TUCS

Anton Tarasyuk | Elena Troubitsyna | Linas Laibinis

Quantitative Verification of System Safety in Event-B

Turku Centre for Computer Science

TUCS Technical Report
No 1010, May 2011

Quantitative Verification of System Safety in Event-B

Anton Tarasyuk

Åbo Akademi University, Department of Information Technologies

Turku Centre for Computer Science

Joukahaisenkatu 3-5, FIN-20520 Turku, Finland

anton.tarasyuk@abo.fi

Elena Troubitsyna

Åbo Akademi University, Department of Information Technologies

Joukahaisenkatu 3-5 A, FIN-20520 Turku, Finland

elena.troubitsyna@abo.fi

Linas Laibinis

Åbo Akademi University, Department of Information Technologies

Joukahaisenkatu 3-5 A, FIN-20520 Turku, Finland

linas.laibinis@abo.fi

TUCS Technical Report

No 1010, May 2011

Abstract

Certification of safety-critical systems requires formal verification of system properties and behaviour as well as quantitative demonstration of safety. Usually, formal modelling frameworks do not include quantitative assessment of safety. This has a negative impact on productivity and predictability of system development. In this paper we present an approach to integrating quantitative safety analysis into formal system modelling and verification in Event-B. The proposed approach is based on an extension of Event-B, which allows us to perform quantitative assessment of safety within proof-based verification of system behaviour. This enables development of systems that are not only correct but also safe by construction. The approach is demonstrated by a case study – an automatic railway crossing system.

Keywords: Event-B; formal modelling; refinement; safety; probabilistic reasoning

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Safety is a property of a system to not endanger human life or environment [4]. To guarantee safety, designers employ various rigorous techniques for formal modeling and verification. Such techniques facilitate formal reasoning about system correctness. In particular, they allow us to guarantee that a safety invariant – a logical representation of safety – is always preserved during system execution. However, real safety-critical systems, i.e., the systems whose components are susceptible to various kinds of faults, are not "absolutely" safe. In other words, certain combinations of failures might lead to an occurrence of a hazard – a potentially dangerous situation that breaches safety requirements. While designing and certifying safety-critical systems, we should demonstrate that the probability of a hazard occurrence is acceptably low. In this paper we propose an approach to combining formal system modeling and quantitative safety analysis.

Our approach is based on a probabilistic extension of Event-B [21]. Event-B is a formal modeling framework for developing systems correct-by-construction [3, 1]. It is actively used in the EU project Deploy [6] for modeling and verifying of complex systems from various domains including railways. The Rodin platform [20] provides the designers with an automated tool support that facilitates formal verification and makes Event-B relevant in an industrial setting.

The main development technique of Event-B is refinement – a top-down process of gradual unfolding of the system structure and elaborating on its functionality. In this paper we propose design strategies that allow the developers to structure safety requirements according to the system abstraction layers. Essentially, such an approach can be seen as a process of extracting a fault tree – a logical representation of a hazardous situation in terms of the primitives used at different layers of abstraction. Eventually, we arrive at the representation of a hazardous situation in terms of failures of basic system components. Since our specification contains an explicit representation of probabilities of component failures, standard calculations allow us to obtain a probabilistic evaluation of a hazard occurrence. As a result, we obtain an algebraic representation of probability of safety requirements violation. This probability is defined using the probabilities of system component failures. To illustrate our approach, we present a formal development and safety analysis of a radio-based railway crossing. We believe the proposed approach can potentially facilitate development, verification and assessment of safety-critical systems.

The rest of the paper is organised as follows. In Section 2 we describe our formal modelling framework – Event-B, and briefly introduce its probabilistic extension. In Section 3 we discuss a general design strategy for specifying Event-B models amenable for probabilistic analysis of system safety. In Sec-

Machine M **Variables** v **Invariants** I **Events** $\quad Init$ $\quad evt_1$ $\quad \cdots$ $\quad evt_N$	$\longrightarrow$	**Context** C **Carrier Sets** d **Constants** c **Axioms** A

Figure 1: Event-B machine and context

tion 4 we demonstrate the presented approach by a case study. Finally, Section 5 presents an overview of the related work and some concluding remarks.

2 Modelling in Event-B

The B Method [2] is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [19, 5]. Event-B is a formal framework derived from the B Method to model parallel, distributed and reactive systems. The Rodin platform provides automated tool support for modelling and verification (by theorem proving) in Event-B. Currently Event-B is used in the EU project Deploy to model several industrial systems from automotive, railway, space and business domains.

Event-B Language. In Event-B, a system specification (model) is defined using the notion of an *abstract state machine* [18]. 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 an 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 Fig. 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 also contains other predicates defining properties that must 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

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

Figure 2: Before-after predicates

follows:

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

where a is the list of local variables, the guard g is a conjunction of predicates over the local variables a and state variables v, while the action S is an assignment to the state variables. If an event does not have local variables, it can be described simply as

$$evt \mathrel{\widehat{=}} \textbf{ when } g \textbf{ then } S \textbf{ end}.$$

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 nondeterministically. 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 nondeterministic assignment is denoted either as $x :\in Set$, where Set is a set of values, or $x :\mid Q(x, y, x')$, where Q 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 satisfying Q.

Event-B Semantics. The semantics of Event-B actions is defined using so-called before-after (BA) predicates [3, 18]. A BA predicate describes a relationship between the system states before and after execution of an event, as shown in Fig. 2. Here x and y are disjoint lists (partitions) of state variables, and x', y' represent their values in the after-state.

The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequents. The full list of proof obligations can be found in [3].

Probabilistic Event-B. In our previous work [21] we have have extended the Event-B modelling language with a new operator – *quantitative proba-*

bilistic choice, denoted $\oplus|$. It has the following syntax

$$x \oplus| \; x_1 \;@\; p_1; \ldots ; x_n \;@\; p_n,$$

where $\sum_{i=1}^{n} p_i = 1$. It assigns to the variable x a new value x_i with the corresponding non-zero probability p_i. The quantitative probabilistic choice (assignment) allows us to precisely represent the probabilistic information about how likely a particular choice is made. In other words, it behaves according to some known probabilistic distribution.

We have restricted the use of the new probabilistic choice operator by introducing it only to replace the existing demonic one. This approach has also been adopted by Hallerstede and Hoang, who have proposed extending the Event-B framework with *qualitative probabilistic choice* [10]. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart [13]. Hence, such an extension is not interfering with the established refinement process. Therefore, we can rely on the Event-B proof obligations to guarantee functional correctness of a refinement step. Moreover, the probabilistic information introduced in new quantitative probabilistic choices can be used to stochastically evaluate certain non-functional system properties.

For instance, in [21] we have shown how the notion of Event-B refinement can be strengthened to quantitatively demonstrate that the refined system is more reliable than its abstract counterpart. In this paper we aim at enabling quantitative safety analysis within Event-B development.

3 Safety Analysis in Event-B

In this paper we focus on modelling of highly dynamic reactive control systems. Such systems provide instant control actions as a result of receiving stimuli from the controlled environment. Such a restriction prevents the system from executing automated error recovery, i.e. once a component fails, its failure is considered to be permanent and the system ceases its automatic functioning.

Generally, control systems are built in a layered fashion and reasoning about their behaviour is conducted by unfolding layers of abstraction. Deductive system safety analysis is performed in a similar way. We start by identifying a hazard – a dangerous undesirable situation associated with the system. By unfolding the layers of abstraction we formulate the hazard in terms of component states of different layers.

In an Event-B model, a hazard can be naturally defined as a predicate over the system variables. Sometimes, it is more convenient to reformulate a hazard as a dual safety requirement (property) that specifies a proper behaviour of a system in a hazardous situation. The general form of such a safety property is the following:

$$SAF \mathrel{\widehat{=}} H(v) \Rightarrow K(v),$$

where the predicate $H(v)$ specifies a hazardous situation and the predicate $K(v)$ defines the safety requirements in the terms of the system variables and their states.

The essential properties of an Event-B model are usually formulated as invariants. However, to represent system behaviour realistically, our specification should include modelling of not only normal behaviour but also component failure occurrence. Since certain combinations of failures will lead to hazardous situations, we cannot guarantee "absolute" preservation of safety invariants. Indeed, the goal of development of safety-critical systems is to guarantee that the probability of violation of safety requirements is sufficiently small.

To assess the preservation of a desired safety property we will unfold it (in the refinement process) until it refers only to concrete system components that have direct impact on the system safety. To quantitatively evaluate this impact we require that these components are probabilistically modelled in Event-B using the available information about their reliability. Next we demonstrate how the process of unfolding the safety property from the abstract to the required concrete representation can be integrated into the system development by refinement in Event-B.

Often, functioning of a system can be structured according to a number of *execution stages*. There is a specific component functionality associated with each stage. Since there is no possibility to replace or repair failed system components, we can divide the process of quantitative safety assessment into several consecutive steps, where each step corresponds to a particular stage of the system functioning. Moreover, a relationship between different failures of components and the system behaviour at a certain execution stage is preserved during all the subsequent stages. On the other hand, different subsystems can communicate with each other, which leads to possible additional dependencies between system failures (not necessarily within the same execution stage). This fact significantly complicates quantitative evaluation of the system safety.

We can unfold system safety properties either in a *backward* or in a *forward* way. In the backward unfolding we start from the last execution stage preceding the stage associated with the potentially hazardous situation. In the forward one we start from the first execution stage of the system and continue until the last stage just before the hazardous situation occurs. In this paper we follow the former approach. The main idea is to perform a stepwise analysis of any possible behaviour of all the subsystems at every execution stage preceding the hazardous situation, while gradually unfolding the abstract safety property in terms of new (concrete) variables representing faulty components of the system.

Specifically, in each refinement step, we have to establish the relationship between the newly introduced variables and the abstract variables present in the safety property. A standard way to achieve this is to formulate the required relationship as a number of safety invariants in Event-B. According to our development strategy, each such invariant establishes a connection between abstract and more concrete variables that have an impact on system safety. Moreover, the preservation of a safety invariant is usually verified for a particular subsystem at a specific stage. Therefore, we can define a general form of such an invariant in the following way:

$$I_s(v,u) \mathrel{\widehat{=}} F(v) \Rightarrow (K(v) \Leftrightarrow L(u)),$$

where the predicate F restricts the execution stage and the subsystems involved, while the predicate $K \Leftrightarrow L$ relates the values of the newly introduced variables u with the values the abstract variables v present in the initially defined safety property or/and in the safety invariants defined in the previous refinement steps.

To calculate the probability of preservation of the safety property, the refinement process should be continued until all the abstract variables, used in the definition of the system safety property, are related to the concrete, probabilistically updated variables, representing various system failures or malfunctioning. The process of probability evaluation is rather straightforward and based on basic definitions and rules for calculating probabilities (see [7] for instance).

Let us consider a small yet generic example illustrating the calculation of probability using Event-B safety invariants. We assume that the safety property SAF is defined as above. In addition, let us define two safety invariants – I_s and J_s – introduced in two subsequent refinement steps. More specifically,

$$I_s \mathrel{\widehat{=}} F \Rightarrow (K(v) \Leftrightarrow L_1(u_1) \vee L_2(u_2)) \text{ and } J_s \mathrel{\widehat{=}} \tilde{F} \Rightarrow (L_2(u_2) \Leftrightarrow N(w)),$$

where $u_1 \subset u, u_1 \neq \varnothing$ are updated probabilistically in the first refinement, while $u_2 = u \setminus u_1$ are still abstract in the first refinement machine and related by J_s to the probabilistically updated variables w in the following one. Let us note that the predicate $\tilde{F}$ must define the earlier stage of the system than the predicate F does. Then the probability that the safety property SAF is preserved by the system is

$$\begin{aligned} P_{SAF} = P\{K(v)\} = P\{L_1(u_1) \vee L_2(u_2)\} = P\{L_1(u_1) \vee N(w)\} = \\ P\{L_1(u_1)\} + P\{N(w)\} - P\{L_1(u_1) \wedge N(w)\}, \end{aligned}$$

where

$$P\{L_1(u_1) \wedge N(w)\} = P\{L_1(u_1)\} \cdot P\{N(w)\}$$

in the case of independent L_1 and N, and

$$P\{L_1(u_1) \wedge N(w)\} = P\{L_1(u_1)\} \cdot P\{N(w) \mid L_1(u_1)\}$$

otherwise. Note that the predicate $H(v)$ is not participating in the calculation of P_{SAF} directly. Instead, it defines "the time and the place" when and where the values of the variables u and v should be considered, and, as long as it specifies the hazardous situation following the stages defined by F and $\tilde{F}$, it can be understood as the *post-state* for all the probabilistic events.

In the next section we will demonstrate the approach presented above by a case study – an automatic railway crossing system.

4 Case Study

To illustrate safety analysis in the probabilistically enriched Event-B method, in this section we present a quantitative safety analysis of a radio-based railway crossing. This case study is included into priority program 1064 of the German Research Council (DFG) prepared in cooperation with Deutsche Bahn AG. The main difference between the proposed technology and traditional control systems of railway crossings is that signals and sensors on the route are replaced by radio communication and software computations performed at the train and railway crossings. Formal system modelling of such a system has been undertaken previously [16, 15]. However, the presented methodology is focused on logical (qualitative) reasoning about safety and does not include quantitative safety analysis. Below we demonstrate how to integrate formal modelling and probabilistic safety analysis.

Let us now briefly describe the functioning of a radio-based railway crossing system. The train on the route continuously computes its position. When it approaches a crossing, it broadcasts a *close* request to the crossing. When the railway crossing receives the command *close*, it performs some routine control to ensure safe train passage. It includes switching on the traffic lights, that is followed by an attempt to close the barriers. Shortly before the train reaches the *latest braking point*, i.e., the latest point where it is still possible for the train to stop safely, it requests the *status* of the railway crossing. If the crossing is secured, it responds with a *release* signal, which indicates that the train may pass the crossing. Otherwise, the train has to brake and stop before the crossing. More detailed requirements can be found in [16] for instance.

In our development we abstract away from modelling train movement, calculating train positions and routine control by the railway crossing. Let us note that, any time when the train approaches to the railway crossing, it sequentially performs a number of predefined operations:

- it sends the *close* request to the crossing controller;

- after a delay it sends the *status* request;
- it awaits for an answer from the crossing controller.

The crossing controller, upon receiving the close request, tries to close the barriers and, if successful, sends the *release* signal to the train. Otherwise, it does not send any signal and in this case the train activates the emergency brakes. Our safety analysis focused on defining the hazardous events that may happen in such a railway crossing system due to different hardware and/or communication failures, and assess the probability of the hazard occurrences. We make the following fault assumptions:

- the radio communication is unreliable and can cause messages to be lost;
- the crossing barrier motors may fail to start;
- the positioning sensors that are used by the crossing controller to determine a physical position of the barriers are unreliable;
- the train emergency brakes may fail.

The abstract model. We start our development with identification of all the high-level subsystems we have to model. Essentially, our system consists of two main components – the train and the crossing controller. The system environment is represented by the physical position of the train. Therefore, each control cycle consists of three main phases – Env, $Train$ and $Crossing$. To indicate the current *phase* the eponymous variable is used.

The type modelling abstract train positions is defined as the enumerated set of nonnegative integers $POS\_SET = \{0, CRP, SRP, SRS, DS\}$, where $0 < CRP < SRP < SRS < DS$. Each value of POS_SET represents a specific position of the train. Here 0 stands for some initial train position outside the communication area, CRP and SRP stand for the close and status request points, and SRS and DS represent the safe reaction and danger spots respectively. The actual train position is modelled by the variable $train\_pos \in POS\_SET$. In addition, we use the boolean variable $emrg\_brakes$ to model the status of the train emergency brakes. We assume that initially they are not triggered, i.e., $emrg\_brakes = FALSE$.

The crossing has two barriers – one at each side of the crossing. The status of the barriers is modelled by the variables bar_1 and bar_2 that can take values *Opened* and *Closed*. We assume that both barriers are initially open.

The initial abstract machine *RailwayCrossing* is illustrated in Fig. 3. We omit showing here the *Initialisation* event and the **Invariants** clause (it merely defines the types of variables). Due to lack of space, in the rest

Machine *RailwayCrossing*
Variables $train\_pos, phase, emrg\_brakes, bar_1, bar_2$
Invariants $\cdots$
Events $\cdots$

$UpdatePosition_1 \;\widehat{=}$
when
$phase = Env \wedge train\_pos < DS \wedge emrg\_brakes = FALSE$
then
$train\_pos := \min(\{p \mid p \in POS\_SET \wedge p > train\_pos\}) \,||\, phase := Train$
end

$UpdatePosition_2 \;\widehat{=}$
when
$phase = Env \wedge ((train\_pos = DS \wedge emrg\_brakes = FALSE) \vee emrg\_brakes = TRUE)$
then
skip
end

$TrainIdle \;\widehat{=}$
when
$phase = Train \wedge train\_pos \neq SRS$
then
$phase := Crossing$
end

$TrainReact \;\widehat{=}$
when
$phase = Train \wedge train\_pos = SRS$
then
$emrg\_brakes :\in BOOL \,||\, phase := Crossing$
end

$CrossingBars \;\widehat{=}$
when
$phase = Crossing \wedge train\_pos = CRP$
then
$bar_1, bar_2 :\in BAR\_POS \,||\, phase := Env$
end

$CrossingIdle \;\widehat{=}$
when
$phase = Crossing \wedge train\_pos \neq CRP$
then
$phase := Env$
end

Figure 3: Railway crossing: the abstract machine

of the section we will also present only some selected excerpts of the model. The full Event-B specifications of the *Railway crossing system* can be found in the appendix.

In the machine *RailwayCrossing* we consider only the basic functionality of the system. Two events $UpdatePosition_1$ and $UpdatePosition_2$ are used to abstractly model train movement. The first event models the train movement outside the danger spot by updating the train abstract position according to the next value of the POS_SET. The event $UpdatePosition_2$ models the train behaviour after it has passed the last braking point or when it has stopped in the safe reaction spot. Essentially, this event represents the system termination (both safe and unsafe cases), which is modelled as infinite stuttering, i.e., keeping the system in the final state forever. Such an approach for modelling of the train movement is sufficient since we only analyse system behaviour within the train-crossing communication area, i.e., the area that consists of the close and status request points, and the safe reaction spot. A more realistic approach for modelling of the train movement is out of the scope of our safety analysis.

For the crossing controller, we abstractly model closing of the barriers by the event $CrossingBar$, which non-deterministically assigns the variables bar_1 and bar_2 from the set BAR_POS. Let us note that in the abstract machine the crossing controller immediately knows when the train enters the close request area and makes an attempt to close the barriers. In further refinement steps we eliminate this unrealistic abstraction by introducing communication between the train and the crossing controller. In addition, in the $Train$ phase the event $TrainReact$ non-deterministically models triggering of the train emergency brakes in the safe reaction spot.

The hazard present in the system is the situation when the train passes the crossing while at least one barrier is not closed. In terms of the introduced system variables and their states it can defined as follows:

$$train\_pos = DS \wedge (bar_1 = Opened \vee bar_2 = Opened).$$

In a more traditional (for Event-B invariants) form, this hazard can be dually reformulated as the following safety property:

$$train\_pos = SRS \wedge phase = Crossing \Rightarrow \\ (bar_1 = Closed \wedge bar_2 = Closed) \vee emrg\_brakes = TRUE. \quad (1)$$

This safety requirement can be interpreted as follows: after the train, being in the safe reaction spot, has reacted on signals from the crossing controller, the system is in the safe state only when both barriers are closed or the emergency brakes are activated. Obviously, this property cannot be formulated as an Event-B invariant – it might be violated due to possible communication and/or hardware failures. Our goal is to assess the probability of violation (or preservation) of the safety property (1). To achieve this, during the refinement process, we have to unfold (1) by introducing into the specification the representation of all the system components that have an impact on the system safety. Moreover, we should establish a relationship between the variables representing these components and the abstract variables presented in (1).

The first refinement. In the first refinement step we examine in detail the system behaviour at the safe reaction spot – the last train position preceding the danger spot where the hazard may occur. As a result, the abstract event $TrainReact$ is refined by three events $TrainRelease_1$, $TrainRelease_2$ and $TrainStop$ that represent reaction of the train on the presence or absence of the release signal from the crossing controller. The first two events are used to model the situations when the release signal has been successfully delivered or lost respectively. The last one models the situation when the release signal has not been sent due to some problems at the crossing controller side. Please note that since the events $TrainRelease_2$ and $TrainStop$ perform the same actions, i.e., trigger the emergency brakes, they differ only in their guards.

The event $CrossingStatusReq$ that "decides" whether to send or not to send the release signal is very abstract at this stage – it does not have any specific guards except those that define the system phase and train position. Moreover, the variable *release_snd* is updated in the event body nondeterministically. To model the failures of communication and emergency brakes, we introduce two new events with *probabilistic* bodies – the events $ReleaseComm$ and $TrainDec$ correspondingly. For convenience, we consider communication as a part of the receiving side behaviour. Thus the release communication failure occurrence is modelled in the $Train$ phase while the train being in the SRS position. Some key details of the Event-B machine $RailwayCrossing\_R1$ that refines the abstract machine $RailwayCrossing$ are shown in Fig. 4.

The presence of concrete variables representing unreliable system components in $RailwayCrossing\_R1$ allows us to formulate two safety invariants (saf_inv_1 and saf_inv_2) that glue the abstract variable *emrg_brakes* participating in the safety requirement (1) with the (more) concrete variables *release_rcv*, $emrg\_brakes\_failure$, *release_snd* and $release\_com\_failure$.

$$\mathbf{saf\_inv_1} : train\_pos = SRS \land phase = Crossing \Rightarrow (emrg\_brakes = TRUE \Leftrightarrow release\_rcv = FALSE \land emrg\_brakes\_failure = FALSE)$$

$$\mathbf{saf\_inv_2} : train\_pos = SRS \land phase = Crossing \Rightarrow (release\_rcv = FALSE \Leftrightarrow release\_snd = FALSE \lor release\_comm\_failure = TRUE)$$

We split the relationship between the variables into two invariant properties just to improve the readability and make the invariants easier to understand. Obviously, since the antecedents of both invariants coincide, one can easily merge them together by replacing the variable *release_rcv* in saf_inv_1 with the right hand side of the equivalence in the consequent of saf_inv_1. Please note that the variable *release_snd* corresponds to a certain combination of system actions and hence should be further unfolded during the refinement process.

The second refinement. In the second refinement step we further elaborate on the system functionality. In particular, we model the request messages that the train sends to the crossing controller, as well as sensors that read the position of the barriers. Selected excerpts from the second refinement machine $RailwayCrossing\_R2$ are shown in Fig. 5. To model sending

Machine $RailwayCrossing\_R1$
Variables $\ldots, release\_snd, release\_rcv, emrg\_brakes\_failure, release\_com\_failure, \ldots$
Invariants $\cdots$
Events $\cdots$

$TrainRelease_1 \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE$
$release\_comm\_failure = FALSE \wedge deceleration = FALSE \wedge comm\_ct = FALSE$
then
$emrg\_brakes := FALSE \,||\, release\_rcv := TRUE \,||\, phase := Crossing$
end

$TrainRelease_2 \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE$
$release\_comm\_failure = TRUE \wedge deceleration = FALSE \wedge comm\_ct = FALSE$
then
$emrg\_brakes :| \; emrg\_brakes' \in BOOL \wedge (emrg\_brakes' = TRUE \Leftrightarrow$
$emrg\_brakes\_failure = FALSE)$
$release\_rcv := TRUE \,||\, phase := Crossing$
end

$TrainStop \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = SRS \wedge release\_snd = FALSE \wedge deceleration = FALSE$
then
$\ldots$
end

$CrossingStatusReq \mathrel{\widehat{=}}$
when
$phase = Crossing \wedge train\_pos = SRP$
then
$release\_snd :\in BOOL \,||\, phase := Env$
end

$ReleaseComm \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE \wedge comm\_ct = TRUE$
then
$release\_comm\_failure \oplus| \; TRUE @ p_1; FALSE @ 1{-}p_1 \,||\, comm\_ct := FALSE$
end

$TrainDec \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = SRS \wedge deceleration = TRUE$
then
$emrg\_brakes\_failure \oplus| \; TRUE @ p_4; FALSE @ 1{-}p_4 \,||\, deceleration := FALSE$
end

Figure 4: Railway crossing: first refinement

of the close and status requests by the train, we refine the event $TrainIdle$ by two simple events $TrainCloseReq$ and $TrainStatusReq$ that activate sending of the close and status requests at the corresponding stages. In the crossing controller part, we refine the event $CrossingBars$ by the event $CrossingCloseReq$ that sets the actuators closing the barriers in response to the close request from the train. Clearly, in the case of communication failure occurrence during the close request transmission, both barriers remain open.

Moreover, the abstract event $CrossingStatusReq$ is refined by two events $CrossingStatusReq_1$ and $CrossingStatusReq_2$ to model a reaction of the crossing controller on the status request. The former event is used to model

Machine $RailwayCrossing\_R2$
Variables $\ldots, close\_snd, close\_rcv, status\_snd, status\_rcv,$
$close\_com\_failure, status\_com\_failure, sensor_1, sensor_2 \ldots$
Invariants $\cdots$
Events $\cdots$
$TrainCloseReq \mathrel{\widehat{=}}$
when
$phase = Train \wedge train\_pos = CRP$
then
$close\_req\_snd := TRUE \,||\, phase := Crossing$
end
$\cdots$
$CrossingCloseReq \mathrel{\widehat{=}}$
when
$phase = Crossing \wedge close\_req\_snd = TRUE \wedge comm\_tc = FALSE$
then
$bar_1, bar_2 :| \; bar'_1 \in BAR\_POS \wedge bar'_2 \in BAR\_POS \wedge$
$(close\_comm\_failure = TRUE \Rightarrow bar'_1 = Opened \wedge bar'_2 = Opened)$
$close\_req\_rcv : |close\_req\_rcv' \in BOOL \wedge (close\_req\_rcv' = TRUE \Leftrightarrow$
$close\_comm\_failure = FALSE)$
$comm\_tc := TRUE \,||\, phase := Env$
end
$CrossingStatusReq_1 \mathrel{\widehat{=}}$
when
$phase = Crossing \wedge status\_req\_snd = TRUE \wedge close\_req\_rcv = TRUE \wedge$
$sens\_reading = FALSE \wedge comm\_tc = FALSE$
then
$release\_snd :| \; release\_snd' \in BOOL \wedge (release\_snd' = TRUE \Leftrightarrow$
$status\_comm\_failure = FALSE \wedge sensor_1 = Closed \wedge sensor_2 = Closed)$
$status\_req\_rcv : |status\_req\_rcv' \in BOOL \wedge (status\_req\_rcv' = TRUE \Leftrightarrow$
$status\_comm\_failure = FALSE)$
$comm\_tc := TRUE \,||\, phase := Env$
end
$\cdots$
$ReadSensors \mathrel{\widehat{=}}$
when
$phase = Crossing \wedge status\_req\_snd = TRUE \wedge sens\_reading = TRUE$
then
$sensor_1 \; :\in \; \{bar_1, bnot(bar_1)\} \,||\, sensor_2 \; :\in \; \{bar_2, bnot(bar_2)\} \,||\, sens\_reading \; :=$
$FALSE$
end

Figure 5: Railway crossing: second refinement

the situation when the close request has been successfully received (at the previous stage) and the latter one models the opposite situation. Notice that in the refined event $CrossingStatusReq_1$ the controller sends the release signal only when it has received both request signals and identified that both barriers are closed. This interconnection is reflected in the safety invariant saf_inv_3.

$$\mathbf{saf\_inv_3} : train\_pos = SRP \wedge phase = Env \Rightarrow$$
$$(release\_snd = TRUE \Leftrightarrow close\_req\_rcv = TRUE \wedge$$
$$status\_req\_rcv = TRUE \wedge sensor_1 = Closed \wedge sensor_2 = Closed)$$

Here the variables $sensor_1$ and $sensor_2$ represent values of the barrier positioning sensors. Let us remind that the sensors are unreliable and can return the actual position of the barriers incorrectly. Specifically, the sensors can get stuck at their previous values or spontaneously change the values to the opposite ones. In addition, to model the communication failures, we add two new events *CloseComm* and *StatusComm*. These events are similar to the *ReleaseComm* event of the *RailwayCrossing_R1* machine. Rather intuitive dependencies between the train requests delivery and communication failure occurrences are defined by a pair of safety invariants saf_inv_4 and saf_inv_5.

$$\mathbf{saf\_inv_4} : train\_pos = SRP \wedge phase = Env \Rightarrow (status\_req\_rcv = TRUE \Leftrightarrow status\_com\_failure = FALSE)$$

$$\mathbf{saf\_inv_5} : train\_pos = CRP \wedge phase = Env \Rightarrow (close\_req\_rcv = TRUE \Leftrightarrow close\_com\_failure = FALSE)$$

The third refinement. In the third refinement step – the Event-B machine *RailwayCrossing_R3* – we refine the remaining abstract representation of components mentioned in the safety requirement (1), i.e., modelling of the barrier motors and positioning sensors. We introduce the new variables $bar\_failure_1$, $bar\_failure_2$, $sensor\_failure_1$ and $sensor\_failure_2$ to model the hardware failures. These variables are assigned probabilistically in the newly introduced events *BarStatus* and *SensorStatus* in the same way as it was done for the communication and emergency brakes failures in the first refinement. We refine *CrossingCloseReq* and *ReadSensors* events accordingly. Finally, we formulate four safety invariants saf_inv_6, saf_inv_7, saf_inv_8 and saf_inv_9 to specify the correlation between the physical position of the barriers, the sensor readings, and the hardware failures.

$$\mathbf{saf\_inv_6} : train\_pos = CRP \wedge phase = Env \Rightarrow (bar_1 = Closed \Leftrightarrow bar\_failure_1 = FALSE \wedge close\_comm\_failure = FALSE)$$

$$\mathbf{saf\_inv_7} : train\_pos = CRP \wedge phase = Env \Rightarrow (bar_2 = Closed \Leftrightarrow bar\_failure_2 = FALSE \wedge close\_comm\_failure = FALSE)$$

$$\mathbf{saf\_inv_8} : train\_pos = SRP \wedge phase = Env \Rightarrow (sensor_1 = Closed \Leftrightarrow ((bar_1 = Closed \wedge sensor\_failure_1 = FALSE) \vee (bar_1 = Opened \wedge sensor\_failure_1 = TRUE)))$$

$$\mathbf{saf\_inv_9} : train\_pos = SRP \wedge phase = Env \Rightarrow (sensor_2 = Closed \Leftrightarrow ((bar_2 = Closed \wedge sensor\_failure_2 = FALSE) \vee (bar_2 = Opened \wedge sensor\_failure_2 = TRUE)))$$

The first two invariants state that the crossing barrier can be closed (in the post-state) only when the controller has received the close request and the barrier motor has not failed to start. The second pair of invariants postulates that the positioning sensor may return the value *Closed* in two cases – when the barrier is closed and the sensor works properly, or when the barrier has got stuck while opened and the sensor misreads its position.

Once we have formulated the last four safety invariants, there is no longer any variable, in the safety property (1), that cannot be expressed via some probabilistically updated variables introduced during the refinement process. This allows us to calculate the probability P_{SAF} that (1) is preserved by the system:

$$\begin{aligned}
P_{SAF} = P\{(bar_1 = Closed \wedge bar_2 = Closed) \vee emrg_brakes = TRUE\} = \\
P\{bar_1 = Closed \wedge bar_2 = Closed\} + P\{emrg_brakes = TRUE\} - \\
P\{bar_1 = Closed \wedge bar_2 = Closed\} \cdot \\
P\{emrg_brakes = TRUE \mid bar_1 = Closed \wedge bar_2 = Closed\}.
\end{aligned}$$

Let us recall that we have idenified four different types of failures in our system – the communication failure, the failure of the barrier motor, the sensor failure and emergency brakes failure. We suppose that the probabilities of all these failures are constant and equal to p_1, p_2, p_3 and p_4 correspondingly. The first probability presented in the sum above can be trivially calculated based on the safety invariants saf_inv_7 and saf_inv_8:

$$\begin{aligned}
P\{bar_1 = Closed \wedge bar_2 = Closed\} = \\
P\{bar_failure_1 = FALSE \wedge bar_failure_2 = FALSE \wedge \\
close_comm_failure = FALSE\} = (1 - p_1) \cdot (1 - p_2)^2.
\end{aligned}$$

Indeed, both barriers are closed only when the crossing controller received the close request and none of the barrier motors has failed. The calculation of the other two probabilities is slightly more complicated. Nevertheless, they can be straightforwardly obtained using all the safety invariants defined in the model and basic rules for calculating probability. We omit the computation details due to a lack of space. The resulting probability of preservation of the safety property (1) is:

$$\begin{aligned}
P_{SAF} \;=\; & (1 - p_1) \cdot (1 - p_2)^2 + \\
& (1 - p_4) \cdot \Big(1 - (1 - p_1)^3 \cdot (p_2 \cdot p_3 + (1 - p_2) \cdot (1 - p_3))^2\Big) - \\
& (1 - p_1) \cdot (1 - p_2)^2 \cdot (1 - p_4) \cdot \big(1 - (1 - p_1)^2 \cdot (1 - p_3)^2\big).
\end{aligned}$$

Please note that P_{SAF} is defined as a function of probabilities of component failures, i.e., probabilities $p_1, \ldots, p_4$. Provided the numerical values of them are given, we can use the obtained formula to verify whether the system achieves the desired safety threshold.

5 Discussion

5.1 Related Work

Formal methods are extensively used for the development and verification of safety-critical systems. In particular, the B Method and Event-B are successfully being applied for formal development of railway systems [12, 5]. A safety analysis of the formal model of a radio-based railway crossing controller has also been performed with the KIV theorem prover [16, 15]. However, the approaches for integrating formal verification and quantitative assessment are still scarce.

Usually, quantitative analysis of safety relies on probabilistic model checking techniques. For instance, in [11], the authors demonstrate how the quantitative model checker PRISM [17] can be used to evaluate system dependability attributes. The work reported in [8] presents model-based probabilistic safety assessment based on generating PRISM specifications from Simulink diagrams annotated with failure logic. A method pFMEA (probabilistic Failure Modes and Effect Analysis) also relies on the PRISM model checker to conduct quantitative analysis of safety [9]. The approach integrates the failure behaviour into the system model described in continuous time Markov chains via failure injection. In [14] the authors propose a method for probabilistic model-based safety analysis for synchronous parallel systems. It has been shown that different types of failures, in particular per-time and per-demand, can be modelled and analysed using probabilistic model checking.

However, in general the methods based on model checking aim at safety evaluation of already developed systems. They extract a model eligible for probabilistic analysis and evaluate impact of various system parameters on its safety. In our approach, we aim at providing the designers with a safety-explicit development method. Indeed, safety analysis is essentially integrated into system development by refinement. It allows us to perform quantitative assessment of safety within proof-based verification of the system behaviour.

5.2 Conclusions

In this paper we have proposed an approach to integrating quantitative safety assessment into formal system development in Event-B. The main merit of our approach is that of merging logical (qualitative) reasoning about correctness of system behaviour with probabilistic (quantitative) analysis of its safety. An application of our approach allows the designers to obtain a probability of hazard occurrence as a function over probabilities of component failures.

Essentially, our approach sets the guidelines for safety-explicit development in Event-B. We have shown how to explicitly define safety properties at different levels of refinement. The refinement process has facilitated not only correctness-preserving model transformations but also establishes a logical link between safety conditions at different levels of abstraction. It leads to deriving a logical representation of hazardous conditions. An explicit modelling of probabilities of component failures has allowed us to calculate the likelihood of hazard occurrence. The B Method and Event-B are successfully and intensively used in the development of safety-critical systems, particularly in the railway domain. We believe that our approach provides the developers with a promising solution unifying formal verification and quantitative reasoning.

In our future work we are planning to further extend the proposed approach to enable probabilistic safety assessment at the architectural level.

References

[1] J.-R. Abrial. Extending B without Changing it (for Developing Distributed Systems). In H. Habiras, editor, *First Conference on the B method*, pages 169–190. 1996.

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

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

[4] A. Avizienis, J.-C. Laprie, and B. Randell. Fundamental Concepts of Dependability. 2001. Research Report No 1145, LAAS-CNRS.

[5] D. Craigen, S. Gerhart, and T. Ralson. Case Study: Paris Metro Signaling System. In *IEEE Software*, pages 32–35, 1994.

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

[7] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, 1967.

[8] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi. Systematic Model-based Safety Assessment via Probabilistic Model Checking. In *ISoLA'10, International Conference on Leveraging Applications of Formal Methods, Verification, and Validation*, pages 625–639. Springer-Verlag, 2010.

[9] L. Grunske, R. Colvin, and K. Winter. Probabilistic Model-Checking Support for FMEA. In *QEST'07, International Conference on Quantitative Evaluation of Systems*, 2007.

[10] S. Hallerstede and T. S. Hoang. Qualitative probabilistic modelling in Event-B. In J. Davies and J. Gibbons, editors, *IFM 2007, Integrated Formal Methods*, pages 293–312, 2007.

[11] M. Kwiatkowska, G. Norman, and D. Parker. Controller Dependability Analysis by Probabilistic Model Checking. In *Control Engineering Practice*, pages 1427–1434, 2007.

[12] T. Lecomte, T. Servat, and G. Pouzancre. Formal Methods in Safety-Critical Railway Systems. In *Brasilian Symposium on Formal Methods*, 2007.

[13] A. K. McIver and C. C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[14] F. Ortmeier and M. Güdemann. Probabilistic Model-Based Safety Analysis. In *QAPL 2010, Workshop on Quantitative Aspects of Programming Languages*, EPTCS, pages 114–128, 2010.

[15] F. Ortmeier, W. Reif, and G. Schellhorn. Formal Safety Analysis of a Radio-Based Railroad Crossing Using Deductive Cause-Consequence Analysis (DCCA). In *EDCC 2005, European Dependable Computing Conference*, pages 139–151. Springer, 2007.

[16] F. Ortmeier and G. Schellhorn. Formal Fault Tree Analysis: Practical Experiences. In *AVoCS 2006, International Workshop on Automated Verification of Critical Systems*, volume 185 of *ENTCS*, pages 139–151. Elsevier, 2007.

[17] PRISM – Probabilistic Symbolic Model Checker. online at http://www.prismmodelchecker.org/.

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

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

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

[21] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Towards Probabilistic Modelling in Event-B. In D. Méry and S. Merz, editors, *IFM 2010, Integrated Formal Methods*. Springer-Verlag, 2010.

Appendix

MACHINE RailroadCrossing

SEES RailroadCrossing_ctx

VARIABLES

train_pos

phase

emrg_brakes

bar_2

bar_1

INVARIANTS

inv1 : $train\_pos \in POS\_SET$

inv2 : $phase \in PHASES$

inv3 : $emrg\_brakes \in BOOL$

inv4 : $phase \in \{Env, Train, Crossing\}$

inv7 : $bar\_2 \in BAR\_POS$

inv6 : $bar\_1 \in BAR\_POS$

inv8 : $phase \neq Env \Rightarrow train\_pos \neq 0$

EVENTS

Initialisation

begin

act1 : $train\_pos := 0$
act2 : $phase := Env$
act3 : $emrg\_brakes := FALSE$
act5 : $bar\_2 := Opened$
act4 : $bar\_1 := Opened$

end

Event $UpdatePos1 \widehat{=}$

when

grd1 : $phase = Env$
grd2 : $emrg\_brakes = FALSE$
grd3 : $train\_pos < DS$

then

act1 : $train\_pos := min(\{p|p \in POS\_SET \land p > train\_pos\})$
act2 : $phase := Train$

end

Event *UpdatePos2* $\widehat{=}$

when

grd1 : $phase = Env$
grd2 : $(emrg\_brakes = FALSE \land train\_pos = DS) \lor emrg\_brakes = TRUE$

then

skip

end

Event *TrainIdle* $\widehat{=}$

when

grd1 : $phase = Train$
grd2 : $train\_pos \neq SRS$

then

act1 : $phase := Crossing$

end

Event *TrainReact* $\widehat{=}$

when

grd1 : $phase = Train$
grd2 : $train\_pos = SRS$

then

act1 : $emrg\_brakes :\in BOOL$
act2 : $phase := Crossing$

end

Event *CrossingBars* $\widehat{=}$

when

grd1 : $phase = Crossing$
grd2 : $train\_pos = CRP$

then

act1 : $bar\_1 :\in BAR\_POS$
act2 : $bar\_2 :\in BAR\_POS$

act3 : $phase := Env$

end

Event $CrossingIdle \mathrel{\widehat{=}}$

when

grd1 : $phase = Crossing$

grd2 : $train\_pos \neq CRP$

then

act1 : $phase := Env$

end

END

CONTEXT RailroadCrossing_ctx

SETS

PHASES

BAR_POS

CONSTANTS

CRP Close Request Period

SRP Status Request Period

SRS Safe Reaction Spot

DS Danger Spot

POS_SET

Env Environment (train movement)

Train Train controller/actuators phase

Crossing Crossing controller/actuators phase

Opened

Closed

bnot

AXIOMS

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

axm2 : $SRP \in \mathbb{N}_1$

axm3 : $SRS \in \mathbb{N}_1$

axm4 : $DS \in \mathbb{N}_1$

axm6 : $CRP < SRP$

axm7 : $SRP < SRS$

axm8 : $SRS < DS$

axm10 : $partition(PHASES, \{Env\}, \{Train\}, \{Crossing\})$

axm11 : $partition(BAR\_POS, \{Opened\}, \{Closed\})$

axm13 : $POS\_SET \subseteq \mathbb{N}$

axm12 : $POS\_SET = \{0, CRP, SRP, SRS, DS\}$

axm14 : $bnot \in BAR\_POS \rightarrow BAR\_POS$

axm15 : $bnot(Opened) = Closed$

axm16 : $bnot(Closed) = Opened$

thm1 : $\forall x \cdot x \in BAR\_POS \Rightarrow (\{x, bnot(x)\} = BAR\_POS)$

END

MACHINE RailroadCrossing_R1

REFINES RailroadCrossing

SEES RailroadCrossing_ctx

VARIABLES

`train_pos`

`phase`

`emrg_brakes`

`bar_2`

`bar_1`

`release_snd`

`release_rcv`

`communication_ct`

`deceleration`

`emrg_brakes_failure`

`release_comm_failure`

INVARIANTS

inv1 : $release\_snd \in BOOL$

inv2 : $release\_rcv \in BOOL$

inv3 : $communication\_ct \in BOOL$

inv4 : $deceleration \in BOOL$

inv5 : $emrg\_brakes\_failure \in BOOL$

inv6 : $release\_comm\_failure \in BOOL$

inv7 : $train\_pos = SRS \land phase = Crossing \Rightarrow (emrg\_brakes = TRUE \Leftrightarrow release\_rcv = FALSE \land emrg\_brakes\_failure = FALSE)$

inv8 : $train\_pos = SRS \land phase = Crossing \Rightarrow (release\_rcv = FALSE \Leftrightarrow release\_snd = FALSE \lor release\_comm\_failure = TRUE)$

EVENTS

Initialisation

extended

begin

act1 : train_pos := 0

act2 : phase := Env

act3 : emrg_brakes := FALSE

act5 : $\texttt{bar\_2} := \texttt{Opened}$
act4 : $\texttt{bar\_1} := \texttt{Opened}$
act6 : $release\_snd := FALSE$
act7 : $release\_rcv := FALSE$
act8 : $communication\_ct := TRUE$
act9 : $deceleration := TRUE$
act10 : $emrg\_brakes\_failure := FALSE$
act11 : $release\_comm\_failure := FALSE$

end

Event *UpdatePos1* $\widehat{=}$

extends *UpdatePos1*

when

grd1 : $\texttt{phase} = \texttt{Env}$
grd2 : $\texttt{emrg\_brakes} = \texttt{FALSE}$
grd3 : $\texttt{train\_pos} < \texttt{DS}$

then

act1 : $\texttt{train\_pos} := \texttt{min}(\{\texttt{p}|\texttt{p} \in \texttt{POS\_SET} \land \texttt{p} > \texttt{train\_pos}\})$
act2 : $\texttt{phase} := \texttt{Train}$

end

Event *UpdatePos2* $\widehat{=}$

extends *UpdatePos2*

when

grd1 : $\texttt{phase} = \texttt{Env}$
grd2 : $(\texttt{emrg\_brakes} = \texttt{FALSE} \land \texttt{train\_pos} = \texttt{DS}) \lor \texttt{emrg\_brakes} = \texttt{TRUE}$

then

skip

end

Event *TrainIdle* $\widehat{=}$

extends *TrainIdle*

when

grd1 : $\texttt{phase} = \texttt{Train}$
grd2 : $\texttt{train\_pos} \neq \texttt{SRS}$

then

act1 : phase := Crossing

end

Event *TrainRelease1* $\widehat{=}$

refines *TrainReact*

when

grd1 : $phase = Train$

grd2 : $train\_pos = SRS$

grd3 : $release\_comm\_failure = FALSE$

grd4 : $release\_snd = TRUE$

grd5 : $communication\_ct = FALSE$

grd6 : $deceleration = FALSE$

then

act1 : $emrg\_brakes := FALSE$

act2 : $phase := Crossing$

act3 : $release\_rcv := TRUE$

end

Event *TrainRelease2* $\widehat{=}$

refines *TrainReact*

when

grd1 : $phase = Train$

grd2 : $train\_pos = SRS$

grd3 : $release\_comm\_failure = TRUE$

grd4 : $release\_snd = TRUE$

grd5 : $communication\_ct = FALSE$

grd6 : $deceleration = FALSE$

then

act1 : $emrg\_brakes :| emrg\_brakes' \in BOOL \wedge (emrg\_brakes' = TRUE \Leftrightarrow emrg\_brakes\_failure = FALSE)$

act2 : $phase := Crossing$

act3 : $release\_rcv := FALSE$

end

Event *TrainStop* $\widehat{=}$

refines *TrainReact*

when

grd1 : $phase = Train$

grd2 : $train\_pos = SRS$

grd3 : $release\_snd = FALSE$

grd4 : $deceleration = FALSE$

then

act1 : $emrg\_brakes :| emrg\_brakes' \in BOOL \wedge (emrg\_brakes' = TRUE \Leftrightarrow emrg\_brakes\_failure = FALSE)$

act2 : $phase := Crossing$

act3 : $release\_rcv := FALSE$

end

Event *CrossingBars* $\widehat{=}$

extends *CrossingBars*

when

grd1 : phase = Crossing

grd2 : train_pos = CRP

then

act1 : bar_1 :∈ BAR_POS

act2 : bar_2 :∈ BAR_POS

act3 : phase := Env

end

Event *CrossingStatusReq* $\widehat{=}$

refines *CrossingIdle*

when

grd1 : $phase = Crossing$

grd2 : $train\_pos = SRP$

then

act1 : $release\_snd :\in BOOL$

act2 : $phase := Env$

end

Event *CrossingIdle* $\widehat{=}$

refines *CrossingIdle*

when

grd1 : $phase = Crossing$

grd2 : $train\_pos \in \{SRS, DS\}$

then

`act1` : $phase := Env$

`act2` : $release\_snd := FALSE$

end

Event $ReleaseComm \;\widehat{=}$

when

`grd1` : $phase = Train$

`grd2` : $train\_pos = SRS$

`grd3` : $release\_snd = TRUE$

`grd4` : $communication\_ct = TRUE$

then

`act1` : $release\_comm\_failure :\in BOOL$

`act2` : $communication\_ct := FALSE$

end

Event $TrainDec \;\widehat{=}$

when

`grd1` : $phase = Train$

`grd2` : $train\_pos = SRS$

`grd3` : $deceleration = TRUE$

then

`act1` : $emrg\_brakes\_failure :\in BOOL$

`act2` : $deceleration := FALSE$

end

END

MACHINE RailroadCrossing_R2

REFINES RailroadCrossing_R1

SEES RailroadCrossing_ctx

VARIABLES

    train_pos
    phase
    emrg_brakes
    bar_2
    bar_1
    release_snd
    release_rcv
    communication_ct
    deceleration
    emrg_brakes_failure
    release_comm_failure
    close_req_snd
    close_req_rcv
    status_req_snd
    status_req_rcv
    sens_reading
    sensor_1
    sensor_2
    communication_tc
    close_comm_failure
    status_comm_failure

INVARIANTS

inv1 : $close\_req\_snd \in BOOL$

inv2 : $close\_req\_rcv \in BOOL$

inv3 : $status\_req\_snd \in BOOL$

inv4 : $status\_req\_rcv \in BOOL$

inv5 : $sens\_reading \in BOOL$

inv6 : $sensor\_1 \in BAR\_POS$

inv7 : $sensor\_2 \in BAR\_POS$

$\texttt{inv8}:$ $communication\_tc \in BOOL$

$\texttt{inv9}:$ $close\_comm\_failure \in BOOL$

$\texttt{inv10}:$ $status\_comm\_failure \in BOOL$

$\texttt{inv11}:$ $train\_pos = SRP \wedge phase = Env \Rightarrow (release\_snd = TRUE \Leftrightarrow close\_req\_rcv = TRUE \wedge status\_req\_rcv = TRUE \wedge sensor\_1 = Closed \wedge sensor\_2 = Closed)$

$\texttt{inv12}:$ $train\_pos = SRP \wedge phase = Env \Rightarrow (status\_req\_rcv = TRUE \Leftrightarrow status\_comm\_failure = FALSE)$

$\texttt{inv13}:$ $train\_pos = CRP \wedge phase = Env \Rightarrow (close\_req\_rcv = TRUE \Leftrightarrow close\_comm\_failure = FALSE)$

$\texttt{inv14}:$ $phase = Crossing \Rightarrow (close\_req\_snd = TRUE \Leftrightarrow train\_pos = CRP)$

$\texttt{inv15}:$ $phase = Crossing \Rightarrow (status\_req\_snd = TRUE \Leftrightarrow train\_pos = SRP)$

$\texttt{inv16}:$ $close\_req\_snd = TRUE \Rightarrow status\_req\_snd = FALSE$

$\texttt{inv17}:$ $train\_pos < SRP \Rightarrow status\_req\_snd = FALSE$

$\texttt{inv18}:$ $train\_pos > SRP \Rightarrow close\_req\_snd = FALSE$

$\texttt{inv19}:$ $train\_pos = SRP \wedge phase = Env \Rightarrow close\_req\_snd = FALSE$

$\texttt{inv20}:$ $train\_pos = DS \Rightarrow status\_req\_snd = FALSE$

$\texttt{inv21}:$ $train\_pos = SRS \wedge phase = Env \Rightarrow status\_req\_snd = FALSE$

EVENTS

Initialisation

extended

begin

$\texttt{act1}:$ $\texttt{train\_pos} := 0$

$\texttt{act2}:$ $\texttt{phase} := \texttt{Env}$

$\texttt{act3}:$ $\texttt{emrg\_brakes} := \texttt{FALSE}$

$\texttt{act5}:$ $\texttt{bar\_2} := \texttt{Opened}$

$\texttt{act4}:$ $\texttt{bar\_1} := \texttt{Opened}$

$\texttt{act6}:$ $\texttt{release\_snd} := \texttt{FALSE}$

$\texttt{act7}:$ $\texttt{release\_rcv} := \texttt{FALSE}$

$\texttt{act8}:$ $\texttt{communication\_ct} := \texttt{TRUE}$

$\texttt{act9}:$ $\texttt{deceleration} := \texttt{TRUE}$

$\texttt{act10}:$ $\texttt{emrg\_brakes\_failure} := \texttt{FALSE}$

$\texttt{act11}:$ $\texttt{release\_comm\_failure} := \texttt{FALSE}$

$\texttt{act12}:$ $close\_req\_snd := FALSE$

$\texttt{act13}:$ $close\_req\_rcv := FALSE$

$\texttt{act14}:$ $status\_req\_snd := FALSE$

act15 : $status\_req\_rcv := FALSE$
act16 : $sens\_reading := TRUE$
act17 : $sensor\_1 := Opened$
act18 : $sensor\_2 := Opened$
act19 : $communication\_tc := TRUE$
act20 : $close\_comm\_failure := FALSE$
act21 : $status\_comm\_failure := FALSE$

end

Event $UpdatePos1 \widehat{=}$

extends $UpdatePos1$

when

grd1 : $\mathtt{phase} = \mathtt{Env}$
grd2 : $\mathtt{emrg\_brakes} = \mathtt{FALSE}$
grd3 : $\mathtt{train\_pos} < \mathtt{DS}$

then

act1 : $\mathtt{train\_pos} := \mathtt{min}(\{\mathtt{p}|\mathtt{p} \in \mathtt{POS\_SET} \land \mathtt{p} > \mathtt{train\_pos}\})$
act2 : $\mathtt{phase} := \mathtt{Train}$

end

Event $UpdatePos2 \widehat{=}$

extends $UpdatePos2$

when

grd1 : $\mathtt{phase} = \mathtt{Env}$
grd2 : $(\mathtt{emrg\_brakes} = \mathtt{FALSE} \land \mathtt{train\_pos} = \mathtt{DS}) \lor \mathtt{emrg\_brakes} = \mathtt{TRUE}$

then

skip

end

Event $TrainCloseReq \widehat{=}$

refines $TrainIdle$

when

grd1 : $phase = Train$
grd2 : $train\_pos = CRP$

then

act1 : $phase := Crossing$

act2 : $close\_req\_snd := TRUE$

end

Event $TrainStatusReq \widehat{=}$

refines $TrainIdle$

when

grd1 : $phase = Train$

grd2 : $train\_pos = SRP$

then

act1 : $status\_req\_snd := TRUE$

act2 : $close\_req\_snd := FALSE$

act3 : $phase := Crossing$

end

Event $TrainRelease1 \widehat{=}$

extends $TrainRelease1$

when

grd1 : phase = Train

grd2 : train_pos = SRS

grd3 : release_comm_failure = FALSE

grd4 : release_snd = TRUE

grd5 : communication_ct = FALSE

grd6 : deceleration = FALSE

then

act1 : emrg_brakes := FALSE

act2 : phase := Crossing

act3 : release_rcv := TRUE

act4 : $status\_req\_snd := FALSE$

end

Event $TrainRelease2 \widehat{=}$

extends $TrainRelease2$

when

grd1 : phase = Train

grd2 : train_pos = SRS

grd3 : release_comm_failure = TRUE

grd4 : release_snd = TRUE

grd5 : $\texttt{communication\_ct} = \texttt{FALSE}$

grd6 : $\texttt{deceleration} = \texttt{FALSE}$

then

act1 : $\texttt{emrg\_brakes} : | \texttt{emrg\_brakes}' \in \texttt{BOOL} \wedge (\texttt{emrg\_brakes}' = \texttt{TRUE} \Leftrightarrow \texttt{emrg\_brakes\_failure} = \texttt{FALSE})$

act2 : $\texttt{phase} := \texttt{Crossing}$

act3 : $\texttt{release\_rcv} := \texttt{FALSE}$

act4 : $status\_req\_snd := FALSE$

end

Event *TrainStop* $\widehat{=}$

extends *TrainStop*

when

grd1 : $\texttt{phase} = \texttt{Train}$

grd2 : $\texttt{train\_pos} = \texttt{SRS}$

grd3 : $\texttt{release\_snd} = \texttt{FALSE}$

grd4 : $\texttt{deceleration} = \texttt{FALSE}$

then

act1 : $\texttt{emrg\_brakes} : | \texttt{emrg\_brakes}' \in \texttt{BOOL} \wedge (\texttt{emrg\_brakes}' = \texttt{TRUE} \Leftrightarrow \texttt{emrg\_brakes\_failure} = \texttt{FALSE})$

act2 : $\texttt{phase} := \texttt{Crossing}$

act3 : $\texttt{release\_rcv} := \texttt{FALSE}$

act4 : $status\_req\_snd := FALSE$

end

Event *TrainDangerSpot* $\widehat{=}$

refines *TrainIdle*

when

grd1 : $phase = Train$

grd2 : $train\_pos = DS$

then

act1 : $phase := Crossing$

end

Event *CrossingCloseReq* $\widehat{=}$

refines *CrossingBars*

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = TRUE$

grd3 : $communication\_tc = FALSE$

then

act1 : $bar\_1, bar\_2 : |bar\_1' \in BAR\_POS \wedge bar\_2' \in BAR\_POS \wedge (close\_comm\_failure = TRUE \Rightarrow bar\_1' = Opened \wedge bar\_2' = Opened)$

act3 : $close\_req\_rcv : |close\_req\_rcv' \in BOOL \wedge (close\_req\_rcv' = TRUE \Leftrightarrow close\_comm\_failure = FALSE)$

act4 : $communication\_tc := TRUE$

act5 : $phase := Env$

end

Event $CrossingStatusReq1 \mathrel{\widehat{=}}$

refines $CrossingStatusReq$

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $close\_req\_rcv = TRUE$

grd4 : $sens\_reading = FALSE$

grd5 : $communication\_tc = FALSE$

then

act1 : $release\_snd : |release\_snd' \in BOOL \wedge (release\_snd' = TRUE \Leftrightarrow (status\_comm\_failure = FALSE \wedge sensor\_1 = Closed \wedge sensor\_2 = Closed))$

act2 : $status\_req\_rcv : |status\_req\_rcv' \in BOOL \wedge (status\_req\_rcv' = TRUE \Leftrightarrow status\_comm\_failure = FALSE)$

act3 : $communication\_tc := TRUE$

act4 : $phase := Env$

end

Event $CrossingStatusReq2 \mathrel{\widehat{=}}$

refines $CrossingStatusReq$

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $close\_req\_rcv = FALSE$

grd4 : $sens\_reading = FALSE$

grd5 : $communication\_tc = FALSE$

then

act1 : $status\_req\_rcv :| status\_req\_rcv' \in BOOL \land (status\_req\_rcv' = TRUE \Leftrightarrow status\_comm\_failure = FALSE)$

act2 : $release\_snd := FALSE$

act3 : $communication\_tc := TRUE$

act4 : $phase := Env$

end

Event $CrossingIdle \widehat{=}$

refines $CrossingIdle$

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = FALSE$

grd3 : $status\_req\_snd = FALSE$

then

act1 : $phase := Env$

act2 : $release\_snd := FALSE$

end

Event $ReleaseComm \widehat{=}$

extends $ReleaseComm$

when

grd1 : $\mathtt{phase = Train}$

grd2 : $\mathtt{train\_pos = SRS}$

grd3 : $\mathtt{release\_snd = TRUE}$

grd4 : $\mathtt{communication\_ct = TRUE}$

then

act1 : $\mathtt{release\_comm\_failure :\in BOOL}$

act2 : $\mathtt{communication\_ct := FALSE}$

end

Event $TrainDec \widehat{=}$

extends $TrainDec$

when

grd1 : $\mathtt{phase = Train}$

grd2 : $\mathtt{train\_pos = SRS}$

grd3 : deceleration = TRUE

then

act1 : emrg_brakes_failure :∈ BOOL

act2 : deceleration := FALSE

end

Event *CloseComm* $\widehat{=}$

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = TRUE$

grd3 : $communication\_tc = TRUE$

then

act1 : $close\_comm\_failure :\in BOOL$

act2 : $communication\_tc := FALSE$

end

Event *StatusComm* $\widehat{=}$

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $communication\_tc = TRUE$

then

act1 : $status\_comm\_failure :\in BOOL$

act2 : $communication\_tc := FALSE$

end

Event *ReadSensors* $\widehat{=}$

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $sens\_reading = TRUE$

then

act1 : $sensor\_1 :\in \{bar\_1, bnot(bar\_1)\}$

act2 : $sensor\_2 :\in \{bar\_2, bnot(bar\_2)\}$

act3 : $sens\_reading := FALSE$

end

END

MACHINE RailwayCrossing_R3

REFINES RailroadCrossing_R2

SEES RailroadCrossing_ctx

VARIABLES

train_pos

phase

emrg_brakes

bar_2

bar_1

release_snd

release_rcv

communication_ct

deceleration

emrg_brakes_failure

release_comm_failure

close_req_snd

close_req_rcv

status_req_snd

status_req_rcv

sens_reading

sensor_1

sensor_2

communication_tc

close_comm_failure

status_comm_failure

bar_failure_1

bar_failure_2

sensor_failure_1

sensor_failure_2

closing

sensing

INVARIANTS

inv1 : $bar\_failure\_1 \in BOOL$

inv2 : $bar\_failure\_2 \in BOOL$

inv3 : $sensor\_failure\_1 \in BOOL$

inv4 : $sensor\_failure\_2 \in BOOL$

inv5 : $closing \in BOOL$

inv6 : $sensing \in BOOL$

inv7 : $train\_pos = CRP \land phase = Env \Rightarrow (bar\_1 = Closed \Leftrightarrow bar\_failure\_1 = FALSE \land close\_comm\_failure = FALSE) \land (bar\_2 = Closed \Leftrightarrow bar\_failure\_2 = FALSE \land close\_comm\_failure = FALSE)$

inv8 : $train\_pos = SRP \land phase = Env \Rightarrow (sensor\_1 = Closed \Leftrightarrow ((bar\_1 = Closed \land sensor\_failure\_1 = FALSE) \lor (bar\_1 = Opened \land sensor\_failure\_1 = TRUE))) \land (sensor\_2 = Closed \Leftrightarrow ((bar\_2 = Closed \land sensor\_failure\_2 = FALSE) \lor (bar\_2 = Opened \land sensor\_failure\_2 = TRUE)))$

inv9 : $sens\_reading = FALSE \Rightarrow (sensor\_1 = Closed \Leftrightarrow ((bar\_1 = Closed \land sensor\_failure\_1 = FALSE) \lor (bar\_1 = Opened \land sensor\_failure\_1 = TRUE))) \land (sensor\_2 = Closed \Leftrightarrow ((bar\_2 = Closed \land sensor\_failure\_2 = FALSE) \lor (bar\_2 = Opened \land sensor\_failure\_2 = TRUE)))$

inv10 : $sensing = TRUE \Rightarrow sens\_reading = TRUE$

inv11 : $sens\_reading = FALSE \Rightarrow train\_pos > CRP$

EVENTS

Initialisation

extended

begin

act1 : train_pos := 0

act2 : phase := Env

act3 : emrg_brakes := FALSE

act5 : bar_2 := Opened

act4 : bar_1 := Opened

act6 : release_snd := FALSE

act7 : release_rcv := FALSE

act8 : communication_ct := TRUE

act9 : deceleration := TRUE

act10 : emrg_brakes_failure := FALSE

act11 : release_comm_failure := FALSE

act12 : close_req_snd := FALSE

act13 : close_req_rcv := FALSE

act14 : status_req_snd := FALSE

act15 : status_req_rcv := FALSE

act16 : sens_reading := TRUE

act17 : $\mathtt{sensor\_1} := \mathtt{Opened}$
act18 : $\mathtt{sensor\_2} := \mathtt{Opened}$
act19 : $\mathtt{communication\_tc} := \mathtt{TRUE}$
act20 : $\mathtt{close\_comm\_failure} := \mathtt{FALSE}$
act21 : $\mathtt{status\_comm\_failure} := \mathtt{FALSE}$
act22 : $bar\_failure\_1 := FALSE$
act23 : $bar\_failure\_2 := FALSE$
act24 : $sensor\_failure\_1 := FALSE$
act25 : $sensor\_failure\_2 := FALSE$
act26 : $closing := TRUE$
act27 : $sensing := TRUE$

end

Event *UpdatePos1* $\widehat{=}$

extends *UpdatePos1*

when

grd1 : $\mathtt{phase} = \mathtt{Env}$
grd2 : $\mathtt{emrg\_brakes} = \mathtt{FALSE}$
grd3 : $\mathtt{train\_pos} < \mathtt{DS}$

then

act1 : $\mathtt{train\_pos} := \mathtt{min}(\{\mathtt{p}|\mathtt{p} \in \mathtt{POS\_SET} \land \mathtt{p} > \mathtt{train\_pos}\})$
act2 : $\mathtt{phase} := \mathtt{Train}$

end

Event *UpdatePos2* $\widehat{=}$

extends *UpdatePos2*

when

grd1 : $\mathtt{phase} = \mathtt{Env}$
grd2 : $(\mathtt{emrg\_brakes} = \mathtt{FALSE} \land \mathtt{train\_pos} = \mathtt{DS}) \lor \mathtt{emrg\_brakes} = \mathtt{TRUE}$

then

skip

end

Event *TrainCloseReq* $\widehat{=}$

extends *TrainCloseReq*

when

grd1 : phase = Train
grd2 : train_pos = CRP

then

act1 : phase := Crossing
act2 : close_req_snd := TRUE

end

Event *TrainStatusReq* $\widehat{=}$

extends *TrainStatusReq*

when

grd1 : phase = Train
grd2 : train_pos = SRP

then

act1 : status_req_snd := TRUE
act2 : close_req_snd := FALSE
act3 : phase := Crossing

end

Event *TrainRelease1* $\widehat{=}$

extends *TrainRelease1*

when

grd1 : phase = Train
grd2 : train_pos = SRS
grd3 : release_comm_failure = FALSE
grd4 : release_snd = TRUE
grd5 : communication_ct = FALSE
grd6 : deceleration = FALSE

then

act1 : emrg_brakes := FALSE
act2 : phase := Crossing
act3 : release_rcv := TRUE
act4 : status_req_snd := FALSE

end

Event *TrainRelease2* $\widehat{=}$

extends *TrainRelease2*

when

grd1 : $\mathtt{phase} = \mathtt{Train}$

grd2 : $\mathtt{train\_pos} = \mathtt{SRS}$

grd3 : $\mathtt{release\_comm\_failure} = \mathtt{TRUE}$

grd4 : $\mathtt{release\_snd} = \mathtt{TRUE}$

grd5 : $\mathtt{communication\_ct} = \mathtt{FALSE}$

grd6 : $\mathtt{deceleration} = \mathtt{FALSE}$

then

act1 : $\mathtt{emrg\_brakes} :| \mathtt{emrg\_brakes}' \in \mathtt{BOOL} \land (\mathtt{emrg\_brakes}' = \mathtt{TRUE} \Leftrightarrow \mathtt{emrg\_brakes\_failure} = \mathtt{FALSE})$

act2 : $\mathtt{phase} := \mathtt{Crossing}$

act3 : $\mathtt{release\_rcv} := \mathtt{FALSE}$

act4 : $\mathtt{status\_req\_snd} := \mathtt{FALSE}$

end

Event *TrainStop* $\widehat{=}$

extends *TrainStop*

when

grd1 : $\mathtt{phase} = \mathtt{Train}$

grd2 : $\mathtt{train\_pos} = \mathtt{SRS}$

grd3 : $\mathtt{release\_snd} = \mathtt{FALSE}$

grd4 : $\mathtt{deceleration} = \mathtt{FALSE}$

then

act1 : $\mathtt{emrg\_brakes} :| \mathtt{emrg\_brakes}' \in \mathtt{BOOL} \land (\mathtt{emrg\_brakes}' = \mathtt{TRUE} \Leftrightarrow \mathtt{emrg\_brakes\_failure} = \mathtt{FALSE})$

act2 : $\mathtt{phase} := \mathtt{Crossing}$

act3 : $\mathtt{release\_rcv} := \mathtt{FALSE}$

act4 : $\mathtt{status\_req\_snd} := \mathtt{FALSE}$

end

Event *TrainDangerSpot* $\widehat{=}$

extends *TrainDangerSpot*

when

grd1 : $\mathtt{phase} = \mathtt{Train}$

grd2 : $\mathtt{train\_pos} = \mathtt{DS}$

then

act1 : $\mathtt{phase} := \mathtt{Crossing}$

end

Event $CrossingCloseReq1 \triangleq$

refines $CrossingCloseReq$

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = TRUE$

grd3 : $communication\_tc = FALSE$

grd4 : $closing = FALSE$

grd5 : $close\_comm\_failure = FALSE$

then

act1 : $bar\_1 :| bar\_1' \in BAR\_POS \land (bar\_1' = Closed \Leftrightarrow bar\_failure\_1 = FALSE)$

act2 : $bar\_2 :| bar\_2' \in BAR\_POS \land (bar\_2' = Closed \Leftrightarrow bar\_failure\_2 = FALSE)$

act3 : $close\_req\_rcv := TRUE$

act4 : $communication\_tc := TRUE$

act5 : $phase := Env$

end

Event $CrossingCloseReq2 \triangleq$

refines $CrossingCloseReq$

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = TRUE$

grd3 : $communication\_tc = FALSE$

grd4 : $closing = FALSE$

grd5 : $close\_comm\_failure = TRUE$

then

act1 : $bar\_1, bar\_2 := Opened, Opened$

act2 : $close\_req\_rcv := FALSE$

act3 : $communication\_tc := TRUE$

act4 : $phase := Env$

end

Event $CrossingStatusReq1 \triangleq$

extends $CrossingStatusReq1$

when

grd1 : phase = Crossing

grd2 : $\mathtt{status\_req\_snd} = \mathtt{TRUE}$

grd3 : $\mathtt{close\_req\_rcv} = \mathtt{TRUE}$

grd4 : $\mathtt{sens\_reading} = \mathtt{FALSE}$

grd5 : $\mathtt{communication\_tc} = \mathtt{FALSE}$

then

act1 : $\mathtt{release\_snd} :| \mathtt{release\_snd}' \in \mathtt{BOOL} \land (\mathtt{release\_snd}' = \mathtt{TRUE} \Leftrightarrow (\mathtt{status\_comm\_failure} = \mathtt{FALSE} \land \mathtt{sensor\_1} = \mathtt{Closed} \land \mathtt{sensor\_2} = \mathtt{Closed}))$

act2 : $\mathtt{status\_req\_rcv} :| \mathtt{status\_req\_rcv}' \in \mathtt{BOOL} \land (\mathtt{status\_req\_rcv}' = \mathtt{TRUE} \Leftrightarrow \mathtt{status\_comm\_failure} = \mathtt{FALSE})$

act3 : $\mathtt{communication\_tc} := \mathtt{TRUE}$

act4 : $\mathtt{phase} := \mathtt{Env}$

end

Event *CrossingStatusReq2* $\widehat{=}$

extends *CrossingStatusReq2*

when

grd1 : $\mathtt{phase} = \mathtt{Crossing}$

grd2 : $\mathtt{status\_req\_snd} = \mathtt{TRUE}$

grd3 : $\mathtt{close\_req\_rcv} = \mathtt{FALSE}$

grd4 : $\mathtt{sens\_reading} = \mathtt{FALSE}$

grd5 : $\mathtt{communication\_tc} = \mathtt{FALSE}$

then

act1 : $\mathtt{status\_req\_rcv} :| \mathtt{status\_req\_rcv}' \in \mathtt{BOOL} \land (\mathtt{status\_req\_rcv}' = \mathtt{TRUE} \Leftrightarrow \mathtt{status\_comm\_failure} = \mathtt{FALSE})$

act2 : $\mathtt{release\_snd} := \mathtt{FALSE}$

act3 : $\mathtt{communication\_tc} := \mathtt{TRUE}$

act4 : $\mathtt{phase} := \mathtt{Env}$

end

Event *CrossingIdle* $\widehat{=}$

extends *CrossingIdle*

when

grd1 : $\mathtt{phase} = \mathtt{Crossing}$

grd2 : $\mathtt{close\_req\_snd} = \mathtt{FALSE}$

grd3 : $\mathtt{status\_req\_snd} = \mathtt{FALSE}$

then

act1 : $\mathtt{phase} := \mathtt{Env}$

act2 : $\texttt{release\_snd} := \texttt{FALSE}$

end

Event *ReleaseComm* $\widehat{=}$

extends *ReleaseComm*

when

grd1 : $\texttt{phase} = \texttt{Train}$

grd2 : $\texttt{train\_pos} = \texttt{SRS}$

grd3 : $\texttt{release\_snd} = \texttt{TRUE}$

grd4 : $\texttt{communication\_ct} = \texttt{TRUE}$

then

act1 : $\texttt{release\_comm\_failure} :\in \texttt{BOOL}$

act2 : $\texttt{communication\_ct} := \texttt{FALSE}$

end

Event *TrainDec* $\widehat{=}$

extends *TrainDec*

when

grd1 : $\texttt{phase} = \texttt{Train}$

grd2 : $\texttt{train\_pos} = \texttt{SRS}$

grd3 : $\texttt{deceleration} = \texttt{TRUE}$

then

act1 : $\texttt{emrg\_brakes\_failure} :\in \texttt{BOOL}$

act2 : $\texttt{deceleration} := \texttt{FALSE}$

end

Event *CloseComm* $\widehat{=}$

extends *CloseComm*

when

grd1 : $\texttt{phase} = \texttt{Crossing}$

grd2 : $\texttt{close\_req\_snd} = \texttt{TRUE}$

grd3 : $\texttt{communication\_tc} = \texttt{TRUE}$

then

act1 : $\texttt{close\_comm\_failure} :\in \texttt{BOOL}$

act2 : $\texttt{communication\_tc} := \texttt{FALSE}$

end

Event *StatusComm* $\widehat{=}$

extends *StatusComm*

when

grd1 : `phase` $=$ `Crossing`

grd2 : `status_req_snd` $=$ `TRUE`

grd3 : `communication_tc` $=$ `TRUE`

then

act1 : `status_comm_failure` $:\in$ `BOOL`

act2 : `communication_tc` $:=$ `FALSE`

end

Event *ReadSensors* $\widehat{=}$

refines *ReadSensors*

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $sens\_reading = TRUE$

grd4 : $sensing = FALSE$

then

act1 : $sensor\_1 :| sensor\_1' \in BAR\_POS \land (sensor\_1' = bar\_1 \Leftrightarrow sensor\_failure\_1 = FALSE)$

act2 : $sensor\_2 :| sensor\_2' \in BAR\_POS \land (sensor\_2' = bar\_2 \Leftrightarrow sensor\_failure\_2 = FALSE)$

act3 : $sens\_reading := FALSE$

end

Event *BarStatus* $\widehat{=}$

when

grd1 : $phase = Crossing$

grd2 : $close\_req\_snd = TRUE$

grd3 : $closing = TRUE$

then

act1 : $bar\_failure\_1 :\in BOOL$

act2 : $bar\_failure\_2 :\in BOOL$

act3 : $closing := FALSE$

end

Event $SensorStatus \;\widehat{=}$

when

grd1 : $phase = Crossing$

grd2 : $status\_req\_snd = TRUE$

grd3 : $sensing = TRUE$

then

act1 : $sensor\_failure\_1 :\in BOOL$

act2 : $sensor\_failure\_2 :\in BOOL$

act3 : $sensing := FALSE$

end

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 978-952-12-2263-4
ISSN 1239-1891