Inna Pereverzeva | Elena Troubitsyna | Linas Laibinis

# A Case Study in a Formal Development of a Fault Tolerant Multi-Robotic System

Turku Centre *for* Computer Science

TUCS Technical Report
No 1052, July 2012

# A Case Study in a Formal Development of a Fault Tolerant Multi-Robotic System

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 1052, July 2012

**Abstract**

Multi-robotic systems are typical examples of complex multi-agent systems. The robots – autonomic agents – cooperate with each other in order to achieve the system goals. While designing multi-robotic systems, we should ensure that these goals remain achievable despite robot failures, i.e., guarantee system fault tolerance. However, designing the fault tolerance mechanisms for multi-agent systems is a notoriously difficult task. In this paper we describe a case study in formal development of a complex fault tolerant multi-robotic system. The system design relies on cooperative error recovery and dynamic reconfiguration. We demonstrate how to specify and verify essential properties of a fault tolerant multi-robotic system in Event-B and derive a detailed formal system specification by refinement. The main objective of the presented case study is to investigate suitability of a refinement approach for specifying a complex multi-agent system with co-operative error recovery.

**Keywords:** Event-B, formal modelling, refinement, fault tolerance, multi-robotic system.

**TUCS Laboratory**
Distributed Systems Laboratory

# 1 Introduction

Over the last decade, the field of autonomous multi-robotic systems has grown dramatically. There are several research directions that are continuously receiving significant attention: autonomous navigation and control, self-organising behaviour, architectures for multi-robot co-operation, to name a few. The robot co-operation is studied from a variety of perspectives: delegation of authority and control, heterogeneous versus homogeneous architectures, communication structure etc. In this paper we focus on studying the fault tolerance aspects of multi-robotic co-operation. Namely, we show by example how to formally derive a specification of a multi-robotic system that relies on dynamic reconfiguration and co-operative error recovery to achieve fault tolerance.

Our paper presents a case study in formal development of a cleaning multi-robotic system. That kind of systems are typically employed in hazardous areas. The system has a heterogenous architecture consisting of several stationary devices, base stations, that coordinate the work of respective groups of robots. A base station assigns a robot to clean a certain segment. Since both base stations and robots can fail, the main objective of our formal development is to formally specify co-operative error recovery and verify that the proposed design ensures goal reachability, i.e., guarantees that the whole territory will be eventually cleaned. The proposed development approach ensures goal reachability "by construction". It is based on refinement in Event-B [2] – a formal top-down approach to correct-by-construction system development. The main development technique – refinement – allows us to ensure that a resulting specification preserves the globally observable behaviour and properties of the specifications it refines. The Rodin platform [7] automates modelling and verification in Event-B.

In this paper we demonstrate how to formally define a system goal and, in a stepwise manner, *derive* a detailed specification of the system architecture. While refining the system specification, we gradually introduce a representation of the main elements of the architecture – base stations and robots – as well as failures and the fault tolerance mechanisms. Moreover, we identify the main properties of a fault tolerant multi-robotic system and demonstrate how to formally specify and verify them as a part of the refinement process. In particular, we show how to derive a mechanism for cooperative error recovery in a systematic way.

Traditionally, the behaviour of multi-robotic systems is verified by simulation and model checking. These approaches allow the designers to investigate only a limited number of scenarios and require a significant reduction of the state space. In our paper, we discuss advantages and limitations of a refinement approach to achieve full-scale verification of a multi-robotic system.

The paper is structured as follows. In Section 2 we briefly overview the

Event-B formalism. Section 3 describes the requirements for our case study – a multi-robotic cleaning system – and outlines the development strategy. Section 4 presents a formal development of the cleaning system and demonstrates how to express and verify its properties in the refinement process. Finally, in Section 5 we conclude by assessing our contributions and reviewing the related work.

# 2 Modelling and Refinement in Event-B

The Event-B formalism – a variation of the B Method [1] – is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine* [2]. An abstract state machine encapsulates the model state represented as a collection of variables and defines operations on the state, i.e., it describes the *behaviour* of the modelled system. Usually, a machine has an accompanying component, called *context*, which may include user-defined carrier sets, constants and their properties given as a list of model axioms. In Event-B, the model variables are strongly typed by the constraining predicates. These predicates and the other important properties that must be preserved by the model constitute model *invariants*.

The dynamic behaviour of the system is defined by a set of atomic *events*. Generally, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where $e$ is the event's name, $a$ is the list of local variables, the *guard* $G_e$ is a predicate over the local variables of the event and the state variables of the system. The body of the event is defined by the next-state relation $R_e$. In Event-B, $R_e$ is defined by a *multiple* (possibly nondeterministic) assignment over the system variables. The guard defines the conditions under which the assignment can be performed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically. If an event does not have local variables, it can be described simply as:

$$e \mathrel{\widehat{=}} \textbf{when } G_e \textbf{ then } R_e \textbf{ end}.$$

Event-B employs a top-down refinement-based approach to system development. A 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 add new events, split events as well as replace abstract variables by their concrete counterparts, i.e., perform *data refinement*. When data refinement is performed,

we should define so called *gluing invariant* as a part of the invariant of the refined machine. The gluing invariant defines the relationship between the abstract and concrete variables.

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 abstraction. Then the event obtains the status *anticipated*. This obliges the designer to prove, at some later refinement step, that the event indeed converges.

The consistency of Event-B models, i.e., verification of well-formedness and invariant preservation as well as correctness of refinement steps, is formally demonstrated by discharging the relevant proof obligations generated by the Rodin platform [7]. Rodin also provides an automated tool support for proving.

# 3 Multi-Robotic Systems

Our paper focuses on formal modelling and development of multi-robotic systems that should function autonomously , i.e., without human intervention. Such kind of systems are often deployed in hazardous areas, e.g., nuclear power plants, disaster areas, minefields, etc.

Typically, the main task or *goal* that a multi-robotic system should accomplish is split between the deployed robots. The robot activities are coordinated by a number of stationary units – base stations. Since both robots and base stations may fail, to ensure success of the overall goal we should incorporate the fault tolerance mechanisms into the system design. These mechanisms rely on co-operative error recovery that allows the system dynamically reallocate functions from the failed agents to the healthy ones.

Designing co-operative error recovery for multi-agent systems is a notoriously complex task. The complexity is caused by several factors: asynchronous communication, a highly decentralised system architecture and the lack of the "universally known" global system state. Yet, the designers should guarantee that the system goals are achievable despite failures. A variety of failure modes and scenarios makes verification of goal reachability of the co-operative error recovery difficult and time-consuming. Therefore, there is a clear need for rigorous approaches that support scalable design and verification in a systematic manner.

Next we present the requirements of our case study – a multi-robotic system for cleaning a territory. Then we will demonstrate how we can formally develop the system in Event-B and prove its essential properties.

## 3.1 A Case Study: Cleaning a Territory

The goal of the system is to get a certain territory cleaned by robots. The whole territory is divided into several *zones*, which in turn are further divided into a number of *sectors*. Each zone has a *base station* that coordinates the cleaning activities within the zone. In general, one base station might coordinate several zones. In its turn, each base station supervises a number of robots attached to it by assigning cleaning tasks to them.

A robot is an autonomous electro-mechanical device that can move and clean. A base station may assign a robot a specific sector to clean. Upon receiving the assignment, the robot autonomously moves to this sector and performs cleaning. After successfully completing its mission, the robot returns back to the base station to receive a new assignment. The base station keeps track of the cleaned and non-cleaned sectors. Moreover, the base stations periodically exchange the information about their cleaned sectors.

While performing the given task, a robot might fail which subsequently leads to a failure to clean the assigned sector. We assume that a base station is able to detect all the failed robots attached to it. In case of a robot failure, the base station may assign another active robot to perform the failed task.

A base station might fail as well. We assume that a failure of a base station can be detected by the others base stations. In that case, the healthy base stations redistribute control over the robots coordinated by the failed base station.

Let us now to formulate the main requirements and properties associated with the multi-robotic system informally described above.

(PR1) *The main system goal: the whole territory has to be cleaned.*

(PR2) *To clean the territory, every its zone has to be cleaned.*

(PR3) *To clean a zone, every its sector has to be cleaned.*

(PR4) *Every cleaned sector or zone remains cleaned during functioning of the system.*

(PR5) *No two robots should clean the same sector.* In other words, a robot gets only non-assigned and non-cleaned sectors to clean.

(PR6) *The information about the cleaned sectors stored in any base station has to be consistent with the current state of the territory.* More specifically, if a base station sees a particular sector in some zone as cleaned,

then this sector is marked as cleaned in the memory of the base station responsible for it. Also, if a sector is marked as non-cleaned in the memory of the base station responsible for it, then any base station sees it as non-cleaned.

(PR7) *Base station cooperation: if a base station has been detected as failed then some base station will take the responsibility for all the zones and robots of the failed base station.*

(PR8) *Base station cooperation: if a base station has no more active robots, a group of robot is sent to this base station from another base station.*

(PR9) *Base station cooperation: if a base station has cleaned all its zones, its active robots may be reallocated under control of another base station.*

The last three requirements essentially describe the co-operative recovery mechanisms that we assume to be present in the described multi-robot system.

## 3.2 Formal Development Strategy

In the next section we will present a formal Event-B development of the described multi-system robotic system. We demonstrate how to specify and verify the given properties (PR1)–(PR9). Let us now give a short overview of this development and highlight formal techniques used to ensure the proposed properties.

We start with a very abstract model, essentially representing the system behaviour as a process iteratively trying to achieve the main goal (PR1). The next couple of data refinement steps decompose the main goal into a set of subgoals, i.e., reformulate it in the terms of zones and sectors. We will define and prove the relevant gluing invariants establishing a formal relationship between goals and the corresponding subgoals.

While the specification remains highly abstract, we postulate goal reachability property by defining *anticipate* status for the involved events. Once, as a result of the refinement process, the model becomes sufficiently detailed, we change the event status into *convergent* and prove their termination by supplying the appropriate variant expression.

Next we introduce different types of agents (i.e., base stations and robots). The base stations coordinate execution of the tasks required to achieve the corresponding subgoal, while the robots execute the tasks allocated on them. We formally define the relationships between different types of agents, as well as agents and respective subgoals. These relationships are specified and proved as invariant properties of the model.

The consequent refinement steps explicitly introduce agent failures, the information exchange as well as co-operation activities between the agents.

The integrity between the local and the global information stored within base stations is again formulated and proved as model invariant properties.

We assume that communication between the base stations as well as the robots and the base stations is reliable. In other words, messages are always transmitted correctly without any loss or errors. The main focus of our development is on specifying and verifying the co-operative recovery mechanisms.

# 4 Development of a Multi-Robotic System in Event-B

## 4.1 Abstract Model

We start our development by abstractly modelling the described multi-robotic system. We aim to ensure the property (PR1). The main system goal is to clean the whole territory. The process of achieving this goal is modelled by the simple event Body presented below. A variable $goal \in STATE$ models the current state of the system goal. It obtains values from the enumerated set $STATE = \{incompl, compl\}$, where the value *compl* corresponds to the situation when the goal is achieved, otherwise it is equal to *incompl*. The system continues its execution until the whole territory is not cleaned, i.e., while *goal* stays *incompl*.

Body $\widehat{=}$
**status** *anticipated*
**when**
$goal \neq compl$
**then**
$goal :\in STATE$
**end**

The event Body has the status *anticipated*. This means that goal reachability is postulated rather than proved. However, at some refinement step it also obliges us to prove that the event or its refinements converge, i.e., to prove that the process of achieving goal eventually terminates.

## 4.2 First Refinement: Zone Cleaning

Our initial model represents the system behaviour at a high level of abstraction. The objective of our first refinement step is to elaborate on the process of cleaning the territory. Specifically, we assume that the whole territory is divided into $n$ zones, where $n \in \mathbb{N}$ and $n \geq 1$, and aim at ensuring the property (PR2).

We augment our model with a representation of subgoals. We also associate the notion of a *subgoal* with the process of *cleaning a particular zone*. A subgoal is achieved only when the corresponding zone is cleaned. A new variable *zones* represents the current subgoal status for every zone:

$$zones \in 1..n \rightarrow STATE.$$

In this refinement step we perform a data refinement: we replace the abstract variable *goal* with a new variable *zones*. To establish the relationship between those variables, we formulate the following gluing invariant:

$$goal = compl \Leftrightarrow zones[1..n] = \{compl\}.$$

The invariant can be understood as follows: the territory is considered to be cleaned if and only if its every zone is cleaned. Hence, hereby we have formalised the property (PR2). The refined event Body is presented below:

Body $\widehat{=}$ refines Body
**status** *anticipated*
**any** $z, res$
**when**
$z \in 1..n \ \wedge \ zones(z) \neq compl \wedge res \in STATE$
**then**
$zones(z) := res$
**end**

Moreover, while a certain subgoal is reached, it stays such, i.e., the system always progresses towards achieving its goals. Thereby we ensure the property (PR4).

## 4.3 Second Refinement: Sector Cleaning

In the next refinement step we further decompose system subgoals into a set of subsubgoals. Specifically, we assume that each zone in our system is divided into $k$ sectors, where $k \in \mathbb{N}$ and $k \geq 1$, and aim at formalising the property (PR3). We establish the relationship between the notion of a subsubgoal (or simply *a task*) and the process of *cleaning a particular sector*. A task is completed when the corresponding sector is cleaned. A new variable *territory* represents the current status of each sector:

$$territory \in 1 \mathinner{..} n \rightarrow (1 \mathinner{..} k \rightarrow STATE).$$

The refinement step is again an example of a data refinement. Indeed, we replace the abstract variable *zones* with a new variable *territory*. The following gluing invariant expresses the relationship between subgoals and subsubgoals (tasks) and correspondingly ensures the property (PR3):

$$\forall j \cdot j \in 1 \mathinner{..} n \Rightarrow (zones(j) = compl \Leftrightarrow territory(j)[1 \mathinner{..} k] = \{compl\}).$$

The invariant postulates that a zone is cleaned if and only if each of its sectors is cleaned.

The abstract event Body is further refined. It is now models cleaning of a previously non-cleaned sector $s$ in a zone $z$. The task is achieved when this sector is eventually cleaned, i.e., *result* becomes *compl*.

Body $\widehat{=}$ refines Body
**status** *anticipated*
**any** $z, s, result$
**when**
$zone \in 1..n \ \wedge s \in 1 \mathinner{..} k \ \wedge territory(z)(s) \neq compl \ \wedge result \in STATE$
**then**
$territory(z) := territory(z) \lhd\!\!\!\!- \{s \mapsto result\}$
**end**

Let us observe that the event Body also preserve the property (PR4).

At this refinement step we have achieved a sufficient level of detail to introduce an explicit representation of the agents – base stations and robots. This constitutes the main objective of our next refinement step.

## 4.4 Third Refinement: Introducing Agents

We start by defining, in the model context, the abstract finite set $AGENTS$ and its disjointed non-empty subsets $RB$ and $BS$ that represent the robots and the base stations respectively. To define a relationship between a zone and its supervising base station, we introduce the variable *responsible*, which is defined as the following total function:

$$responsible \in 1 \mathinner{..} n \rightarrow BS.$$

Each robot is supervised by a certain base station. During system execution robots might become inactive (failed). We model the relationship between robots and their supervised station by a variable *attached*, defined as partial function:

$$attached \in RB \nrightarrow BS.$$

The new function variables $asgn\_z$ and $asgn\_s$ model respectively the zone and the sector assigned to a robot to clean. When a robot is idle, i.e., it does not have a task assigned to it, the corresponding function value is 0:

$$asgn\_z \in RB \nrightarrow 0 \mathinner{..} n,\ asgn\_s \in RB \nrightarrow 0 \mathinner{..} k.$$

We require that only the robots that have a supervisory base station might receive a cleaning task:

$$dom(attached) = dom(asgn\_z),\ dom(asgn\_z) = dom(asgn\_s).$$

Now we can formulate the property (PR5) – *no two robots can clean the certain sector at the same time* – as a model invariant:

$$\forall rb1, rb2 \cdot rb1 \in dom(attached) \wedge rb2 \in dom(attached) \wedge asgn\_z(rb1) = asgn\_z(rb2) \wedge$$
$$asgn\_s(rb1) \neq 0 \wedge asgn\_s(rb2) \neq 0 \wedge asgn\_s(rb1) = asgn\_s(rb2) \Rightarrow rb1 = rb2.$$

To coordinate the cleaning process, a base station stores the information about its own cleaned sectors and periodically updates information about the status of the other cleaned sectors. Therefore, we assume that each base station has a "map" – a knowledge about all sectors of the whole territory. To model this, we introduce a new variable, *local_map*:

$$local\_map \in BS \rightarrow (1 \mathinner{..} n \nrightarrow (1 \mathinner{..} k \rightarrow STATE)).$$

The "maps" are defined only for the base stations that have any zone cleaning to coordinate, i.e., $bs \in ran(responsible)$:

$$\forall bs \cdot bs \in ran(responsible) \Rightarrow local\_map(bs) \in 1 \mathinner{..} n \rightarrow (1 \mathinner{..} k \rightarrow STATE),$$
$$\forall bs \cdot bs \in BS \wedge bs \notin ran(responsible) \Rightarrow local\_map(bs) = \varnothing.$$

The abstract variable *territory* represents the global knowledge on the whole territory. For any sector and zone, this global knowledge has to be consistent with the information stored by the base stations. Namely, if in the local knowledge of any base station $bs$ a sector $s$ is marked as cleaned, i.e., $local\_map(bs)(z)(s) = compl$, then it should be cleaned according to the global knowledge as well, i.e., $territory(z)(s) = compl$; and vice versa: if a sector $s$ is marked as non-cleaned in the global knowledge, i.e., $territory(z)(s) = incompl$, then it remains non-cleaned according the local knowledge of any base station $bs$, i.e., $local\_map(bs)(z)(s) = incompl$. To establish those relationships, we formulate and prove the following invariants:

$$\forall bs, z, s \cdot bs \in ran(responsible) \land z \in 1 \mathinner{..} n \land s \in 1 \mathinner{..} k \Rightarrow (local\_map(bs)(z)(s) = compl \Rightarrow territory(z)(s) = compl),$$

$$\forall bs, z, s \cdot bs \in ran(responsible) \land z \in 1 \mathinner{..} n \land s \in 1 \mathinner{..} k \Rightarrow (territory(z)(s) = incompl \Rightarrow local\_map(bs)(z)(s) = incompl).$$

For each base station, the local information about its zones and sectors always coincides with the global knowledge about these zones and sectors:

$$\forall bs, z, s \cdot bs \in ran(responsible) \land z \in 1 \mathinner{..} n \land responsible(z) = bs \land s \in 1 \mathinner{..} k \Rightarrow (territory(z)(s) = incompl \Leftrightarrow local\_map(bs)(z)(s) = incompl).$$

All together, these three invariants formalise the property (PR6).

A base station assigns a cleaning task to its attached robots. This behaviour is modelled by a new event NewTask. In the event guard, we check that the assigned sector $s$ is not cleaned yet, i.e., $local\_map(bs)(z)(s) = incompl$, and no other robot is currently cleaning it. The last condition can be formally expressed as $s \notin ran((dom(asgn\_z \rhd \{z\})) \lhd asgn\_s)$, i.e., the sector $s$ is not assigned to any robot that performs cleaning in the zone $z$:

NewTask $\widehat{=}$
**any** $bs, rb, z, s$
**when**
$bs \in BS \land rb \in dom(attached) \land attached(rb) = bs \land z \in 1 \mathinner{..} n \land$
$responsible(z) = bs \land asgn\_z(rb) = 0 \land s \in 1 \mathinner{..} k \land asgn\_s(rb) = 0 \land$
$local\_map(bs)(z)(s) = incompl \land s \notin ran((dom(asgn\_z \rhd \{z\})) \lhd asgn\_s)$
**then**
$asgn\_s(rb) := s$
$asgn\_z(rb) := z$
**end**

The robot failures have impact on execution of the cleaning process. The cleaning task cannot be performed if a robot assigned for this task has failed. To reflect this behaviour in our model, we refine the abstract event Body by two events TaskSuccess and TaskFailure, which respectively model successful and unsuccessful execution of the task. If the task has been successfully performed by the assigned robot $rb$, its supervising base station $bs$ changes the status of the sector $s$ to cleaned, i.e., we override the previous value of $local\_map(bs)(z)(s)$ by the value $compl$.

TaskSuccess $\widehat{=}$ refines Body
**status** *convergent*
**any** $bs, rb, z, s$
**when**
$bs \in BS \wedge rb \in dom(attached) \wedge attached(rb) = bs \wedge$
$z \in 1 \mathinner{..} n \ \wedge responsible(z) = bs \ \wedge asgn\_z(rb) = z \wedge$
$s \in 1 \mathinner{..} k \ \wedge asgn\_s(rb) = s \wedge local\_map(bs)(z)(s) = incompl$
**then**
$territory(z) := territory(z) \Leftarrow \{s \mapsto compl\}$
$local\_map(bs) := local\_map(bs) \Leftarrow \{z \mapsto local\_map(bs)(z) \Leftarrow \{s \mapsto compl\}\}$
$asgn\_s(rb) := 0$
$asgn\_z(rb) := 0$
$counter := counter - 1$
end

The dual event TaskFailure abstractly models the opposite situation caused by a robot failure. As a result, all the relationships concerning the failed robot $rb$ are removed:

TaskFailure $\widehat{=}$ refines Body
**status** *convergent*
**any** $bs, rb, z, s$
**when**
$bs \in BS \wedge rb \in dom(attached) \wedge attached(rb) = bs \wedge$
$z \in 1 \mathinner{..} n \ \wedge responsible(z) = bs \ \wedge asgn\_z(rb) = z \wedge$
$s \in 1 \mathinner{..} k \ \wedge asgn\_s(rb) = s \wedge local\_map(bs)(z)(s) = incompl$
**then**
$territory(z) := territory(z) \Leftarrow \{s \mapsto incompl\}$
$asgn\_s := \{rb\} \lhd asgn\_s$
$asgn\_z := \{rb\} \lhd asgn\_z$
$attached := \{rb\} \lhd attached$
end

At this refinement step, we are ready to demonstrate that the events TaskSuccess and TaskFailure converge. To prove it, we define the following variant expression over system variables:

$$counter \ + \ card(dom(attached)),$$

where *counter* is an auxiliary variable that stores the number of all non-cleaned sectors of the whole territory. The initial value of *counter* is equal to $n * k$. When a robot fails to perform a task, it is removed from the corresponding set of the attached robots $dom(attached)$. This in turn decreases the value of $card(dom(attached))$ and consequently the whole variant expression. On the other hand, when a robot succeeds in cleaning a sector, the variable *counter* decreases and consequently the whole variant expression decreases as well. If there are no sectors to clean, the events become disabled and the system terminates.

A base station keeps track of the cleaned and non-cleaned sectors and repeatedly receives the information from the other base stations about their cleaned sectors. This knowledge is inaccurate for the period when the information is sent but not yet received. In this refinement step we abstractly model receiving the information by a base station. In the next refinement step, we are going to define this process of information broadcasting more precisely.

The new event UpdateMap models updating the local map of a base station $bs$. Here we have to ensure that the obtained information is always consistent with the global one. Specifically, the base station updates a sector $s$ as cleaned only if it has this status according to the global knowledge, i.e., $territory(z)(s) = compl$.

UpdateMap $\widehat{=}$
**any** $bs, z, s$
**when**
$bs \in BS \wedge z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \ \wedge responsible(z) \neq bs \wedge$
$bs \in ran(responsible) \wedge territory(z)(s) = compl$
**then**
$local\_map(bs) := local\_map(bs) \lhd\!\!\!+ \{z \mapsto local\_map(bs)(z) \lhd\!\!\!+ \{s \mapsto compl\}\}$
**end**

In this refinement step we also introduce an abstract representation of the base station co-operation defined by the property (PR7). Namely, we allow to reassign a group of robots from one base station to another. This behaviour is defined by the event ReassignRB. In the next refinement steps we will elaborate on this event and define the conditions under which this behaviour takes place.

Additionally, we model a possible redistribution between the base stations their pre-assigned responsibility for zones and robots. This behaviour is defined in the new event GetAdditionalResponsibility presented below. The event guard defines the conditions when such a change is allowed. A base station $bs\_j$ can take the responsibility for a set of new zones $zss$ if it has the accurate knowledge about these zones, i.e., the information about their cleaned and non-cleaned sectors. Specifically, in the guard we check that the global status of each sector $s$ from the zone $z$, i.e., $territory(z)(s)$, coincides with the local information that the base station $bs\_j$ has about this sector. In that case, we reassign responsibility for the zone(s) $zss$ and the robots $rbs$ to the base station $bs\_j$:

GetAdditionalResponsibility $\widehat{=}$
**any** $bs\_i, bs\_j, rbs, zs$
**when**
$bs\_i \in BS \wedge bs\_j \in BS \wedge$
$zs \subset 1 \mathinner{..} n \ \wedge zs = dom(responsible \rhd \{bs\_i\}) \wedge$
$rbs \subset dom(attached) \ \wedge rbs = dom(attached \rhd \{bs\_i\}) \wedge$
$bs\_i \neq bs\_j \ \wedge bs\_j \in ran(responsible) \ \wedge bs\_j \in ran(responsible) \wedge$
$(\forall z, s \cdot z \in zs \wedge s \in 1 \mathinner{..} k \Rightarrow territory(z)(s) = local\_map(bs\_j)(z)(s))$
**then**
$responsible := responsible \lhd\!\!\!+ (zs \times \{bs\_j\})$
$attached := attached \lhd\!\!\!+ (rbs \times \{bs\_j\})$
$asgn\_s := asgn\_s \lhd\!\!\!+ (rbs \times \{0\})$
$asgn\_z := asgn\_z \lhd\!\!\!+ (rbs \times \{0\})$
$local\_map(bs\_i) := \varnothing$
**end**

Modelling this behaviour allows us to formalise the property (PR9). Our next refinement step will elaborate on our chosen communication model that is needed to achieve such co-operative recovery.

## 4.5 Fourth Refinement: a Model of Broadcasting

In the fourth refinement step we aim at defining an abstract model of broadcasting. After receiving a notification from a robot about successful cleaning the assigned sector, a base station updates its local map and broadcasts the message about the cleaned sector to the other base stations. In its turn, upon receiving the message, each base station correspondingly updates its own local map. We assume that the communication between base stations is reliable: no message is lost and eventually every base station receives it. In further refinement steps, this model of the broadcasting can be further refined by a more concrete mechanism.

To model the described behaviour, we introduce a new relational variable, $msg$, that models the message broadcasting buffer:

$$msg \in BS \leftrightarrow (1\,..\,n \times 1\,..\,k).$$

If a message $(bs \mapsto (z \mapsto s))$ belongs to this buffer, this means that the sector $s$ from the zone $z$ has been cleaned, i.e., $territory(z)(s) = compl$. The first element of the message, $bs$, determines the base station the message is sent to. We formulate this property by the following system invariant:

$$\forall z, s{\cdot}z \in 1\,..\,n \wedge s \in 1\,..\,k \wedge (z \mapsto s) \in ran(msg) \Rightarrow territory(z)(s) = compl.$$

If there are no messages in the $msg$ buffer for any particular base station then the local map of this base station is accurate, i.e., it coincides with the global knowledge about the teritory:

$$\forall bs, z, s{\cdot}z \in 1\,..\,n \wedge s \in 1\,..\,k \wedge bs \in ran(responsible) \wedge (bs \mapsto (z \mapsto s)) \notin msg \Rightarrow territory(z)(s) = local\_map(bs)(z)(s),$$

$$\forall bs{\cdot}bs \in ran(responsible) \wedge bs \notin dom(msg) \Rightarrow (\forall z, s{\cdot}z \in 1\,..\,n \wedge s \in 1\,..\,k \Rightarrow territory(z)(s) = local\_map(bs)(z)(s)).$$

After receiving a notification about successful cleaning of a sector, a base station marks this sector as cleaned in its local map and then broadcasts the message about it to other base stations. To model this, we refine the abstract event TaskSuccess. Specifically, in the event body we add a new assignment $msg := msg \cup (bss \times \{z \mapsto s\})$ to add a new message to the broadcasting buffer.

We also refine the abstract event UpdateMap. In particular, we replace the guard $territory(z)(s) = compl$ by the guard $(bs \mapsto (z \mapsto s)) \in msg$. This guard checks that there is a message for the base station $bs$ about the cleaned sector $s$ from the zone $z$. As a result of the event, the base station $bs$ reads the message and marks the sector $s$ in the zone $z$ as cleaned in its local map.

UpdateMap $\widehat{=}$ refines UpdateMap
**any** $bs, z, s$
**when**
$bs \in BS \wedge z \in 1\,..\,n \wedge s \in 1\,..\,k \;\wedge responsible(z) \neq bs \wedge$
$bs \in ran(responsible) \;\wedge (bs \mapsto (z \mapsto s)) \in msg$
**then**
$local\_map(bs) := local\_map(bs) \Leftarrow \{z \mapsto local\_map(bs)(z) \Leftarrow \{s \mapsto compl\}\}$
$msg := msg \setminus \{bs \mapsto (z \mapsto s)\}$

## 4.6 Fifth Refinement: Introducing Robot Failures

Now we aim at modelling possible robot failures and elaborate on the abstract events concerning robot and zone reassigning. We start by partitioning the robots into active and failed ones. The current set of all active robots is defined by a new variable *active* with the following invariant properties:

$$active \subseteq dom(attached),\ active \subseteq dom(asgn\_s),\ active \subseteq dom(asgn\_z).$$

Initially all robots are active, i.e., $active = RB$. A new event RobotFailure models possible robot failures that can happen at any time during system execution:

RobotFailure $\widehat{=}$
**any** $rb$
**when**
$rb \in active \wedge card(active) > 1$
**then**
$active := active \setminus \{rb\}$
**end**

We make an assumption that the last active robot can not fail and add the corresponding guard $card(active) > 1$ to the event RobotFailure to restrict possible robot failures. Let us note that for multi-robotic systems with many homogeneous robots this constraint is not unreasonable.

A base station monitors all its robots and detects the failed ones. The abstract event TaskFailure abstractly models such robot detection.

To formalise the property (PR8), we should model a situation when some base station $bs\_j$ does not have active robots anymore, i.e., $dom(attached \rhd \{bs\_j\}) \nsubseteq active$. In that case, some group of active robots $rbs$ has to be sent to this base station $bs\_j$ from another base station $bs\_i$. This behaviour is modelled by the event ReassignNewBStoRBs that refines the abstract event ReassignRB. As a result, all the robots from $rbs$ become attached to the base station $bs\_j$:

ReassignNewBStoRBs $\widehat{=}$ refines ReassignRB
**any** $bs\_i, bs\_j, rbs$
**when**
$bs\_i \in BS \wedge bs\_j \in BS \wedge rbs \subset active \wedge$
$ran(rbs \lhd attached) = \{bs\} \;\wedge bs\_i \in ran(responsible) \wedge$
$ran(rbs \lhd asgn\_s) = \{0\} \;\wedge rbs \neq \varnothing \;\wedge bs\_j \in ran(responsible) \wedge$
$bs\_i \neq bs\_j \;\wedge bs\_i \in ran(rbs \lhd attached) \;\wedge dom(attached \rhd \{bs\_j\}) \nsubseteq active$
**then**
$attached := attached \Leftarrow (rbs \times \{bs\_j\})$
**end**

This event can be further refined by a concrete procedure to choose a particular base station that will share its robots (e.g., based on load balancing).

Finally, to ensure the property (PR9) , let us consider the situation when all the sectors for which a base station is responsible are cleaned. In that case, all the active robots of the base station may be sent to some other base station that still has some unfinished cleaning to co-ordinate. This functionality is specified by the event SendRobotsToBS (a refinement of the event ReassignRB):

SendRobotsToBS $\widehat{=}$ refines ReassignRB
**any** $bs\_i, bs\_j, rbs$
**when**
$bs\_i \in operating \wedge bs\_j \in operating \wedge rbs \subset active \wedge$
$ran(rbs \lhd attached) = \{bs\_i\} \wedge bs\_i \in ran(responsible) \wedge$
$ran(rbs \lhd asgn\_s) = \{0\} \wedge rbs \neq \varnothing \wedge bs\_j \in ran(responsible) \wedge$
$bs\_i \neq bs\_j \wedge bs\_i \in ran(rbs \lhd\!\!\!- attached) \wedge rbs = dom(attached \rhd \{bs\_i\}) \wedge$
$(\forall z \cdot z \in 1 \mathinner{..} n \wedge responsible(z) = bs\_i \Rightarrow local\_map(bs\_i)(z)[1 \mathinner{..} k] = \{compl\})$
**then**
$attached := attached \lhd\!\!\!+ (rbs \times \{bs\_j\})$
**end**

## 4.7 Sixth Refinement: Introducing Base Station Failures

In the final refinement step presented in the paper, we aim at specifying the base station failures. Each base station might be either operating or failed. We introduce a new variable *operating* to define the set of all operating base stations. The corresponding invariant properties are as follows.

$$operating \subseteq BS,$$
$$\forall bs \cdot bs \in BS \wedge local\_map(bs) = \varnothing \;\Rightarrow\; bs \notin operating.$$

Also, similarly to the event RobotFailure, we introduce a new event BaseStationFailure to model a possible base station failure.

In the fourth refinement step we assumed that a base station can take over the responsibility for the robots and zones of another base station. This behaviour was modelled by the event GetAdditionalResponsibility. Now we can refine this event by introducing an additional condition – only if a base station is detected as failed, another base station can take over its responsibility for the respective zones and robots:

GetAdditionalResponsibility $\widehat{=}$ refines GetAdditionalResponsibility
**any** $bs\_i, bs\_j, za, rbs$
**when**
$bs\_i \in BS \wedge bs\_j \in operating \wedge zs \subset 1 \mathinner{..} n \wedge$
$zs = dom(responsible \rhd \{bs\_i\}) \wedge rbs \subset active \wedge$
$rbs = dom(attached \rhd \{bs\_i\}) \wedge bs\_i \neq bs\_j \wedge bs\_j \notin dom(msg) \wedge bs\_i \notin operating$
**then**
$responsible := responsible \lhd\!\!\!+ (zs \times \{bs\_j\})$
$attached := attached \lhd\!\!\!+ (rbs \times \{bs\_j\})$
$asgn\_s := asgn\_s \lhd\!\!\!+ (rbs \times \{0\})$
$asgn\_z := asgn\_z \lhd\!\!\!+ (rbs \times \{0\})$
$local\_map(bs\_i) := \varnothing$
**end**

As a result of the presented refinement chain, we arrived at a centralised model of the multi-robotic system. We can further refine the system to derive its distributed implementation, relying on the modularisation extension of Event-B to achieve this.

# 5 Discussion

**Assessment of the development.** The development of the presented multi-robotic system has been carried out with the support of the Rodin platform [7]. We have derived a complex system specification in six refinement steps. In general, the refinement approach has demonstrated a good scalability and allowed us to model intricate dependencies between the system components. We have been able to express and verify all the desired properties defined for our system. Therefore, we can make a general conclusion about suitability of the refinement technique for formal development and verification of the multi-robotic systems.

However, we have also identified a number of problems. Firstly, in spite of seeming simplicity, the relationships between the base stations, zones and sectors have been modelled using quite complex nested data structures (functions). The Rodin platform could not comfortably handle the proofs involving manipulations with the nested functions and required rather time-consuming interactive proving efforts. Secondly, the Rodin platform does not support the direct assignment to a function with nested arguments. For instance, instead of simply specifying $local\_map(bs)(z)(s) := compl$, we have to express it as the following intricate statement $local\_map(bs) \Leftarrow \{z \mapsto local\_map(bs)(z) \Leftarrow \{s \mapsto compl\}\}$, i.e., use the overriding relation twice. These two problems can be alleviated with a mathematical extension of the Rodin platform that is currently under development.

Despite certain technical difficulties, we have found the refinement approach as such to be beneficial for deriving precise requirements and the corresponding model of a multi-robotic system. In the refinement process, we have discovered a number of subtleties in the system requirements. The proving effort has helped us to localise the present problems and ambiguities and find the appropriate solutions. For instance, we had to impose extra restrictions on the situations when a base station takes a new responsibility for other zones and robots. Moreover, we had to make our assumptions about robot failures more precise.

**Related work.** Formal modelling of multi-agent systems has been undertaken in [9, 8, 10]. The authors have proposed an extension of the Unity framework to explicitly define such concepts as mobility and context-awareness. Our modelling have pursued a different goal – we have aimed at formally guaranteeing that the specified agent behaviour achieves the pre-

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 incorporate well-known (static) fault tolerance mechanisms into formal models. In our approach, we have implemented a more advanced fault tolerance scheme that relies on goal reallocation and dynamic reconfiguration to guarantee goal reachability.

The foundational work on goal-oriented development has been done by van Lamsweerde [11]. 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 it with formal modelling [4, 5, 6]. These works aimed at expressing temporal logic properties in Event-B. In our work, we have relied on goals to facilitate structuring of the system behaviour and derived a detailed system model that satisfies the desired properties by refinement.

**Conclusions.** In this paper we have presented a formal development of a fault tolerant multi-robotic system. The development has been carried out by refinement in Event-B. As a result of the formal development process, we have achieved the desired goal – formally specified the complex system behaviour and proved the desired properties. The formal development has allowed us to uncover missing requirements and rigorously define the relationships between agents. The refinement approach has also allowed us to derive a complex mechanism for cooperative error recovery in a systematic manner.

Our approach has demonstrated a number of advantages comparing to various process-algebraic approaches used for modelling multi-agent systems. The reliance on a proof-based verification has allowed us to derive a quite complex model of the behaviour of a multi-agent robotic system. We have not needed to avoid complex data types and could comfortably express intricate relationships between the system goals and the employed agents. As a result, our approach scales well with respect to the number of system states, agents, and their complex interactions. We believe that, once the mentioned technical difficulties of handling complex nested functions are resolved in the Rodin platform, Event-B and the associated tool set will provide a suitable framework for formal modelling of complex multi-robotic systems.

# 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. Springer, 2009.

[4] 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.

[5] 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.

[6] 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.

[7] Rodin. Event-B Platform, online at http://www.event-b.org/.

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

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

[10] 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.

[11] Axel van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE'01*, pages 249–263. IEEE Computer Society, 2001.

# Appendix: Event-B Development

---

**MACHINE** Abs
**SEES** cntx
**VARIABLES**
    goal
**INVARIANTS**
    inv1 : $goal \in STATE$
**EVENTS**
**Initialisation**
    **begin**
        act1 : $goal := incompl$
    **end**
**Event** $Body \mathrel{\widehat{=}}$
**Status** anticipated
    **when**
        grd1 : $goal \neq compl$
    **then**
        act1 : $goal :\in STATE$
    **end**
**END**

**CONTEXT** cntx
**SETS**
    STATE
**CONSTANTS**
    incompl
    compl
**AXIOMS**
    axm1 : $partition(STATE, \{incompl\}, \{compl\})$
**END**

**MACHINE** RS_ref1
**REFINES** Abs
**SEES** cntx1
**VARIABLES**
zones
**INVARIANTS**
inv1 : $zones \in 1 \mathinner{..} n \rightarrow STATE$
inv2 : $zones[1 \mathinner{..} n] = \{compl\} \Leftrightarrow goal = compl$
**EVENTS**
**Initialisation**
**begin**
act1 : $zones := 1 \mathinner{..} n \times \{incompl\}$
**end**
**Event** *Body* $\widehat{=}$
**Status** anticipated
**refines** *Body*
**any**
$z$
$res$
**where**
grd1 : $z \in 1 \mathinner{..} n$
grd2 : $zones(z) \neq compl$
grd3 : $res \in STATE$
**with**
goal′ : $\mathtt{goal}' = \mathtt{fnc}(\mathtt{bool}(\mathtt{zones}'[\mathtt{1} \mathinner{..} \mathtt{n}] = \{\mathtt{compl}\}))$
**then**
act1 : $zones(z) := res$
**end**
**END**

**CONTEXT** cntx1
**EXTENDS** cntx
**CONSTANTS**
n, fnc
**AXIOMS**
axm1 : $n \in \mathbb{N}_1$
axm2 : $fnc \in BOOL \rightarrowtail STATE$
axm3 : $fnc(TRUE) = compl$
axm4 : $fnc(FALSE) = incompl$
**END**

**MACHINE** RS_ref2
**REFINES** RS_ref1
**SEES** cntx2
**VARIABLES**
territory
**INVARIANTS**
inv1 : $territory \in 1 \mathinner{..} n \rightarrow (1 \mathinner{..} k \rightarrow STATE)$
inv2 : $\forall j \cdot j \in 1 \mathinner{..} n \Rightarrow (zones(j) = compl \Leftrightarrow territory(j)[1 \mathinner{..} k] = \{compl\})$
**EVENTS**
**Initialisation**
**begin**
act1 : $territory := 1 \mathinner{..} n \times \{1 \mathinner{..} k \times \{incompl\}\}$
**end**
**Event** *Body* $\widehat{=}$
**Status** anticipated
**refines** *Body*
**any**
$z$
$s$
*result*
**where**
grd1 : $z \in 1 \mathinner{..} n$
grd2 : $s \in 1 \mathinner{..} k$
grd3 : $territory(z)(s) \neq compl$
grd4 : $result \in STATE$
**with**
res : $\mathtt{res} = \mathtt{fnc}(\mathtt{bool}(\mathtt{territory}'(\mathtt{z})[1 \mathinner{..} \mathtt{k}] = \{\mathtt{compl}\}))$
**then**
act1 : $territory(z) := territory(z) \lhd\!\!\!- \{s \mapsto result\}$
**end**
**END**

**CONTEXT** cntx2
**EXTENDS** cntx1
**CONSTANTS**
k
**AXIOMS**
axm1 : $k \in \mathbb{N}_1$
**END**

**MACHINE** RS_ref3

**REFINES** RS_ref2

**SEES** cntx3

**VARIABLES**

`territory`

`responsible`

`attached`

`local_map`

`asgn_z`

`asgn_s`

`counter`

**INVARIANTS**

`inv1` : $responsible \in 1 \mathinner{..} n \rightarrow BS$

`inv2` : $attached \in RB \nrightarrow BS$

`inv4` : $asgn\_s \in RB \nrightarrow 0 \mathinner{..} k$

`inv5` : $asgn\_z \in RB \nrightarrow 0 \mathinner{..} n$

`inv14` : $dom(asgn\_z) = dom(asgn\_s)$

`inv15` : $dom(attached) = dom(asgn\_z)$

`inv6` : $local\_map \in BS \rightarrow (1 \mathinner{..} n \nrightarrow (1 \mathinner{..} k \rightarrow STATE))$

`inv7` : $\forall bs \cdot bs \in ran(responsible) \Rightarrow local\_map(bs) \in 1 \mathinner{..} n \rightarrow (1 \mathinner{..} k \rightarrow STATE)$

`inv8` : $\forall bs \cdot bs \notin ran(responsible) \wedge bs \in BS \Rightarrow local\_map(bs) = \varnothing$

`inv9` : $\forall bs, z, s \cdot bs \in ran(responsible) \wedge z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \Rightarrow (territory(z)(s) = incompl \Rightarrow local\_map(bs)(z)(s) = incompl)$

`inv10` : $\forall bs, z, s \cdot bs \in ran(responsible) \wedge z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \Rightarrow (local\_map(bs)(z)(s) = compl \Rightarrow territory(z)(s) = compl)$

`inv11` : $\forall bs, z, s \cdot bs \in ran(responsible) \wedge z \in 1 \mathinner{..} n \wedge responsible(z) = bs \wedge s \in 1 \mathinner{..} k \Rightarrow (territory(z)(s) = incompl \Leftrightarrow local\_map(bs)(z)(s) = incompl)$

`inv12` : $\forall rb \cdot rb \in dom(asgn\_z) \Rightarrow (asgn\_z(rb) = 0 \Leftrightarrow asgn\_s(rb) = 0)$

`inv13` : $\forall rb1, rb2 \cdot rb1 \in dom(asgn\_z) \wedge rb2 \in dom(asgn\_z) \wedge asgn\_z(rb1) = asgn\_z(rb2) \wedge asgn\_s(rb1) \neq 0 \wedge asgn\_s(rb2) \neq 0 \wedge asgn\_s(rb1) = asgn\_s(rb2) \Rightarrow rb1 = rb2$

`inv16` : $counter \in 0 \mathinner{..} n * k$

**EVENTS**

**Initialisation**

*extended*

**begin**

act1 : $\texttt{territory} := \texttt{1} \mathinner{..} \texttt{n} \times \{\texttt{1} \mathinner{..} \texttt{k} \times \{\texttt{incompl}\}\}$
act2 : $responsible :\in 1 \mathinner{..} n \twoheadrightarrow BS$
act3 : $attached :\in RB \twoheadrightarrow BS$
act4 : $local\_map := BS \times \{1 \mathinner{..} n \times \{1 \mathinner{..} k \times \{incompl\}\}\}$
act5 : $asgn\_s := RB \times \{0\}$
act6 : $asgn\_z := RB \times \{0\}$
act7 : $counter := n * k$

**end**

**Event** *NewTask* $\widehat{=}$

**any**

$bs$
$rb$
$z$
$s$

**where**

grd1 : $bs \in BS$
grd2 : $rb \in dom(attached)$
grd3 : $attached(rb) = bs$
grd4 : $asgn\_s(rb) = 0$
grd5 : $z \in 1 \mathinner{..} n$
grd6 : $responsible(z) = bs$
grd7 : $asgn\_z(rb) = 0$
grd8 : $s \in 1 \mathinner{..} k$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $s \notin ran((dom(asgn\_z \rhd \{z\})) \lhd asgn\_s)$

**then**

act1 : $asgn\_s(rb) := s$
act2 : $asgn\_z(rb) := z$

**end**

**Event** *TaskSuccess* $\widehat{=}$

**Status** convergent

**refines** *Body*

**any**

$z$
$s$
$bs$
$rb$

**where**

grd1 : $bs \in BS$
grd2 : $rb \in dom(attached)$
grd3 : $attached(rb) = bs$
grd4 : $z \in 1 \mathinner{..} n$

grd5 : $responsible(z) = bs$
grd6 : $asgn\_z(rb) = z$
grd7 : $s \in 1 \mathinner{..} k$
grd8 : $asgn\_s(rb) = s$
grd9 : $local\_map(bs)(z)(s) = incompl$

**with**

result : result = compl

**then**

act1 : $territory(z) := territory(z) \lhd \{s \mapsto compl\}$
act2 : $local\_map(bs) := local\_map(bs) \lhd \{z \mapsto local\_map(bs)(z) \lhd \{s \mapsto compl\}\}$
act3 : $asgn\_s(rb) := 0$
act4 : $asgn\_z(rb) := 0$
act5 : $counter := counter - 1$

**end**

**Event** *TaskFailure* $\widehat{=}$

**Status** convergent

**refines** *Body*

**any**

$bs$
$rb$
$z$
$s$

**where**

grd1 : $bs \in BS$
grd2 : $rb \in dom(attached)$
grd3 : $attached(rb) = bs$
grd4 : $z \in 1 \mathinner{..} n$
grd5 : $responsible(z) = bs$
grd6 : $asgn\_z(rb) = z$
grd7 : $s \in 1 \mathinner{..} k$
grd8 : $asgn\_s(rb) = s$
grd9 : $local\_map(bs)(z)(s) = incompl$

**with**

result : result = incompl

**then**

act1 : $territory(z) := territory(z) \lhd \{s \mapsto incompl\}$
act3 : $asgn\_s := \{rb\} \lhd asgn\_s$
act4 : $asgn\_z := \{rb\} \lhd asgn\_z$
act5 : $attached := \{rb\} \lhd attached$

**end**

**Event** *ReassignRB* $\widehat{=}$

**any**

$bs\_i$

$bs\_j$

$rbs$

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $rbs \subset dom(attached)$

grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$

grd5 : $bs\_i \in ran(responsible)$

grd6 : $ran(rbs \lhd asgn\_s) = \{0\}$

grd7 : $rbs \neq \varnothing$

grd8 : $bs\_j \in ran(responsible)$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_i \in ran(rbs \lhd attached)$

**then**

act1 : $attached := attached \lhd\!\!+ (rbs \times \{bs\_j\})$

**end**

**Event** *GetAdditionalResponsibility* $\widehat{=}$

**any**

$bs\_i$

$bs\_j$

$zs$

$rbs$

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $zs \subset 1 \mathinner{..} n$

grd4 : $zs = dom(responsible \rhd \{bs\_i\})$

grd5 : $rbs \subset dom(attached)$

grd6 : $rbs = dom(attached \rhd \{bs\_i\})$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_j \in ran(responsible)$

grd11 : $\forall z, s \cdot z \in zs \land s \in 1 \mathinner{..} k \Rightarrow territory(z)(s) = local\_map(bs\_j)(z)(s)$

**then**

act1 : $responsible := responsible \lhd\!\!+ (zs \times \{bs\_j\})$

act3 : $attached := attached \lhd\!\!+ (rbs \times \{bs\_j\})$

act4 : $asgn\_s := asgn\_s \lhd\!\!+ (rbs \times \{0\})$

act5 : $asgn\_z := asgn\_z \lhd\!\!+ (rbs \times \{0\})$

act6 : $local\_map(bs\_i) := \varnothing$

**end**

**Event** *UpdateMap* $\widehat{=}$

**any**

$bs$

$z$

$s$

**where**

grd1 : $bs \in BS$

grd2 : $z \in 1 \mathinner{..} n$

grd3 : $s \in 1 \mathinner{..} k$

grd4 : $responsible(z) \neq bs$

grd5 : $bs \in ran(responsible)$

grd6 : $territory(z)(s) = compl$

**then**

act1 : $local\_map(bs) := local\_map(bs) \lhd\!\!+ \{z \mapsto local\_map(bs)(z) \lhd\!\!+ \{s \mapsto compl\}\}$

**end**

**VARIANT**

$\texttt{counter} + \texttt{card}(\texttt{dom}(\texttt{attached}))$

**END**

**CONTEXT** cntx3

**EXTENDS** cntx2

**SETS**

AGENTS

TSTATE

**CONSTANTS**

RB

BS

**AXIOMS**

axm1 : $partition(AGENTS, RB, BS)$

axm2 : $finite(AGENTS)$

axm3 : $RB \neq \varnothing$

axm4 : $BS \neq \varnothing$

**END**

**MACHINE** RS_ref4

**REFINES** RS_ref3

**SEES** cntx3

**VARIABLES**

territory

responsible

attached

local_map

asgn_z

asgn_s

msg

counter

**INVARIANTS**

inv1 : $msg \in BS \leftrightarrow (1 \mathinner{..} n \times 1 \mathinner{..} k)$

inv2 : $\forall z, s \cdot z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \wedge (z \mapsto s) \in ran(msg) \Rightarrow territory(z)(s) = compl$

inv3 : $\forall bs \cdot bs \in ran(responsible) \wedge bs \notin dom(msg) \Rightarrow (\forall z, s \cdot z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \Rightarrow territory(z)(s) = local\_map(bs)(z)(s))$

inv4 : $\forall bs, z, s \cdot z \in 1 \mathinner{..} n \wedge s \in 1 \mathinner{..} k \wedge bs \in ran(responsible) \wedge (bs \mapsto (z \mapsto s)) \notin msg \Rightarrow local\_map(bs)(z)(s) = territory(z)(s)$

**EVENTS**

**Initialisation**

*extended*

**begin**

act1 : territory $:= 1 \mathinner{..} n \times \{1 \mathinner{..} k \times \{incompl\}\}$

act2 : responsible $:\in 1 \mathinner{..} n \rightarrow\!\!\!\!\!\!\rightarrow BS$

act3 : attached $:\in RB \rightarrow\!\!\!\!\!\!\rightarrow BS$

act4 : local_map $:= BS \times \{1 \mathinner{..} n \times \{1 \mathinner{..} k \times \{incompl\}\}\}$

act5 : asgn_s $:= RB \times \{0\}$

act6 : asgn_z $:= RB \times \{0\}$

act7 : counter $:= n * k$

act8 : $msg := BS \times \varnothing$

**end**

**Event** *NewTask* $\widehat{=}$

**extends** *NewTask*

**any**

bs

rb

z

s

**where**

grd1 : $\mathtt{bs} \in \mathtt{BS}$

grd2 : $\mathtt{rb} \in \mathtt{dom}(\mathtt{attached})$

grd3 : $\mathtt{attached}(\mathtt{rb}) = \mathtt{bs}$

grd4 : $\mathtt{asgn\_s}(\mathtt{rb}) = 0$

grd5 : $\mathtt{z} \in 1 \mathinner{..} \mathtt{n}$

grd6 : $\mathtt{responsible}(\mathtt{z}) = \mathtt{bs}$

grd7 : $\mathtt{asgn\_z}(\mathtt{rb}) = 0$

grd8 : $\mathtt{s} \in 1 \mathinner{..} \mathtt{k}$

grd9 : $\mathtt{local\_map}(\mathtt{bs})(\mathtt{z})(\mathtt{s}) = \mathtt{incompl}$

grd10 : $\mathtt{s} \notin \mathtt{ran}((\mathtt{dom}(\mathtt{asgn\_z} \rhd \{\mathtt{z}\})) \lhd \mathtt{asgn\_s})$

**then**

act1 : $\mathtt{asgn\_s}(\mathtt{rb}) := \mathtt{s}$

act2 : $\mathtt{asgn\_z}(\mathtt{rb}) := \mathtt{z}$

**end**

**Event** *TaskSuccess* $\widehat{=}$

**extends** *TaskSuccess*

**any**

z

s

bs

rb

*bss*

**where**

grd1 : $\mathtt{bs} \in \mathtt{BS}$

grd2 : $\mathtt{rb} \in \mathtt{dom}(\mathtt{attached})$

grd3 : $\mathtt{attached}(\mathtt{rb}) = \mathtt{bs}$

grd4 : $\mathtt{z} \in 1 \mathinner{..} \mathtt{n}$

grd5 : $\mathtt{responsible}(\mathtt{z}) = \mathtt{bs}$

grd6 : $\mathtt{asgn\_z}(\mathtt{rb}) = \mathtt{z}$

grd7 : $\mathtt{s} \in 1 \mathinner{..} \mathtt{k}$

grd8 : $\mathtt{asgn\_s}(\mathtt{rb}) = \mathtt{s}$

grd9 : $\mathtt{local\_map}(\mathtt{bs})(\mathtt{z})(\mathtt{s}) = \mathtt{incompl}$

grd10 : $bss = BS \setminus \{bs\}$

**then**

act1 : $\mathtt{territory}(\mathtt{z}) := \mathtt{territory}(\mathtt{z}) \lhd\!\!+ \{\mathtt{s} \mapsto \mathtt{compl}\}$

act2 : $\mathtt{local\_map}(\mathtt{bs}) := \mathtt{local\_map}(\mathtt{bs}) \lhd\!\!+ \{\mathtt{z} \mapsto \mathtt{local\_map}(\mathtt{bs})(\mathtt{z}) \lhd\!\!+ \{\mathtt{s} \mapsto \mathtt{compl}\}\}$

act3 : $\mathtt{asgn\_s}(\mathtt{rb}) := 0$

act4 : $\mathtt{asgn\_z}(\mathtt{rb}) := 0$

act5 : $\mathtt{counter} := \mathtt{counter} - 1$

act6 : $msg := msg \cup (bss \times \{z \mapsto s\})$

**end**

**Event** *TaskFailure* $\widehat{=}$

**extends** *TaskFailure*

**any**

bs

rb

z

s

**where**

grd1 : $bs \in BS$

grd2 : $rb \in dom(attached)$

grd3 : $attached(rb) = bs$

grd4 : $z \in 1 \mathinner{..} n$

grd5 : $responsible(z) = bs$

grd6 : $asgn\_z(rb) = z$

grd7 : $s \in 1 \mathinner{..} k$

grd8 : $asgn\_s(rb) = s$

grd9 : $local\_map(bs)(z)(s) = incompl$

**then**

act1 : $territory(z) := territory(z) \leftarrow \{s \mapsto incompl\}$

act3 : $asgn\_s := \{rb\} \lhd asgn\_s$

act4 : $asgn\_z := \{rb\} \lhd asgn\_z$

act5 : $attached := \{rb\} \lhd attached$

**end**

**Event** *ReassignBS* $\widehat{=}$

**extends** *ReassignRB*

**any**

bs_i

bs_j

rbs

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $rbs \subset dom(attached)$

grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$

grd5 : $bs\_i \in ran(responsible)$

grd6 : $ran(rbs \lhd asgn\_s) = \{0\}$

grd7 : $rbs \neq \varnothing$

grd8 : $bs\_j \in ran(responsible)$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_i \in ran(rbs \lhd attached)$

**then**

act1 : $\texttt{attached} := \texttt{attached} \lhd\!\!\!- (\texttt{rbs} \times \{\texttt{bs\_j}\})$

**end**

**Event** *GetAdditionalResponsibility* $\widehat{=}$

**refines** *GetAdditionalResponsibility*

**any**

$bs\_i$
$bs\_j$
$zs$
$rbs$

**where**

grd1 : $bs\_i \in BS$
grd2 : $bs\_j \in BS$
grd3 : $zs \subset 1 \mathinner{..} n$
grd4 : $zs = dom(responsible \rhd \{bs\_i\})$
grd5 : $rbs \subset dom(attached)$
grd6 : $rbs = dom(attached \rhd \{bs\_i\})$
grd9 : $bs\_i \neq bs\_j$
grd10 : $bs\_j \in ran(responsible)$
grd12 : $bs\_j \notin dom(msg)$

**then**

act1 : $responsible := responsible \lhd\!\!\!- (zs \times \{bs\_j\})$
act3 : $attached := attached \lhd\!\!\!- (rbs \times \{bs\_j\})$
act4 : $asgn\_s := asgn\_s \lhd\!\!\!- (rbs \times \{0\})$
act5 : $asgn\_z := asgn\_z \lhd\!\!\!- (rbs \times \{0\})$
act6 : $local\_map(bs\_i) := \varnothing$

**end**

**Event** *UpdateMap* $\widehat{=}$

**refines** *UpdateMap*

**any**

$bs$
$z$
$s$

**where**

grd1 : $bs \in BS$
grd2 : $z \in 1 \mathinner{..} n$
grd3 : $s \in 1 \mathinner{..} k$
grd4 : $responsible(z) \neq bs$
grd5 : $bs \in ran(responsible)$
grd6 : $(bs \mapsto (z \mapsto s)) \in msg$

**then**

act1 : $local\_map(bs) := local\_map(bs) \Leftarrow \{z \mapsto local\_map(bs)(z) \Leftarrow \{s \mapsto compl\}\}$

act2 : $msg := msg \setminus \{bs \mapsto (z \mapsto s)\}$

**end**

**END**

**MACHINE** RS_ref5

**REFINES** RS_ref4

**SEES** cntx3

**VARIABLES**

territory

responsible

attached

local_map

asgn_z

asgn_s

msg

active

counter

**INVARIANTS**

inv1 : $active \subseteq dom(attached)$

inv2 : $active \subseteq dom(asgn\_s)$

inv3 : $active \subseteq dom(asgn\_z)$

**EVENTS**

**Initialisation**

*extended*

**begin**

act1 : $\mathtt{territory} := \mathtt{1} \mathinner{..} \mathtt{n} \times \{\mathtt{1} \mathinner{..} \mathtt{k} \times \{\mathtt{incompl}\}\}$

act2 : $\mathtt{responsible} :\in \mathtt{1} \mathinner{..} \mathtt{n} \nrightarrow \mathtt{BS}$

act3 : $\mathtt{attached} :\in \mathtt{RB} \nrightarrow \mathtt{BS}$

act4 : $\mathtt{local\_map} := \mathtt{BS} \times \{\mathtt{1} \mathinner{..} \mathtt{n} \times \{\mathtt{1} \mathinner{..} \mathtt{k} \times \{\mathtt{incompl}\}\}\}$

act5 : $\mathtt{asgn\_s} := \mathtt{RB} \times \{\mathtt{0}\}$

act6 : $\mathtt{asgn\_z} := \mathtt{RB} \times \{\mathtt{0}\}$

act7 : $\mathtt{counter} := \mathtt{n} * \mathtt{k}$

act8 : $\mathtt{msg} := \mathtt{BS} \times \varnothing$

act9 : $active := RB$

**end**

**Event** *NewTask* $\widehat{=}$

**refines** *NewTask*

**any**

*bs*

*rb*

*z*

*s*

**where**

grd1 : $bs \in BS$
grd2 : $rb \in active$
grd3 : $attached(rb) = bs$
grd4 : $asgn\_s(rb) = 0$
grd5 : $z \in 1 \mathinner{..} n$
grd6 : $responsible(z) = bs$
grd7 : $asgn\_z(rb) = 0$
grd8 : $s \in 1 \mathinner{..} k$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $s \notin ran((dom(asgn\_z \rhd \{z\})) \lhd asgn\_s)$

**then**

act1 : $asgn\_s(rb) := s$
act2 : $asgn\_z(rb) := z$

**end**

**Event** *TaskSuccess* $\widehat{=}$

**refines** *TaskSuccess*

**any**

$z$
$s$
$bs$
$rb$
$bss$

**where**

grd1 : $bs \in BS$
grd2 : $rb \in active$
grd3 : $attached(rb) = bs$
grd4 : $z \in 1 \mathinner{..} n$
grd5 : $responsible(z) = bs$
grd6 : $asgn\_z(rb) = z$
grd7 : $s \in 1 \mathinner{..} k$
grd8 : $asgn\_s(rb) = s$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $bss = BS \setminus \{bs\}$

**then**

act1 : $territory(z) := territory(z) \lhd\!\!\!- \{s \mapsto compl\}$
act2 : $local\_map(bs) := local\_map(bs) \lhd\!\!\!- \{z \mapsto local\_map(bs)(z) \lhd\!\!\!- \{s \mapsto compl\}\}$
act3 : $asgn\_s(rb) := 0$
act4 : $asgn\_z(rb) := 0$
act5 : $msg := msg \cup (bss \times \{z \mapsto s\})$
act6 : $counter := counter - 1$

**end**

**Event** *TaskFailure* $\widehat{=}$

**refines** *TaskFailure*

**any**

$bs$

$rb$

$z$

$s$

**where**

grd1 : $bs \in BS$

grd2 : $rb \in dom(attached)$

grd3 : $attached(rb) = bs$

grd4 : $z \in 1 \mathinner{..} n$

grd5 : $responsible(z) = bs$

grd6 : $asgn\_z(rb) = z$

grd7 : $s \in 1 \mathinner{..} k$

grd8 : $asgn\_s(rb) = s$

grd9 : $local\_map(bs)(z)(s) = incompl$

grd10 : $rb \notin active$

**then**

act1 : $territory(z) := territory(z) \lhd\!\!\!- \{s \mapsto incompl\}$

act3 : $asgn\_s := \{rb\} \lhd\!\!\!- asgn\_s$

act4 : $asgn\_z := \{rb\} \lhd\!\!\!- asgn\_z$

act5 : $attached := \{rb\} \lhd\!\!\!- attached$

**end**

**Event** *ReassignNewBStoRBs* $\widehat{=}$

**refines** *ReassignBS*

**any**

$bs\_i$

$bs\_j$

$rbs$

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $rbs \subset active$

grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$

grd5 : $bs\_i \in ran(responsible)$

grd6 : $ran(rbs \lhd asgn\_s) = \{0\}$

grd7 : $rbs \neq \varnothing$

grd8 : $bs\_j \in ran(responsible)$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_i \in ran(rbs \lhd\!\!\!- attached)$

grd11 : $dom(attached \rhd \{bs\_j\}) \not\subseteq active$

**then**

act1 : $attached := attached \lhd\!\!\!- (rbs \times \{bs\_j\})$

**end**

**Event** *GetAdditionalResponsibility* $\widehat{=}$

**refines** *GetAdditionalResponsibility*

**any**

$bs\_i$

$bs\_j$

$zs$

$rbs$

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $zs \subset 1 \mathinner{..} n$

grd4 : $zs = dom(responsible \rhd \{bs\_i\})$

grd5 : $rbs \subset active$

grd6 : $rbs = dom(attached \rhd \{bs\_i\})$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_j \in ran(responsible)$

grd11 : $bs\_j \notin dom(msg)$

**then**

act1 : $responsible := responsible \lhd\!\!\!- (zs \times \{bs\_j\})$

act3 : $attached := attached \lhd\!\!\!- (rbs \times \{bs\_j\})$

act4 : $asgn\_s := asgn\_s \lhd\!\!\!- (rbs \times \{0\})$

act5 : $asgn\_z := asgn\_z \lhd\!\!\!- (rbs \times \{0\})$

act6 : $local\_map(bs\_i) := \varnothing$

**end**

**Event** *UpdateMap* $\widehat{=}$

**extends** *UpdateMap*

**any**

bs

z

s

**where**

grd1 : $\mathtt{bs} \in \mathtt{BS}$

grd2 : $\mathtt{z} \in 1 \mathinner{..} \mathtt{n}$

grd3 : $\mathtt{s} \in 1 \mathinner{..} \mathtt{k}$

grd4 : $\mathtt{responsible}(\mathtt{z}) \neq \mathtt{bs}$

grd5 : $\mathtt{bs} \in \mathtt{ran}(\mathtt{responsible})$

grd6 : $(\mathtt{bs} \mapsto (\mathtt{z} \mapsto \mathtt{s})) \in \mathtt{msg}$

**then**

act1 : $\texttt{local\_map(bs)} := \texttt{local\_map(bs)} \lhd\!\!+ \{\texttt{z} \mapsto \texttt{local\_map(bs)(z)} \lhd\!\!+ \{\texttt{s} \mapsto \texttt{compl}\}\}$

act2 : $\texttt{msg} := \texttt{msg} \setminus \{\texttt{bs} \mapsto (\texttt{z} \mapsto \texttt{s})\}$

**end**

**Event** *RobotFailure* $\widehat{=}$

**any**

$rb$

**where**

grd1 : $rb \in active$

grd2 : $card(active) > 1$

**then**

act1 : $active := active \setminus \{rb\}$

**end**

**Event** *SendRobotsToBS* $\widehat{=}$

**refines** *ReassignBS*

**any**

$bs\_i$

$bs\_j$

$rbs$

**where**

grd1 : $bs\_i \in BS$

grd2 : $bs\_j \in BS$

grd3 : $rbs \subset active$

grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$

grd5 : $bs\_i \in ran(responsible)$

grd6 : $ran(rbs \lhd asgn\_s) = \{0\}$

grd7 : $rbs \neq \varnothing$

grd8 : $bs\_j \in ran(responsible)$

grd9 : $bs\_i \neq bs\_j$

grd10 : $bs\_i \in ran(rbs \lhd\!\!| attached)$

grd11 : $rbs = dom(attached \rhd \{bs\_i\})$

grd12 : $\forall z \cdot z \in 1..n \wedge responsible(z) = bs\_i \Rightarrow local\_map(bs\_i)(z)[1..k] = \{compl\}$

**then**

act1 : $attached := attached \lhd\!\!+ (rbs \times \{bs\_j\})$

**end**

**END**

**MACHINE** RS_ref6

**REFINES** RS_ref5

**SEES** cntx3

**VARIABLES**

territory

responsible

attached

local_map

asgn_z

asgn_s

msg

active

operating

counter

**INVARIANTS**

inv1 : $operating \subseteq BS$

inv2 : $operating \subseteq ran(responsible)$

inv3 : $\forall bs \cdot bs \in BS \land local\_map(bs) = \varnothing \Rightarrow bs \notin operating$

**EVENTS**

**Initialisation**

*extended*

**begin**

act1 : $\mathtt{territory} := \mathtt{1}\,..\,\mathtt{n} \times \{\mathtt{1}\,..\,\mathtt{k} \times \{\mathtt{incompl}\}\}$

act2 : $\mathtt{responsible} :\in \mathtt{1}\,..\,\mathtt{n} \nrightarrow \mathtt{BS}$

act3 : $\mathtt{attached} :\in \mathtt{RB} \nrightarrow \mathtt{BS}$

act4 : $\mathtt{local\_map} := \mathtt{BS} \times \{\mathtt{1}\,..\,\mathtt{n} \times \{\mathtt{1}\,..\,\mathtt{k} \times \{\mathtt{incompl}\}\}\}$

act5 : $\mathtt{asgn\_s} := \mathtt{RB} \times \{\mathtt{0}\}$

act6 : $\mathtt{asgn\_z} := \mathtt{RB} \times \{\mathtt{0}\}$

act7 : $\mathtt{counter} := \mathtt{n} * \mathtt{k}$

act8 : $\mathtt{msg} := \mathtt{BS} \times \varnothing$

act9 : $\mathtt{active} := \mathtt{RB}$

act10 : $operating := BS$

**end**

**Event** *NewTask* $\widehat{=}$

**refines** *NewTask*

**any**

*bs*

*rb*

*z*

$s$

**where**

grd1 : $bs \in operating$
grd2 : $rb \in active$
grd3 : $attached(rb) = bs$
grd4 : $asgn\_s(rb) = 0$
grd5 : $z \in 1 \mathinner{..} n$
grd6 : $responsible(z) = bs$
grd7 : $asgn\_z(rb) = 0$
grd8 : $s \in 1 \mathinner{..} k$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $s \notin ran((dom(asgn\_z \rhd \{z\})) \lhd asgn\_s)$

**then**

act1 : $asgn\_s(rb) := s$
act2 : $asgn\_z(rb) := z$

**end**

**Event** *TaskSuccess* $\widehat{=}$

**refines** *TaskSuccess*

**any**

$z$
$s$
$bs$
$rb$
$bss$

**where**

grd1 : $bs \in operating$
grd2 : $rb \in active$
grd3 : $attached(rb) = bs$
grd4 : $z \in 1 \mathinner{..} n$
grd5 : $responsible(z) = bs$
grd6 : $asgn\_z(rb) = z$
grd7 : $s \in 1 \mathinner{..} k$
grd8 : $asgn\_s(rb) = s$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $bss = BS \setminus \{bs\}$
grd11 : $counter > 0$

**then**

act1 : $territory(z) := territory(z) \lhd\!\!\!\!- \{s \mapsto compl\}$
act2 : $local\_map(bs) := local\_map(bs) \lhd\!\!\!\!- \{z \mapsto local\_map(bs)(z) \lhd\!\!\!\!- \{s \mapsto compl\}\}$
act3 : $asgn\_s(rb) := 0$
act4 : $asgn\_z(rb) := 0$

act5 : $msg := msg \cup (bss \times \{z \mapsto s\})$
act6 : $counter := counter - 1$

**end**

**Event** *TaskFailure* $\widehat{=}$

**refines** *TaskFailure*

**any**

$bs$
$rb$
$z$
$s$

**where**

grd1 : $bs \in operating$
grd2 : $rb \in dom(attached)$
grd3 : $attached(rb) = bs$
grd4 : $z \in 1 \mathinner{..} n$
grd5 : $responsible(z) = bs$
grd6 : $asgn\_z(rb) = z$
grd7 : $s \in 1 \mathinner{..} k$
grd8 : $asgn\_s(rb) = s$
grd9 : $local\_map(bs)(z)(s) = incompl$
grd10 : $rb \notin active$

**then**

act1 : $territory(z) := territory(z) \lhd\!\!+ \{s \mapsto incompl\}$
act3 : $asgn\_s := \{rb\} \lhd asgn\_s$
act4 : $asgn\_z := \{rb\} \lhd asgn\_z$
act5 : $attached := \{rb\} \lhd attached$

**end**

**Event** *ReassignNewBStoRBs* $\widehat{=}$

**refines** *ReassignNewBStoRBs*

**any**

$bs\_i$
$bs\_j$
$rbs$

**where**

grd1 : $bs\_i \in operating$
grd2 : $bs\_j \in operating$
grd3 : $rbs \subset active$
grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$
grd5 : $ran(rbs \lhd asgn\_s) = \{0\}$
grd6 : $rbs \neq \varnothing$
grd7 : $bs\_i \neq bs\_j$
grd8 : $bs\_i \in ran(rbs \lhd\!\!+ attached)$

grd9 : $dom(attached \rhd \{bs\_j\}) \not\subseteq active$

**then**

act1 : $attached := attached \lhd\!\!\!- (rbs \times \{bs\_j\})$

**end**

**Event** *GetAdditionalResponsibility* $\widehat{=}$

**refines** *GetAdditionalResponsibility*

**any**

$bs\_i$
$bs\_j$
$zs$
$rbs$

**where**

grd1 : $bs\_i \in BS$
grd2 : $bs\_j \in operating$
grd3 : $zs \subset 1 \mathinner{..} n$
grd4 : $zs = dom(responsible \rhd \{bs\_i\})$
grd5 : $rbs \subset active$
grd6 : $rbs = dom(attached \rhd \{bs\_i\})$
grd7 : $bs\_i \neq bs\_j$
grd8 : $bs\_j \notin dom(msg)$
grd9 : $bs\_i \notin operating$

**then**

act1 : $responsible := responsible \lhd\!\!\!- (zs \times \{bs\_j\})$
act3 : $attached := attached \lhd\!\!\!- (rbs \times \{bs\_j\})$
act4 : $asgn\_s := asgn\_s \lhd\!\!\!- (rbs \times \{0\})$
act5 : $asgn\_z := asgn\_z \lhd\!\!\!- (rbs \times \{0\})$
act6 : $local\_map(bs\_i) := \varnothing$

**end**

**Event** *UpdateMap* $\widehat{=}$

**refines** *UpdateMap*

**any**

$bs$
$z$
$s$

**where**

grd1 : $bs \in operating$
grd2 : $z \in 1 \mathinner{..} n$
grd3 : $s \in 1 \mathinner{..} k$
grd4 : $responsible(z) \neq bs$
grd5 : $(bs \mapsto (z \mapsto s)) \in msg$

**then**

act1 : $local\_map(bs) := local\_map(bs) \Leftarrow \{z \mapsto local\_map(bs)(z) \Leftarrow \{s \mapsto compl\}\}$

act2 : $msg := msg \setminus \{bs \mapsto (z \mapsto s)\}$

**end**

**Event** *RobotFailure* $\widehat{=}$

**extends** *RobotFailure*

**any**

rb

**where**

grd1 : $\texttt{rb} \in \texttt{active}$

grd2 : $\texttt{card}(\texttt{active}) > 1$

**then**

act1 : $\texttt{active} := \texttt{active} \setminus \{\texttt{rb}\}$

**end**

**Event** *SendRobotsToBS* $\widehat{=}$

**refines** *SendRobotsToBS*

**any**

$bs\_i$

$bs\_j$

$rbs$

**where**

grd1 : $bs\_i \in operating$

grd2 : $bs\_j \in operating$

grd3 : $rbs \subset active$

grd4 : $ran(rbs \lhd attached) = \{bs\_i\}$

grd5 : $ran(rbs \lhd asgn\_s) = \{0\}$

grd6 : $rbs \neq \varnothing$

grd7 : $bs \neq bs\_j$

grd8 : $bs\_i \in ran(rbs \lhd\!\!\cdot attached)$

grd9 : $rbs = dom(attached \rhd \{bs\_i\})$

grd10 : $\forall z \cdot z \in 1..n \wedge responsible(z) = bs\_i \Rightarrow local\_map(bs\_i)(z)[1..k] = \{compl\}$

**then**

act1 : $attached := attached \Leftarrow (rbs \times \{bs\_j\})$

**end**

**Event** *BaseStationFailure* $\widehat{=}$

**Status** convergent

**any**

$bs$

**where**

grd1 : $bs \in operating$

grd2 : $card(operating) > 1$

**then**

act1 : $operating := operating \setminus \{bs\}$

**end**

**VARIANT**

card(operating)

**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 978-952-12-2766-0
ISSN 1239-1891