Inna Pereverzeva | Elena Troubitsyna | Linas Laibinis

Rigorous Development of a Safe Multi-Agent System

Turku Centre *for* Computer Science

TUCS Technical Report
No 1004, April 2011

Rigorous Development of a Safe Multi-Agent System

Inna Pereverzeva
Åbo Akademi University, Department of 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 1004, April 2011

Abstract

It is widely recognised that system complexity poses the major threat to dependability. Yet, such complex distributed systems as multi-agent systems are increasingly used in critical applications. To ensure their dependability, we need powerful development techniques that would allow us to master complexity inherent to multi-agent systems and formally verify correctness of agent interactions while performing safety-critical collaborative activities. In this paper we propose a rigorous approach to the development of a critical multi-agent system by refinement in Event-B. Our approach offers the developers a scalable method for modelling and verification of complex agent interactions and formal verification of their correctness and safety. We present a formal development of a hospital multi-agent system and show that refinement in Event-B facilitates development of complex dependable systems.

Keywords: Event-B, refinement, formal modelling, multi-agent systems, safety

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Multi-agent systems (MAS) are complex decentralised distributed systems composed of agents asynchronously communicating with each other. Agents are computer programs acting autonomously on behalf of a person or organisation, while coordinating their activities by communication [10]. MAS are increasingly used in various critical application such as factories, hospitals, rescue operations in disaster areas etc. However, the wide-spread use of MAS is hindered by the lack of methods for ensuring their dependability.

In this paper we focus on studying complex agent interactions while conducting safety-critical collaborative activities. In critical MAS, incorrect execution of these activities might have devastating consequences.

In this paper we consider a hospital MAS. We focus on modelling how main safety-critical collaborative activities – handling emergency situations (caused by the occurrence of sudden critical conditions of a patient) and updating patients records. Obviously, incorrect execution of these activities might lead to patient's death. Hence, there is a clear need for a development method that would guarantee correct provision of these operations.

However, ensuring correctness in a hospital MAS is a challenging issue due to faults caused by agent disconnections, dynamic role allocation (different shifts of medical personnel) and autonomy of the agent behaviour. To address these challenges, we need the system-level modelling approaches that would support formal verification of correctness and facilitate discovery of restrictions that should be imposed on a system to guarantee its safety.

In this paper we demonstrate how to develop a critical MAS by refinement in Event-B. Event-B [2, 11] is a formal framework for developing complex systems. The main development technique of Event-B is refinement – a process of a gradual transformation of an abstract specification into a specification directly translatable into an implementation. Correctness of each refinement step is verified by proofs. The Rodin platform [12] provides the developers with an automated tool support for constructing and verifying formal system models.

In our development we adopt a system's approach, i.e., abstractly model the entire system, so that the specifications of its individual components can be obtained from it by decomposition. At each refinement step we introduce certain details of complex agent interaction and prove the essential conditions associated with them.

The formal verification process facilitates not only safety assurance but also discovery of restrictions that should be imposed on the system behaviour to guarantee its safety. We believe that the formal development in Event-B offers a scalable technique for development and verification of complex critical MAS.

The paper is structured as follows. In Section 2 we present our formal modelling framework – Event-B. In Section 3 we describe a hospital MAS and show how to abstractly model a MAS and introduce fault tolerance by refinement. In

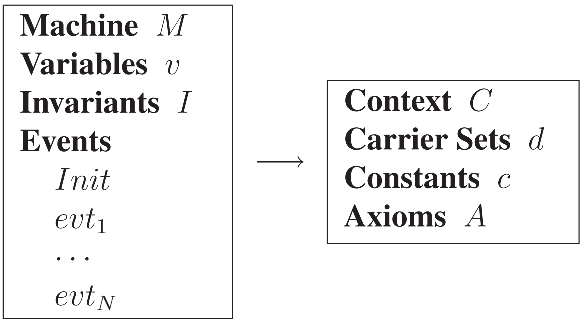

Figure 1: Event-B machine and context

Section 4 we show how to introduce complex collaborative agent interactions by refinement and verify their safety. Finally, in Section 5 we overview the related work, discuss the achieved results and outline the future work.

2 Formal Modeling and Refinement in Event B

We start by briefly describing our development framework. The Event-B formalism is an extension of the B Method [1], a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. Event-B is actively used within the FP7 ICT project DEPLOY to develop dependable systems from various domains [3].

2.1 Modelling and Refinement in Event B

In Event-B, a system specification (model) is defined using the notion of an *abstract state machine* [11]. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state. Therefore, it describes the dynamic part (behaviour) of the modelled system. A machine may also have the accompanying component, called *context*, which contains the static part of the system. In particular, a context can include user-defined carrier sets, constants and their properties, which are given as a list of model axioms. A general form of Event-B models is given in Figure 1.

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

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

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

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

Figure 2: Before-after predicates

where vl is a list of new local variables (parameters), the guard g is a state predicate, and the action S is a statement (assignment). In case when vl is empty, the event syntax becomes **when** g **then** S **end**. If g is always true, the syntax can be further simplified to **begin** S **end**.

The occurrence of events represents the observable behaviour of the system. The guard 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 assignments. The assignments can be either deterministic or non-deterministic. 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.

Event-B Semantics The semantics of an Event-B model is formulated as a collection of *proof obligations* – logical sequents. Below we describe only the most important proof obligations that should be verified (proved) for the initial and refined models. The full list of proof obligations can be found in [2].

The semantics of Event-B actions is defined using so called before-after (BA) predicates [2, 11]. A before-after predicate describes a relationship between the system states before and after execution of an event, as shown in Figure 2. Here x and y are disjoint lists (partitions) of state variables, and x', y' represent their values in the after-state.

The initial Event-B model should satisfy the event feasibility and invariant preservation properties. For each event of the model, evt_i, its feasibility means that, whenever the event is enabled, its before-after predicate (BA) is well-defined, i.e., exists some reachable after-state:

$$A(d,c),\ I(d,c,v),\ g_i(d,c,v) \ \vdash \ \exists v' \cdot BA_i(d,c,v,v') \qquad \text{(FIS)}$$

where A is model axioms, I is the model invariant, g_i is the event guard, d are model sets, c are model constants, and v, v' are the variable values before and after the event execution.

Each event evt_i of the initial Event-B model should also preserve the given model invariant:

$$A(d,c),\ I(d,c,v),\ g_i(d,c,v),\ BA_i(d,c,v,v') \ \vdash\ I(d,c,v') \qquad \text{(INV)}$$

Since the initialisation event has no initial state and guard, its proof obligation is simpler:

$$A(d,c),\ BA_{Init}(d,c,v') \ \vdash\ I(d,c,v') \qquad \text{(INIT)}$$

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible at the abstract level. Moreover, Event-B formal development supports data refinement, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of the refined machine formally defines the relationship between the abstract and concrete variables.

To verify correctness of a refinement step, we need to prove a number of proof obligations for the refined model. For brevity, here we show only a few essential ones.

Let us first introduce a shorthand $H(d,c,v,w)$ to stand for the hypotheses $BA(d,c),\ I(d,c,v),\ I'(d,c,v,w)$, where $I,\ I'$ are respectively the abstract and refined invariants. Then the invariant preservation property for an event evt_i of the refined model can be presented as follows:

$$H(d,c,v,w),\ g'_i(d,c,w),\ BA'_i(d,c,w,w') \ \vdash\ I'(d,c,w') \qquad \text{(REF\_INV)}$$

where g'_i is the refined guard, BA'_i is a before-after predicate of the refined event, and v,w are respectively the abstract and concrete variables.

The event guards in the refined model can be only strengthened in a refinement step:

$$H(d,c,v,w),\ g'_i(d,c,w) \ \vdash\ g_i(d,c,v) \qquad \text{(REF\_GRD)}$$

where g_i, g'_i are respectively the abstract and concrete guards of the event evt_i.

Finally, the *simulation* proof obligation requires to show that the "execution" of the refined event is not contradictory with its abstract version:

$$H(d,c,v,w),\ g'_i(d,c,w),\ BA'_i(d,c,w,w') \ \vdash\ BA_i(d,c,v,v') \qquad \text{(REF\_SIM)}$$

where BA_i, BA'_i are respectively the abstract and concrete before-after predicates of the same event evt_i.

The Event-B refinement process allows us to gradually introduce implementation details, while preserving functional correctness. The model verification effort, in particular, automatic generation and proving of the required proof obligations, is significantly facilitated by the Rodin platform [12]. Proof-based verification as well as reliance on abstraction and decomposition adopted in Event-B offers the designers a scalable support in the development of such complex distributed systems as multi-agent systems. Next we demonstrate our approach to formal modelling of a multi-agent system in Event-B.

3 Abstract Modelling of a Hospital MAS

3.1 A case study description

In this paper we present a formal development of a hospital MAS. The system consists of two types of agents – patients and medical personnel (called doctors for simplicity). The condition of each patient is monitored by the corresponding medical equipment – an agent representing a patient. The doctor agents are running on Pocket PC-based devices – Personal Digital Assistants (PDA). The hospital provides the wireless connectivity to the doctor agents. Each doctor is associated with one agent. From now on, we will use the terms ”patient” and ”doctor” to designate both agents and people that they represent.

The medical equipment continuously updates in the patient's medical record consisting of different medical measurements and detects emergencies – dangerous changes of critical parameters (e.g., blood pressure, pulse rate and etc.). In case of emergency, the patient agent generates an emergency call that is communicated to the doctors treating the patient. An important safety requirement imposed on the system is that *all emergencies should be promptly handled by the doctors*. In spite of its seeming simplicity, this requirement is hard to ensure. Indeed, a MAS operates in a volatile communication environment, i.e., agents might experience temporal disconnections. Hence the design of our system should incorporate certain fault tolerance mechanisms that would guarantee that each emergency call is eventually handled by some doctor. Moreover, different doctors can be associated with the same patient during different shifts. Therefore, we have to ensure that at the end of a doctor's shift all his/her patients are handed over to another doctor.

Another important safety requirement associated with the system is to guarantee that *a doctor always accesses the most recent patient record and the patient's data are always kept in a consistent state*. We assume that the patient record is stored at the equipment associated with her. To ensure that these requirements are satisfied, we should regulate the access to the patient's data. A specific delivery of a medicine, prescription of a treatment and so on are introduced into the patient's log by the medical personnel via their PDAs. To ensure that the data are updated consistently we only allow the doctor to modify patient's data when he or she is in a close proximity to the patient. When the doctor arrives to the patient location, the patient data become available at the doctor's PDA and the doctor can modify them. All the modifications are synchronised with the data stored by the patient's equipment. When the doctor finishes examining the patient or delivering a medicine and leaves, the connection to the patient's data is lost. Such a restriction allows us to ensure that only a doctor who is in a close proximity to a patient is allowed to modify the patient's record. Moreover, it also ensures that the doctor has the access to the freshest info about the patient. This precludes, e.g., a possibility of delivering the medicine twice. [1]

The safety-critical requirements imposed on the system should be fulfilled in the course of complex agent interactions. Next we demonstrate how refinement process in Event-B can facilitate modelling of intertangled agent interactions and verification of safety properties.

3.2 Towards modelling agent interdependencies

Our abstract specification – the machine *Hospital* shown in Fig. 3 – is very simple. It models the behaviour of the entire hospital MAS in a highly abstract way. We define the variable med_agents – the set of active agents of the type $MEDSTAFF$. The events $Activate$ and $Deactivate$ model joining and leaving the hospital location by the agents. While an agent is active, it can perform certain activities which is abstractly modelled by the event $Activity$.

Machine $Hospital$
Variables med_agents
Invariants
  $inv_1 : med\_agents \subseteq MEDSTAFF$
Events
  $Initialisation \ \widehat{=}$
    begin
      $med\_agents := \varnothing$
    end
  $Activate \ \widehat{=}$
    any ma
    when
      $ma \in MEDSTAFF$
      $ma \notin med\_agents$
    then
      $med\_agents := med\_agents \cup \{ma\}$
    end
  $Activity \ \widehat{=}$
    then
      $skip$
    end
  $Deactivate \ \widehat{=}$
    any ma
    when
      $ma \in med\_agents$
    then
      $med\_agents := med\_agents \setminus \{ma\}$
    end

Figure 3: Hospital: abstract specification

[1]Modeling the security requirements ensuring patient's data integrity is outside of the scope of this paper.

In our first refinement step (the excerpt from which is shown in Fig. 4) we augment our model with representation of patients. The variable $patients$ defines a set of patients admitted to the hospital. Each patient arriving at the hospital is associated with a doctor who has a primarily responsibility for treating the patient. To model the interdependence between patients and medical personnel we introduce the variable *assigned_doctor*, which is defined as total function associating patients with active doctor agents. In addition, we add the new events $PatientArrival$ and $PatientDischarge$ to model patient arrival and discharge from the hospital correspondingly.

Machine $Hospital1$ **Refines** $Hospital$
Variables ... $patients, assigned\_doctor, last\_visit, visited$
Invariants
$inv_1 : patients \subseteq PATIENTS$
$inv_2 : assigned\_doctor \in patients \rightarrow med\_agents$
$inv_3 : last\_visit \in patients \nrightarrow MEDSTAFF$
$inv_4 : visited \subseteq patients$
$inv_5 : last\_visit[visited] \subseteq med\_agents$
$inv_6 : visited \subseteq dom(last\_visit)$
Events
...
$PatientArrival \mathrel{\widehat{=}}$
any ma, pa
when
$pa \in PATIENTS \wedge pa \notin patients \wedge ma \in med\_agents$
then
$patients := patients \cup \{pa\}$
$assigned\_doctor(pa) := ma$
end
$VisitBegin \mathrel{\widehat{=}}$
Refines $Activity$
any ma, pa
when
$ma \in med\_agents \wedge pa \in patients \wedge pa \notin visited \wedge ma \notin last\_visit[visited]$
then
$last\_visit(pa) := ma$
$visited := visited \cup \{pa\}$
end
$AgentLeaving \mathrel{\widehat{=}}$
Refines $Deactivate$
any ma
when
$ma \in med\_agents \wedge ma \notin ran(assigned\_doctor) \wedge ma \notin last\_visited[visited]$
then
$med\_agents := med\_agents \setminus \{ma\}$
end
$ReassignDoctor \mathrel{\widehat{=}}$
Refines $Deactivate$
any ma, ma_new
when
$ma \in ran(assigned\_doctor) \wedge ma \notin last\_visited[visited] \wedge$
$ma\_new \in med\_agents \wedge ma\_new \neq ma$
then
$med\_agents := med\_agents \setminus \{ma\}$
$assigned\_doctor := assigned\_doctor \lhd\!\!- (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$
end
...

Figure 4: Hospital: the first refinement step

Moreover, in the refined specification we also elaborate on the event $Activity$. Essentially, the medical personnel should examine the patients and deliver the prescribed medicine. We generalise these actions under the general term "visiting a patient". In our refined model, we define the variable $visited$ representing a subset of patients that are currently being examined. The new variable $last\_visited$ stores for every patient the id of the last doctor agent that has visited her. The events $VisitBegin$ and $VisitEnd$ refine the abstract event $Activity$ and model the visiting procedure.

In our abstract specification we have assumed that doctor agents can leave the hospital at any time. However, to guarantee safety of the patients, we must impose certain restrictions on when the doctors can actually leave the hospital. Before a doctor agent can leave the hospital, we should reassign his/her patients to another doctor. Moreover, we assume that the doctor cannot leave the hospital in the middle of a patient visit. We split the abstract event $Deactivate$ into two corresponding events: $AgentLeaving$ and $ReassignDoctor$. The event $AgentLeaving$ models leaving the location by a doctor. Here we check that the doctor does not have any assigned patients and is not currently involved in examining a patient.

Due to a lack of space, we show only the excerpts from our formal specifications. The complete specifications can be found in the Appendix.

3.3 Introducing fault tolerance by refinement

In the specification $Hospital1$, while defining the events $AgentLeaving$ and $ReassignDoctor$, we have abstracted away from the reasons behind of doctor leaving and patient reassignment. Essentially, a doctor agent might leave the location because the doctor's shift is over or because the agent has irrecoverably failed and should be permanently disconnected. At the second refinement step we introduce a distinction between the normal agent leaving and its disconnection due to failure.

In a MAS, the agents often lose connection only for a short period of time. After the connection is restored, the agent should be able to continue its operations. Therefore, after detection a loss of connection, the location should not immediately disengage the disconnected agent but rather set a deadline before which the agent should reconnect. If the disconnected agent restores its connection before the deadline then it can continue its normal activities. However, if the agent fails to do so, the location should permanently disengage the agent.

To model such a behaviour, in our next refinement step shown in Fig. 5 we introduce the variable $disconnected$ representing the subset of active agents that are detected as disconnected. Moreover, to model the timeout mechanism, we define the variable $timer$ of the type $\{inactive, active, timeout\}$. Initially, for every active agent, the $timer$ value is set to $inactive$. As soon as active agent loses connection with the location, its id is added to the set $disconnected$ and its timer value becomes $active$. This behaviour is specified in the new event $DisconnectAgent$. A temporally disconnected agent can succeed or fail to reconnect as modelled by the events $ReconnectionSuccessful$ and $ReconnectionFailed$ respectively. If the agent reconnects before the value of timer becomes $timeout$, the timer value is changed to $inactive$ and the agent continues its activities virtually uninterrupted. Otherwise, the agent is removed from the set of active agents.

Machine $Hospital2$ **Refines** $Hospital1$
Variables ... $disconnected, timer$
Invariants
  $inv_1 : disconnected \subseteq med\_agents$
  $inv_2 : timer \in med\_agents \rightarrow STATE$
  $inv_3 : \forall ma \cdot (ma \in med\_agents \wedge timer(ma) \neq inactive \Leftrightarrow ma \in disconnected)$
Events
  ...
  $DisconnectAgent \mathrel{\widehat{=}}$
    any ma
    when
      $ma \in med\_agents \wedge ma \notin disconnected$
    then
      $disconnected := disconnected \cup \{ma\}$
      $timer(ma) := active$
    end
  $ReconectionFailed \mathrel{\widehat{=}}$
    any ma
    when
      $ma \in disconnected \wedge timer(ma) = active$
    then
      $timer(ma) := timeout$
    end
  $DetectFailedAgent \mathrel{\widehat{=}}$
    Refines $ReassignDoctor$
    any ma, ma_new
    when
      $ma \in ran(assigned\_doctor) \wedge ma \notin last\_visited[visited] \wedge ma\_new \in med\_agents \wedge$
      $ma\_new \neq ma \wedge ma \in disconnected \wedge timer(ma) = timeout \wedge$
      $ma\_new \notin disconnected \vee (ma\_new \in disconnected \wedge timer(ma\_new) = active)$
    then
      $med\_agents := med\_agents \setminus \{ma\}$
      $assigned\_doctor := assigned\_doctor \mathbin{⩤\!\!+} (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$
      $disconnected := disconnected\{ma\}$
      $timer := \{ma\} \mathbin{⩤} timer$
    end
  $DetectFailedFreeAgent \mathrel{\widehat{=}}$
    Refines $AgentLeaving$
    any ma
    when
      $ma \in med\_agents \wedge ma \notin ran(assigned\_doctor) \wedge ma \notin last\_visited[visited] \wedge$
      $ma \in disconnected \wedge timer(ma) = timeout$
    then
      $med\_agents := med\_agents \setminus \{ma\}$
      $disconnected := disconnected\{ma\}$
      $timer := \{ma\} \mathbin{⩤} timer$
    end
  ...

Figure 5: Hospital: the second refinement step

The introduction of an agent disconnection also affects the some abstract events. To model separate case of doctor leaving the location because of the end of shift or due to the disconnection timeout, we split the event $AgentLeaving$ into two events $NormalAgentLeaving$ and $DetectFailedFreeAgent$. Moreover, if a disconnected agent has the associated patients, we have to reassign them to another doctor. Hence, similarly with a $AgentLeaving$, the event $ReassignDoctor$ is decomposed into two events $NormalReassignDoctor$ and $DetectFailedAgent$.

4 Ensuring Correctness of Cooperative Agent Actions

4.1 Modelling emergency calls

Our next refinement step introduces abstract modelling of the emergency calls, that are generated by patient monitoring equipment. We must ensure that each call will be properly handled by a corresponding doctor.

We introduce the variable $emergency\_calls$, which is defined as a partial function associating the emergency calls with the patients. Moreover, we define the variable $accepted\_calls$ that establishes the correspondence between the emergency calls and the doctors that answer them.

At this refinement step we abstract away from the actual implementation of how a doctor is chosen to handle an emergency call. A detailed model of it will be introduced at the next refinement step. Here we add new events *EmergencyCall* and *HandlingEmergencyCall* to abstractly model occurrence of an emergency and finding a responsible doctor to handle it. In addition, we distinguish two types of patient visit – a regular visit (scheduled examination or delivery of a medicine) and a visit for handling an emergency call. To model it we decompose the event *VisitBegin* into the events *RegularVisitBegin* and *EmergencyVisitBegin*.

We define a system variant to ensure that the newly introduced events *EmergencyCall* and *HandlingEmergencyCall* do not take the control forever. We define the variant as follows:

$$card(ALARMS \setminus dom(emergency\_calls)) + card(ALARMS \setminus dom(accepted\_calls)),$$

and prove that it is decreased by new events. An extract from the machine $Hospital3$ is shown in Fig. 6.

4.2 Introducing a procedure to select a doctor in emergencies

In the previous refinement step we have introduced modelling of emergency calls and non-deterministic assignment of the responsible doctors to handle them. The goal of our next refinement step is to introduce a detailed procedure of selecting

Machine $Hospital3$ **Refines** $Hospital2$
Variables ... $emergency\_calls, accepted\_calls$
Invariants
$inv_1 : emergency\_calls \in ALARMS \nrightarrow patients$
$inv_2 : accepted\_calls \in ALARMS \nrightarrow med\_agents$
$inv_3 : dom(accepted\_calls) \subseteq dom(emergency\_calls)$
Events
...
$EmergencyCall \mathrel{\hat=}$
Status $convergent$
any pa, ec
when
$pa \in patients \land ec \in ALARMS \land ec \notin emergency\_calls \land pa \notin ran(emergency\_calls)$
then
$emergency\_calls := emergency\_calls \cup \{ec \mapsto pa\}$
end
$HandlingEmergencyCall \mathrel{\hat=}$
Status $convergent$
any ec, ma
when
$ec \in emergency\_calls \land ec \notin ran(accepted\_calls) \land$
$ma \in med\_agent \land ma \notin disconnected$
then
$accepted\_calls := accepted\_calls \cup \{ec \mapsto ma\}$
end
...

Figure 6: Hospital: the third refinement step

a doctor in case of an emergency call. It follows the steps graphically depicted in Fig. 7.

The proposed procedure can be described as follows. We start by selecting an emergency call to answer. Then we model a loop of finding a suitable candidate and sending a request to him/her. If the doctor rejects it then we choose the next candidate. The procedure is repeated until we get an acceptance on the request.

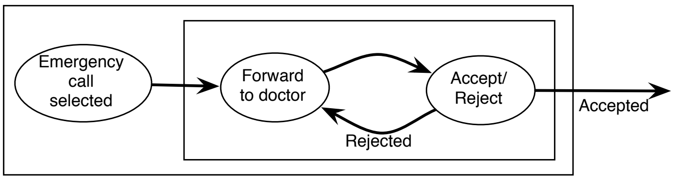

Figure 7: Procedure of choosing a doctor for a certain call

To model the described procedure, we refine the machine $Hospital3$ by introducing a number of new variables and events. The event $ChooseCurrentCall$ starts handling of a particular emergency call. The event $CallFeed$ directs the call to the assigned doctor, while $ForwardCall$ forwards the call to next suitable candidate. The events $AcceptCall$ and $RejectCall$ model acceptance and rejection respectively. A special event $ForcedAcceptCall$ is needed to “force“ the last available doctor to accept the call. To make this decision, the variable *occupied* is used to accumulate the doctors that have already refused the call (for example, a doctor is performing a surgery etc.).

We assume that the whole procedure finding a doctor for a certain emergency call takes a short period of time and during this period no disconnection of agents can occur. As a result, we strengthen the guards in the event $DisconnectAgent$ to disallow any disconnection while an emergency call is handled. An extract from the machine $Hospital4$ is shown in Fig. 8.

Moreover, we define an additional system variant to ensure that event *Reject-Call* is convergent, which means that eventually we get an acceptance from a doctor to answer a call. The variant is $card(med\_agents \setminus occupied)$ and it is decreased by the event.

Machine $Hospital4$ **Refines** $Hospital3$
Variables ... $ec\_handling, directed, candidate\_found, occupied, current\_call$
Invariants
  $inv_1 : ec\_handling \in BOOL$
  $inv_2 : candidate\_found \in BOOL$
  $inv_3 : directed \in ALARMS \twoheadrightarrow med\_agents$
  $inv_4 : occupied \subseteq med\_agents$
  $inv_5 : dom(directed) \subseteq dom(emergency\_calls)$
  $inv_6 : current\_call \in ALARMS$
  ...
Events
  ...
  $CallFeed \mathrel{\widehat{=}}$
    when
      $ec\_handling = TRUE \wedge candidate\_found = FALSE \wedge$
      $assigned\_doctor(emergency\_calls(current\_call)) \notin disconnected \wedge$
      $assigned\_doctor(emergency\_calls(current\_call)) \notin occupied$
    then
      $directed(current\_call) := assigned\_doctor(emergency\_calls(current\_call))$
      $candidate\_found := TRUE$
    end
  $AcceptCall \mathrel{\widehat{=}}$
    Refines $HandlingEmergencyCall$
    when
      $ec\_handling = TRUE \wedge candidate\_found = TRUE \wedge$
      $current\_call \in dom(emergency\_calls) \wedge current\_call \notin dom(accepted\_calls) \wedge$
      $directed(current\_call) \notin disconnected$
    with
      ec: $ec = current\_call$
      ma: $ma = directed(current\_call)$
    then
      $accepted\_calls(current\_call) := directed(current\_call)$
      $ec\_handling := FALSE$
      $candidate\_found := FALSE$
      $occupied := \varnothing$
    end
  ...

Figure 8: Hospital: the fourth refinement step

5 Data integrity

To ensure that a patient gets the correct treatment, we should guarantee that the medical personnel always access the most recent patient record. As we discussed in Section 3, we allow the doctor to access and modify the patient's data only when he/she is in a close proximity to a patient. We implement this requirement via the scoping mechanism [7, 8, 9]. A scope provide a shared data space for a doctor and a patient. We assume that each patient agent has the scope associated with it. As soon as a doctor agent appears at a close vicinity of the patient agent, it automatically joins the scope. While in the scope, the doctor can modify the patient record (e.g., prescribe a new medicine, log the information about the delivered medicine, prescribe a new procedure etc.).

To model this behaviour, we refine the abstract events *RegularVisitBegin*, *EmergencyVisitBegin*, *VisitEnd* by the events *RegularEnterScope*, *EmergencyEnterScope*, *LeaveScope* and add a new event *ModifyRecord*. The event *ModifyRecord* models an update of the patient record by a doctor, when he/she is in the scope of a patient. Thereby we ensure here that the patient record are always up-to-date. The corresponding safety property stating that the medical personnel always access the most recent record is formulated as the invariant (inv_7, Fig. 9).

Machine $Hospital5$ **Refines** $Hospital4$
Variables ... $record, ma\_data, scopes$
Invariants
$inv_1 : record \in patients \rightarrow \mathbb{P}(DATA)$
$inv_2 : ma\_data \in med\_agents \nrightarrow \mathbb{P}(DATA)$
$inv_3 : scopes \in ScopeName \rightarrowtail\!\!\!\!\!\nrightarrow med\_agents$
$inv_4 : \forall ma \cdot ma \in ran(scopes) \Leftrightarrow ma \in dom(ma\_data)$
$inv_5 : \forall ma \cdot ma \in disconnected \Rightarrow ma \notin ran(scopes)$
$inv_6 : \forall ma \cdot ma \in ran(visited \lhd last\_visit) \Rightarrow ma \in ran(scopes)$
$inv_7 : \forall ma, pa \cdot (pa \mapsto ma) \in (visited \lhd last\_visit) \Rightarrow ma\_data(ma) = record(pa)$
$inv_8 : \forall pa \cdot pa \in visited \Rightarrow last\_visit(pa) \in ran(scopes)$
$inv_9 : (visited \lhd last\_visit) \in patients \rightarrowtail\!\!\!\!\!\nrightarrow med\_agents$
Events
...
$EmergencyEnterScope \;\widehat{=}$
Refines $EmergencyVisitBegin$
any ec, sn
when
$ec \in dom(accepted\_calls) \land emergency\_calls(ec) \notin visited \land$
$accepted\_calls(ec) \notin last\_visit[visited] \land accepted\_calls(ec) \notin disconnected \land$
$sn \in ScopeName \land sn \notin dom(scopes) \land$
$accepted\_calls(ec) \notin ran(scopes) \land accepted\_calls(ec) \notin disconnected$
then
$last\_visit(emergency\_calls(ec)) := accepted\_calls(ec)$
$visited := visited \cup \{emergency\_calls(ec)\}$
$emergency\_calls := emergency\_calls \setminus \{ec \mapsto emergency\_calls(ec)\}$
$accepted\_calls := accepted\_calls \setminus \{ec \mapsto accepted\_calls(ec)\}$
$directed := \{ec\} \lhd\!\!\!- directed$
$scopes(sn) := accepted\_calls(ec)$
$ma\_data(accepted\_calls(ec)) := record(emergency\_calls(ec))$
end
$ModifyRecord \;\widehat{=}$
any ma, sn, pa, da_new
when
$(sn \mapsto ma) \in scopes \land pa \in dom(last\_visit) \land last\_visit(pa) = ma \land$
$pa \in visited \land da\_new \in \mathbb{P}(DATA) \land da\_new \neq \varnothing$
then
$ma\_data(ma) := da\_new$
$record(pa) := da\_new$
end
...

Figure 9: Hospital: the fifth refinement step

Moreover, we introduce the variable $record$ that represents the medical history for every patient. The variable ma_data stores the data that appear on the doctor's PDA screen. When the doctor agent is in a close vicinity of a patient, its ma_data becomes equal to the value of the patient data. Finally, we define the variable $scopes$, which is defined as a partial function associating the active scopes with the doctors participating in them. An extract from the machine $Hospital4$ is shown in Fig. 9.

6 Conclusion

In this paper we have presented a formal development of a hospital MAS. We have focused on modelling and verification of safety for two central safety-critical activities – handling emergencies and consistent update of patient data. Ensuring correctness of these activities is especially challenging due to highly dynamic nature of a hospital, volatile error-prone communication environment and autonomous agent behaviour.

In our development we have explicitly modelled the fault tolerance mechanism that ensures correct system functioning in the presence of agent disconnections. We have verified by proofs the correctness and safety of these two activities. Formal verification process has not only allowed us to systematically capture complex requirements but also facilitated derivation of the constraints that should be imposed on the system to guarantee its safety. Indeed, while proving convergence of the emergency handling procedure, we had to explicitly state the assumptions that the system must fulfil. These assumptions can be seen as a contract that should be checked during system deployment to guarantee its safety. In our development we have also demonstrated that the scoping mechanism provides a useful abstraction for ensuring consistent update of the patient data.

The work presented in this paper is inspired by our previous work on modelling context-aware mobile agent systems [7, 8, 9] in the CAMA framework [6, 5]. Similarly to [7, 8, 9], we rely on the timeout mechanism to tolerate agent disconnections and employ the scoping mechanism to provide shared data space for patient and doctor agents. However, in this paper we have focused on modelling and verification of safety properties of complex agent interactions rather than on reasoning about general mechanisms for agent interaction with middleware.

Formal modelling of MAS has been undertaken by [14, 13, 15]. The authors have proposed an extension of the Unity framework to explicitly define such concepts as mobility and context-awareness. In our approach we also have studied the problem of ensuring access to the fresh context. However, in [14] it is solved at the level of the matching agent attributes while in our approach we rely on the scoping mechanism to achieve this.

A formal modelling of MAS for the health care in Z has be undertaken by

Gruer at al. [4]. The work has focused on specifying a multi-agent system for a medical help system. The authors aimed at studying how to formally represent agent interactions, e.g., during negotiations. In our approach we not only model the agent interactions but also formally prove their properties. Hence, our approach is especially suitable for developing critical MAS systems.

In our future work we are planning to further investigate how to model adaptive agent behaviour that depends on the surrounding context as well as explore different reconfiguration mechanisms.

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

[4] P. Gruer, V. Hilaire, A. Koukam, and K. Cetnarowicz. A formal framework for multi-agent systems analysis and design. In *Expert Systems with Applications*, volume 23, pages 349–355, 2002.

[5] A. Iliasov, V. Khomenko, M. Koutny, and A. Romanovsky. On Specification and Verification of Location-based Fault Tolerant Mobile Systems. In A. Romanovsky M. Butler, C. Jones and E. Troubitsyna, editors, *Rigorous Development of Complex Fault-Tolerant Systems*. Springer.

[6] A. Iliasov and A. Romanovsky. CAMA: Structured Coordination Space and Exception Propagation Mechanism for Mobile Agents. In *ECOOP 2005, Workshop on Exception Handling in Object Oriented Systems: Developing Systems that Handle Exceptions*, 2005.

[7] L. Laibinis, E. Troubitsyna, A. Iliasov, and A. Romanovsky. Rigorous Development of Fault-Tolerant Agent Systems. In A. Romanovsky M. Butler, C. Jones and E. Troubitsyna, editors, *Rigorous Development of Complex Fault-Tolerant Systems*, volume 4157 of *LNCS*, pages 241–260. Springer, 2006.

[8] L. Laibinis, E. Troubitsyna, A. Iliasov, and A. Romanovsky. Formal Development of Cooperative Exception Handling for Mobile Agent Systems. In *SERENE 2008, International Workshop on Software Engineering for Resilient Systems*, 2008.

[9] L. Laibinis, E. Troubitsyna, A. Iliasov, and A. Romanovsky. Fault Tolerant Middleware for Agent Systems: A Refinement Approach. In *EWDC 2009, European Workshop on Dependable Computing*, 2009.

[10] OMG Mobile Agents Facility (MASIF). Available at www.omg.org.

[11] Rigorous Open Development Environment for Complex Systems (RODIN). Deliverable D7, Event-B Language, online at http://rodin.cs.ncl.ac.uk/.

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

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

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

[15] 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.

Appendix

MACHINE Hospital
SEES c0
VARIABLES
    med_agents
INVARIANTS
    inv1 : $med\_agents \subseteq MEDSTAFF$
EVENTS
Initialisation
    begin
        act1 : $med\_agents := \varnothing$
    end
Event *Activate* $\widehat{=}$
    any
        ma
    where
        grd1 : $ma \in MEDSTAFF$
        grd2 : $ma \notin med\_agents$
    then
        act1 : $med\_agents := med\_agents \cup \{ma\}$
    end
Event *Activity* $\widehat{=}$
    begin
        skip
    end
Event *Deactivate* $\widehat{=}$
    any
        ma
    where
        grd1 : $ma \in med\_agents$
    then
        act1 : $med\_agents := med\_agents \setminus \{ma\}$
    end
END

CONTEXT c0

SETS

`MEDSTAFF`

AXIOMS

`axm1` : $MEDSTAFF \neq \varnothing$

`axm2` : $finite(MEDSTAFF)$

END

MACHINE Hospital1
REFINES Hospital
SEES c1
VARIABLES

med_agents
patients
assigned_doctor
last_visit
visited

INVARIANTS

inv1 : $patients \subseteq PATIENTS$
inv2 : $assigned\_doctor \in patients \rightarrow med\_agents$
inv3 : $last\_visit \in patients \leftrightarrow MEDSTAFF$
inv4 : $visited \subseteq patients$
inv5 : $last\_visit[visited] \subseteq med\_agents$
inv6 : $visited \subseteq dom(last\_visit)$

EVENTS

Initialisation

extended

begin

act1 : med_agents $:= \varnothing$
act2 : $patients := \varnothing$
act3 : $assigned\_doctor := \varnothing$
act4 : $last\_visit := \varnothing$
act5 : $visited := \varnothing$

end

Event *ActivateAgent* $\widehat{=}$
extends *Activate*

any

ma

where

grd1 : ma $\in$ MEDSTAFF
grd2 : ma $\notin$ med_agents

then

act1 : med_agents $:=$ med_agents $\cup$ {ma}

end

Event *PatientArrival* $\widehat{=}$

any

pa

ma

where

grd1 : $pa \in PATIENTS$

grd2 : $pa \notin patients$

grd3 : $ma \in med\_agents$

then

act1 : $patients := patients \cup \{pa\}$

act2 : $assigned\_doctor(pa) := ma$

end

Event *PatientDischarge* $\widehat{=}$

any

pa

where

grd1 : $pa \in patients$

grd2 : $pa \notin visited$

then

act1 : $patients := patients \setminus \{pa\}$

act2 : $assigned\_doctor := \{pa\} \lhd assigned\_doctor$

act3 : $last\_visit := \{pa\} \lhd last\_visit$

end

Event *VisitBegin* $\widehat{=}$

extends *Activity*

any

ma

pa

where

grd1 : $ma \in med\_agents$

grd2 : $pa \in patients$

grd3 : $pa \notin visited$

grd4 : $ma \notin last\_visit[visited]$

then

act1 : $last\_visit(pa) := ma$

act2 : $visited := visited \cup \{pa\}$

end

Event *VisitEnd* $\widehat{=}$

extends *Activity*

any

pa

where

grd1 : $pa \in visited$

then

act1 : $visited := visited \setminus \{pa\}$

end

Event *AgentLeaving* $\widehat{=}$

extends *Deactivate*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$

grd2 : $ma \notin ran(assigned\_doctor)$

grd3 : $ma \notin last\_visit[visited]$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$

end

Event *ReassignDoctor* $\widehat{=}$

refines *Deactivate*

any

ma

ma_new

where

grd1 : $ma \in ran(assigned\_doctor)$

grd2 : $ma \notin last\_visit[visited]$

grd3 : $ma\_new \in med\_agents$

grd4 : $ma \neq ma\_new$

then

act1 : $med\_agents := med\_agents \setminus \{ma\}$

act2 : $assigned\_doctor := assigned\_doctor \lhd\!\!\!- (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$

end

END

CONTEXT c1

EXTENDS c0

SETS

PATIENTS

AXIOMS

axm1 : $finite(PATIENTS)$

END

MACHINE Hospital2
REFINES Hospital1
SEES c2
VARIABLES

med_agents
patients
assigned_doctor
disconnected
timer
last_visit
visited

INVARIANTS

inv1 : $disconnected \subseteq med\_agents$
inv2 : $timer \in med\_agents \rightarrow STATE$
inv4 : $\forall ma \cdot (ma \in med\_agents \wedge timer(ma) \neq inactive \Leftrightarrow ma \in disconnected)$

EVENTS

Initialisation

extended

begin

act1 : med_agents := $\varnothing$
act2 : patients := $\varnothing$
act3 : assigned_doctor := $\varnothing$
act4 : last_visit := $\varnothing$
act5 : visited := $\varnothing$
act6 : $disconnected := \varnothing$
act7 : $timer := \varnothing$

end

Event *ActivateAgent* $\widehat{=}$
extends *ActivateAgent*

any

ma

where

grd1 : ma $\in$ MEDSTAFF
grd2 : ma $\notin$ med_agents

then

act1 : med_agents := med_agents $\cup$ {ma}
act2 : $timer(ma) := inactive$

end

Event *PatientArrival* $\widehat{=}$

extends *PatientArrival*

any

pa

ma

where

grd1 : $pa \in PATIENTS$

grd2 : $pa \notin patients$

grd3 : $ma \in med\_agents$

then

act1 : $patients := patients \cup \{pa\}$

act2 : $assigned\_doctor(pa) := ma$

end

Event *PatientDischarge* $\widehat{=}$

extends *PatientDischarge*

any

pa

where

grd1 : $pa \in patients$

grd2 : $pa \notin visited$

then

act1 : $patients := patients \setminus \{pa\}$

act2 : $assigned\_doctor := \{pa\} \lhd\!\!\!- assigned\_doctor$

act3 : $last\_visit := \{pa\} \lhd\!\!\!- last\_visit$

end

Event *VisitBegin* $\widehat{=}$

extends *VisitBegin*

any

ma

pa

where

grd1 : $ma \in med\_agents$

grd2 : $pa \in patients$

grd3 : $pa \notin visited$

grd4 : $ma \notin last\_visit[visited]$

then

act1 : $last\_visit(pa) := ma$

act2 : $visited := visited \cup \{pa\}$

end

Event *VisitEnd* $\widehat{=}$

extends *VisitEnd*

any

pa

where

grd1 : $\texttt{pa} \in \texttt{visited}$

then

act1 : $\texttt{visited} := \texttt{visited} \setminus \{\texttt{pa}\}$

end

Event *DisconnectAgent* $\widehat{=}$

any

ma

where

grd1 : $ma \in med\_agents$

grd2 : $ma \notin disconnected$

then

act1 : $disconnected := disconnected \cup \{ma\}$

act2 : $timer(ma) := active$

end

Event *ReconectionFailed* $\widehat{=}$

any

ma

where

grd1 : $ma \in disconnected$

grd2 : $timer(ma) = active$

then

act1 : $timer(ma) := timeout$

end

Event *ReconnectionSuccessful* $\widehat{=}$

any

ma

where

grd1 : $ma \in disconnected$

grd2 : $timer(ma) = active$

then

act1 : $timer(ma) := inactive$

act2 : $disconnected := disconnected \setminus \{ma\}$

end

Event *NormalAgentLeaving* $\widehat{=}$

extends *AgentLeaving*

any

ma

where

grd1 : $\texttt{ma} \in \texttt{med\_agents}$

$\mathtt{grd2}:\ \mathtt{ma} \notin \mathtt{ran}(\mathtt{assigned\_doctor})$
$\mathtt{grd3}:\ \mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
$\mathtt{grd4}:\ ma \notin disconnected$

then

$\mathtt{act1}:\ \mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
$\mathtt{act2}:\ timer := \{ma\} \lhd timer$

end

Event *NormalReassignDoctor* $\widehat{=}$

extends *ReassignDoctor*

any

ma
ma_new

where

$\mathtt{grd1}:\ \mathtt{ma} \in \mathtt{ran}(\mathtt{assigned\_doctor})$
$\mathtt{grd2}:\ \mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
$\mathtt{grd3}:\ \mathtt{ma\_new} \in \mathtt{med\_agents}$
$\mathtt{grd4}:\ \mathtt{ma} \neq \mathtt{ma\_new}$
$\mathtt{grd5}:\ ma \notin disconnected$

then

$\mathtt{act1}:\ \mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
$\mathtt{act2}:\ \mathtt{assigned\_doctor} := \mathtt{assigned\_doctor} \Leftarrow\!\!\!+ (\mathtt{dom}(\mathtt{assigned\_doctor} \rhd \{\mathtt{ma}\}) \times \{\mathtt{ma\_new}\})$
$\mathtt{act3}:\ timer := \{ma\} \lhd timer$

end

Event *DetectFailedFreeAgent* $\widehat{=}$

extends *AgentLeaving*

any

ma

where

$\mathtt{grd1}:\ \mathtt{ma} \in \mathtt{med\_agents}$
$\mathtt{grd2}:\ \mathtt{ma} \notin \mathtt{ran}(\mathtt{assigned\_doctor})$
$\mathtt{grd3}:\ \mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
$\mathtt{grd4}:\ ma \in disconnected$
$\mathtt{grd5}:\ timer(ma) = timeout$

then

$\mathtt{act1}:\ \mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
$\mathtt{act2}:\ disconnected := disconnected \setminus \{ma\}$
$\mathtt{act3}:\ timer := \{ma\} \lhd timer$

end

Event *DetectFailedAgent* $\widehat{=}$

extends *ReassignDoctor*

any

ma

ma_new

where

grd1 : $ma \in ran(assigned\_doctor)$

grd2 : $ma \notin last\_visit[visited]$

grd3 : $ma\_new \in med\_agents$

grd4 : $ma \neq ma\_new$

grd5 : $ma \in disconnected$

grd6 : $timer(ma) = timeout$

grd7 : $ma\_new \notin disconnected \lor (ma\_new \in disconnected \land timer(ma\_new) = active)$

then

act1 : $med\_agents := med\_agents \setminus \{ma\}$

act2 : $assigned\_doctor := assigned\_doctor \Leftarrow\!\!\!+ (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$

act3 : $disconnected := disconnected \setminus \{ma\}$

act4 : $timer := \{ma\} \lhd timer$

end

END

CONTEXT c2

EXTENDS c1

SETS

STATE

CONSTANTS

inactive

active

timeout

AXIOMS

axm1 : $partition(STATE, \{inactive\}, \{active\}, \{timeout\})$

END

MACHINE Hospital3

REFINES Hospital2

SEES c3

VARIABLES

assigned_doctor

disconnected

med_agents

patients

timer

emergency_calls

accepted_calls

last_visit

visited

INVARIANTS

inv1 : $emergency\_calls \in ALARMS \nrightarrow patients$

inv2 : $accepted\_calls \in ALARMS \nrightarrow med\_agents$

inv3 : $dom(accepted\_calls) \subseteq dom(emergency\_calls)$

EVENTS

Initialisation

extended

begin

act1 : med_agents := ∅

act2 : patients := ∅

act3 : assigned_doctor := ∅

act4 : last_visit := ∅

act5 : visited := ∅

act6 : disconnected := ∅

act7 : timer := ∅

act8 : $emergency\_calls := \varnothing$

act9 : $accepted\_calls := \varnothing$

end

Event *ActivateAgent* $\widehat{=}$

extends *ActivateAgent*

any

ma

where

grd1 : ma ∈ MEDSTAFF

grd2 : ma ∉ med_agents

then

act1 : med_agents := med_agents ∪ {ma}
act2 : timer(ma) := inactive
end
Event *PatientArrival* $\widehat{=}$
extends *PatientArrival*
any
pa
ma
where
grd1 : pa ∈ PATIENTS
grd2 : pa ∉ patients
grd3 : ma ∈ med_agents
then
act1 : patients := patients ∪ {pa}
act2 : assigned_doctor(pa) := ma
end
Event *PatientDischarge* $\widehat{=}$
extends *PatientDischarge*
any
pa
where
grd1 : pa ∈ patients
grd2 : pa ∉ visited
grd3 : $pa \notin ran(emergency\_calls)$
then
act1 : patients := patients \ {pa}
act2 : assigned_doctor := {pa} ⩤ assigned_doctor
act3 : last_visit := {pa} ⩤ last_visit
end
Event *DisconnectAgent* $\widehat{=}$
extends *DisconnectAgent*
any
ma
where
grd1 : ma ∈ med_agents
grd2 : ma ∉ disconnected
then
act1 : disconnected := disconnected ∪ {ma}
act2 : timer(ma) := active
end
Event *ReconnectionSuccessful* $\widehat{=}$

extends *ReconnectionSuccessful*

any

ma

where

grd1 : $\texttt{ma} \in \texttt{disconnected}$

grd2 : $\texttt{timer(ma)} = \texttt{active}$

then

act1 : $\texttt{timer(ma)} := \texttt{inactive}$

act2 : $\texttt{disconnected} := \texttt{disconnected} \setminus \{\texttt{ma}\}$

end

Event *ReconectionFailed* $\widehat{=}$

extends *ReconectionFailed*

any

ma

where

grd1 : $\texttt{ma} \in \texttt{disconnected}$

grd2 : $\texttt{timer(ma)} = \texttt{active}$

then

act1 : $\texttt{timer(ma)} := \texttt{timeout}$

end

Event *NormalAgentLeaving* $\widehat{=}$

extends *NormalAgentLeaving*

any

ma

where

grd1 : $\texttt{ma} \in \texttt{med\_agents}$

grd2 : $\texttt{ma} \notin \texttt{ran(assigned\_doctor)}$

grd3 : $\texttt{ma} \notin \texttt{last\_visit[visited]}$

grd4 : $\texttt{ma} \notin \texttt{disconnected}$

grd5 : $ma \notin ran(accepted\_calls)$

then

act1 : $\texttt{med\_agents} := \texttt{med\_agents} \setminus \{\texttt{ma}\}$

act2 : $\texttt{timer} := \{\texttt{ma}\} \lhd \texttt{timer}$

end

Event *NormalReassignDoctor* $\widehat{=}$

extends *NormalReassignDoctor*

any

ma

ma_new

where

grd1 : $\texttt{ma} \in \texttt{ran(assigned\_doctor)}$

grd2 : $\mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
grd3 : $\mathtt{ma\_new} \in \mathtt{med\_agents}$
grd4 : $\mathtt{ma} \neq \mathtt{ma\_new}$
grd5 : $\mathtt{ma} \notin \mathtt{disconnected}$
grd6 : $ma \notin ran(accepted\_calls)$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
act2 : $\mathtt{assigned\_doctor} := \mathtt{assigned\_doctor} \Leftarrow\!\!+ (\mathtt{dom}(\mathtt{assigned\_doctor} \rhd \{\mathtt{ma}\}) \times \{\mathtt{ma\_new}\})$
act3 : $\mathtt{timer} := \{\mathtt{ma}\} \lhd \mathtt{timer}$

end

Event *DetectFailedAgent* $\widehat{=}$
extends *DetectFailedAgent*

any

ma
ma_new

where

grd1 : $\mathtt{ma} \in \mathtt{ran}(\mathtt{assigned\_doctor})$
grd2 : $\mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
grd3 : $\mathtt{ma\_new} \in \mathtt{med\_agents}$
grd4 : $\mathtt{ma} \neq \mathtt{ma\_new}$
grd5 : $\mathtt{ma} \in \mathtt{disconnected}$
grd6 : $\mathtt{timer}(\mathtt{ma}) = \mathtt{timeout}$
grd7 : $\mathtt{ma\_new} \notin \mathtt{disconnected} \lor (\mathtt{ma\_new} \in \mathtt{disconnected} \land \mathtt{timer}(\mathtt{ma\_new}) = \mathtt{active})$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
act2 : $\mathtt{assigned\_doctor} := \mathtt{assigned\_doctor} \Leftarrow\!\!+ (\mathtt{dom}(\mathtt{assigned\_doctor} \rhd \{\mathtt{ma}\}) \times \{\mathtt{ma\_new}\})$
act3 : $\mathtt{disconnected} := \mathtt{disconnected} \setminus \{\mathtt{ma}\}$
act4 : $\mathtt{timer} := \{\mathtt{ma}\} \lhd \mathtt{timer}$
act5 : $accepted\_calls := accepted\_calls \rhd \{ma\}$

end

Event *DetectFailedFreeAgent* $\widehat{=}$
extends *DetectFailedFreeAgent*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$
grd2 : $\mathtt{ma} \notin \mathtt{ran}(\mathtt{assigned\_doctor})$
grd3 : $\mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$
grd4 : $\mathtt{ma} \in \mathtt{disconnected}$

$\texttt{grd5}: \ \mathtt{timer(ma) = timeout}$

then

$\texttt{act1}: \ \mathtt{med\_agents := med\_agents \setminus \{ma\}}$

$\texttt{act2}: \ \mathtt{disconnected := disconnected \setminus \{ma\}}$

$\texttt{act3}: \ \mathtt{timer := \{ma\} \lhd timer}$

$\texttt{act4}: \ accepted\_calls := accepted\_calls \rhd \{ma\}$

end

Event *EmergencyCall* $\widehat{=}$

Status convergent

any

pa

ec

where

$\texttt{grd1}: \ pa \in patients$

$\texttt{grd2}: \ ec \in ALARMS$

$\texttt{grd3}: \ ec \notin dom(emergency\_calls)$

$\texttt{grd4}: \ pa \notin ran(emergency\_calls)$

then

$\texttt{act1}: \ emergency\_calls := emergency\_calls \cup \{ec \mapsto pa\}$

end

Event *HandlingEmergencyCall* $\widehat{=}$

Status convergent

any

ec

ma

where

$\texttt{grd1}: \ ec \in dom(emergency\_calls)$

$\texttt{grd2}: \ ec \notin dom(accepted\_calls)$

$\texttt{grd3}: \ ma \in med\_agents$

$\texttt{grd4}: \ ma \notin disconnected$

then

$\texttt{act1}: \ accepted\_calls := accepted\_calls \cup \{ec \mapsto ma\}$

end

Event *EmergencyVisitBegin* $\widehat{=}$

refines *VisitBegin*

any

ec

where

$\texttt{grd1}: \ ec \in dom(accepted\_calls)$

$\texttt{grd2}: \ emergency\_calls(ec) \notin visited$

$\texttt{grd3}: \ accepted\_calls(ec) \notin last\_visit[visited]$

with

ma : $\texttt{ma} = \texttt{accepted\_calls}(\texttt{ec})$

pa : $\texttt{pa} = \texttt{emergency\_calls}(\texttt{ec})$

then

act1 : $last\_visit(emergency\_calls(ec)) := accepted\_calls(ec)$

act2 : $visited := visited \cup \{emergency\_calls(ec)\}$

act3 : $emergency\_calls := emergency\_calls \setminus \{ec \mapsto emergency\_calls(ec)\}$

act4 : $accepted\_calls := accepted\_calls \setminus \{ec \mapsto accepted\_calls(ec)\}$

end

Event *RegularVisitBegin* $\widehat{=}$

extends *VisitBegin*

any

ma

pa

where

grd1 : $\texttt{ma} \in \texttt{med\_agents}$

grd2 : $\texttt{pa} \in \texttt{patients}$

grd3 : $\texttt{pa} \notin \texttt{visited}$

grd4 : $\texttt{ma} \notin \texttt{last\_visit}[\texttt{visited}]$

grd5 : $pa \notin ran(emergency\_calls)$

then

act1 : $\texttt{last\_visit}(\texttt{pa}) := \texttt{ma}$

act2 : $\texttt{visited} := \texttt{visited} \cup \{\texttt{pa}\}$

end

Event *VisitEnd* $\widehat{=}$

extends *VisitEnd*

any

pa

where

grd1 : $\texttt{pa} \in \texttt{visited}$

then

act1 : $\texttt{visited} := \texttt{visited} \setminus \{\texttt{pa}\}$

end

VARIANT

$\texttt{card}(\texttt{ALARMS} \setminus \texttt{dom}(\texttt{emergency\_calls})) + \texttt{card}(\texttt{ALARMS} \setminus \texttt{dom}(\texttt{accepted\_calls}))$

END

CONTEXT c3
EXTENDS c2
SETS

`ALARMS`

AXIOMS

`axm1 :` $\mathit{finite}(ALARMS)$

END

MACHINE Hospital4
REFINES Hospital3
SEES c3
VARIABLES

assigned_doctor
accepted_calls
disconnected
emergency_calls
last_visit
med_agents
patients
timer
visited
ec_handling
directed
candidate_found
occupied
current_call

INVARIANTS

inv1 : $ec\_handling \in BOOL$
inv2 : $candidate\_found \in BOOL$
inv3 : $directed \in ALARMS \nrightarrow med\_agents$
inv4 : $occupied \subseteq med\_agents$
inv5 : $dom(directed) \subseteq dom(emergency\_calls)$
inv6 : $accepted\_calls \subseteq directed$
inv7 : $current\_call \in ALARMS$
inv8 : $ec\_handling = TRUE \Rightarrow current\_call \in dom(emergency\_calls)$
inv9 : $candidate\_found = TRUE \land ec\_handling = TRUE \Rightarrow current\_call \in dom(directed)$
inv10 : $ec\_handling = FALSE \Rightarrow candidate\_found = FALSE$
inv11 : $ec\_handling = TRUE \land candidate\_found = TRUE \Rightarrow directed(current\_call) \notin occupied$
inv12 : $ec\_handling = FALSE \Rightarrow occupied = \varnothing$
inv13 : $ec\_handling = TRUE \Rightarrow current\_call \notin dom(accepted\_calls)$

EVENTS

Initialisation

extended

begin

act1 : $\mathtt{med\_agents} := \varnothing$
act2 : $\mathtt{patients} := \varnothing$
act3 : $\mathtt{assigned\_doctor} := \varnothing$
act4 : $\mathtt{last\_visit} := \varnothing$
act5 : $\mathtt{visited} := \varnothing$
act6 : $\mathtt{disconnected} := \varnothing$
act7 : $\mathtt{timer} := \varnothing$
act8 : $\mathtt{emergency\_calls} := \varnothing$
act9 : $\mathtt{accepted\_calls} := \varnothing$
act10 : $ec\_handling := FALSE$
act11 : $candidate\_found := FALSE$
act12 : $directed := \varnothing$
act13 : $occupied := \varnothing$
act14 : $current\_call :\in ALARMS$

end

Event *ActivateAgent* $\widehat{=}$
extends *ActivateAgent*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{MEDSTAFF}$
grd2 : $\mathtt{ma} \notin \mathtt{med\_agents}$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \cup \{\mathtt{ma}\}$
act2 : $\mathtt{timer}(\mathtt{ma}) := \mathtt{inactive}$

end

Event *PatientArrival* $\widehat{=}$
extends *PatientArrival*

any

pa
ma

where

grd1 : $\mathtt{pa} \in \mathtt{PATIENTS}$
grd2 : $\mathtt{pa} \notin \mathtt{patients}$
grd3 : $\mathtt{ma} \in \mathtt{med\_agents}$

then

act1 : $\mathtt{patients} := \mathtt{patients} \cup \{\mathtt{pa}\}$
act2 : $\mathtt{assigned\_doctor}(\mathtt{pa}) := \mathtt{ma}$

end

Event *PatientDischarge* $\widehat{=}$

extends *PatientDischarge*

any

pa

where

grd1 : $\mathtt{pa} \in \mathtt{patients}$

grd2 : $\mathtt{pa} \notin \mathtt{visited}$

grd3 : $\mathtt{pa} \notin \mathtt{ran(emergency\_calls)}$

then

act1 : $\mathtt{patients} := \mathtt{patients} \setminus \{\mathtt{pa}\}$

act2 : $\mathtt{assigned\_doctor} := \{\mathtt{pa}\} \lhd \mathtt{assigned\_doctor}$

act3 : $\mathtt{last\_visit} := \{\mathtt{pa}\} \lhd \mathtt{last\_visit}$

end

Event *DisconnectAgent* $\widehat{=}$

extends *DisconnectAgent*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$

grd2 : $\mathtt{ma} \notin \mathtt{disconnected}$

grd3 : $ec\_handling = FALSE$

then

act1 : $\mathtt{disconnected} := \mathtt{disconnected} \cup \{\mathtt{ma}\}$

act2 : $\mathtt{timer(ma)} := \mathtt{active}$

end

Event *ReconnectionSuccessful* $\widehat{=}$

extends *ReconnectionSuccessful*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{disconnected}$

grd2 : $\mathtt{timer(ma)} = \mathtt{active}$

then

act1 : $\mathtt{timer(ma)} := \mathtt{inactive}$

act2 : $\mathtt{disconnected} := \mathtt{disconnected} \setminus \{\mathtt{ma}\}$

end

Event *ReconectionFailed* $\widehat{=}$

extends *ReconectionFailed*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{disconnected}$

grd2 : timer(ma) = active

then

act1 : timer(ma) := timeout

end

Event *NormalAgentLeaving* $\widehat{=}$

extends *NormalAgentLeaving*

any

ma

where

grd1 : ma $\in$ med_agents

grd2 : ma $\notin$ ran(assigned_doctor)

grd3 : ma $\notin$ last_visit[visited]

grd4 : ma $\notin$ disconnected

grd5 : ma $\notin$ ran(accepted_calls)

grd6 : $ma \notin ran(directed)$

then

act1 : med_agents := med_agents $\setminus$ {ma}

act2 : timer := {ma} $\lhd$ timer

act3 : $occupied := occupied \setminus \{ma\}$

end

Event *NormalReassignDoctor* $\widehat{=}$

extends *NormalReassignDoctor*

any

ma

ma_new

where

grd1 : ma $\in$ ran(assigned_doctor)

grd2 : ma $\notin$ last_visit[visited]

grd3 : ma_new $\in$ med_agents

grd4 : ma $\neq$ ma_new

grd5 : ma $\notin$ disconnected

grd6 : ma $\notin$ ran(accepted_calls)

grd7 : $ma \notin ran(directed)$

then

act1 : med_agents := med_agents $\setminus$ {ma}

act2 : assigned_doctor := assigned_doctor $\Leftarrow\!\!\!+$ (dom(assigned_doctor $\rhd$ {ma}) $\times$ {ma_new})

act3 : timer := {ma} $\lhd$ timer

act4 : $occupied := occupied \setminus \{ma\}$

end

Event *DetectFailedFreeAgent* $\widehat{=}$

extends *DetectFailedFreeAgent*

any

ma

where

grd1 : $ma \in med\_agents$

grd2 : $ma \notin ran(assigned\_doctor)$

grd3 : $ma \notin last\_visit[visited]$

grd4 : $ma \in disconnected$

grd5 : $timer(ma) = timeout$

grd6 : $ma \notin ran(directed)$

then

act1 : $med\_agents := med\_agents \setminus \{ma\}$

act2 : $disconnected := disconnected \setminus \{ma\}$

act3 : $timer := \{ma\} \lhd\!\!\!- timer$

act4 : $accepted\_calls := accepted\_calls \rhd\!\!\!- \{ma\}$

act5 : $occupied := occupied \setminus \{ma\}$

end

Event *DetectFailedAgent* $\widehat{=}$

extends *DetectFailedAgent*

any

ma

ma_new

where

grd1 : $ma \in ran(assigned\_doctor)$

grd2 : $ma \notin last\_visit[visited]$

grd3 : $ma\_new \in med\_agents$

grd4 : $ma \neq ma\_new$

grd5 : $ma \in disconnected$

grd6 : $timer(ma) = timeout$

grd7 : $ma\_new \notin disconnected \lor (ma\_new \in disconnected \land timer(ma\_new) = active)$

grd8 : $ma \notin ran(directed)$

then

act1 : $med\_agents := med\_agents \setminus \{ma\}$

act2 : $assigned\_doctor := assigned\_doctor \Leftarrow\!\!\!+ (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$

act3 : $disconnected := disconnected \setminus \{ma\}$

act4 : $timer := \{ma\} \lhd\!\!\!- timer$

act5 : $accepted\_calls := accepted\_calls \rhd\!\!\!- \{ma\}$

act6 : $occupied := occupied \setminus \{ma\}$

end

Event *EmergencyCall* $\widehat{=}$

extends *EmergencyCall*

any

pa

ec

where

grd1 : $\texttt{pa} \in \texttt{patients}$

grd2 : $\texttt{ec} \in \texttt{ALARMS}$

grd3 : $\texttt{ec} \notin \texttt{dom(emergency\_calls)}$

grd4 : $\texttt{pa} \notin \texttt{ran(emergency\_calls)}$

then

act1 : $\texttt{emergency\_calls} := \texttt{emergency\_calls} \cup \{\texttt{ec} \mapsto \texttt{pa}\}$

end

Event *ChooseCurrentCall* $\widehat{=}$

any

ec

where

grd1 : $ec \in dom(emergency\_calls)$

grd2 : $ec \notin dom(directed)$

grd3 : $ec\_handling = FALSE$

then

act1 : $ec\_handling := TRUE$

act2 : $current\_call := ec$

end

Event *CallFeed* $\widehat{=}$

when

grd1 : $ec\_handling = TRUE$

grd2 : $candidate\_found = FALSE$

grd3 : $assigned\_doctor(emergency\_calls(current\_call)) \notin disconnected$

grd4 : $assigned\_doctor(emergency\_calls(current\_call)) \notin occupied$

then

act1 : $directed(current\_call) := assigned\_doctor(emergency\_calls(current\_call))$

act2 : $candidate\_found := TRUE$

end

Event *ForwardCall* $\widehat{=}$

any

ma_new

where

grd1 : $ec\_handling = TRUE$

grd2 : $candidate\_found = FALSE$

grd3 : $assigned\_doctor(emergency\_calls(current\_call)) \in disconnected \vee$
$assigned\_doctor(emergency\_calls(current\_call)) \in occupied$

grd4 : $ma\_new \in med\_agents$
grd5 : $ma\_new \notin disconnected$
grd6 : $ma\_new \notin occupied$

then

act1 : $directed(current\_call) := ma\_new$
act2 : $candidate\_found := TRUE$

end

Event *AcceptCall* $\widehat{=}$

refines *HandlingEmergencyCall*

when

grd1 : $ec\_handling = TRUE$
grd2 : $candidate\_found = TRUE$
grd3 : $current\_call \in dom(emergency\_calls)$
grd4 : $current\_call \notin dom(accepted\_calls)$
grd5 : $directed(current\_call) \notin disconnected$

with

ec : $\mathtt{ec} = \mathtt{current\_call}$
ma : $\mathtt{ma} = \mathtt{directed(current\_call)}$

then

act1 : $accepted\_calls(current\_call) := directed(current\_call)$
act2 : $ec\_handling := FALSE$
act3 : $candidate\_found := FALSE$
act4 : $occupied := \varnothing$

end

Event *RejectCall* $\widehat{=}$

Status convergent

when

grd1 : $ec\_handling = TRUE$
grd2 : $candidate\_found = TRUE$
grd3 : $card(med\_agents \setminus occupied) \geq 2$

then

act1 : $occupied := occupied \cup \{directed(current\_call)\}$
act2 : $candidate\_found := FALSE$

end

Event *ForcedAcceptCall* $\widehat{=}$

refines *HandlingEmergencyCall*

when

grd1 : $ec\_handling = TRUE$
grd2 : $candidate\_found = TRUE$
grd3 : $current\_call \in dom(emergency\_calls)$
grd4 : $current\_call \notin dom(accepted\_calls)$

grd5 : $directed(current\_call) \notin disconnected$
grd6 : $card(med\_agents \setminus occupied) = 1$

with

ec : $\mathtt{ec} = \mathtt{current\_call}$
ma : $\mathtt{ma} = \mathtt{directed(current\_call)}$

then

act1 : $accepted\_calls(current\_call) := directed(current\_call)$
act2 : $ec\_handling := FALSE$
act3 : $candidate\_found := FALSE$
act4 : $occupied := \varnothing$

end

Event *EmergencyVisitBegin* $\widehat{=}$

extends *EmergencyVisitBegin*

any

ec

where

grd1 : $\mathtt{ec} \in \mathtt{dom(accepted\_calls)}$
grd2 : $\mathtt{emergency\_calls(ec)} \notin \mathtt{visited}$
grd3 : $\mathtt{accepted\_calls(ec)} \notin \mathtt{last\_visit[visited]}$
grd4 : $accepted\_calls(ec) \notin disconnected$

then

act1 : $\mathtt{last\_visit(emergency\_calls(ec))} := \mathtt{accepted\_calls(ec)}$
act2 : $\mathtt{visited} := \mathtt{visited} \cup \{\mathtt{emergency\_calls(ec)}\}$
act3 : $\mathtt{emergency\_calls} := \mathtt{emergency\_calls} \setminus \{\mathtt{ec} \mapsto \mathtt{emergency\_calls(ec)}\}$
act4 : $\mathtt{accepted\_calls} := \mathtt{accepted\_calls} \setminus \{\mathtt{ec} \mapsto \mathtt{accepted\_calls(ec)}\}$
act5 : $directed := \{ec\} \lhd directed$

end

Event *RegularVisitBegin* $\widehat{=}$

extends *RegularVisitBegin*

any

ma
pa

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$
grd2 : $\mathtt{pa} \in \mathtt{patients}$
grd3 : $\mathtt{pa} \notin \mathtt{visited}$
grd4 : $\mathtt{ma} \notin \mathtt{last\_visit[visited]}$
grd5 : $\mathtt{pa} \notin \mathtt{ran(emergency\_calls)}$

then

act1 : $\mathtt{last\_visit(pa)} := \mathtt{ma}$
act2 : $\mathtt{visited} := \mathtt{visited} \cup \{\mathtt{pa}\}$

end

Event *VisitEnd* $\widehat{=}$

extends *VisitEnd*

any

pa

where

grd1 : $\mathtt{pa} \in \mathtt{visited}$

then

act1 : $\mathtt{visited} := \mathtt{visited} \setminus \{\mathtt{pa}\}$

end

VARIANT

$\mathtt{card}(\mathtt{med\_agents} \setminus \mathtt{occupied})$

END

MACHINE Hospital5

REFINES Hospital4

SEES c4

VARIABLES

accepted_calls
assigned_doctor
directed
disconnected
emergency_calls
last_visit
med_agents
patients
timer
record
scopes
ec_handling
ma_data
visited
occupied
candidate_found
current_call

INVARIANTS

inv1 : $record \in patients \rightarrow \mathbb{P}(DATA)$

inv2 : $ma\_data \in med\_agents \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}(DATA)$

inv3 : $scopes \in ScopeName \rightarrowtail\!\!\!\!\!\!+ \; med\_agents$

inv4 : $\forall ma \cdot ma \in ran(scopes) \Leftrightarrow ma \in dom(ma\_data)$

inv5 : $\forall ma \cdot ma \in disconnected \Rightarrow ma \notin ran(scopes)$

inv6 : $\forall ma \cdot ma \in ran(visited \lhd last\_visit) \Rightarrow ma \in ran(scopes)$

inv7 : $\forall ma, pa \cdot (pa \mapsto ma) \in (visited \lhd last\_visit) \Rightarrow ma\_data(ma) = record(pa)$

inv8 : $\forall pa \cdot pa \in visited \Rightarrow last\_visit(pa) \in ran(scopes)$

inv9 : $(visited \lhd last\_visit) \in patients \rightarrowtail\!\!\!\!\!\!+ \; med\_agents$

EVENTS

Initialisation

extended

begin

act1 : med_agents := $\varnothing$

act2 : patients := ∅
act3 : assigned_doctor := ∅
act4 : last_visit := ∅
act5 : visited := ∅
act6 : disconnected := ∅
act7 : timer := ∅
act8 : emergency_calls := ∅
act9 : accepted_calls := ∅
act10 : ec_handling := FALSE
act11 : candidate_found := FALSE
act12 : directed := ∅
act13 : occupied := ∅
act14 : current_call :∈ ALARMS
act15 : *record* := ∅
act16 : *scopes* := ∅
act17 : *ma_data* := ∅

end

Event *ActivateAgent* $\widehat{=}$

extends *ActivateAgent*

any

ma

where

grd1 : ma ∈ MEDSTAFF
grd2 : ma ∉ med_agents

then

act1 : med_agents := med_agents ∪ {ma}
act2 : timer(ma) := inactive

end

Event *PatientArrival* $\widehat{=}$

extends *PatientArrival*

any

pa
ma
da

where

grd1 : pa ∈ PATIENTS
grd2 : pa ∉ patients
grd3 : ma ∈ med_agents
grd4 : *da* ⊆ *DATA*

then

act1 : patients := patients ∪ {pa}
act2 : assigned_doctor(pa) := ma

act3 : $record(pa) := da$

end

Event *PatientDischarge* $\widehat{=}$

extends *PatientDischarge*

any

pa

where

grd1 : $\mathtt{pa} \in \mathtt{patients}$

grd2 : $\mathtt{pa} \notin \mathtt{visited}$

grd3 : $\mathtt{pa} \notin \mathtt{ran(emergency\_calls)}$

then

act1 : $\mathtt{patients} := \mathtt{patients} \setminus \{\mathtt{pa}\}$

act2 : $\mathtt{assigned\_doctor} := \{\mathtt{pa}\} \lhd \mathtt{assigned\_doctor}$

act3 : $\mathtt{last\_visit} := \{\mathtt{pa}\} \lhd \mathtt{last\_visit}$

act4 : $record := \{pa\} \lhd record$

end

Event *DisconnectAgent* $\widehat{=}$

extends *DisconnectAgent*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$

grd2 : $\mathtt{ma} \notin \mathtt{disconnected}$

grd3 : $\mathtt{ec\_handling} = \mathtt{FALSE}$

grd4 : $ma \notin ran(scopes)$

then

act1 : $\mathtt{disconnected} := \mathtt{disconnected} \cup \{\mathtt{ma}\}$

act2 : $\mathtt{timer(ma)} := \mathtt{active}$

end

Event *ReconnectionSuccessful* $\widehat{=}$

extends *ReconnectionSuccessful*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{disconnected}$

grd2 : $\mathtt{timer(ma)} = \mathtt{active}$

then

act1 : $\mathtt{timer(ma)} := \mathtt{inactive}$

act2 : $\mathtt{disconnected} := \mathtt{disconnected} \setminus \{\mathtt{ma}\}$

end

Event *ReconectionFailed* $\widehat{=}$

extends *ReconectionFailed*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{disconnected}$

grd2 : $\mathtt{timer(ma) = active}$

then

act1 : $\mathtt{timer(ma)} := \mathtt{timeout}$

end

Event *NormalAgentLeaving* $\widehat{=}$

extends *NormalAgentLeaving*

any

ma

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$

grd2 : $\mathtt{ma} \notin \mathtt{ran(assigned\_doctor)}$

grd3 : $\mathtt{ma} \notin \mathtt{last\_visit[visited]}$

grd4 : $\mathtt{ma} \notin \mathtt{disconnected}$

grd5 : $\mathtt{ma} \notin \mathtt{ran(accepted\_calls)}$

grd6 : $\mathtt{ma} \notin \mathtt{ran(directed)}$

grd7 : $ma \notin ran(scopes)$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$

act2 : $\mathtt{timer} := \{\mathtt{ma}\} \lhd \mathtt{timer}$

act3 : $\mathtt{occupied} := \mathtt{occupied} \setminus \{\mathtt{ma}\}$

end

Event *NormalReassignDoctor* $\widehat{=}$

extends *NormalReassignDoctor*

any

ma

ma_new

where

grd1 : $\mathtt{ma} \in \mathtt{ran(assigned\_doctor)}$

grd2 : $\mathtt{ma} \notin \mathtt{last\_visit[visited]}$

grd3 : $\mathtt{ma\_new} \in \mathtt{med\_agents}$

grd4 : $\mathtt{ma} \neq \mathtt{ma\_new}$

grd5 : $\mathtt{ma} \notin \mathtt{disconnected}$

grd6 : $\mathtt{ma} \notin \mathtt{ran(accepted\_calls)}$

grd7 : $\mathtt{ma} \notin \mathtt{ran(directed)}$

grd8 : $ma \notin ran(scopes)$

then

$act1 : med\_agents := med\_agents \setminus \{ma\}$
$act2 : assigned\_doctor := assigned\_doctor \Leftarrow (dom(assigned\_doctor \rhd \{ma\}) \times \{ma\_new\})$
$act3 : timer := \{ma\} \lhd timer$
$act4 : occupied := occupied \setminus \{ma\}$

end

Event *DetectFailedFreeAgent* $\widehat{=}$

extends *DetectFailedFreeAgent*

any

ma

where

$grd1 : ma \in med\_agents$
$grd2 : ma \notin ran(assigned\_doctor)$
$grd3 : ma \notin last\_visit[visited]$
$grd4 : ma \in disconnected$
$grd5 : timer(ma) = timeout$
$grd6 : ma \notin ran(directed)$

then

$act1 : med\_agents := med\_agents \setminus \{ma\}$
$act2 : disconnected := disconnected \setminus \{ma\}$
$act3 : timer := \{ma\} \lhd timer$
$act4 : accepted\_calls := accepted\_calls \rhd \{ma\}$
$act5 : occupied := occupied \setminus \{ma\}$
$act6 : scopes := scopes \rhd \{ma\}$
$act7 : ma\_data := \{ma\} \lhd ma\_data$

end

Event *DetectFailedAgent* $\widehat{=}$

extends *DetectFailedAgent*

any

ma
ma_new

where

$grd1 : ma \in ran(assigned\_doctor)$
$grd2 : ma \notin last\_visit[visited]$
$grd3 : ma\_new \in med\_agents$
$grd4 : ma \neq ma\_new$
$grd5 : ma \in disconnected$
$grd6 : timer(ma) = timeout$
$grd7 : ma\_new \notin disconnected \lor (ma\_new \in disconnected \land timer(ma\_new) = active)$
$grd8 : ma \notin ran(directed)$

then

act1 : $\mathtt{med\_agents} := \mathtt{med\_agents} \setminus \{\mathtt{ma}\}$
act2 : $\mathtt{assigned\_doctor} := \mathtt{assigned\_doctor} \mathbin{\lhd\!\!-} (\mathrm{dom}(\mathtt{assigned\_doctor} \rhd \{\mathtt{ma}\}) \times \{\mathtt{ma\_new}\})$
act3 : $\mathtt{disconnected} := \mathtt{disconnected} \setminus \{\mathtt{ma}\}$
act4 : $\mathtt{timer} := \{\mathtt{ma}\} \lhd \mathtt{timer}$
act5 : $\mathtt{accepted\_calls} := \mathtt{accepted\_calls} \rhd \{\mathtt{ma}\}$
act6 : $\mathtt{occupied} := \mathtt{occupied} \setminus \{\mathtt{ma}\}$
act7 : $scopes := scopes \rhd \{ma\}$
act8 : $ma\_data := \{ma\} \lhd ma\_data$

end

Event *EmergencyCall* $\widehat{=}$
extends *EmergencyCall*

any
pa
ec

where
grd1 : $\mathtt{pa} \in \mathtt{patients}$
grd2 : $\mathtt{ec} \in \mathtt{ALARMS}$
grd3 : $\mathtt{ec} \notin \mathrm{dom}(\mathtt{emergency\_calls})$
grd4 : $\mathtt{pa} \notin \mathrm{ran}(\mathtt{emergency\_calls})$

then
act1 : $\mathtt{emergency\_calls} := \mathtt{emergency\_calls} \cup \{\mathtt{ec} \mapsto \mathtt{pa}\}$

end

Event *ChooseCurrentCall* $\widehat{=}$
extends *ChooseCurrentCall*

any
ec

where
grd1 : $\mathtt{ec} \in \mathrm{dom}(\mathtt{emergency\_calls})$
grd2 : $\mathtt{ec} \notin \mathrm{dom}(\mathtt{directed})$
grd3 : $\mathtt{ec\_handling} = \mathtt{FALSE}$

then
act1 : $\mathtt{ec\_handling} := \mathtt{TRUE}$
act2 : $\mathtt{current\_call} := \mathtt{ec}$

end

Event *CallFeed* $\widehat{=}$
extends *CallFeed*

when
grd1 : $\mathtt{ec\_handling} = \mathtt{TRUE}$
grd2 : $\mathtt{candidate\_found} = \mathtt{FALSE}$
grd3 : $\mathtt{assigned\_doctor}(\mathtt{emergency\_calls}(\mathtt{current\_call})) \notin \mathtt{disconnected}$

grd4 : $\mathtt{assigned\_doctor(emergency\_calls(current\_call))} \notin \mathtt{occupied}$

then

act1 : $\mathtt{directed(current\_call)} := \mathtt{assigned\_doctor(emergency\_calls(current\_ca}$

act2 : $\mathtt{candidate\_found} := \mathtt{TRUE}$

end

Event *AcceptCall* $\widehat{=}$

extends *AcceptCall*

when

grd1 : $\mathtt{ec\_handling} = \mathtt{TRUE}$

grd2 : $\mathtt{candidate\_found} = \mathtt{TRUE}$

grd3 : $\mathtt{current\_call} \in \mathtt{dom(emergency\_calls)}$

grd4 : $\mathtt{current\_call} \notin \mathtt{dom(accepted\_calls)}$

grd5 : $\mathtt{directed(current\_call)} \notin \mathtt{disconnected}$

then

act1 : $\mathtt{accepted\_calls(current\_call)} := \mathtt{directed(current\_call)}$

act2 : $\mathtt{ec\_handling} := \mathtt{FALSE}$

act3 : $\mathtt{candidate\_found} := \mathtt{FALSE}$

act4 : $\mathtt{occupied} := \varnothing$

end

Event *RejectCall* $\widehat{=}$

extends *RejectCall*

when

grd1 : $\mathtt{ec\_handling} = \mathtt{TRUE}$

grd2 : $\mathtt{candidate\_found} = \mathtt{TRUE}$

grd3 : $\mathtt{card(med\_agents} \setminus \mathtt{occupied)} \geq 2$

then

act1 : $\mathtt{occupied} := \mathtt{occupied} \cup \{\mathtt{directed(current\_call)}\}$

act2 : $\mathtt{candidate\_found} := \mathtt{FALSE}$

end

Event *ForwardCall* $\widehat{=}$

extends *ForwardCall*

any

ma_new

where

grd1 : $\mathtt{ec\_handling} = \mathtt{TRUE}$

grd2 : $\mathtt{candidate\_found} = \mathtt{FALSE}$

grd3 : $\mathtt{assigned\_doctor(emergency\_calls(current\_call))} \in \mathtt{disconnected} \lor$
$\mathtt{assigned\_doctor(emergency\_calls(current\_call))} \in \mathtt{occupied}$

grd4 : $\mathtt{ma\_new} \in \mathtt{med\_agents}$

grd5 : $\mathtt{ma\_new} \notin \mathtt{disconnected}$

grd6 : $\mathtt{ma\_new} \notin \mathtt{occupied}$

then

act1 : $\mathtt{directed(current\_call) := ma\_new}$
act2 : $\mathtt{candidate\_found := TRUE}$

end

Event *ForcedAcceptCall* $\widehat{=}$
extends *ForcedAcceptCall*

when

grd1 : $\mathtt{ec\_handling = TRUE}$
grd2 : $\mathtt{candidate\_found = TRUE}$
grd3 : $\mathtt{current\_call \in dom(emergency\_calls)}$
grd4 : $\mathtt{current\_call \notin dom(accepted\_calls)}$
grd5 : $\mathtt{directed(current\_call) \notin disconnected}$
grd6 : $\mathtt{card(med\_agents \setminus occupied) = 1}$

then

act1 : $\mathtt{accepted\_calls(current\_call) := directed(current\_call)}$
act2 : $\mathtt{ec\_handling := FALSE}$
act3 : $\mathtt{candidate\_found := FALSE}$
act4 : $\mathtt{occupied := \varnothing}$

end

Event *EmergencyEnterScope* $\widehat{=}$
extends *EmergencyVisitBegin*

any

$\mathtt{ec}$
sn

where

grd1 : $\mathtt{ec \in dom(accepted\_calls)}$
grd2 : $\mathtt{emergency\_calls(ec) \notin visited}$
grd3 : $\mathtt{accepted\_calls(ec) \notin last\_visit[visited]}$
grd4 : $\mathtt{accepted\_calls(ec) \notin disconnected}$
grd5 : $sn \in ScopeName$
grd6 : $sn \notin dom(scopes)$
grd7 : $accepted\_calls(ec) \notin ran(scopes)$
grd8 : $accepted\_calls(ec) \notin disconnected$

then

act1 : $\mathtt{last\_visit(emergency\_calls(ec)) := accepted\_calls(ec)}$
act2 : $\mathtt{visited := visited \cup \{emergency\_calls(ec)\}}$
act3 : $\mathtt{emergency\_calls := emergency\_calls \setminus \{ec \mapsto emergency\_calls(ec)\}}$
act4 : $\mathtt{accepted\_calls := accepted\_calls \setminus \{ec \mapsto accepted\_calls(ec)\}}$
act5 : $\mathtt{directed := \{ec\} \lhd directed}$
act6 : $scopes(sn) := accepted\_calls(ec)$
act7 : $ma\_data(accepted\_calls(ec)) := record(emergency\_calls(ec))$

end

Event *RegularEnterScope* $\widehat{=}$
extends *RegularVisitBegin*

any

$\mathtt{ma}$

$\mathtt{pa}$

sn

where

grd1 : $\mathtt{ma} \in \mathtt{med\_agents}$

grd2 : $\mathtt{pa} \in \mathtt{patients}$

grd3 : $\mathtt{pa} \notin \mathtt{visited}$

grd4 : $\mathtt{ma} \notin \mathtt{last\_visit}[\mathtt{visited}]$

grd5 : $\mathtt{pa} \notin \mathtt{ran}(\mathtt{emergency\_calls})$

grd6 : $sn \in ScopeName$

grd7 : $sn \notin dom(scopes)$

grd8 : $ma \notin ran(scopes)$

grd9 : $ma \notin disconnected$

then

act1 : $\mathtt{last\_visit}(\mathtt{pa}) := \mathtt{ma}$

act2 : $\mathtt{visited} := \mathtt{visited} \cup \{\mathtt{pa}\}$

act3 : $scopes := scopes \cup \{sn \mapsto ma\}$

act4 : $ma\_data(ma) := record(pa)$

end

Event *ModifyRecord* $\widehat{=}$

any

ma

sn

pa

da_new

where

grd1 : $(sn \mapsto ma) \in scopes$

grd2 : $pa \in dom(last\_visit)$

grd3 : $last\_visit(pa) = ma$

grd4 : $pa \in visited$

grd5 : $da\_new \in \mathbb{P}(DATA)$

grd6 : $da\_new \neq \varnothing$

then

act1 : $ma\_data(ma) := da\_new$

act2 : $record(pa) := da\_new$

end

Event *LeaveScope* $\widehat{=}$
extends *VisitEnd*

any

pa

sn

where

grd1 : pa ∈ visited

grd2 : $(sn \mapsto last\_visit(pa)) \in scopes$

then

act1 : visited := visited \ {pa}

act2 : $scopes := scopes \rhd \{last\_visit(pa)\}$

act3 : $ma\_data := \{last\_visit(pa)\} \lhd ma\_data$

end

END

CONTEXT c4

EXTENDS c3

SETS

DATA

ScopeName

AXIOMS

axm1 : $finite(DATA)$

axm2 : $finite(ScopeName)$

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 978-952-12-2572-7
ISSN 1239-1891