategy is extended in order to prove global properties such as convergence and resilience. Our proposal was applied to the foraging ants case study. While the proof

obligations were used to prove the correctness of the micro level models, it was necessary to turn to *TLA* in order to prove the properties at the macro-level. We think that this combination of *TLA* and Event-B is very promising for formal reasoning about *SOMAS*.

Our ambitions for future works are summarized in the following three points:

- Defining formal refinement patterns by means of *Event-B*. Our goal is to provide a *SOMAS* designer a helpful tool allowing the use of formal techniques in the development process.
- Integration of the proposed formal framework within *SOMAS* development methods in order to ensure formal proofs at the early stages of the system development. This integration will be made by using model-driven engineering techniques.
- Introduction of the self-organization mechanisms, based on the cooperation in particular, at the proposed refinement strategy of the local agents behaviour and the analysis of the impact of these mechanisms on the resilience of the system. For the foraging ants, for example, the objective is to analyse the ability of the ants to improve the rapidity of reaching and exploiting food thanks to their cooperative attitude. To achieve this aim, we plan to use a probabilistic approach coupled with *Event-B*.

References

[1] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organization in multi-agent systems. *Knowledge Eng. Review*, 2005.

[2] Marie-Pierre Gleizes. Self-adaptive Complex Systems (regular paper). In Massimo Cossentino, Mickael Kaisers, Karl Tuyls, and Gerhard Weiss, editors, *European Workshop on Multi-Agent Systems (EUMAS), Maastricht, The Netherlands, 13/11/2011-16/11/2011*, volume 7541, pages 114–128, http://www.springerlink.com/, 2012. Springer-Verlag.

[3] G. Di Marzo Serugendo. Robustness and dependability of self-organizing systems - a safety engineering perspective. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 254–268, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] Steven Carl Bankes. Robustness, adaptivity, and resiliency analysis. In *AAAI Fall Symposium: Complex Adaptive Systems*, volume FS-10-03 of *AAAI Technical Report*. AAAI, 2010.

[5] Carole Bernon, Marie Pierre Gleizes, and Gauthier Picard. Enhancing self-organising emergent systems design with simulation. In Gregory M. P. O'Hare, Alessandro Ricci, Michael J. O'Grady, and Oguz Dikenelli, editors, *ESAW*, volume 4457 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2006.

[6] E. Kaddoum, C. Raibulet, J-P. GeorgÃ©, G. Picard, and M-P. Gleizes. Criteria for the evaluation of self-* systems. In *Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2010.

[7] J.-R. Abrial. *Modelling in Event-B*. Cambridge University Press, 2010.

[8] Dominique Méry and Michael Poppleton. Formal modelling and verification of population protocols. In *IFM*, pages 208–222, 2013.

[9] Thai Son Hoang and Jean-Raymond Abrial. Reasoning about liveness properties in Event-B. In *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, pages 456–471, 2011.

[10] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[11] Luca Gardelli, Mirko Viroli, and Andrea Omicini. Exploring the dynamics of self-organising systems with stochastic π-calculus: Detecting abnormal behaviour in MAS. In Robert Trappl, editor, *Cybernetics and Systems 2006*, volume 2, pages 539–544, Vienna, Austria, 18–21 April 2006. Austrian Society for Cybernetic Studies. 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5). Proceedings.

[12] Matteo Casadei and Mirko Viroli. Using probabilistic model checking and simulation for designing self-organizing systems. In *SAC*, pages 2103–2104, 2009.

[13] S. Konur, D. Clare, and M. Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robot. Auton. Syst.*, 60(2):199–213, February 2012.

[14] I. Pereverzeva, E. Troubitsyna, and L. Laibinis. Formal development of critical multi-agent systems: A refinement approach. In *EDCC*, pages 156–161, 2012.

[15] I. Pereverzeva, E. Troubitsyna, and L. Laibinis. Development of fault tolerant mas with cooperative error recovery by refinement in event-b. *CoRR*, abs/1210.7035, 2012.

[16] Thai Son Hoang, Hironobu Kuruma, David A. Basin, and Jean-Raymond Abrial. Developing topology discovery in event-b. *Sci. Comput. Program.*, 74(11-12):879–899, 2009.

[17] Vincent Hilaire, Pablo Gruer, Abderrafiaa Koukam, and Olivier Simonin. Formal driven prototyping approach for multiagent systems. *IJAOSE*, 2(2):246–266, 2008.

[18] Olivier Simonin, Arnaud Lanoix, Alexis Scheuer, and François Charpillet. Specifying in B the Influence/Reaction Model to Study Situated MAS: Application to vehicles platooning. In *V2CS : First International workshop on Verification and Validation of multi-agent models for complex systems*, page 15 pages, France, November 2011.