HAL open science

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

To cite this version:

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. 2nd International workshop on Engineering Multi-Agent Systems (EMAS 2014), May 2014, Paris-, France. 10.1007/978-3-319-14484-9_3 . hal-03260611

HAL Id: hal-03260611

https://hal.science/hal-03260611v1

Submitted on 16 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Zeineb Graja[1,2], Frédéric Migeon[2], Christine Maurel[2], Marie-Pierre Gleizes[2], and Ahmed Hadj Kacem[1]

[1] Research on Development and Control of Distributed Applications Laboratory (ReDCAD)
Faculty of Economics and Management
University of Sfax, Tunisia
zeineb.graja@redcad.org, ahmed.hadjkacem@fsegs.rnu.tn

[2] Institute for Research in Computer Science in Toulouse (IRIT)
Paul Sabatier University
Toulouse, France
{graja,migeon,maurel,gleizes}@irit.fr

Abstract. This paper proposes a formal modeling for Self-Organizing Multi-Agent Systems (SOMAS) based on stepwise refinements, with the Event-B language and the Temporal Logic of Actions (TLA). This modeling 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 individual level and the global level. Our work is illustrated by the foraging ants case study.

Keywords: Self-organizing MAS, foraging ants, formal verification, refinement, Event-B, TLA.

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. Thanks to their self-organizing mechanisms, SOMAS are able to adjust their behavior and cope with the environment changes [14].

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

One of the main challenges when engineering a SOMAS is about giving assurances and guarantees related to its correctness, robustness and resilience. Correctness refers to fulfillment of the different constraints related to the

agents activities. Robustness ensures that the system is able to cope with changes and perturbations [5]. Whereas resilience informs about the capability of the system to adapt when robustness fails or a better performance is possible [2].

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 [3], others define metrics for evaluating the resulting behavior of the system [9]. Our proposal to deal with SOMAS verification is to take advantage of formal methods. We propose a formal modeling for the local behavior of the agents based on stepwise refinement steps and the *Event-B* formalism [1]. Our refinement strategy guarantees the correctness of the system. In order to prove the desired global properties related to robustness and resilience, we make use of 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 to prove liveness and convergence properties and fits well with SOMAS. Our work is illustrated with 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 and TLA. In section 3, our refinement strategy of SOMAS is presented. An illustration of this strategy on the foraging ants is given in section 4. Section 5 presents a summary of related works dealing with verification of SOMAS. Section 6 concludes the paper and draws future perspectives.

2 Background

2.1 Event-B

The *Event-B* formalism was proposed by J.R. Abrial [1] 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 modeling notation which enables, contrary to process algebra approaches, to support scalable solutions for system modeling. In order to make formal verification, *Event-B* is based on theorem proving. This technique avoids the problem of explosion in the number of the system states encountered with the model checkers.

The concept used to make a formal development is that of a *model*. A model is formed of components which can be of two types: *machine* and *context*. A context is 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 behavior 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 and an event in *Event-B* are described as follows.

```
Machine M
  SEES
    CM_i
  VARIABLES
    V_i
  INVARIANTS
    Inv(V_i)
  EVENT ev_1
  ...
  EVENT ev_i
END
```

```
EVENT ev_i
  ANY
    p
  WHERE
    grd_evi : G_evi(p, v)
  THEN
    act_evi : A_evi(p, v, v')
END
```

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 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) $x :\mid Q(p, v, v')$ 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).

Proof Obligations. Proof Obligations (POs) 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 POs. INV PO is necessary to prove that invariants hold after the execution of each event. Proving (or discharging) FIS PO means that when an event guard holds, every action can be executed. This PO 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 POs GRD and SIM. 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 gluing invariant. 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 POs.

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 [11]. Its syntax is based on four elements: 1) constants, and constant formulas - 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 [8]. In the remainder of this section, we give some concepts that will be used further in section 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: $[A]_f \mathrel{\widehat{=}} A \vee (f' = f)$. $\langle A \rangle$ asserts that A occurs and at least one variable in f changes.
$\langle A \rangle_f \mathrel{\widehat{=}} A \wedge (f' \neq f)$.

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 $WF_f(A)$; which asserts that an operation must be executed if it remains possible to do so for a long enough time and 2) Strong Fairness for action A denoted $SF_f(A)$; asserts that an operation must be executed if it is often enough possible to do so [11]. Formally $WF_f(A)$ and $SF_f(A)$ are defined as follows.

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

$\Box$ 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.
$Enabled\langle A \rangle_f$ 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 two proof rules $WF1$ and $SF2$ given below. $WF1$ 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

[1] `http://www.event-b.org/`

$\langle A \rangle_f$ 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.

$WF1$		$SF1$	
$WF1.1$	$P \wedge [N]_f \Rightarrow (P' \vee Q')$	$SF1.1$	$P \wedge [N]_f \Rightarrow (P' \vee Q')$
$WF1.2$	$P \wedge \langle N \wedge A \rangle_f \Rightarrow Q'$	$SF1.2$	$P \wedge \langle N \wedge A \rangle_f \Rightarrow Q'$
$WF1.3$	$P \Rightarrow Enabled\langle A \rangle_f$	$SF1.3$	$\Box P \wedge \Box[N]_f \Rightarrow \Diamond Enabled\langle A \rangle_f$
$\Box[N]_f \wedge WF_f(A) \Rightarrow P \rightsquigarrow Q$		$\Box[N]_f \wedge SF_f(A) \Rightarrow P \rightsquigarrow Q$	

3 Formal Modeling of Self-Organizing MAS

The formal modeling is based on two levels of abstraction; i.e. the micro level which corresponds to the local behavior of the agents and the macro level which describes the global behavior of the system. In this subsection, 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 Modeling of the Agents Local Behavior

The main concern at this level is the design of the behavior of the agents and their interactions. In a very abstract way, the behavior 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 (rep), a set of decision rules telling it which decisions to make ($decisions$), the set of actions it can perform ($actions$) and the set of operations ($perceptions$) allowing it to update its representations of the environment. Moreover, an agent is identified by its intrinsic characteristics such as the representations it has on itself ($prop$), its sensors ($sensors$) and its actuators ($actuators$). More formally, an agent is described by the following expression:

$$agent \triangleq \; < prop, rep, sensors, actuators, decisions, actions, perceptions >$$

In Event-B, the characteristics of agents, their representations of the environment, sensors and actuators are modeled 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 ag, belonging to the set of agents noted $Agents$, can be considered as a set of *before-after-predicates* denoted $Decide_i(ag, d, d')$, where d is the set of variables corresponding to the properties and actuators of ag. Moreover, the actions of each agent ag can be considered as a set of *before-after predicates* having the form $Act_i(ag, a, a')$, where a is the set of variables

corresponding to the properties and sensors of ag. Indeed, an action event is responsible for getting the agent to the perception step. Since the actions of an agent can affect its local environment, the set a can also contain variables describing the environment state. Finally, *perceptions* is the event enabling an agent to update its perceptions. It is described by the *before-after predicate*: $Perceive(ag, rep, rep')$. The local agents behavior described earlier is said "correct", if the following properties are satisfied.

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

$$\begin{aligned} LocProp2 \triangleq\ & \forall ag \cdot ag \in Agents \wedge Decide_i(ag, d, d') = TRUE \Rightarrow \\ & \exists Act_i \cdot Act_i \in actions \wedge G_Act_i(ag, a) = TRUE \end{aligned}$$

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

$$\begin{aligned} LocProp3 \triangleq\ & \forall ag \cdot ag \in Agents \wedge Perceive_i(ag, rep, rep') = TRUE \Rightarrow \\ & \exists Decide_i \cdot Decide_i \in decisions \wedge G_Decide_i(ag, d) = TRUE \end{aligned}$$

3.2 Global Properties of the Macro-level

At the macro level, the main concern is to prove that the agents behavior, designed at the micro-level, will lead to the desired global properties. The aim is to discover, in the case of proof failure, design errors and thus make the necessary corrections at the micro-level. One of the most relevant global properties that should be proved, when designing self-organizing systems, is robustness. Serugendo ([5]) defines four attributes for the analysis of robustness:

- Convergence[2]: indicates the system ability to reach its goal,
- Stability: informs about the system capacity to maintain its goal once reached,
- Speed of convergence, and
- Scalability: shows if the system is affected by the number of agents.

Besides robustness, resilience represents another relevant property that should be analyzed for SOMAS. Resilience refers to the ability of the system to self-adapt when facing changes and perturbations. The analysis of resilience allows assessment of the aptitude of self-organizing mechanisms to recover from errors without explicitly detecting an error ([5],[2]).
In this paper, we only focus on proving the stability property. We give an example from the foraging ants case study and some guidelines to prove it in the next section. The formalization and proof of the remaining properties is still an ongoing work.

[2] Convergence here is different from the convergence of an event in Event-B, i.e. termination.

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. This abstract model guarantees $LocProp1$. An overview of this machine is given in figure 1.

```
Machine Agents0
  SEES
    Context0
  VARIABLES
    stepAgent
  INVARIANTS
    defStepAg : stepAgent ∈ Agents → Steps
  EVENTS
  INITIALISATION
  THEN
    initStep : stepAgent := Agents × {perceive}
  END
  EVENT Perceive
  EVENT Decide
  EVENT Act
  ANY
    agent
  WHERE
    checkStep : agent ∈ Agents ∧ stepAgent(agent) = act
  THEN
    updStepAg : stepAgent(agent) := perceive
  END
END
```

Fig. 1. The $Agents0$ machine

The first refinement consists in identifying the different actions performed by the agents. Thus, the refinement of the machine $Agents0$ by $Agents1$ is achieved by splitting the *Act* event into the different actions an agent can perform. This refinement should ensure $LocProp2$. Figure 2 is an excerpt from the $Agents1$ machine modeling the actions of an agent.

In the second refinement step, 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 3 describes how the decision and action events are refined.

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 4 shows an excerpt from the $Agents3$ machine that refines the $Agents2$. The $gluInvSensorsPercept$ invariant is a gluing invaraint 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 4 allowing the agent to determine the state of its local environment based on the global system state.

```
Machine Agents1
  SEES
    Context1
  EVENTS
...
  EVENT Act_Action_i
  REFINES Act
  ANY
    agent
    action
  WHERE
    checkStep : agent ∈ Agents ∧ stepAgent(agent) = act
    checkAction : action = Action_i
  THEN
    updStepAg : stepAgent(agent) := perceive
  END
END
```

Fig. 2. The refinement of the event *Act* in the *Agents*1 machine

```
EVENT Decide_Perform_Action_i
REFINES Decide
ANY
  agent
WHERE
  checkStep : agent ∈ Agents ∧ stepAgent(agent) = decide
THEN
  updStepAg : stepAgent(agent) := act
  updActAg : actuators(agent) := enabled
END
EVENT Act_Action_i
REFINES Act_Action_i
ANY
  agent
WHERE
  checkStep : agent ∈ Agents ∧ stepAgent(agent) = act
  checkActuator : actuators(agent) = enabled
WITH
  action : action =
  Act_Action_i ⇔ actuators(agent) = enabled
THEN
  updStepAg : stepAgent(agent) := perceive
END
```

Fig. 3. The refinement of the *Act* and *Decide* events in the *Agents*2 machine

4 Application to the Foraging Ants

The case study is a formalization of the behavior 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 perturbations coming from the environment are mainly

```
Machine Agents3
  SEES
    Context3
  VARIABLES
    sensors
    rep
    ActualSysState
  INVARIANTS
    defSensorAg : sensors ∈ Agents → Activation
    defRepAg : rep ∈ Agents → Value
    defGlobalStateSys : ActualSysState ∈ SysStates
    gluInvSensorsPercept : ∀ag·ag ∈ Agents⇒
        (stepAgent(ag) = perceive
        ⇔sensors(ag) = enabled)
  EVENTS
  EVENT Perceive
  REFINES Perceive
  ANY
    agent
  WHERE
    grdAgent : agent ∈ Agents
    grdChekSensors : sensors(agent) = enabled
  THEN
    updStepAg : stepAgent(agent) := decide
    updRepAg : rep(agent) :=
AbilityToPerceive(ActualSysState)
    updSensorAg : sensors := disabled
  END
END
```

Fig. 4. Refinement of the *Perceive* event in the machine *Agents*3

pheromone evaporation and appearance of obstacles. The behavior 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 on a 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 *LocProp*1, *LocProp*2 and *LocProp*3 (described in section 3), the following properties should be verified at the micro-level.

- *LocInv*1: the ant should avoid obstacles
- *LocInv*2: a given location cannot contain both obstacle and food.

The main global properties associated with the foraging ants system are described in the following[3].

- $C1$: the ants are able to reach any source of food
- $C2$: the ants are able to bring all the food to the nest
- $S1$: when a source of food is detected, the ants are able to focus on its exploitation
- $R1$: the ants focusing on exploiting a source of food, are able to continue their foraging activity when this source of food suddenly disappear from the environment.

In the remainder of this section, we only focus on the properties related to the correctness ($LocProp1$, $LocProp2$, $LocProp3$, $LocInv1$ and $LocInv2$) and the stability ($S1$) of the system. The proofs of convergence and resilience are still an ongoing work. The next section illustrates the proposed refinement strategy.

4.1 Formalization of the Ants Local Behavior

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 *Act* describing the agent behavioral 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.

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

As an example, we give below the event *Act* modeling the action step. The only action specified at this level is to switch the ant to the perception step.

```
EVENT Act
  ANY
    ant
  WHERE
    grd12 : ant ∈ Ants ∧ stepAgent(ant) = act
  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 $QuFood$, *Obstacles* modeling respectively the food and the obstacles distribution in the

[3] C refers to Convergence, S to Stability and R to Resilience.

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. $\rhd$ denotes a range subtraction. Thus, $QuFood \rhd \{0\}$ is a subset of the relation $QuFood$ that contains all pairs whose second element is not equal to zero.

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

Moreover, the Act event is refined by the four following events:

1. Act_Mov: the ant moves in the environment
2. $Act_Mov_Drop_Phero$: the ant moves and drops pheromone when coming back to the nest
3. Act_Harv_Food: the ant picks up food
4. Act_Drop_Food: the ant drops of food at the nest

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

```
EVENT Act_Mov
REFINES Act
  ANY
    ant, loc, decideAct
  WHERE
    grd12 : ant ∈ Ants ∧ stepAgent(ant) = act
    grd34 : loc ∈ Next(currentLoc(ant)) ∧ 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 modeled 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 Act_Mov event.

The majority of the generated POs are related to proving the refinement correctness (the SIM PO) and the preservation of invariants. With the presented version of the Act_Mov event, it is impossible to discharge the $inv5$ preservation PO ($inv5$ states that an ant cannot be in a location containing obstacles). In fact, if loc belongs to the set $Obstacles$, Act_Mov will enable ant to move to a location containing an obstacle, which is forbidden by $inv5$. In order to discharge the $inv5$ preservation PO, we need to add the guard $grd5 : loc \notin Obstacles$ to Act_Mov event. Finally, in order to guarantee the property $LocProp2$ for the Act_Mov event, it is necessary to add another event $Act_Mov_Impossible$ that refines Act and allows to take into account the situation where the move to loc is not possible because of obstacles. $Act_Mov_Impossible$ will just allow ant to return to the perception step. The same reasoning is applied for

$Act_Mov_Drop_Phero$. For Act_Harv_Food, 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 decision made 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. Dec_Mov_Exp: decide to move for exploring the environment
2. Dec_Mov_Back: decide to come back to the nest
3. $Dec_Mov_Drop_Back$: decide to come back wile dropping pheromone
4. Dec_Harv_Food: decide to take the food
5. Dec_Drop_Food: decide to drop food in the nest

As an example, we give the event Dec_Mov_Exp above.

```
EVENT Dec_Mov_Exp
REFINES Decide
  ANY
    ant, loc
  WHERE
    grd12 : ant ∈ Ants ∧ stepAgent(ant) = decide
    grd3 : loc ∈ Next(currentLoc(ant)) ∧ loc ≠ Nest
  THEN
    act123 : stepAgent(ant) := act||nextLocation(ant) := loc||paw(ant) := activate
END
```

As a result of event Dec_Mov_Exp 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 Act_Mov by adding a $Witness$ relating the parameter $decideAct$ in the event Act_Mov with the variable paw.

```
EVENT Act_Mov
REFINES Act_Mov
  ANY
    ant
  WHERE
    grd123 : ant ∈ Ants ∧ stepAgent(ant) = perceive ∧ loc ∈ Next(currentLoc(ant))
    grd4 : paw(ant) = activate
  WITNESSES
    decideAct : decideAct = Move ⇔ 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 $DePhero$ modeling the distribution of pheromone in the environment. The event $Perceive$ (here above) 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 ∈ Ants ∧ stepAgent(ant) = perceive ∧ loc = currentLoc(ant)
    grd45 : fp ∈ Locations × Locations ⇸ ℕ ∧ fp = FPerc(QuFood)
    grd67 : php ∈ Locations × Locations ⇸ ℕ ∧ php = PhPerc(DePhero)
  THEN
    act1 : stepAgentCycle(ant) := decide
    act2 : food(ant) := {loc ↦ fp(loc ↦ dir)|dir ∈ Next(loc)}
    act3 : pheromone(ant) := {loc ↦ php(loc ↦ dir)|dir ∈ Next(loc)}
END
```

$FPerc$ (guard $grd45$) and $PhPerc$ (guard $grd67$)[4] 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 Dec_Mov_Exp into three events:

1. Dec_Mov_Rand: decide to move to a location chosen randomly because no scent is smelt
2. $Dec_Mov_Fol_F$: decide to move towards the direction where the food smell is maximum
3. $Dec_Mov_Fol_Ph$: 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 Act_Mov 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 behavior for the ants. Let us now focus on the ability of the modeled behavior to reach the desired global properties. As we already mentioned, the focus of this paper is on the stability property ($S1$) which informs about the capability of ants to exploit entirely a source of food detected.

Recall in the machine $Ants3$, we have three events describing an exploration movement namely $Act_Mov_Fol_F$, $Act_Mov_Fol_Ph$, Act_Mov_Rand

[4] In the guards $grd45$ and $grd67$, $\nrightarrow$ denotes a partial function.

plus the event Act_Harv_Food corresponding to the action of picking up food. All these events are defined according to the parameter loc which refers to any location. In order to prove the stability property, we refine these events by instantiating the parameter loc with a precise location of food $loc1$. This refinement gives rise to the machine $Ants4$. Our aim is to prove that once $loc1$ is reached, the quantity of food in it will decrease until reaching zero. In *Event-B*, this kind of reasoning is possible by proving *convergence* (or *termination*) of the event responsible for decreasing this value, i.e. the event Act_Harv_Food. For carrying out the proof of termination in *Event-B*, we need to use a variant, i.e. a natural number expression or a finite set and prove that event Act_Harv_Food decreases it in each execution. Finding an implicit variant is trivial under weak fairness assumptions on the actions of this event ([8]). In our case, the non-determinism introduced by the movement actions makes such an assumption impossible. Indeed, Act_Harv_Food is not always enabled since once an ant reaches a source of food, the others can need time to reach this source.

For proving convergence, our work is inspired by the proofs done by D. Méry and M. Poppleton in [8] where they demonstrate how to prove convergence under fairness assumption by the use of the Temporal Logic of Actions (TLA) [11] and *Event-B*.

Let us consider the two states P and $Q_{Harvest}$ describing the quantity of food on $loc1$ and defined as follows:

$$P \mathrel{\widehat{=}} Inv_{Ants4} \wedge QuFood(loc1) = n+1, Q_{Harvest} \mathrel{\widehat{=}} Inv_{Ants4} \wedge QuFood(loc1) = n$$

Inv_{Ants4} denotes the conjunction of invariants of machine $Ants4$. Proving the termination of Act_Harv_Food is reformulated by the formula:

$$P \rightsquigarrow Q_{Harvest}.$$

We define N and $A_{Harvest}$ as follows.

$N \mathrel{\widehat{=}} Act_Harv_Food \vee Act_Mov_Fol_F \vee$
$Act_Mov_Fol_Ph \vee Act_Mov_Rand$ and
$A_{Harvest} \mathrel{\widehat{=}} Act_Harv_Food.$

By applying SF1, we prove $P \rightsquigarrow Q_{Harvest}$:

$$\begin{array}{ll}
SF1.1 & P \wedge [N]_{QuFood(loc1)} \Rightarrow (P' \vee Q'_{Harvest}) \\
SF1.2 & P \wedge \langle N \wedge A_{Harvest} \rangle_{QuFood(loc1)} \Rightarrow Q'_{Harvest} \\
SF1.3 & \Box P \wedge \Box[N]_{QuFood(loc1)} \Rightarrow \Diamond Enabled \langle A_{Harvest} \rangle_{QuFood(loc1)} \\
\hline
SF1.H & \Box[N]_{QuFood(loc1)} \wedge SF_{QuFood(loc1)}(A_{harvest}) \Rightarrow P \rightsquigarrow Q_{Harvest}
\end{array}$$

Condition $SF1.1$ describes a progress step where either state P or $Q_{Harvest}$ can be produced.

Condition $SF1.2$ describes the inductive step where $\langle A_{Harvest} \rangle_{QuFood(loc1)}$ produces state $Q_{Harvest}$.

Condition $SF1.3$ ensures that $\langle A_{Harvest} \rangle_{QuFood(loc1)}$ will be *eventually* enabled. Note that both conditions $SF1.1$ and $SF1.2$ do not contain any temporal

operator. As a consequence, they are expressible in Event-B. $SF1.3$ is a temporal formula that can be expressed in the *leads to* form. Thus, we can define SF1.31 as:

$$SF1.31 \mathrel{\widehat{=}} \Box[N]_{QuFood(loc1)} \Rightarrow P \rightsquigarrow \Diamond Enabled\langle A_{Harvest}\rangle_{QuFood(loc1)}$$

To demonstrate that condition $SF1.31$ is true, we need to prove that the formula $\Diamond Enabled\langle AHarvest\rangle_{QuFood(loc1)}$ holds.

Ants are able to reach food thanks to their movements for following food. Thus if we assume that once an ant smells food, it will be able to follow it (we do not consider case where food disappears suddenly), we can argue that the event Act_Harv_Food is always eventually $Enabled$. Consequently, we can prove $SF1.31$ under weak fairness assumption.
We consider:
$Q_{followFood} \mathrel{\widehat{=}} Enabled\langle A_{Harvest}\rangle_{QuFood(loc1)}$ and
$A_{FollowFood} \mathrel{\widehat{=}} Act_Follow_Food.$
We apply $WF1$:

$$\begin{array}{ll} WF1.311 & P \wedge [N]_{QuFood(loc1)} \Rightarrow (P' \vee Q'_{FollowFood}) \\ WF1.312 & P \wedge \langle N \wedge A_{FollowFood}\rangle_{QuFood(loc1)} \Rightarrow Q'_{FollowFood} \\ WF1.313 & P \Rightarrow Enabled\langle A_{FollowFood}\rangle_{QuFood(loc1)} \\ \hline \end{array}$$

$$WF1.31 \quad \Box[N]_{QuFood(loc1)} \wedge WF_{QuFood(loc1)}(A_{FollowFood}) \Rightarrow P \ \rightsquigarrow \ Q_{FollowFood}$$

$WF1.311, WF1.312$ and $WF1.313$ do not contain any temporal operator, so that they are directly expressible in *Event-B*.

5 Related Work

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

Formal Modeling of Self-organizing Systems
In [6], *Gardelli* uses *stochastic Pi-Calculus* for modeling SOMAS for intrusion detection. This formalization was used to perform simulations using the $SPIM$ tool to assess the impact of certain parameters, such as the number of agents and frequency of inspections, on the system behavior. In [4], a hybrid approach for modeling 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 ([10]) use also the $PRISM$ tool and probabilistic model checking to verify the behavior of robot swarm, 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 behavior 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 behavior 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.

Formal Modeling Using the Event-B Language

In [13], the authors propose a formal modeling 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. *Event-B* modeling for fault tolerant MAS was proposed in [12]. 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 [7] 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 techniques. 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 modeling of individual behaviors of agents, unlike the proposed refinement strategies that use a top-down development of the entire system. We made this choice to be as closely as possible to the bottom-up nature of self-organizing systems.

6 Conclusion

We have presented in this paper a formal modeling 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 behavior. This refinement strategy is extended in order to prove global properties such as robustness 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 stability property 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 four points:

- Reasoning about the convergence of SOMAS by means of TLA.
- Introduction of the self-organization mechanisms, based on the cooperation in particular, at the proposed refinement strategy of the local agents behavior 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 analyze 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*.
- Definition of design patterns for modeling and refinement of SOMAS and their application to real case studies.
- 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.

References

1. Abrial, J.R.: Modelling in Event-B. Cambridge University Press (2010)
2. Bankes, S.C.: Robustness, adaptivity, and resiliency analysis. In: AAAI Fall Symposium: Complex Adaptive Systems. AAAI Technical Report, vol. FS-10-03. AAAI (2010)
3. Bernon, C., Gleizes, M.-P., Picard, G.: Enhancing self-organising emergent systems design with simulation. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 284–299. Springer, Heidelberg (2007), `http://dblp.uni-trier.de/db/conf/esaw/esaw2006.html#BernonGP06`
4. Casadei, M., Viroli, M.: Using probabilistic model checking and simulation for designing self-organizing systems. In: SAC, pp. 2103–2104 (2009)
5. Serugendo, G.D.M.: Robustness and dependability of self-organizing systems - A safety engineering perspective. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 254–268. Springer, Heidelberg (2009), `http://dx.doi.org/10.1007/978-3-642-05118-0_18`

6. Gardelli, L., Viroli, M., Omicini, A.: Exploring the dynamics of self-organising systems with stochastic π-calculus: Detecting abnormal behaviour in MAS. In: Trappl, R. (ed.) Cybernetics and Systems 2006, April 18-21, vol. 2, pp. 539–544. Austrian Society for Cybernetic Studies, Vienna (2006), 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), Proceedings of the 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5)
7. Hoang, T.S., Kuruma, H., Basin, D.A., Abrial, J.R.: Developing topology discovery in Event-B. Sci. Comput. Program. 74(11-12), 879–899 (2009)
8. Méry, D., Poppleton, M.: Formal modelling and verification of population protocols. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 208–222. Springer, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-38613-8_15
9. Kaddoum, E., Raibulet, C., George, J.P., Picard, G., Gleizes, M.P.: Criteria for the evaluation of self-* systems. In: Workshop on Software Engineering for Adaptive and Self-Managing Systems (2010)
10. Konur, S., Clare, D., Fisher, M.: Analysing robot swarm behaviour via probabilistic model checking. Robot. Auton. Syst. 60(2), 199–213 (2012)
11. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16(3), 872–923 (1994)
12. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Development of fault tolerant MAS with cooperative error recovery by refinement in Event-B. CoRR abs/1210.7035 (2012)
13. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal development of critical multi-agent systems: A refinement approach. In: EDCC, pp. 156–161 (2012)
14. Serugendo, G.D.M., Gleizes, M.P., Karageorgos, A.: Self-organization in multi-agent systems. Knowledge Eng. Review (2005)