Formal Modelling and Verification of Cooperative Ant Behaviour in Event-B

Linas Laibinis, Elena Troubitsynå, Zeineb Graja, Frédéric Migeon, Ahmed Hadj Kacem

Formal Modelling and Verification of Cooperative Ant Behaviour in Event-B

Linas Laibinis[1], Elena Troubitsyna[1], Zeineb Graja[2,3], Frédéric Migeon[3], and Ahmed Hadj Kacem[2]

[1] Åbo Akademi University, Turku, Finland
{Linas.Laibinis,Elena.Troubitsyna}@abo.fi
[2] Research Laboratory on Development and Control of Distributed Applications (ReDCAD), Faculty of Economics and Management, University of Sfax, Tunisia
zeineb.graja@redcad.org, ahmed.hadjkacem@fsegs.rnu.tn
[3] Institute for Research in computer Science in Toulouse (IRIT), Paul Sabatier University, Toulouse, France
{zeineb.graja,frederic.migeon}@irit.fr

Abstract. Multi-agent technology is a promising approach to development of complex decentralised systems that dynamically adapt to changing environmental conditions. The main challenge while designing such multi-agent systems is to ensure that reachability of the system-level goals emerges through collaboration of autonomous agents despite changing operating conditions. In this paper, we present a case study in formal modelling and verification of a colony of foraging ants. We formalise the behaviour of cooperative ants in Event-B and verify by proofs that the desired system-level properties become achievable via agent collaboration. The applied refinement-based approach weaves proof-based verification into the formal development. It allows us to rigorously define constraints on the environment and the ant behaviour at different abstraction levels and systematically explore the relationships between system-level goals, environment and autonomous ants. We believe that the proposed approach helps to structure complex system requirements, facilitates formal analysis of various system interdependencies, and supports formalisation of intricate mechanisms of agent collaboration.

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

1 Introduction

Self-organising multi-agent systems (MAS) are decentralised systems composed of a number of autonomic actors – agents – that cooperate with each other to achieve system-level goals [6]. Each autonomic agent follows a number of rules that govern its own behaviour as well as agent interactions. The absence of a centralised controlling mechanism and a loosely-coupled system architecture enhance system adaptability. However, they also make the design of self-adaptive MAS a challenging task, since the designers should demonstrate that the desired system-level behaviour emerges from the behaviour of individual agents.

In this paper, we propose an approach to formal development of a self-organising MAS by refinement in Event-B. Event-B [1] is a formal approach for designing distributed systems correct-by-construction. The main development technique of Event-B – refinement – allows the designers to transform an abstract specification into a detailed model through a chain of correctness-preserving transformations. Each refinement step is verified by proofs guaranteeing that a refined model preserves the externally observable behaviour. Refinement also allows us to formally define relations between formal models representing the system behaviour at different levels of abstraction. Hence it constitutes a suitable mechanism for establishing relationships between the system-level goals, the behaviour of autonomic agents, and their interactions.

In this paper, we undertake a formal development of a colony of foraging agents. We adopt the systems approach [11] that promotes an integrated modelling of the system with its environment. In our modelling, we further extend the systems approach by integrating the third component – the observer. The observer detects that the system level goal has been reached and the system can successfully terminate.

We start from an abstract specification in which all three layers – the system environment (the grid with distributed food), the ant colony, and the observer are modelled in a formal abstract way. In the chain of model refinements, we introduce a detailed representation of the ant behaviour and link their actions with the changes in the environment while, at the same time, elaborating on the logical conditions of system-level goal reachability. Our models incorporate the perceive-decide-act pattern for modelling the ant behaviour as well as the ant decision rules, including the heuristics for moving and harvesting food [2]. We discuss the benefits of formal modelling, the introduced modelling assumptions, and point out the modelling aspects that require integration with other modelling techniques such as stochastic analysis and simulation.

The paper is structured as follows. In Section 2 we briefly describe the basics of self-organising MAS, while Section 3 presents our formal modelling framework – Event-B. In Section 4 we describe our case study – the colony of foraging ants – and outline the formal development strategy. Section 5 presents our formal development in detail. In Section 6 we discusses the results achieved by our approach. Finally, in Section 7 we conclude and overview the related work.

2 Self-organising Cooperative MAS

Multi-Agent Systems (MAS) exhibiting a self-organised behaviour is a promising approach to design complex decentralised software systems. The main challenge when designing self-organising MAS is to ensure that the desired system-level behaviour emerges from the interactions of individual agents. Since self-organising systems do not have a centralised controlling mechanism, each individual agent should adapt its behaviour according to its individual perception of the operating environment and the rules governing its behaviour. The mechanism of self-adaptation should be described by the means of local information. Therefore, the functionality of the overall system should emerge from the interactions between the agents [4]. While designing a MAS, we assume that each agent has a

life cycle, called the perceive-decide-act cycle, which consists of sensing its local environment, then deciding according to its own environment perceptions which actions to perform, and, finally, executing them.

This paper focuses on studying cooperative MAS. The main idea that stems from the adaptive MAS theory is to ensure that each agent acts in cooperation with its neighbours and in accordance with the state of its operational environment [8,6]. This behavioural pattern has resulted in the following three meta-rules [3] of the cooperative MAS design:

- The agent should be able to understand every received signal from its environment and its neighbours;
- The representations that the agent has about its environment should allow it to make decisions;
- The decisions that an agent make should enable it to perform an action which is useful for the other agents and the environment.

Natural self-organising systems, such as, e.g., ant colonies, provide us with the valuable behavioural patterns that can facilitate design of decentralised cooperative interaction mechanisms [6]. The individual capabilities of ants to drop pheromone, smell nest, food or other agents lead to discovery of the cooperative mechanisms to perceive the environment, make decisions and act.

Traditionally, the behaviour of self-organising systems is studied via simulation and model-checking. Simulation allows the designers to experiment with various system parameters and create certain heuristics facilitating the system design [2]. Model checking provides support in the discovery of deadlocks and property violations [7]. However, to cope with the complexity of self-organising MAS, the designer also need techniques that support not only verification, but also the development process itself. Moreover, such techniques should support disciplined development and facilitate reasoning about various aspects of the system behaviour at different levels of abstraction.

We believe that the Event-B framework provides a suitable basis for formal model-driven development of cooperative MAS. In the next section we give a brief overview of the Event-B framework, while in Section 5 we will demonstrate our approach to development of cooperative MAS in Event-B.

3 Formal Development in Event-B

The Event B formalism [1] is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. Event B is particularly suitable for modelling distributed and reactive systems and had been actively used within several EU projects for modelling complex software-intensive systems from various domains.

In Event-B, a system specification (model) consists of two parts – machine and context, as shown in Fig. 1. The dynamic part of the model — a machine – is defined using the notion of an *abstract state machine* [1]. A machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state, i.e., contains the the dynamic part (behaviour) of the modelled system. Another part of the model, called *context*, contains the static

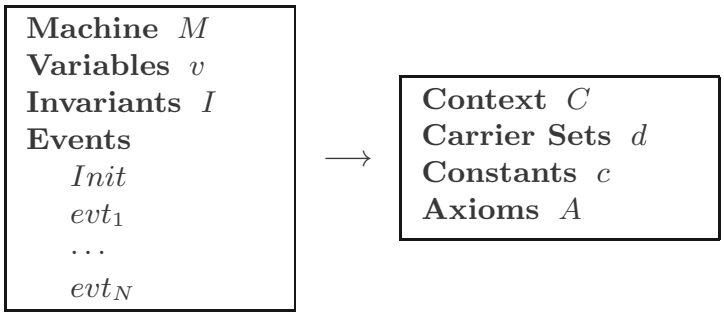

Fig. 1. Event-B machine and context

part of the system. In particular, a context can include user-defined carrier sets, constants and their properties, which are given as a list of model axioms.

The machine is uniquely identified by its name M. The state variables, v, are declared in the **Variables** clause and initialised in the $Init$ event. The variables are strongly typed by the constraining predicates I given in the **Invariants** clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$\textbf{ANY } vl \textbf{ WHERE } g \textbf{ THEN } S \textbf{ END},$$

where vl is a list of new local variables (parameters), the guard g is a state predicate, and the action S is a statement (assignment). In case when vl is empty, the event syntax becomes **WHEN** g **THEN** S **END**.

The occurrence of events represents the observable behaviour of the system. The guard defines the conditions for the action to be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution. If none of the events is enabled, the system deadlocks.

The action of an event is a parallel composition of assignments. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x :\in Set$, where Set is a set of values, or $x :| P(x, y, x')$, where P is a predicate relating initial values of x, y to some final value of x'. As a result, x can get any value belonging to Set or according to P.

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. Moreover, Event-B formal development supports data refinement, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of the refined machine formally defines the relationship between the abstract and concrete variables.

The consistency of Event-B models, i.e., verification of model well-formedness, invariant preservation as well as correctness of refinement steps, is demonstrated by discharging the relevant proof obligations. The Rodin platform [13] provides an automated support for modelling and verification. In particular, it automatically generates the required proof obligations and attempts to discharge them.

4 The Foraging Ants Case Study

Case Study Description. A colony of foraging ants is a nature-inspired cooperative MAS. The global objective of the colony is to bring the food, scattered in the environment, to the nest. Each ant has the ability to perceive, primarily by smell, different characteristics (e.g., closeness of food, nest or other ants) of its environment. The ant perception depends on its position in the environment. We assume that the stronger is the smell, the closer is the food. Each ant can autonomically perform such actions as moving, harvesting and carrying-up food, unloading food at the nest, as well as dropping pheromone. Pheromone is a chemical substance that ants put for marking paths to the discovered food. The autonomic behaviour of an ant can be summarised by the following rules:

1. Each ant starts by exploring the environment and moving randomly;
2. If it smells food, it moves to the direction where the smell is strongest;
3. If it smells pheromone, it moves to the direction where the smell is strongest;
4. When reaching the food at a certain location, an ant harvests as much food as it can carry and returns to the nest;
5. While an ant returns to the nest carrying food, it drops pheromone along the way to attract other ants to the food source.

We assume that the environment is composed of a set of connected locations. One specific location is reserved for the nest of the colony. A location might contain a certain limited amount of food. Moreover, a location can be marked by some quantity of (gradually evaporating) pheromone.

The aim of the formal development is to specify the following requirements.

- (R1) The main goal of the colony is to bring all the food to the nest;
- (R2) Before performing an action, an ant should make a decision based on its current perceptions of the environment;
- (R3) A combination of the ant perceptions should allow it to make an unambiguous decision;
- (R4) The decision an ant makes should lead to its specific action;
- (R5) An ant should avoid conflicts with other ants on the same food source;
- (R6) An ant should avoid conflicts with other ants on the same area.

The requirements (R5) and (R6) essentially mean that, given a choice, an ant should avoid the areas already exploited by other ants. This ensures a more efficient food foraging by exploring more territory for yet undiscovered food.

The Formal Refinement Strategy. Traditionally, modelling of self-organising MAS is structured according to three views:

- modelling the environment,
- modelling autonomic agents and their interactions,
- modelling the system-level properties (observer-view).

One of the advantages of Event-B is that it supports the systems [11] approach, i.e., allows us to model the system behaviour together with its environment. In this paper, we further extend the systems approach by defining our model as

an integration of the environment, agent and observer views. It allows us to formally derive the interconnections between these layers through the development process. Moreover, such an integrated modelling approach allows us to discover the constraints that the environment should satisfy, precisely define how the agent behaviour affects the environment, and link the system-level goals with both agent and environment dynamics.

Next we present the strategy that we will follow in our formal development of the case study. The main idea behind the proposed strategy is to derive a detailed model demonstrating that the local ant behaviour leads to achieving the defined system-level goal (the requirement R1). The reachability is proved as a result of formalisation of all three views: environment, agent and observer as well as their inter-relationships. The overall refinement strategy is as follows.

1. **Initial model.** The initial abstract model introduces the location grid (including the nest) with the distributed food and models the effect of ants activity – all the food is gradually transferred to the nest. Reaching this goal is eventually observed by the observer.
2. **First refinement.** The specification obtained at this level introduces into the model a representation of the ants and their behaviour, following the perceive- decide-act cycle. Moreover, we elaborate of the act stage.
3. **Second refinement.** At this level, we focus on the decision stage. We introduce different types of possible ant decisions as well as the dynamic food load the ants are carrying. This refinement allows us also to establish a link between the act and decide stages.
4. **Third refinement.** At this level, we complete refinement of the perceive-decide-act cycle by introducing different ant perceptions and the decision rules based on these perceptions.
5. **Fourth refinement.** At the final refinement step, we elaborate on the link between the cooperative ant behaviour and system-level properties. We introduce the conditions guaranteeing that an ant in a particular functioning mode (exploration, going after perceived food, returning to the nest, etc.) always gets closer to its current target.

In our refinement strategy, each subsequent refinement step elaborates on the previously-introduced models, verifying at the same time the consistency between the models. Moreover, different refinement steps focus on modelling and verifying different requirements (R1)–(R6). For instance, the second refinement step ensures the requirement (R4), while the third refinement step focuses on formalising the requirements (R3), (R5), and (R6).

5 Formal Development of the Foraging Ants Case Study

Next we present our formal development of the colony of foraging ants. Due to the space limit, we will present excerpts from our formal models and only discuss the main modelling solutions as well as verified properties.

The Initial Model. In our formal development, we aim at establishing an explicit link between the system-level goals and the ant behaviour. In our case,

we have to demonstrate that the ants will harvest all the food distributed in the environment. In the initial specification, we abstractly model the process of harvesting food and reaching the goal – reaching the state in which all the food initially distributed over the environment is transferred to the colony nest.

In the context component of our abstract model, we introduce constants and properties required to represent the grid (the environment) on which the ants are moving to harvest the food. We model the grid as a finite adirectional graph over a set of interconnected locations. The locations are modelled as the abstract set *Locations*, with $Nest$ – a fixed location in the grid – representing the nest of the colony to which the ants should bring the food.

axm1: $Nest \in Locations \ \land \ Locations \backslash \{Nest\} \neq \varnothing$
axm2: $Grid \in Locations \leftrightarrow\!\!\!\!\!\rightarrow Locations \ \land \ Grid = Grid\sim$

Here $\leftrightarrow\!\!\!\!\!\rightarrow$ designates a total relation, i.e., each location has at least one adjacent location, connected by *Grid*. *Grid* is also symmetric (the axiom axm2). To make a decision about where to move next, each ant should analyse its close vicinity – a set of nearby locations. To represent this set, we explicitly introduce the function $Next$, defined as a relational image of *Grid* for the given location *loc*.

axm3 : $Next \in Locations \rightarrow \mathbb{P}(Locations)$
axm4 : $\forall loc.\, loc \in Locations \ \Rightarrow \ Next(loc) \ = \ Grid[\{loc\}]$

Finally, in the context component we also introduce the initial food distribution in the grid, defined as a constant function over the grid locations:

axm5 : $QuantityFoodMax \in \mathbb{N}1$
axm6 : $InitFoodDistr \in Locations \rightarrow 0..QuantityFoodMax$
axm7 : $InitFoodDistr(Nest) = 0 \ \land \ (\exists loc.\, loc \in Locations \backslash \{Nest\} \ \land \ InitFoodDistr(loc) > 0)$

Here *QuantityFoodMax* is the constant restricting the maximal amount of food for any single location. In other axioms, we require that the initial food amount in the nest is equal to 0, and there is at least one location outside the nest that contains some non-zero amount of food. Without these constraints, the system-level goal would be automatically satisfied, i.e., the system would terminate right after initialisation.

In our abstract model, the main complexity lies in the context that defines the system environment. The dynamic part of the model – the machine – is rather simple. After the system initialisation, the event Change becomes enabled. It models non-deterministic changes in food distribution on the grid. Eventually the event Observer becomes enabled, indicating that all the food is now transferred to *Nest*. We also introduce two variables *GoalReached* and *QuantityFood*:

inv1 : $GoalReached \in \text{bool}$
inv2 : $QuantityFood \in Locations \rightarrow 0..QuantityFoodMax$

The current state of food distribution in the grid is modelled by the variable *QuantityFood*. It is defined as a function (array) over the grid locations, associating each location with the food amount currently stored in it. The variable is initialised by *InitFoodDistr*, introduced in the context. The boolean variable *GoalReached* indicates whether the system reached its main goal.

The system goal is reached when there is no food left in the grid, i.e.,

$$\forall loc.\, loc \in Locations \backslash \{Nest\} \;\Rightarrow\; QuantityFood(loc) \;=\; 0$$

Once this happens, the variable $GoalReached$ is assigned $TRUE$ (by Observer). This in turn disables the event Change, effectively terminating the system.

The event Change abstractly models possible changes in the grid food distribution. Essentially, it non-deterministically specifies the general tendency for the food to be transferred from non-nest locations to *Nest*. This behaviour is enforced by permitting non-deterministic decrease (or at least non-increase) of the food amount outside *Nest* or, similarly, its non-deterministic increase in *Nest*.

EVENT Change
  ANY loc, $newQF$ WHERE
    $grd1 : loc \in Locations \wedge newQF \in \mathbb{N}$
    $grd2 : loc = Nest \Rightarrow newQF \geq QuantityFood(loc)$
    $grd3 : loc \neq Nest \Rightarrow newQF \leq QuantityFood(loc)$
    $grd4 : GoalReached = FALSE$
  THEN
    $act1 : QuantityFood(loc) := newQF$
  END

Even though modelling of ants is abstracted away in the initial model, it is implicitly assumed that all the changes in the grid food distribution happen because of some ant activities. In the subsequent refinements, this relationship will be made explicit, constraining the nondeterminism of our abstract model.

The First Refinement. In our first refinement step, we introduce abstract representation of ants and their behaviour stages. In particular, we adopt the widely used pattern $Perceive \rightarrow Decide \rightarrow Act$ to model the cyclic ant behaviour. Moreover, we distinguish two groups of ants – the ants that are currently engaged in the food foraging and the ants that are resting in the nest.

In the context of the refined model, we introduce the abstract set $Ants$ to model the colony of ants. Moreover, we define the enumerated set $StepCycle = \{perceive, decide, act\}$ that defines the constants to indicate specific cyclic stages of the ant behaviour. In the dynamic part of the refined model, we introduce three new variables to model ants and their behaviour.

inv8 : $WorkingAnts \subseteq Ants$
inv9 : $AgentStage \in WorkingAnts \rightarrow StepCycle$
inv10 : $currentLoc \in Ants \rightarrow Locations$

The variable $WorkingAnts$ stores the set of the ants currently involved in food foraging. The variable $AgentStage$ indicates the current behaviour stage for each working ant. Finally, the variable $currentLoc$ associates each ant (whether working or resting) with its current location on the grid.

The abstract event Change is now refined (and renamed into Act) to model possible actions of a particular ant. The event is parameterised with a local variable ant, which is required to be a working ant in the stage act (the guard $grd5$). Note that the local variable (parameter) loc of the abstract event Change, which signified an arbitrary possible grid location, is now constrained to be equal to $currentLoc(ant)$, i.e., the current location of ant.

The ant action may also result in the ant moving to an adjacent location. The new action $act2$ specifies this: the ant may move to a new location (one of possible locations specified by $Next(currentLoc(ant))$) or stay where it is now.

EVENT Act
REFINES Change
ANY $newQF$, ant WHERE
$grd1 : newQF \in \mathbb{N}$
$grd2 : currentLoc(ant) = Nest \Rightarrow newQF \geq QuantityFood(currentLoc(ant))$
$grd3 : currentLoc(ant) \neq Nest \Rightarrow newQF \leq QuantityFood(currentLoc(ant))$
$grd4 : GoalReached = FALSE$
$grd5 : ant \in WorkingAnts \wedge AgentStage(ant) = act$
THEN
$act1 : QuantityFood(currentLoc(ant)) := newQF$
$act2 : currentLoc(ant) :\in Next(currentLoc(ant)) \cup \{currentLoc(ant)\}$
END

The new events Perceive, Decide, and NewCycle are introduced to model the cyclic ant behaviour. The events Perceive and Decide abstractly model the perceive and decide stages of the ant behaviour. These events will be elaborated (refined) in the subsequent refinement steps. The event NewCycle is enabled when all the working ants completed their act stage (i.e., executed Act). It starts a new cycle by moving all the working ants into the perceive stage.

As a part of verifying model correctness, Event-B allows us to formally prove convergence of the newly introduced events (system transitions) in the refined models. The convergence is proved by providing a natural number expression (variant) and then formally demonstrating that this expression is decreased by any execution of new events. We have proved convergence of the new events Perceive and Decide (thus guaranteeing that the act stage will be reached for all the working ants), using the following variant expression:

$$card(\{a \cdot a \in WorkingAnts \wedge AgentStage(a) = perceive\}) + card(\{a \cdot a \in WorkingAnts \wedge AgentStage(a) \in \{perceive, decide\}),$$

where *card* is the set cardinality operator.

The Second Refinement. The first refinement step has allowed us to establish the connection between the dynamical changes in the system environment (the food distribution in the grid) and the actors (ants) that cause these changes. In the second refinement, we elaborate on the ant decision stage. We also introduce the notions of ant load (i.e., the amount of food an ant is carrying) and grid pheromone distribution (i.e., quantities of ant pheromone in grid locations).

To model the outcome of the decision stage, in the model context we introduce the enumerated set $Decision = \{move, harvest, dropPheromoneAndMove, dropPheromone, dropFood, doNothing\}$, with the constants for respective ant decisions. The ant decisions are constrained by the corresponding environment and its own conditions, such as the presence of food and currently carried load. To reason about this, we add the constant $MaxLoad$ (for the maximal amount a single ant can carry) as well as the functions $TotalLoad$ and $TotalFood$, returning the total amount of food for a set of ants or a set of locations.

axm2 : $MaxLoad \in \mathbb{N}1$
axm3 : $TotalLoad \in (Ants \to \mathbb{N}) \to (\mathbb{P}(Ants) \to \mathbb{N})$
axm4 : $TotalFood \in (Locations \to \mathbb{N}) \to (\mathbb{P}(Locations) \to \mathbb{N})$

In the dynamic part of the model, we introduce three new variables:

inv2 : $load \in WorkingAnts \to Decision$
inv3 : $AgentDecision \in WorkingAnts \to Decision$
inv4 : $DensityPheromone \in Locations \to \mathbb{N}$

The first variable, *load*, models the current load each ant is carrying. The second one, *AgentDecision*, stores the latest decision made by every working ant. Finally, the variable *DensityPheromone* reflects the current amount of dropped pheromone in specific grid locations. Having the corresponding type and variable for ant decisions allows us to refine the abstract event Decide as follows:

EVENT Decide
REFINES Decide
  ANY ant WHERE
    $grd1 : ant \in WorkingAnts \ \wedge \ AgentStage(ant) = decide$
    $grd2 : GoalReached = FALSE$
  THEN
    $act1 : AgentDecision(ant) :\in Decision$
    $act2 : AgentStage(ant) := act$
  END

The event is still abstract since the environment perceptions of an ant, which are the basis for making ant decisions, will be introduced later. As a result, here the ant decision is made nondeterministically from the set *Decision*. Nevertheless, we can now rely on the information about the last decision of each ant (stored in *AgentDecision*) to elaborate on the act stage. In the previous model, this stage is represented by a single event Act. Now we refine the event Act to introduce the instances of Act corresponding to each possible ant decision. For example, below we show the event that models dropping food by an ant after reaching the nest. This event is proved to be a valid refinement of Act.

EVENT Act_Drop_Food
REFINES Act
  ANY ant WHERE
    $grd1 : ant \in WorkingAnts \ \wedge \ AgentStage(ant) = act$
    $grd2 : AgentDecision(ant) = dropFood$
    $grd3 : currentLoc(ant) = Nest \ \wedge \ GoalReached = FALSE$
  THEN
    $act1 : QuantityFood(Nest) := QuantityFood(Nest) + load(ant)$
    $act2 : load(ant) := 0$
  END

In a similar way, such events as, for example, Act_Move, Act_HarvestFood, and Act_DropPheromone, are introduced and proved to be specific refinements of the abstract event Act. The first event changes the ant's current location (not affecting the food and pheromone distributions), while the second and third ones update respectively the grid pheromone and food distributions.

In the abstract model, the food distribution in the event Change is updated with a high degree of nondeterminism. After two refinements, the introduced system details and constraints allow us to eliminate this nondeterminism completely. In fact, we can prove (as an invariant property) that no food is lost:

inv5 : $TotalFood(QuantityFood)(Locations) \ + TotalLoad(load)(Ants) = TotalFood(InitFoodDistribution)(Locations)$

Here we use the context functions $TotalFood$ and $TotalLoad$ to state that all the food from the initial food distribution is now either in the grid locations or is carried up by the ants. We can also explicitly relate the reaching of the main goal with the food absence outside the nest.

inv6 : $GoalReached = TRUE \ \Rightarrow \ TotalFood(QuantityFood)(Ants \backslash \{Nest\}) = 0$
inv7 : $GoalReached = TRUE \ \Rightarrow$
$QuantityFood(Nest) = TotalFood(InitFoodDistribution)(Locations)$

Finally, we can now prove that all the events affecting the food distribution are convergent, using the following variant expression:

$$TotalFood(QuantityFood)(Locations \backslash \{Nest\})$$

The Third Refinement. The second refinement step has allowed us to build a link between the ant decisions and its subsequent actions. While further elaborating on the ants behaviour, we have also refined the definition of the system level goal and the conditions that lead to it. Next, we focus on introducing different ant perceptions and formulate the decision rules allowing an ant to decide on its next action based on a combination of its current perception values.

We assume that each ant has the ability to perceive from a distance the food, dropped pheromone, nest and other ants. The exact strength of each perception depends on the ant's current location and the direction (i.e., the next location) it is facing. In other words, knowing the current food, pheromone or ant distribution on the grid, as well as the ant's location and direction, we can argue that the value of a specific perception can be unambiguously determined. This reasoning allows us to introduce the ant perceptions as abstract functions in the context. Specifically, the food perception can be defined as follows:

axm5 : $MaxFoodSmell \in \mathbb{N}1$
axm6 : $FoodPerception \in (Locations \to \mathbb{N}) \to (Locations \times Locations \nrightarrow 0..MaxFoodSmell)$

The first parameter of $FoodPerception$ is the current food distribution on the grid. The second parameter is a pair of locations, the first element of which is the current location and the second one is a possible next location (a position in the vicinity). The resulting value is the perception strength. We also assume that there is a upper limit for it (e.g., the maximal food smell).

In a similar way, we introduce $PheromonePerception$ and $AntPerception$. The final perception, $NestPerception$, can be defined slightly simpler: since the nest location is stationary, the first parameter can be omitted.

axm11 : $MaxNestSmell \in \mathbb{N}1$
axm12 : $NestPerception \in (Locations \times Locations \nrightarrow 0..MaxNestSmell)$

When foraging, an ant should evaluate different alternatives in its vicinity based on a combination of all its perceptions: food, pheromone, other ants, and nest. In other words, four perception values should be merged into a single value, which is then can be compared with the corresponding values for each possible direction. In the concrete system implementations, the perception values are often partitioned into the distinct intervals with the corresponding attached weights. In our formal development, we abstract away from the concrete heuristics allowing for producing a single perception value[1]. Instead, we define a generic abstract function, $Favour$,

axm14 : $Favour \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

which can be instantiated in many different ways.

For the given food, pheromone, and ants distributions fd, pd, ad, and a pair of locations $(loc \mapsto next)$, the overall perception value can be calculated as follows:

[1] How such heuristics can be obtained, see, for instance, [2].

$$Favour\ (FoodPerception\ fd\ (loc \mapsto next),\ PheromonePerception\ pd\ (loc \mapsto next),$$
$$AntsPerception\ ad\ (loc \mapsto next),\ NestPerception\ (loc \mapsto next))$$

In the dynamic part of the model, we introduce two new variables:

inv4 : $nextLocation \in WorkingAnts \to Location$
inv5 : $DensityAnts \in Locations \to \mathbb{N}$

The first variable, *nextLocation*, stores the next location an ants decides to move to based on its environment perceptions. The variable *DensityAnts* contains the dynamic information about the quantity of ants in grid locations.

Based on the perception functions introduced in the model context, we can now elaborate on the ant decide stage. Specifically, the abstract event Decide is now split into its several versions (Decide_MoveExplore, Decide_MoveReturn, Decide_HarvestFood, ...) that cover different ant decisions. The introduced perception functions are most useful for the events where an ant should decide the next location to proceed to. For instance, the event Decide_MoveExplore presented below should rely on all the ant perceptions.

EVENT Decide_MoveExplore
REFINES Decide
  ANY ant, $nextDir$, $maxFav$ WHERE
    $grd1 : ant \in WorkingAnts \ \wedge \ AgentStage(ant) = decide$
    $grd2 : GoalReached = FALSE \ \wedge \ nextDir \in Next(currentLoc(ant))$
    $grd3 : maxFav = FAVOUR(ant \mapsto nextDir)$
    $grd4 : maxFav = max(\{dir \cdot dir \in Next(currentLoc(ant)) \mid FAVOUR(ant \mapsto dir)\}$
  THEN
    $act1 : AgentDecision(ant) := move$
    $act2 : AgentStage(ant) := act$
    $act2 : NextLocation(ant) := nextDir$
  END

where $FAVOUR(ant \mapsto nextDir)$ stands for

$Favour($ $FoodPerception\ (QuantityFood)\ (currentLoc(ant) \mapsto nextDir)$,
$PheromonePerception\ (DensityPheromone)\ (currentLoc(ant) \mapsto nextDir)$,
$AntsPerception\ (DensityAnts)\ (currentLoc(ant) \mapsto nextDir)$,
$NestPerception\ (currentLoc(ant) \mapsto nextDir))$

The event allows an ant to choose the most favourable direction to move next. It is the location *nextDir* belonging to $Next(currentLoc(ant))$ and giving the maximal $FAVOUR(...)$ value based on the current ant perceptions.

The Fourth Refinement. One of the main purposes of the presented formal development is to formally establish the reachability of the main system goal: "All the distributed food will be eventually transferred to the nest". In the second refinement, we already proved that all the events affecting the food distribution are convergent and the amount of food outside the nest is constantly decreasing.

However, this result does not concern the events modelling the ants moving in search of the food or drawn by the left pheromone, ants returning to the nest, etc. We have to ensure that the ants do not stay forever in such modes of operation. We can achieve this by deriving the necessary conditions (constraints) on the ant perception functions that essentially control ant movements.

When ants are returning to the nest (or going after the food/pheromone smell), they have a specific target to reach, after which they switch to a different

activity (operational mode). The property we have to ensure is that, if an ant moves to the next location according to the used perception functions, it always gets closer to the target of its current operational mode. We ensure this by adding additional expected constraints (axioms) for the abstract perception functions in the model context. Then we are going to use these constraints in the model machine component to prove termination of the ants in particular operation modes. For instance, we constrain the definition of $NestPerception$ as follows:

$$\begin{array}{l}\mathsf{axm17}: \ \forall loc, next, prc.\ loc \in Locations \wedge next \in Next(loc) \wedge \\ \quad prc = NestPerception(loc \mapsto next) \wedge \\ \quad prc = max(\{dir \cdot dir \in Next(loc) \mid NestPerception(loc \mapsto dir)\}) \ \Rightarrow \\ \quad\quad next = Nest \ \vee \ (\exists loc'.\ loc' \in Next(next) \wedge NestPerception(loc' \mapsto dir) > prc)\end{array}$$

This axiom states that, if an ant proceeds to the direction with the maximal nest perception value, it either immediately reaches the nest or there exists the next location after that with an even higher perception value. Since we have introduced the constant for the maximal nest smell, the nest perception value cannot go up indefinitely. In similar fashion, we add the corresponding constraints to the other perception as well as the $Favour$ function.

In the model machine component, we explicitly introduce ant operation modes by partitioning the working ants into separate classes. For instance, the variable AntsApproachingNest is introduced for the ants returning to the nest.

$$\mathsf{inv7}: \ AntsApproachingNest \subseteq WorkingAnts$$

The corresponding model events are refined to update this variable if necessary.

Moreover, we formulate the variant in order to formally demonstrate termination of the ants in this operation mode:

$$\begin{array}{l}\Sigma\ (ant \cdot\ ant \in AntsApproachingNest \mid (maxNestSmell - max(\\ \quad \{\, dir \cdot dir \in Next(currentLoc(ant)) \mid NestPerception(currentLoc(ant) \mapsto dir)\})))\end{array}$$

The decreasing of this variant (for the corresponding events) is proved by relying on the axiom axm17 presented above. In a similar way, we formulate the necessary conditions and prove termination for the other ant operation modes.

6 Discussion

The presented formal development of the foraging ants case study has been carried out within the Rodin platform [13] – the integrated tool support for Event-B. The Rodin platform has significantly facilitated both modelling and verification of our models. In particular, it has generated over 480 proof obligations, most of which were automatically discharged. Majority of those proof obligations came from the last two refinement steps, indicating the rising level of complexity.

By formulating many important notions as abstract sets and functions (with only essential properties postulated) in the model context, we have not only achieved better understanding of the environment-system interdependencies but also arrived at a parametric system model. Indeed, the obtained generic definitions can be instantiated with different system-specific parameters and hence the proposed models can be reused to model a family of cooperative MAS. For instance, generic definitions of the ant decision rules (perception and $Favour$ functions) allow us to instantiate them in many ways, assigning different weights for various perceptions or their combinations.

In the last refinement step we derived the constraints for ensuring termination of ants staying in particular operation modes. These constraints can be seen as the conditions to be checked for concrete instances of the perception functions.

Our derived models also demonstrate the interplay between the global and local reasoning. Even though the ant perception functions (which are the basis for local ant decisions) are defined globally, they merely represent our global assumptions that each ant has particular capabilities to perceive its vicinity.

We formalised the problem of system-level goal reachability as a termination problem. We had to constrain the environment by requiring that no new food sources appear on the grid, otherwise the system would become non-convergent. The proved termination for ants in particular modes can be seen as piece-wise invariant, since it can be violated at the points of ants switching the operating modes. The termination proof is based on the standard Event-B technique using variants. To obtain a general termination result, one can consider *almost certain termination* approach [9] based on the probabilistic reasoning. However, such an approach would complicate the refinement process because of intricate properties of models containing both probabilistic and demonic non-determinism.

To evaluate quantitative characteristics of the modelled system (e.g., how effective are cooperation strategies of concrete instances of the decision rules), the designers should bridge Event-B with other approaches. We are planning to investigate how runtime simulation or model checking can be used for this aim.

7 Related Work and Conclusions

Self-organising systems have attracted significant research attention over the last decade. Majority of the approaches rely on simulation and model checking to explore the impact of different parameters on the system behaviour. In [7], *Gardelli* uses stochastic Pi-Calculus for modelling self-organising MAS for intrusion detection capabilities. The *SPIM* tool is used to assess the impact of, e.g., the number of agents and frequency of inspections, on the system behaviour. In [5], a hybrid approach for modelling and verifying self-organising systems has been proposed. This approach uses stochastic simulations to model the system described as a Markov chain and probabilistic model checking (using the *PRISM* tool) for verification. Konur et al. [10] also use *PRISM* and probabilistic model checking to verify the behaviour of a robot swarm. The authors verify the system properties expressed in the *PCTL* logic for several scenarios.

In this paper, we have experimented with a technique that not only allows the designers to verify certain system-level properties of self-organising MAS, but also provides a support throughout the development process. In our work, we start from a high-level system model and *derive* the specification that details the individual agent mechanisms leading to reaching the desired goal.

The derivational approach has been also adopted in our previous work [12]. There we have studied the mechanisms of goal decomposition by refinement and ensuring data integrity in cooperative MAS. In contrast, in this paper we propose how to create a parameterised generic model of the system environment and establish the link between actions of autonomic agents and the environment

state. Moreover, we demonstrate how to formally represent agent perception and decision rules as generic system parameters.

In this paper, we presented a case study in formal development of a nature-inspired self-organising MAS. We demonstrated how to derive a detailed specification of a colony of foraging ants by refinement. Formal derivation has provided us with a structured and disciplined framework for the development of a complex system with intricate agent interactions. We believe that the proposed approach is promising for modelling the logical aspects of self-organising systems.

Self-organising MAS are complex multi-facet phenomena and hence require a range of approaches for their modelling and analysis. The proposed approach should be integrated with stochastic analysis techniques, in order to identify the most optimal system parameters that would allow the system to achieve its objectives not only in terms of logical correctness but also performance, reliability and required resources. Integration with such techniques constitutes one of the directions of our future research.

References

1. Abrial, J.-R.: Modelling in Event-B. Cambridge University Press (2010)
2. Bonjean, N., Mefteh, W., Gleizes, M.-P., Maurel, C., Migeon, F.: ADELFE 2.0. In: Handbook on Agent-Oriented Design Processes. Springer
3. Camps, V.: Vers une théorie de l'auto-organisation dans les systèmes multi-agents basée sur la coopération: application à la recherche d'information dans un système d'information répartie. PhD thesis, Paul Sabatier university (1998)
4. Capera, D., Georgé, J.-P., Gleizes, M.P., Glize, P.: The AMAS theory for complex problem solving based on self-organizing cooperative agents. In: WETICE, pp. 383–388 (2003)
5. Casadei, M., Viroli, M.: Using probabilistic model checking and simulation for designing self-organizing systems. In: Proceedings of the 2009 ACM Symposium on Applied Computing, pp. 2103–2104. ACM, New York (2009)
6. Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A. (eds.): Self-organising Software - From Natural to Artificial Adaptation. Natural Computing Series. Springer (October 2011), `http://www.springerlink.com`
7. 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.) Proceedings of 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), Vienna, Austria, vol. 2, pp. 539–544 (2006)
8. Gleizes, M.-P., Camps, V., Georgé, J.-P., Capera, D.: Engineering systems which generate emergent functionalities. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS (LNAI), vol. 5049, pp. 58–75. Springer, Heidelberg (2008)
9. Hallerstede, S., Hoang, T.S.: Qualitative Probabilistic Modelling in Event-B. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 293–312. Springer, Heidelberg (2007)
10. Konur, S., Clare, D., Fisher, M.: Analysing robot swarm behaviour via probabilistic model checking. Robot. Auton. Syst. 60(2), 199–213 (2012)
11. Laibinis, L., Troubitsyna, E.: Refinement of Fault Tolerant Control Systems in B. In: Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.) SAFECOMP 2004. LNCS, vol. 3219, pp. 254–268. Springer, Heidelberg (2004)
12. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal Development of Critical Multi-agent Systems: A Refinement Approach. In: EDCC, pp. 156–161 (2012)
13. Rodin platform. Automated tool environment for Event-B, `http://rodin-b-sharp.sourceforge.net/`