TUCS

Inna Pereverzeva | Elena Troubitsyna | Linas Laibinis

Formal Goal-Oriented Development of Resilient MAS in Event-B

Turku Centre *for* Computer Science

TUCS Technical Report
No 1033, January 2012

Formal Goal-Oriented Development of Resilient MAS in Event-B

Inna Pereverzeva

Åbo Akademi University, Department of Computer Science,
Turku Centre for Computer Science
inna.pereverzeva@abo.fi

Elena Troubitsyna

Åbo Akademi University, Department of Computer Science
elena.troubitsyna@abo.fi

Linas Laibinis

Åbo Akademi University, Department of Computer Science
linas.laibinis@abo.fi

TUCS Technical Report

No 1033, January 2012

Abstract

Goal-Oriented Development facilitates structuring complex requirements. To ensure resilience the designers should guarantee that the system achieves its goals despite changes, e.g., caused by failures of system components. In this paper we propose a formal goal-oriented approach to development of resilient MAS. We formalize the notion of goal and goal achievement in Event-B and propose the specification and refinement patterns that allow us to guarantee that the targeted goals are reached despite agent failures. We illustrate our approach by a case study – development of an autonomous multi-robotic system.

Keywords: Event-B, formal modelling, refinement, goal-oriented development, multi-agent system.

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Goal-Oriented Development [15] has been recognised as an useful framework for structuring and specifying complex system requirements. In goal-oriented development, the system requirements are defined in terms of goals – the functional and non-functional objectives that a system should achieve. Often changes in system operational environment, e.g., caused by failures of agents – independent system components of various types – might hinder achieving the desired goals. Hence, to ensure system resilience [7], i.e., guarantee its dependability in spite of the changes, we need formally verify reachability of the targeted goals. Traditionally, such a verification is undertaken by abstracting implementation up to requirements level and model-checking satisfiability of goals. However, such an approach suffers from a state explosion that is especially prohibitive for such applications as multi-robotic systems [5].

In this paper we propose a formal development approach that ensures goal reachability "by construction". Our approach is based on refinement in Event-B. Event-B [2] is a formal top-down development approach to correct-by-construction system development. The main development technique – refinement – allows us to ensure that a concrete specification preserves globally observable behaviour and properties of abstract specification. Verification of each refinement step is done by proofs. Rodin platform [11] automates modelling and verification in Event-B. Currently Event-B is actively used within EU project Deploy [4] to model dependable systems from various domains.

We formalise goal-oriented development by defining a set of specification and refinement patterns. Our formalisation reflects the main concepts of the goal-oriented engineering. In particular, we demonstrate how to define system goals at different levels of abstraction and guarantee goal reachability while specifying collaborative agent behaviour. Moreover, we propose refinement patterns that allow the system to dynamically reallocate goals from failed agents to healthy ones and per se, guarantee resilience. A development of an autonomous multi-robotic system illustrates application of the proposed patterns. We believe that our approach offers a scalable technique for development and formal verification of complex resilient MAS.

The paper has the following structure. In Section 2 we briefly present our modelling framework – Event-B. In Section 3 we present the set of specification and refinement patterns that facilitate goal-oriented development in Event-B. In Section 4 we present a case study – development of an autonomous multi-robotic system by refinement. In Section 5 we overview the related work, discuss the presented approach and outline the directions for the future research.

2 Formal Modelling and Refinement in Event B

In this section we present our formal development framework – Event-B. The Event-B formalism is an extension of the B Method [1]. It is a state-based formal

Table 1: Before-after predicates

Action (S)	$BA(S)$
$x := E(x,y)$	$x' = E(x,y) \ \wedge \ y' = y$
$x :\in Set$	$\exists z \cdot (z \in Set \wedge x' = z) \ \wedge \ y' = y$
$x :\mid P(x,y,x')$	$\exists z \cdot (P(x,z,y) \wedge x' = z) \ \wedge \ y' = y$

approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. Event-B has been specifically designed to model and reason about parallel, distributed and reactive systems.

2.1 Modelling in Event-B

In Event-B, a system model is specified using the notion of an *abstract state machine* [2]. An abstract state machine encapsulates the system state represented as a collection of model variables, and defines operations on this state, i.e., it describes the dynamic *behaviour* of the modelled system. A machine may also have the accompanying component, called *context*. A context might include user-defined carrier sets, constants and their properties, which are given as a list of model axioms. In Event-B, the variables are strongly typed by the constraining predicates called **invariants**. Moreover, the invariant specify important properties that should be preserved during system execution.

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

$$\textbf{evt} \mathrel{\widehat{=}} \textbf{any } vl \textbf{ where } g \textbf{ then } S \textbf{ end}$$

where vl is a list of new local variables (parameters), g is the event **guard**, and S is the event **action**. The guard is a state predicate that defines the conditions under which the action can 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 non-deterministically. If none of the events is enabled then the system deadlocks. In general, the action of an event is a parallel composition of deterministic or non-deterministic assignments. A deterministic assignment, $x := E(x,y)$, has the standard syntax and meaning. A non-deterministic assignment is denoted either as $x :\in Set$, where Set is a set of values, or $x :\mid P(x,y,x')$, where P is a predicate relating initial values of x, y to some final value of x'. As a result of such a non-deterministic assignment, x can get any value belonging to Set or according to P.

The semantics of Event-B actions is defined using so called before-after (BA) predicates [2]. A before-after predicate describes a relationship between the system states before and after execution of an event, as shown in Table 1. Here x and y are disjoint lists (partitions) of state variables, and x', y' represent their values in the after-state.

The semantics of an Event-B model is formulated as a collection of *proof obligations* – logical sequents, which must be proved to show that a machine is well-defined and the events preserve invariant. The full list of proof obligations can be found in [2].

2.2 Event-B Refinement

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that non-deterministically models the most essential functional requirements. In a sequence of refinement steps we gradually reduce non-determinism and introduce detailed design decisions. In particular, we can replace abstract variables by their concrete counterparts, i.e., perform data refinement. In this case, the invariant of the refined machine formally defines the relationship between the abstract and concrete variables. Via such a *gluing* invariant we establish a correspondence between the state spaces of the refined and the abstract machines.

Often a refinement step introduces new events and variables into the abstract specification. The new events correspond to the stuttering steps that are not visible at the abstract level, i.e., they refine implicit *skip*. To guarantee that the refined specification preserves the global behaviour of the abstract machine, we should demonstrate that the newly introduced events *converge*. To prove it, we need to define a *variant* – an expression over a finite subset of natural numbers – and show that the execution of new events decreases it. Sometimes, convergence of an event cannot be proved due to a high level of non-determinism. Then the event obtains the status *anticipated*. This obliges the designer to prove at some later refinement step, that the event indeed converges. Then the status of the events is changed to the *convergent*.

Refinement relation is transitive. It allows us to build complex specifications in a number of small (and hence rather simple and highly-automated) correctness-preserving model transformations. Each refinement step requires to verify a number of proof obligations that ensure that the refined specification adheres to its abstract counterpart. The verification efforts, in particular, automatic generation and proving of the required proof obligations, are significantly facilitated by the Rodin platform [10].

Refinement and proof-based verification of Event-B offers the designers a scalable support for the development of such complex distributed systems as MAS. In the next section we show how refinement process can facilitate modelling MAS and reasoning about goal reachability.

3 A Formal View of Goal-Oriented Multi-Agent System.

3.1 Patterns for Goal-Oriented Development

The goal-oriented engineering facilitates structuring complex system requirements in terms of *goals* – objectives that the system should meet [15]. In this paper we focus on modelling functional goals, i.e., the goals defining objectives of the services that the system should deliver. We propose a number of *specification and refinement patterns* that interpret essential activities of goal-oriented engineering in terms of Event-B refinement.

A pattern in Event-B is an abstract machine that defines a generic modelling solution that can be reused in similar developments via instantiation. Usually an Event-B pattern contains generic (abstract) types, constants and variables. The context of such a model constraints the instantiation by defining the properties that should be satisfied by concrete representations (instantiations) of abstract data structures. The invariant properties of a pattern, once proven, remain valid for all instantiations.

The aim of defining a pattern is to capture experience gained in modelling a certain problem. To illustrate how patterns are defined let us now present a pattern that allow the designers to explicitly define goals while modelling a system in Event-B. We call it *Abstract Goal Modelling Pattern*.

3.2 Abstract Goal Modelling Pattern

Let $GSTATE$ be an abstract type defining the system state space[1]. Moreover, let *Goal* be a non-empty proper subset of $GSTATE$ that abstractly defines the given system goals. We say that the system has achieved the desired goals if its current state belongs to $Goal$. Both $GSTATE$ and $Goal$ are the abstract types. Together with their properties they are defined in the model context as follows:

$$\textit{Goal} \neq \varnothing \text{ and } Goal \subset GSTATE.$$

Let us note that *GSTATE* and *Goal* are generic parameters of the initial pattern. During a system development, we should supply their concrete instantiations that satisfy the properties shown above.

While modelling a system in Event-B, we should ensure that the system under construction achieves the desired goal. We can formally express this by requiring that the system terminates in a state satisfying $Goal$. The machine M_AGM is defined according to the *Goal Modelling Pattern*:

```
Machine M_AGM
Variables gstate
Invariants
  inv : gstate ∈ GSTATE
Events
  Initialisation ≙
      begin
          gstate :∈ GSTATE \ Goal
      end
  Reaching_Goal ≙
      status anticipated
      when
          gstate ∈ GSTATE \ Goal
      then
          gstate :∈ GSTATE
      end
end
```

[1] In fact, it is sufficient to consider the states that our goal depends on.

The dynamic behaviour of the system is abstractly modelled by the event Reaching_Goal. The system terminates when Reaching_Goal becomes disable, i.e., when a state satisfying *Goal* is reached.

The event Reaching_Goal has the status $anticipated$. Hence, in the machine M_AGM goal reachability is postulated rather than proved. However, it also obliges us to prove (at some refinement step) that the event or its refinements converge. Therefore, while refining a concrete specification defined according to *Abstract Goal Modelling Pattern*, we will be forced to prove goal reachability.

Let us assume that we have a collection of Event-B patterns: $P_1, P_2, ..., P_n$ that refine each other in the following way:

P_1 **is refined by** P_2 **... is refined by** P_n.

Such a refinement chain expresses a generic development by refinement. Abstract data structures of all the involved patterns become generic parameters of the development. Each pattern abstractly defines a solution for specifying a certain modelling aspect. Therefore, each refinement step has a rationale behind it – its meta-level description. We use it to formulate modelling aspects that the refinement transformation aims at defining. The result of refinement transformation is called a refinement pattern.

Next we propose several refinement patterns that allow us to implement the ideas of goal-oriented engineering in Event-B refinement. We start from defining *Goal Decomposition Refinement pattern*.

3.3 Goal Decomposition Pattern

The main idea of goal-oriented development is to decompose the high-level system goals into a set of subgoals. This is an iterative process that aims at building the hierarchy of system goals. Essentially, subgoals define intermediate stages of the process of achieving the main goal.

The purpose of *Goal Decomposition Pattern* is to explicitly model subgoals in the system specification. While defining this pattern we should ensure that high-level goals remain achievable. Hence, our refinement pattern should reflect the relation between the high-level goals and subgoals. Moreover, it should ensure that high-level goal reachibility is preserved and can be defined via reachibility of lower-layer subgoals.

In this paper we assume that subgoals are independent of each other. This means that reachability of any subgoal does not affect reachibility of another one. Moreover, while a certain subgoal is reached, it remains reached, i.e., the system always progresses towards achieving its goals. Formally, it can be expressed as a stability property with respect to some state predicate P:

Stable(P) $\Leftrightarrow$ *once P becomes true it remains true.*

Intuitively, a stability property can be understood as a postponed invariant property that does not need to be true initially.

In Event-B, stability properties can be easily expressed by introducing auxiliary variables for storing the previous value of the state and then formulating stability properties as the invariant properties of the form:

$$P(prev\_state) = TRUE \;\Rightarrow\; P(state) = TRUE.$$

To express a goal decomposition in terms of Event-B, let us define a corresponding refinement pattern. We present it by the machine M_GD shown below. The new pattern allows us to introduce a number of subgoals into our system model and express their reachability. Moreover, the refinement relation between patterns allows us to express reachability of the main goal via reachability of its subgoals.

Let us assume for simplicity, that system goal *Goal* is achieved by reaching three subgoals. The subgoals are defined as corresponding variables of the M_GD machine: $SubGoal_1$, $SubGoal_2$, and $SubGoal_3$. The goal independence assumption allows us to partition high-level goal state space $GSTATE$ into three non-empty subsets: $SG\_STATE1, SG\_STATE2, SG\_STATE3$. We define the subgoals as follows:

$$SubGoal_i \neq \varnothing \;\text{ and }\; SubGoal_i \subset SG\_STATEi,\; i \in 1..3.$$

To establish a relashionship between the new state spaces $SG\_STATEi,\; i \in 1..3,$ of the M_GD machine and the abstract state space of M_AGM machine we define the following function:

$$State\_map \in SG\_STATE1 \times SG\_STATE2 \times SG\_STATE3 \rightarrowtail\!\!\!\!\rightarrow GSTATE,$$

where $\rightarrowtail\!\!\!\!\rightarrow$ designates a bijection function. Essentially it partitions the original goal state space into three independent parts.

To postulate that the main goal is reached if and only if all three subgoals are reached, we add the axiom into the context of the M_GD machine:

$$\forall sg1, sg2, sg3.\; sg1 \in Subgoal_1 \wedge sg2 \in Subgoal_2 \wedge sg3 \in Subgoal_3$$
$$\Leftrightarrow State\_map(sg1 \mapsto sg2 \mapsto sg3) \in Goal.$$

Refinement performed according to the *Goal Decomposition Pattern* is an example of the Event-B data refinement. We replace the abstract variable $gstate$ with the new variables $gstate_i \in SG\_STATEi$, $i \in 1..3$. The new variables model the state of the corresponding subgoals. The following gluing invariant allows us to prove data refinement:

$$gstate = State\_map(gstate1 \mapsto gstate2 \mapsto gstate3).$$

Essentially the M_GD machine decomposes the Reaching_Goal event of the M_AGM machine into three similar events $\mathsf{Reaching\_SubGoal_i}$, $i \in 1..3$:

Machine M_GD
Reaching_SubGoal$_i$ $\widehat{=}$ refines Reaching_Goal
  status $anticipated$
  when
    $gstate_i \in SG\_STATEi \setminus Subgoal_i$
  then
    $gstate_i :\in SG\_STATEi$
  end
...

Let us observe that we can easily verify that the following stability property holds for the pattern M_GD:

$$Stable(gstate_1 \in Subgoal_1) \wedge Stable(gstate_2 \in Subgoal_2) \wedge Stable(gstate_3 \in Subgoal_3).$$

The proposed *Goal Decomposition Pattern* can be repeatedly used to refine subgoals into the subgoals of finer granularity until the desired level of details is reached.

3.4 Agent Modelling Pattern

Our elaborated *Abstract Goal Modelling* and *Goal Decomposition* patterns allow us to specify the system goal(s) at different levels of abstraction. In multi-agent systems, (sub)goals are usually achieved by system agents. Agents are independent entities that are capable of performing certain tasks. In general, the system might have several types of agents that are distinguished by the type of tasks that they are capable of performing. Our next refinement pattern – *Agent Modelling Pattern* – allows us to model agents and associate them with goals.

We introduce the set *AGENTS* that abstractly defines the set of system agents. In this refinement pattern we also introduce a concept of agent *eligibility*. An agent is *eligible* if it is capable of achieving a certain task (subgoal). We define the non-empty sets *EL_AG1*, *EL_AG2*, and *EL_AG3* of the agents eligible to achieve each particular subgoal.

Agent might fail while trying to achieve a certain subgoal. Then it is removed from the dynamic set of eligible agents represented by the variable $elig_i$:

$$elig_i \subseteq EL\_AGi, \; i \in 1..3.$$

A goal is achieved if there is at least one eligible agent associated with it. This is formulated as the corresponding invariant property of our pattern:

$$elig_1 \neq \varnothing \text{ and } elig_2 \neq \varnothing \text{ and } elig_3 \neq \varnothing.$$

The dynamic part of *Agent Modelling Pattern* is defined in the machine M_AM. Since we assumed that the agents can fail, the goal assigned to the failed agent cannot be reached. To reflect this assumption in our model, we refine the abstract event $\mathsf{Reaching\_SubGoal_i}$ by two events $\mathsf{Successful\_Reaching\_SubGoal_i}$ and $\mathsf{Failed\_Reaching\_SubGoal_i}$, $i \in 1..3$, which respectively model successful and unsuccessful reaching of the subgoal by some eligible agent:

Machine M_AM
Successful_Reaching_SubGoal$_i$ $\widehat{=}$ refines Reaching_SubGoal$_i$
  status *convergent*
  any ag
  when
    $gstate_i \in SG\_STATEi \setminus Subgoal_i \wedge ag \in elig_i$
  then
    $gstate_i :\in Subgoal_i$
  end
Failed_Reaching_SubGoal$_i$ $\widehat{=}$ refines Reaching_SubGoal$_i$
  status *convergent*
  any ag
  when
    $gstate_i \in SG\_STATEi \setminus Subgoal_i \wedge ag \in elig_i \wedge card(elig_i) > 1$
  then
    $gstate_i :\in SG\_STATEi \setminus Subgoal_i$
    $elig_i := elig_i \setminus \{ag\}$
  end

In the guard of the event Failed_Reaching_SubGoal$_i$ we restrict possible agent failures by postulating that at least one agent associated with the subgoal remains operational: $card(elig_i) > 1,\ i \in 1..3$. This assumption allows us to change the event status from anticipated to convergent. In other words, we are now able to prove that, for each subgoal, the process of reaching it eventually terminates. To prove the convergence we define the following variant expression:

$$card(elig_1)\ +\ card(elig_2)\ +\ card(elig_3) +$$

$$bnat_1(gstate_1)\ +\ bnat_2(gstate_2)\ +\ bnat_3(gstate_3).$$

When an agent fails, it is removed from a corresponding set of eligible agents $elig_i$. This in turn decreases the value of $card(elig_i)$ and consequently the whole variant expression. On the other hand, when an agent succeeds in reaching the goal, all the events become disabled, thus ensuring system termination as well. To show decreasing of the variant expression when the subgoal is reached, we introduce the auxiliary functions $bnat_i$:

$$bnat_i \in SG\_STATEi \rightarrow \mathbb{N},$$

$$\forall s \cdot s \in Subgoal_i \Rightarrow bnat_i(s) = 0,$$

$$\forall s \cdot s \in SG\_STATEi \setminus Subgoal_i \Rightarrow bnat_i(s) = 1.$$

These functions have two possible values – 0 and 1. Until the subgoal is not reached, the corresponding value of function $bnat_i$ equals to 1. When the subgoal is reached, the value becomes 0 and this consequently decreases the whole variant expression.

In practice, the constraint to have at least one operational agent associated with our model can be validated by probabilistic modelling of goal reachability, which is planned as a future work. Let us also note that for multi-robotic systems with many homogenous agents this constraint is usually satisfied.

3.5 Agent Refinement Pattern

Above we have defined the notion of agent eligibility quite abstractly. We establish the relationship between subgoals (tasks) and agents that are capable of achieving them. Our last refinement pattern, *Agent Refinement Pattern*, aims at unfolding the notion of agent eligibility. Here we define the agent eligibility by introducing agent attributes – *agent types* and *statuses*. An eligible agent will be an operational agent that belongs to particular agent type.

We define an enumerated set of agent types $AG\_TYPE = \{TYPE1, TYPE2, TYPE3\}$ and establish the correspondence between abstract sets of eligible agents and the corresponding agent types by the following axioms:

$$\forall ag \cdot ag \in EL\_AGi \Leftrightarrow atype(ag) = TYPEi,\ i \in 1..3.$$

An agent is eligible to perform a certain subgoal if it has the type associated with this subgoal.

An agent might be operational or failed. To model the notion of agent status we define an enumerated set $AG\_STATUS = \{OK, KO\}$, where constants OK and KO designate operational and failed agents correspondingly.

Below we present an excerpt from the dynamic part of the *Agent Refinement Pattern* – the machine M_AR. We add a new variable $astatus$ to store the dynamic status of each agent:

$$astatus \in AGENTS \rightarrow AG\_STATUS.$$

Moreover, we data refine the variables $elig_i$. The following gluing invariants relate them with the concrete sets:

$$elig_i = \{a | a \in AGENTS \wedge atype(a) = TYPEi \wedge astatus(a) = OK\},\ i \in 1..3.$$

In our case, the dynamic set of eligible agents to perform a sertain subgoal becomes a set of active agents of the particular type.

```
Machine M_AR
Successful_Reaching_SubGoal_i ≙ refines Successful_Reaching_SubGoal_i
  any ag
  when
     gstate_i ∈ SG_STATEi \ Subgoal_i ∧ astatus(ag) = OK ∧ atype(ag) = TYPE_i
  then
     gstate_i :∈ Subgoal_i
  end
Failed_Reaching_SubGoal_i ≙ refines Failed_Reaching_SubGoal_i
  any ag
  when
     gstate_i ∈ SG_STATEi \ Subgoal_i ∧ astatus(ag) = OK ∧ atype(ag) = TYPE_i ∧
     card({a|a ∈ AGENTS ∧ atype(a) = TYPE_i ∧ astatus(a) = OK}) > 1
  then
     gstate_i :∈ SG_STATEi \ Subgoal_i
     astatus(ag) := KO
  end
```

The event $\mathsf{Failed\_Reaching\_SubGoal_i}$ is now refined to take into account the concrete definition of agent eligibility. The event also updates the status of the failed agent.

Further refinement patterns can be defined to model various fault tolerance mechanism. However, in this paper instead of building further the collection of patterns, we will demostrate how to instantiate and use the described patterns in a concrete development.

4 Case Study: a Multi-Robotic System

4.1 A Case Study Description

As a case study we consider a multi-robotic system. The goal of the system is to coordinate identical robots to get a certain area cleaned. The area is divided into several zones, which can be further divided into a number of sectors. Each zone has a base station – a static computing and communicating device – that coordinates the cleaning of the zone. In its turn, each base station supervises a number of robots by assigning cleaning tasks to them.

A robot is an autonomous electro-mechanical device – a special kind of a rover that can move and clean. The base station may assign a robot a sector – a certain area in the zone – to clean. As soon as the robot receives a new cleaning task, it autonomously travels to this area and starts to clean it. After successfully completing its mission, it returns back to the base station to receive a new order.

The base station keeps track of the cleaned sectors. A robot may fail to clean the assigned sector. In that case, the base station assigns another robot to perform this task. To ensure that the whole area is eventually cleaned, each base station in its turn should ensure that its zone is eventually cleaned.

The system should function autonomously, i.e., without human intervention. Such kind of systems are often deployed in hazardous areas (nuclear power plants, disaster areas, mine fields etc.). Hence guaranteeing system resilience is an important requirement. Therefore, we should formally demonstrate that the system goal is achievable despite possible robot failures.

Next, we will show how to develop a multi-robotic system by refinement in Event-B and demonstrate how to rely on the patterns proposed in Section 3 to formally specify the system behaviour to ensure reachability of the overall system goal.

4.2 Pattern-Driven Refinement of a Multi-Robotic System

In this section we will describe our formal development of a multi-robotic system in Event-B. The development is concluded via instantiation of the proposed patterns, with the goal decomposition pattern being applied twice in a row.

Abstract model. The initial model defined by the machine $\mathsf{MRS\_Abs}$ specifies the behaviour of a multi-robotic system according to the *Abstract Goal Modelling*

Pattern. We apply this pattern by instantiating abstract variables with the concrete values and specifying events that model system behaviour.

The state space of the initial model is defined by the type $BOOL$. The value TRUE corresponds to the situation when the desired goal is achieved (i.e., the whole territory is cleaned), while FALSE represents the opposite situation.

Similarly to the pattern machine M_AGM, the machine MRS_Abs contains an event, CleaningTerritory, that models system behaviour. It abstractly represents the process of cleaning the territory, where a variable $completed \in BOOL$ models the current state of the system goal. This event is constructed according to the pattern event Reaching_Goal by taking all the instantiations into account, as shown below:

```
Machine AbsMRS
Variables completed
Invariants
  inv : completed ∈ BOOL
Events
  ...
CleaningTerritory ≘
  status anticipated
  when
     completed = FALSE
  then
     completed :∈ BOOL
  end
```

The system continues its execution until the whole territory is cleaned, i.e., as long as $completed$ stays FALSE. At this level of abstraction, the event CleaningTerritory has the *anticipated* status. In other words, similarly to the abstract pattern, we delay the proof that the event eventually converges to subsequent refinements. It is easy to see that the machine AbsMRS is an instantiation of the pattern machine M_AGM, where the abstract type *GSTATE* its replaced with $BOOL$, the constant *Goal* is instantiated with a singleton set $\{$TRUE$\}$, and the variable *gstate* is renamed into *completed*.

First refinement. Our initial model specifies system behaviour in a highly abstract way. It models the process of cleaning the whole territory. The goal of the first refinement is to model the cleaning of the territory zones. Refinement is performed according to the *Goal Decomposition Pattern*.

In the first refinement step resulting in the machine MRS_Ref1, we augment our model with representation of subgoals. The whole territory is divided into n zones, $n \in \mathbb{N}$ and $n \geq 1$. We associate the notion of a *subgoal* with the process of *cleaning a particular zone*. Thus a subgoal is achieved when the corresponding zone is cleaned. A new variable $zone\_completed$ represents the current subgoal status for every zone. The value TRUE corresponds to the situation when the certain zone is cleaned:

$$zone\_completed \in 1..n \rightarrow BOOL.$$

The refined model MRS_Ref1 is built as an instantiation of the *Goal Decomposition Pattern* machine M_GD, where the subgoal states are defined as elements of the variable $zone\_completed$, i.e.,

$$gstate_i = zone\_completed(i), \text{ for } i \in 1..n.$$

This observation suggests the following gluing invariant between the initial and the refined models:

$$completed = TRUE \Leftrightarrow zone\_completed[1..n] = \{TRUE\}.$$

The invariant can be understood as follows: the territory is considered to be cleaned if and only if its every zone is cleaned.

The pattern events $\mathsf{Reaching\_Subgoal}_i$ correspond to a single event CleaningZone:

```
Machine MRS_Ref1
CleaningZone ≙ refines CleaningTeritory
  status anticipated
  any zone, zone_result
  when
     zone ∈ 1..n ∧ zone_completed(zone) = FALSE ∧
     zone_result ∈ BOOL
  then
     zone_completed(zone) := zone_result
  end
```

Second refinement. In our development of a multi-robotic system we should apply the goal decomposition pattern twice, until we reach the level of "primitive" goals, i.e., the goals for which we define the classes of agents eligible for execution of these goals.

Every zone in our system is divided into k sectors, $k \in \mathbb{N}$ and $k \geq 1$. A robot is responsible for cleaning a certain sector. We associate the notion of a *subsubgoal* (or simply *task*) with the process of *cleaning a particular sector*. The task is completed when the sector is cleaned. A new array variable $sector\_completed$ represents the current task status for every sector:

$$sector\_completed \in 1..n \rightarrow (1..k \rightarrow BOOL).$$

The refined model is again built as an instantiation of the *Goal Decomposition Pattern*, where the subsubgoal states are defined as the elements of the variable $sector\_completed$, i.e.,

$$gstate_{ij} = sector\_completed(i)(j), \text{ for } i \in 1..n, \ j \in 1..k.$$

A gluing invariant expresses the relationship between subgoals and subsubgoals:

$$\forall\, zone \cdot zone \in 1\,..\,n \Rightarrow (zone\_completed(zone) = TRUE \Leftrightarrow$$
$$sector\_completed(zone)[1\,..\,k] = \{TRUE\}).$$

The invariant postulates that any zone is cleaned if and only if its every sector is cleaned. The abstract event CleaningZone is refined by the event CleaningSector. The subsubgoal will be achieved if this section is eventually cleaned:

Machine MRS_Ref2
CleaningSector $\widehat{=}$ refines CleaningZone
status $anticipated$
any $zone, sector, sector\_result$
when
$zone \in 1..n \ \wedge sector \in 1 .. k \wedge$
$sector\_completed(zone)(sector) = FALSE \wedge$
$sector\_result \in BOOL$
then
$sector\_completed(zone) := sector\_completed(zone) \Leftarrow \{sector \mapsto sector\_result\}$
end

Now we have reached the desire level of granularity of our subgoals. In the next refinement step (the machine MRS_Ref3) we are going to augment our model with an abstract representation of agents.

Third refinement. The next refined model of our development is constructed according to the refinement *Agent Modelling Pattern*. As a result, we introduce the abstract set $AGENTS$, and its subset $ELIG$ containing the eligible agents for executing the tasks. A new variable $elig$ represents the dynamic set of (currently available) eligible agents. Following the proposed pattern, we should also guarantee that there will be at least one eligible agent for cleaning the sector. This property is formulated as an additional invariant: $elig \neq \varnothing$.

Moreover, according to the pattern, we need abstractly introduce agent failures. This is achieved by refining the abstract event CleaningSector by two events SuccessfulCleaningSector and FailedCleaningSector, which respectively model successful and unsuccessful execution of the task by some eligible agent:

Machine MRS_Ref3
SuccessfulCleaningSector $\widehat{=}$ refines CleaningSector
status $convergent$
any $zone, sector, ag$
when
$zone \in 1..n \wedge sector \in 1 .. k \wedge$
$sector\_completed(zone)(sector) = FALSE \wedge$
$ag \in elig$
then
$sector\_completed(zone) := sector\_completed(zone) \Leftarrow \{sector \mapsto TRUE\}$
end
FailedCleaningSector $\widehat{=}$ refines CleaningSector
status $convergent$
any $zone, sector, ag$
when
$zone \in 1..n \wedge sector \in 1 .. k \wedge$
$sector\_completed(zone)(sector) = FALSE \wedge$
$ag \in elig \ \wedge card(elig) > 1$
then
$sector\_completed(zone) \ := \ sector\_completed(zone) \Leftarrow \{sector \mapsto FALSE\}$
$elig := elig \setminus \{ag\}$
end

Following the proposed pattern, we add in the event FailedCleaningSector the guard $card(elig) > 1$ to restrict possible agent failure in task performance. Let us also note that for multi-robotic systems with many homogenous agents this constraint is not unreasonable. This assumption allows us to prove the convergence of the goal-reaching events, i.e., to prove that the process of cleaning the territory eventually terminates.

Fourth refinement. Finally, the *Agent Refinement Pattern* for introducing agent types and their status is applied to produce the last refined model of our multi-robotic system. In this refinement step we explicitly define the agent types – robots and base stations. We partition our abstract set $AGENTS$ by disjointed non-empty subsets RB and BS, that represent robots and base station respectively. In this case study robots perform the cleaning task. Hence our abstract set of eligible agents is completely represented by robots: $ELIG = RB$. Robots might be active or failed. We introduce the enumerated set $STATUS$, which in our case has two elements $\{active, failed\}$.

At previous refinement step we have modelled agents faults while performing their tasks in a very abstract way. Now we will specify them more concretely. We assume that only robots may fail in our multi-robotic system. Their dynamic status is stored in the variable rb_status:

$$rb\_status \in RB \rightarrow STATUS.$$

The abstract variable $elig$ is now data refined by the concrete set:

$$elig \;=\; \{a|a \in AGENTS \wedge atype(a) = RB \wedge rb\_status(a) = active\}.$$

The concrete events are also built according to the proposed pattern. For instance, the event FailCleaningSectors can now be specified as follows:

```
Machine MRS_Ref4
FailedCleaningSector ≙ refines FailedCleaningSector
  any zone, sector, ag
  when
     zone ∈ 1..n ∧ sector ∈ 1 .. k ∧
     sector_completed(zone)(sector) = FALSE ∧
     ag ∈ RB ∧ card({a|a ∈ RB ∧ rb_status(a) = active}) > 1
     rb_status(ag) = active
  then
     sector_completed(zone) := sector_completed(zone) ⩤ {sector ↦ FALSE}
     rb_status(ag) := failed
  end
```

An overview of the development of an autonomous multi-robotic system according to the proposed specification and refinement patterns is shown in the Fig. 1.

5 Conclusions

5.1 Discussion

In this paper we have proposed a formal goal-oriented approach to development of resilient MAS. We have demonstrated how to rigorously define goals in Event-B and ensure goal reachability by refinement. We have defined a set of modelling

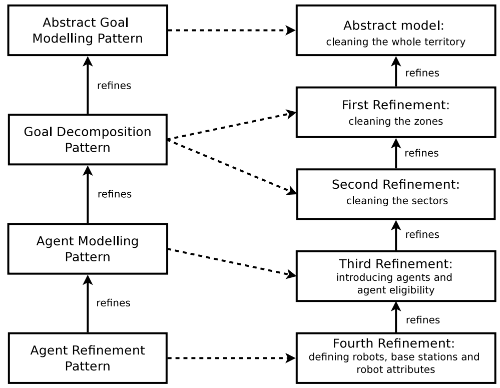

Figure 1: Overview of the development

and refinement patterns that describe generic solutions common to formal modelling of MAS. Rigorous modelling of the impact of agent failures on goal achieving allowed us to propose a dynamic goal reallocation mechanism that guarantees system resilience in presence of agent failures. We have illustrated our approach by a case study – development of an autonomic multi-robotic system.

While modelling the behaviour of multi-robotic system, we have shown that refinement process allows us also to discover restrictions that we have to impose on system behaviour to guarantee its resilience. In our case, the goal was achievable only if at least one robot remains healthy. Feasibility of such a restriction can be checked probabilistically based on the failure rates of robots. In our future work we are planning to integrate stochastic reasoning in our formal development. Moreover, it would be also interesting to experiment with different schemes for goal decomposition and dynamic goal reallocation.

5.2 Related Work

Our approach is different from numerous process-algebraic approaches used for modeling MAS. Firstly, we relied on proof-based verification that does not impose restrictions on the size of the model, number of agents etc. Secondly, we adopted a system's approach, i.e., we modeled the entire system and extracted the specifications of its individual components by decomposition. Such an approach allows us to ensure resilience by enabling goal reallocation at different architectural levels. Furthermore, by incrementally increasing complexity of our models, we have successfully managed to cope both with complexity of requirements and verification.

Formal modelling of MAS has been undertaken by [13, 12, 14]. The authors have proposed an extension of the Unity framework to explicitly define such concepts as mobility and context-awareness. Our modelling pursued a different goal – we aimed at formally guaranteeing that the specified agent behaviour achieves the defined goals. Formal modelling of fault tolerant MAS in Event-B has been undertaken by Ball and Butler [3]. They have proposed a number of informally described patterns that allow the designers to add well-known fault tolerance mechanisms to the specifications. In our approach, we implemented goal reallocation to guarantee goal reachability that can be also considered as a goal-specific fault tolerance.

The foundational work on goal-oriented development has been done by van Lamswerde [15]. The original motivation behind the goal-oriented development was to structure the requirements and derive properties in the form of temporal logic formulas that the system design should satisfy. Over the last decade the goal-oriented approach has received several extensions that allow the designers to link git with formal modelling [9, 6, 8]. These works aimed at expressing temporal logic properties in Event-B. In our work, we have relied on goals to facilitate structuring of system behaviour but derived system specification that satisfies the desired properties by refinement.

References

[1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.

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

[3] Elisabeth Ball and Michael Butler. Event-b patterns for specifying fault-tolerance in multi-agent interaction. In *Methods, Models and Tools for Fault Tolerance*, pages 104–129. 2009.

[4] EU-project DEPLOY. online at http://www.deploy-project.eu/.

[5] Marko Kääramees, Jüri Vain, and Kullo Raiend. Synthesis of on-line planning tester for non-deterministic efsm models. In *TAIC PART*, pages 147–154, 2010.

[6] R. De Landtsheer, E. Letier, and A. van Lamsweerde. Deriving tabular event-based specifications from goal-oriented requirements models. In *Requirements Engineering, 9(2)*, pages 104–120, 2004.

[7] J.C. Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, pages G8–G9, 2008.

[8] Abderrahman Matoussi, Frederic Gervais, and Regine Laleau. A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification. In *16th International Conference on Engineering of Complex Computer Systems*. IEEE, 2011.

[9] Christophe Ponsard, Gautier Dallons, and Massone Philippe. From Rigorous Requirements Engineering to Formal System Design of Safety-Critical Systems. In *ERCIM News (75)*, pages 22–23, 2008.

[10] Rigorous Open Development Environment for Complex Systems (RODIN). IST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/.

[11] RODIN. Event-B Platform. online at http://www.event-b.org/.

[12] G.-C. Roman, Ch. Julien, and J. Payton. A Formal Treatment of Context-Awareness. In *FASE'2004*, volume 2984 of *LNCS*. Springer, 2004.

[13] G.-C. Roman, Ch. Julien, and J. Payton. Modeling adaptive behaviors in Context UNITY. In *Theoretical Computure Science*, volume 376, pages 185–204, 2007.

[14] G.-C. Roman, P.McCann, and J. Plun. Mobile UNITY: Reasoning and Specification in Mobile Computing. In *ACM Transactions of Software Engineering and Methodology*, 1997.

[15] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE*, pages 249–263, 2001.

Appendix

MACHINE Top

SEES TopContext

VARIABLES

    gstate

INVARIANTS

    inv1 : $gstate \in GSTATE$

EVENTS

Initialisation

    begin

        act1 : $gstate :\in GSTATE \setminus Goal$

    end

Event *Reaching_Goal* $\widehat{=}$

Status anticipated

    when

        grd1 : $gstate \in GSTATE \setminus Goal$

    then

        act1 : $gstate :\in GSTATE$

    end

END

CONTEXT TopContext

SETS

    GSTATE

CONSTANTS

    Goal

AXIOMS

    axm1 : $Goal \subset GSTATE$

    axm2 : $Goal \neq \varnothing$

END

MACHINE Subgoals

REFINES Top

SEES SubgoalContext

VARIABLES

gstate1

gstate2

gstate3

prev_gstate1

prev_gstate2

prev_gstate3

INVARIANTS

inv1 : $gstate1 \in SG\_STATE1$

inv2 : $gstate2 \in SG\_STATE2$

inv3 : $gstate3 \in SG\_STATE3$

inv4 : $gstate = State\_map(gstate1 \mapsto gstate2 \mapsto gstate3)$

inv5 : $prev\_gstate1 \in SG\_STATE1$

inv6 : $prev\_gstate1 \in Subgoal1 \Rightarrow gstate1 \in Subgoal1$

inv7 : $prev\_gstate2 \in SG\_STATE2$

inv8 : $prev\_gstate2 \in Subgoal2 \Rightarrow gstate2 \in Subgoal2$

inv9 : $prev\_gstate3 \in SG\_STATE3$

inv10 : $prev\_gstate3 \in Subgoal3 \Rightarrow gstate3 \in Subgoal3$

EVENTS

Initialisation

begin

with

gstate′ : $\mathtt{gstate}' = \mathtt{State\_map}(\mathtt{gstate1}' \mapsto \mathtt{gstate2}' \mapsto \mathtt{gstate3}')$

act1 : $gstate1, prev\_gstate1 : |gstate1' \in SG\_STATE1 \setminus Subgoal1 \wedge prev\_gstate1' \in SG\_STATE1 \setminus Subgoal1 \wedge gstate1' = prev\_gstate1'$

act2 : $gstate2, prev\_gstate2 : |gstate2' \in SG\_STATE2 \setminus Subgoal2 \wedge prev\_gstate2' \in SG\_STATE2 \setminus Subgoal2 \wedge gstate2' = prev\_gstate2'$

act3 : $gstate3, prev\_gstate3 : |gstate3' \in SG\_STATE3 \setminus Subgoal3 \wedge prev\_gstate3' \in SG\_STATE3 \setminus Subgoal3 \wedge gstate3' = prev\_gstate3'$

end

Event *Reaching_SubGoal1* $\widehat{=}$

Status anticipated

refines *Reaching_Goal*

when

grd1 : $gstate1 \in SG\_STATE1 \setminus Subgoal1$

with

gstate' : $\mathtt{gstate}' = \mathtt{State\_map}(\mathtt{gstate1}' \mapsto \mathtt{gstate2} \mapsto \mathtt{gstate3})$

then

act1 : $gstate1 :\in SG\_STATE1$

act2 : $prev\_gstate1 := gstate1$

end

Event *Reaching_SubGoal2* $\widehat{=}$

Status anticipated

refines *Reaching_Goal*

when

grd1 : $gstate2 \in SG\_STATE2 \setminus Subgoal2$

with

gstate' : $\mathtt{gstate}' = \mathtt{State\_map}(\mathtt{gstate1} \mapsto \mathtt{gstate2}' \mapsto \mathtt{gstate3})$

then

act1 : $gstate2 :\in SG\_STATE2$

act2 : $prev\_gstate2 := gstate2$

end

Event *Reaching_SubGoal3* $\widehat{=}$

Status anticipated

refines *Reaching_Goal*

when

grd1 : $gstate3 \in SG\_STATE3 \setminus Subgoal3$

with

gstate' : $\mathtt{gstate}' = \mathtt{State\_map}(\mathtt{gstate1} \mapsto \mathtt{gstate2} \mapsto \mathtt{gstate3}')$

then

act1 : $gstate3 :\in SG\_STATE3$

act2 : $prev\_gstate3 := gstate3$

end

END

CONTEXT SubgoalContext

EXTENDS TopContext

SETS

SG_STATE1

SG_STATE2

SG_STATE3

CONSTANTS

Subgoal1

Subgoal2

Subgoal3

State_map

AXIOMS

axm1 : $Subgoal1 \subset SG\_STATE1$

axm2 : $Subgoal1 \neq \varnothing$

axm3 : $Subgoal2 \subset SG\_STATE2$

axm4 : $Subgoal2 \neq \varnothing$

axm5 : $Subgoal3 \subset SG\_STATE3$

axm6 : $Subgoal3 \neq \varnothing$

axm7 : $State\_map \in SG\_STATE1 \times SG\_STATE2 \times SG\_STATE3 \twoheadrightarrow GSTATE$

axm8 : $\forall sg1, sg2, sg3 \cdot sg1 \in Subgoal1 \wedge sg2 \in Subgoal2 \wedge sg3 \in Subgoal3 \Leftrightarrow State\_map(sg1 \mapsto sg2 \mapsto sg3) \in Goal$

END

MACHINE Agents

REFINES Subgoals

SEES AgentContex

VARIABLES

$\mathtt{gstate1}$

$\mathtt{gstate2}$

$\mathtt{gstate3}$

$\mathtt{elig1}$

$\mathtt{elig2}$

$\mathtt{elig3}$

$\mathtt{prev\_gstate1}$

$\mathtt{prev\_gstate2}$

$\mathtt{prev\_gstate3}$

INVARIANTS

$\mathtt{inv1}: \ elig1 \subseteq EL\_AG1$

$\mathtt{inv2}: \ elig1 \neq \varnothing$

$\mathtt{inv3}: \ elig2 \subseteq EL\_AG2$

$\mathtt{inv4}: \ elig2 \neq \varnothing$

$\mathtt{inv5}: \ elig3 \subseteq EL\_AG3$

$\mathtt{inv6}: \ elig3 \neq \varnothing$

EVENTS

Initialisation

extended

begin

$\mathtt{act1}: \ \mathtt{gstate1}, \mathtt{prev\_gstate1} : |\mathtt{gstate1}' \in \mathtt{SG\_STATE1} \setminus \mathtt{Subgoal1} \wedge$
$\mathtt{prev\_gstate1}' \in \mathtt{SG\_STATE1} \setminus \mathtt{Subgoal1} \wedge \mathtt{gstate1}' = \mathtt{prev\_gstate1}'$

$\mathtt{act2}: \ \mathtt{gstate2}, \mathtt{prev\_gstate2} : |\mathtt{gstate2}' \in \mathtt{SG\_STATE2} \setminus \mathtt{Subgoal2} \wedge$
$\mathtt{prev\_gstate2}' \in \mathtt{SG\_STATE2} \setminus \mathtt{Subgoal2} \wedge \mathtt{gstate2}' = \mathtt{prev\_gstate2}'$

$\mathtt{act3}: \ \mathtt{gstate3}, \mathtt{prev\_gstate3} : |\mathtt{gstate3}' \in \mathtt{SG\_STATE3} \setminus \mathtt{Subgoal3} \wedge$
$\mathtt{prev\_gstate3}' \in \mathtt{SG\_STATE3} \setminus \mathtt{Subgoal3} \wedge \mathtt{gstate3}' = \mathtt{prev\_gstate3}'$

$\mathtt{act5}: \ elig1 := EL\_AG1$

$\mathtt{act6}: \ elig2 := EL\_AG2$

$\mathtt{act7}: \ elig3 := EL\_AG3$

end

Event *Fail_in_Reaching_SubGoal1* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal1*

any

ag

where

grd1 : $gstate1 \in SG\_STATE1 \setminus Subgoal1$

grd2 : $ag \in elig1$

grd3 : $card(elig1) \geq 2$

then

act1 : $gstate1 :\in SG\_STATE1 \setminus Subgoal1$

act2 : $prev\_gstate1 := gstate1$

act3 : $elig1 := elig1 \setminus \{ag\}$

end

Event *Reaching_SubGoal1* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal1*

any

ag

where

grd1 : $gstate1 \in SG\_STATE1 \setminus Subgoal1$

grd2 : $ag \in elig1$

then

act1 : $gstate1 :\in Subgoal1$

act2 : $prev\_gstate1 := gstate1$

end

Event *Fail_in_Reaching_SubGoal2* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal2*

any

ag

where

grd1 : $gstate2 \in SG\_STATE2 \setminus Subgoal2$

grd2 : $ag \in elig2$

grd3 : $card(elig2) \geq 2$

then

act1 : $gstate2 :\in SG\_STATE2 \setminus Subgoal2$

act2 : $prev\_gstate2 := gstate2$

act3 : $elig2 := elig2 \setminus \{ag\}$

end

Event *Reaching_SubGoal2* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal2*

any

ag

where

grd1 : $gstate2 \in SG\_STATE2 \setminus Subgoal2$

grd2 : $ag \in elig2$

then

act1 : $gstate2 :\in Subgoal2$

act2 : $prev\_gstate2 := gstate2$

end

Event *Fail_in_Reaching_SubGoal3* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal3*

any

ag

where

grd1 : $gstate3 \in SG\_STATE3 \setminus Subgoal3$

grd2 : $ag \in elig3$

grd3 : $card(elig3) \geq 2$

then

act1 : $gstate3 :\in SG\_STATE3 \setminus Subgoal3$

act2 : $prev\_gstate3 := gstate3$

act3 : $elig3 := elig3 \setminus \{ag\}$

end

Event *Reaching_SubGoal3* $\widehat{=}$

Status convergent

refines *Reaching_SubGoal3*

any

ag

where

grd1 : $gstate3 \in SG\_STATE3 \setminus Subgoal3$

grd2 : $ag \in elig3$

then

act1 : $gstate3 :\in Subgoal3$

act2 : $prev\_gstate3 := gstate3$

end

VARIANT

$\mathtt{card}(\mathtt{elig1}) + \mathtt{card}(\mathtt{elig2}) + \mathtt{card}(\mathtt{elig3}) +$

$\mathtt{bnat1}(\mathtt{gstate1}) + \mathtt{bnat2}(\mathtt{gstate2}) + \mathtt{bnat3}(\mathtt{gstate3})$

END

CONTEXT AgentContex
EXTENDS SubgoalContext
SETS

AGENTS

CONSTANTS

EL_AG1

EL_AG2

EL_AG3

bnat1

bnat2

bnat3

AXIOMS

axm1 : $AGENTS \neq \varnothing$

axm2 : $EL\_AG1 \cup EL\_AG2 \cup EL\_AG3 \subseteq AGENTS$

axm3 : $EL\_AG1 \neq \varnothing$

axm4 : $EL\_AG2 \neq \varnothing$

axm5 : $EL\_AG3 \neq \varnothing$

axm6 : $finite(AGENTS)$

axm7 : $bnat1 \in SG\_STATE1 \rightarrow \mathbb{N}$

axm8 : $\forall s \cdot s \in Subgoal1 \Rightarrow bnat1(s) = 0$

axm9 : $\forall s \cdot s \in SG\_STATE1 \setminus Subgoal1 \Rightarrow bnat1(s) = 1$

axm10 : $bnat2 \in SG\_STATE2 \rightarrow \mathbb{N}$

axm11 : $\forall s \cdot s \in Subgoal2 \Rightarrow bnat2(s) = 0$

axm12 : $\forall s \cdot s \in SG\_STATE2 \setminus Subgoal2 \Rightarrow bnat2(s) = 1$

axm13 : $bnat3 \in SG\_STATE3 \rightarrow \mathbb{N}$

axm14 : $\forall s \cdot s \in Subgoal3 \Rightarrow bnat3(s) = 0$

axm15 : $\forall s \cdot s \in SG\_STATE3 \setminus Subgoal3 \Rightarrow bnat3(s) = 1$

END

MACHINE AgentsRef

REFINES Agents

SEES AgentContexExtended

VARIABLES

gstate1

gstate2

gstate3

astatus

prev_gstate1

prev_gstate2

prev_gstate3

INVARIANTS

inv1 : $astatus \in AGENTS \rightarrow AG\_STATUS$

inv4 : $\{a|a \in AGENTS \land atype(a) = TYPE1 \land astatus(a) = OK\} = elig1$

inv5 : $\{a|a \in AGENTS \land atype(a) = TYPE2 \land astatus(a) = OK\} = elig2$

inv6 : $\{a|a \in AGENTS \land atype(a) = TYPE3 \land astatus(a) = OK\} = elig3$

EVENTS

Initialisation

begin

act1 : $gstate1, prev\_gstate1 :| gstate1' \in SG\_STATE1 \setminus Subgoal1 \land prev\_gstate1' \in SG\_STATE1 \setminus Subgoal1 \land gstate1' = prev\_gstate1'$

act2 : $gstate2, prev\_gstate2 :| gstate2' \in SG\_STATE2 \setminus Subgoal2 \land prev\_gstate2' \in SG\_STATE2 \setminus Subgoal2 \land gstate2' = prev\_gstate2'$

act3 : $gstate3, prev\_gstate3 :| gstate3' \in SG\_STATE3 \setminus Subgoal3 \land prev\_gstate3' \in SG\_STATE3 \setminus Subgoal3 \land gstate3' = prev\_gstate3'$

act6 : $astatus := AGENTS \times \{OK\}$

end

Event *Fail_in_Reaching_SubGoal1* $\widehat{=}$

Status anticipated

refines *Fail_in_Reaching_SubGoal1*

any

ag

where

grd1 : $gstate1 \in SG\_STATE1 \setminus Subgoal1$

grd2 : $astatus(ag) = OK$

grd4 : $atype(ag) = TYPE1$

grd5 : $card(\{a|a \in AGENTS \land atype(a) = TYPE1 \land astatus(a) = OK\}) \geq 2$

then

act1 : $gstate1 :\in SG\_STATE1 \setminus Subgoal1$

act2 : $prev\_gstate1 := gstate1$

act3 : $astatus(ag) := KO$

end

Event *Reaching_SubGoal1* $\widehat{=}$

refines *Reaching_SubGoal1*

any

ag

where

grd1 : $gstate1 \in SG\_STATE1 \setminus Subgoal1$

grd2 : $astatus(ag) = OK$

grd3 : $atype(ag) = TYPE1$

then

act1 : $gstate1 :\in Subgoal1$

act2 : $prev\_gstate1 := gstate1$

end

Event *Fail_in_Reaching_SubGoal2* $\widehat{=}$

Status anticipated

refines *Fail_in_Reaching_SubGoal2*

any

ag

where

grd1 : $gstate2 \in SG\_STATE2 \setminus Subgoal2$

grd2 : $astatus(ag) = OK$

grd4 : $atype(ag) = TYPE2$

grd5 : $card(\{a|a \in AGENTS \land atype(a) = TYPE2 \land astatus(a) = OK\}) \geq 2$

then

act1 : $gstate2 :\in SG\_STATE2 \setminus Subgoal2$

act2 : $prev\_gstate2 := gstate2$

act3 : $astatus(ag) := KO$

end

Event *Reaching_SubGoal2* $\widehat{=}$

refines *Reaching_SubGoal2*

any

ag

where

grd1 : $gstate2 \in SG\_STATE2 \setminus Subgoal2$

grd2 : $astatus(ag) = OK$
grd3 : $atype(ag) = TYPE2$

then

act1 : $gstate2 :\in Subgoal2$
act2 : $prev\_gstate2 := gstate2$

end

Event *Fail_in_Reaching_SubGoal3* $\widehat{=}$

Status anticipated

refines *Fail_in_Reaching_SubGoal3*

any

ag

where

grd1 : $gstate3 \in SG\_STATE3 \setminus Subgoal3$
grd2 : $astatus(ag) = OK$
grd4 : $atype(ag) = TYPE3$
grd5 : $card(\{a|a \in AGENTS \land atype(a) = TYPE3 \land astatus(a) = OK\}) \geq 2$

then

act1 : $gstate3 :\in SG\_STATE3 \setminus Subgoal3$
act2 : $prev\_gstate3 := gstate3$
act3 : $astatus(ag) := KO$

end

Event *Reaching_SubGoal3* $\widehat{=}$

refines *Reaching_SubGoal3*

any

ag

where

grd1 : $gstate3 \in SG\_STATE3 \setminus Subgoal3$
grd2 : $astatus(ag) = OK$
grd3 : $atype(ag) = TYPE3$

then

act1 : $gstate3 :\in Subgoal3$
act2 : $prev\_gstate3 := gstate3$

end

END

CONTEXT AgentContexExtended

EXTENDS AgentContex

SETS

AG_STATUS

AG_TYPES

CONSTANTS

OK

KO

TYPE1

TYPE2

TYPE3

atype

AXIOMS

axm1 : $partition(AG\_STATUS, \{OK\}, \{KO\})$

axm2 : $partition(AG\_TYPES, \{TYPE1\}, \{TYPE2\}, \{TYPE3\})$

axm3 : $atype \in AGENTS \rightarrow AG\_TYPES$

axm4 : $\forall ag \cdot ag \in EL\_AG1 \Leftrightarrow atype(ag) = TYPE1$

axm8 : $\forall ag \cdot ag \in EL\_AG2 \Leftrightarrow atype(ag) = TYPE2$

axm9 : $\forall ag \cdot ag \in EL\_AG3 \Leftrightarrow atype(ag) = TYPE3$

END

MACHINE MRS_Abs
VARIABLES

`completed`

INVARIANTS

`inv2` : $completed \in BOOL$

EVENTS
Initialisation

begin

`act1` : $completed := FALSE$

end

Event *CleaningTerritory* $\widehat{=}$
Status anticipated

when

`grd1` : $completed = FALSE$

then

`act1` : $completed :\in BOOL$

end

END

MACHINE MRS_Ref1
REFINES MRS_Abs
SEES cntx1
VARIABLES

zone_completed

INVARIANTS

inv1 : $zone\_completed \in 1 \,..\, n \rightarrow BOOL$
inv2 : $zone\_completed[1 \,..\, n] = \{TRUE\} \Leftrightarrow completed = TRUE$

EVENTS
Initialisation

begin

act2 : $zone\_completed := 1 \,..\, n \times \{FALSE\}$

end

Event *CleaningZones* $\widehat{=}$
Status anticipated
refines *CleaningTerritory*

any

$zone$
$zone\_result$

where

grd2 : $zone \in 1 \,..\, n$
grd3 : $zone\_completed(zone) = FALSE$
grd4 : $zone\_result \in BOOL$

with

completed′ : $\mathtt{completed}' = \mathtt{bool}(\mathtt{zone\_completed}'[1..n] = \{\mathtt{TRUE}\})$

then

act2 : $zone\_completed(zone) := zone\_result$

end

END

CONTEXT cntx1
CONSTANTS

n

AXIOMS

axm1 : $n \in \mathbb{N}_1$

END

MACHINE MRS_Ref2
REFINES MRS_Ref1
SEES cntx2
VARIABLES

    sector_completed

INVARIANTS

    inv1 : $sector\_completed \in 1 \,..\, n \rightarrow (1 \,..\, k \rightarrow BOOL)$
    inv2 : $\forall sg \cdot sg \in 1..n \Rightarrow (zone\_completed(sg) = TRUE \Leftrightarrow sector\_completed(sg)[1..k] = \{TRUE\})$

EVENTS

Initialisation

    begin
        act1 : $sector\_completed := 1 \,..\, n \times \{1 \,..\, k \times \{FALSE\}\}$
    end

Event *CleaningSector* $\widehat{=}$
Status anticipated
refines *CleaningZones*

    any
        $zone$
        $sector$
        $sector\_result$
    where
        grd1 : $zone \in 1 \,..\, n$
        grd2 : $sector \in 1 \,..\, k$
        grd3 : $sector\_completed(zone)(sector) = FALSE$
        grd4 : $sector\_result \in BOOL$
    with
        zone_result : $\mathtt{zone\_result} = \mathtt{bool}(\mathtt{sector\_completed}'(\mathtt{zone})[1..\mathtt{k}] = \{\mathtt{TRUE}\})$
    then
        act1 : $sector\_completed(zone) := sector\_completed(zone) \Leftarrow \{sector \mapsto sector\_result\}$
    end

END

CONTEXT cntx2

EXTENDS cntx1

CONSTANTS

k

AXIOMS

axm1 : $k \in \mathbb{N}_1$

END

An Event-B Specification of MRS_Ref3
Creation Date: 14 Feb 2012 @ 00:42:23 PM

MACHINE MRS_Ref3
REFINES MRS_Ref2
SEES cntx3
VARIABLES

sector_completed
elig
counter

INVARIANTS

inv1 : $elig \subseteq ELIG$
inv2 : $elig \neq \varnothing$
inv3 : $counter \in 0 \,..\, n * k$

EVENTS

Initialisation

extended

begin

act1 : $\mathtt{sector\_completed} := 1 \,..\, \mathtt{n} \times \{1 \,..\, \mathtt{k} \times \{\mathtt{FALSE}\}\}$
act2 : $elig := ELIG$
act3 : $counter := n * k$

end

Event *FailedCleaningSector* $\widehat{=}$
Status convergent
refines *CleaningSector*

any

$zone$
$sector$
ag

where

grd1 : $zone \in 1 \,..\, n$
grd2 : $sector \in 1 \,..\, k$
grd3 : $sector\_completed(zone)(sector) = FALSE$
grd4 : $card(elig) \geq 2$
grd5 : $ag \in elig$

with

sector_result : $\mathtt{sector\_result} = \mathtt{FALSE}$

then

act1 : $sector\_completed(zone) := sector\_completed(zone) \Leftarrow\!\!+ \{sector \mapsto FALSE\}$
act2 : $elig := elig \setminus \{ag\}$

end

Event *SuccessfulCleaningSector* $\widehat{=}$

Status convergent

refines *CleaningSector*

any

$zone$
$sector$
ag

where

grd1 : $zone \in 1 \,..\, n$
grd2 : $sector \in 1 \,..\, k$
grd3 : $sector\_completed(zone)(sector) = FALSE$
grd4 : $ag \in elig$
grd5 : $counter > 0$

with

sector_result : $\mathtt{sector\_result} = \mathtt{TRUE}$

then

act1 : $sector\_completed(zone) := sector\_completed(zone) \Leftarrow\!\!+ \{sector \mapsto TRUE\}$
act2 : $counter := counter - 1$

end

VARIANT

$\mathtt{card}(\mathtt{elig}) + \mathtt{counter}$

END

CONTEXT cntx3
EXTENDS cntx2
SETS

    AGENTS

CONSTANTS

    ELIG
    bnat

AXIOMS

axm1 : $finite(AGENTS)$
axm2 : $AGENTS \neq \varnothing$
axm3 : $ELIG \subseteq AGENTS$
axm4 : $ELIG \neq \varnothing$

END

MACHINE MRS_Ref4
REFINES MRS_Ref3
SEES cntx4
VARIABLES

sector_completed
rb_status
counter

INVARIANTS

inv1 : $rb\_status \in RB \rightarrow AG\_STATUS$
inv2 : $\{a | a \in RB \wedge rb\_status(a) = active\} = elig$

EVENTS
Initialisation

begin
act1 : $sector\_completed := 1 .. n \times \{1 .. k \times \{FALSE\}\}$
act2 : $rb\_status := RB \times \{active\}$
act3 : $counter := n * k$
end

Event *SuccessfulCleaningSector* $\widehat{=}$
refines *SuccessfulCleaningSector*

any
$zone$
$sector$
ag
where
grd1 : $zone \in 1 .. n$
grd2 : $sector \in 1 .. k$
grd3 : $sector\_completed(zone)(sector) = FALSE$
grd4 : $ag \in RB$
grd5 : $rb\_status(ag) = active$
grd6 : $counter > 0$
then
act1 : $sector\_completed(zone) := sector\_completed(zone) \Leftarrow \{sector \mapsto TRUE\}$
act2 : $counter := counter - 1$
end

Event *FailedCleaningSector* $\widehat{=}$
refines *FailedCleaningSector*

any
$zone$

$sector$
ag

where

grd1 : $zone \in 1 \,..\, n$
grd2 : $sector \in 1 \,..\, k$
grd3 : $sector\_completed(zone)(sector) = FALSE$
grd4 : $card(\{a|a \in RB \wedge rb\_status(a) = active\}) \geq 2$
grd5 : $ag \in RB$
grd6 : $rb\_status(ag) = active$

then

act1 : $sector\_completed(zone) := sector\_completed(zone) \lhd\!\!- \{sector \mapsto FALSE\}$
act2 : $rb\_status(ag) := failed$

end

END

CONTEXT cntx4
EXTENDS cntx3
SETS

AG_STATUS

CONSTANTS

BS
RB
active
failed

AXIOMS

axm2 : $RB \subset AGENTS$
axm3 : $BS \subset AGENTS$
axm4 : $RB \neq \varnothing$
axm5 : $BS \neq \varnothing$
axm6 : $partition(AGENTS, RB, BS)$
axm7 : $partition(AG\_STATUS, \{active\}, \{failed\})$
axm8 : $ELIG = RB$

END

TURKU

CENTRE *for*

COMPUTER

SCIENCE

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-2701-1
ISSN 1239-1891