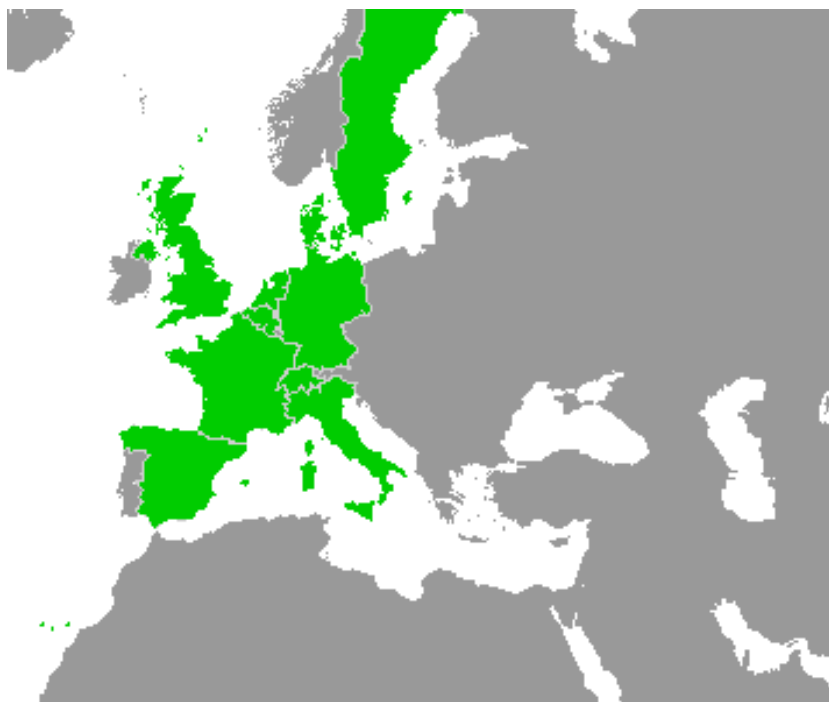


Work-Package 7: “Toolchain”

Event-B Model of Subset 026, Section 5.9

Matthias Güdemann
Systerel, France

April 2013



This page is intentionally left blank

Work-Package 7: “Toolchain”

**OETCS
April 2013**

Event-B Model of Subset 026, Section 5.9

Matthias GÜdemann
Systerel, France
Systerel

Model Description

Prepared for openETCS@ITEA2 Project

Disclaimer: This work is licensed under the "openETCS Open License Terms" (oOLT) dual Licensing: European Union Public Licence (EURL v.1.1+) AND Creative Commons Attribution-ShareAlike 3.0 – (cc by-sa 3.0)

THE WORK IS PROVIDED UNDER openETCS OPEN LICENSE TERMS (oOLT) WHICH IS A DUAL LICENSE AGREEMENT INCLUDING THE TERMS OF THE EUROPEAN UNION PUBLIC LICENSE (VERSION 1.1 OR ANY LATER VERSION) AND THE TERMS OF THE CREATIVE COMMONS PUBLIC LICENSE ("CCPL"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS OLT LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

<http://creativecommons.org/licenses/by-sa/3.0/>

<http://joinup.ec.europa.eu/software/page/eupl/licence-eupl>

Table of Contents

1	Short Introduction to Event-B	5
2	Modeling Strategy	6
3	Model Overview	6
4	Model Benefits	6
5	Detailed Model Description.....	7
5.1	Machine 0 - Basic Flowchart	8
5.2	Context 0 - Train Modes	8
5.3	Machine 1 - Train Modes	9
5.4	Context 1 - Mode Profiles	11
5.5	Machine 2 - Mode Profiles	12
5.6	Machine 3 - Driver Acknowledge	14
5.7	Machine 4 - Timeout.....	17
5.8	Machine 5 - Speed Supervision	18
	References	19

Figures and Tables

Figures

Figure 1. Overview on State Machine and Context Hierarchy	6
Figure 2. Flowchart for "On-Sight" Procedure [Eur12]	7
Figure 3. Basic Flowchart Representation	8
Figure 4. First Refinement with Train Modes	9
Figure 5. Second Refinement	13
Figure 6. Third Refinement with Driver Acknowledge	15
Figure 7. Fourth Refinement State Machine	17

Tables

Table 1. Glossary	5
-------------------------	---

This document describes a formal model of the requirements of section 5.9 of the subset 026 of the ETCS specification 3.3.0 [Eur12]. This section describes the on-sight procedure.

The model is expressed in the formal language Event-B [Abr10] and developed within the Rodin tool [Jas12]. This formalism allows an iterative modeling approach. In general, one starts with a very abstract description of the basic functionality and step-wise adds additional details until the desired level of accuracy of the model is reached. Rodin provides the necessary proof support to ensure the correctness of the refined behavior.

In this document we present an Event-B model of the procedure on-sight. We use the iUML plugin which allows for modeling in UML state-charts to create a graphical model of the procedure which is as close as possible as its description as flowchart in the section 5.9. The state machine is iteratively developed using the refinement feature of Event-B. At each refinement step, we present the reasoning for the step, together with newly introduced variables and events.

For a short introduction on Event-B and the usage of Rodin with models on github see https://github.com/openETCS/model-evaluation/blob/master/model/B-Systemrel/Event_B/rodin-projects-github.pdf?raw=true

Table 1. Glossary

1 Modeling Strategy

The section 5.9 of the SRS describes the procedure on-sight, in particular it describes the sequence of mode changes, necessary driver acknowledge and train brake to enter OS mode, dependent on the current train mode.

For better understanding and to automate many tasks for state based modeling, we use the iUML plugin [?] which automatically generates Event-B code representing a state machine specification.

2 Model Overview

Figure 1 shows the structure of the Event-B model. The left column represents the abstract state machines, the right column the contexts. An arrow from one machine to another machine represents a refinement relation, an arrow from a machine to a context represents a sees relation and arrow from one context to another represents an extension relation.

The modeling starts with the very abstract possibility to establish and to terminate a communication session in the machine $m0$, the set of entities is defined in the context $c0$. This basic functionality is refined in the succeeding machines to incorporate a more detailed description of the flowchart.

3 Model Benefits

The Event-B model in Rodin has some interesting properties which are highlighted here. Some stem from the fact that Rodin is well integrated into the Eclipse platform which renders many useful plugins available, both those explicitly developed for integration with Rodin, but also other without Rodin in mind. Other interesting properties stem from the fact that Rodin and Event-B provide an extensive proof support for properties.

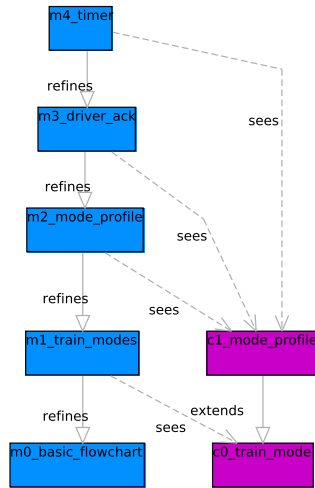


Figure 1. Overview on State Machine and Context Hierarchy

- **Graphical Modeling** Through the iUML plugin, Rodin supports graphical modeling of UML/SysML state machines. Transitions are labeled with events and a fully automatic transformation [SBS09] creates an Event-B representation of the state machine models.
- **Refinement** In addition to the general refinement which is possible in the Event-B approach, the graphical modeling allows to refine the graphical state chart models too. For each refinement step, the new details are graphically emphasized.
- **Model Animation** Through the ProB plugin, the graphical models can be animated just as textual Event-B models. In this case active transitions can be highlighted which helps understand model behavior.
- **Safety Properties** Using Rodin's proof support and the formalization as invariants, it is possible to formalize and prove the identified safety properties of the case study (see Section ??).

4 Detailed Model Description

This section describes in more detail the formal model, beginning from the most abstract Event-B machine. For each refinement, the state machine will be shown and in general only the important manual changes in the model generated from the state machine. The full generated code and the manual changes are available as a Rodin project. At each step the additional modeled functionality and its representation will be described. In particular the initialization event is not shown for the refined machines. If not mentioned explicitly, sets are initialized empty, integers with value 0 and Boolean variables with false.

4.1 Machine 0 - Basic Flowchart

The first state machine $m0$ (see Fig. 3) represents an abstract view of the flowchart describing the on-sight procedure which is shown in §5.9.7 of the SRS [Eur12] (see Fig. 2).

The flowchart is translated into a iUML state machine as follows: the initial state represents the initial situation of the procedure flowchart. The diamonds of the flowchart represent different cases and are therefore into transitions with different target states in the state chart. The nodes of the flowchart are combined for abstraction by combining nodes with multiple incoming flows (or an initial node) with direct successor nodes.

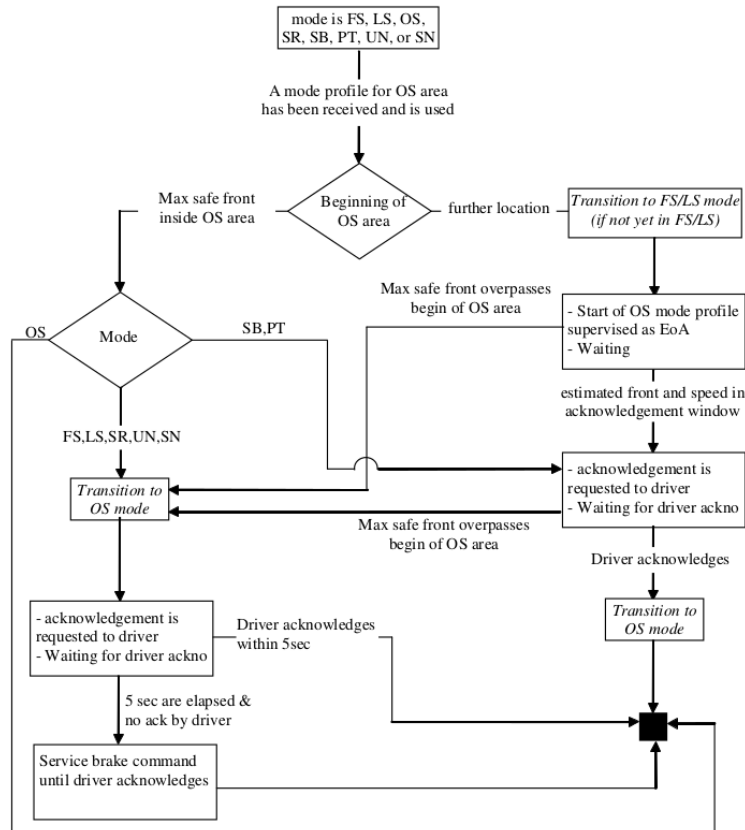


Figure 2. Flowchart for "On-Sight" Procedure [Eur12]

For example the state *ack_and_transition* can be reached from the initial state via the event *use_profile_OS_inside_area_mode_SB_PT* and corresponds to the two lower right nodes of the flowchart. This is justified, as the flow passes two diamonds in the flowchart, verifying that the i) max safe front of the train is inside the OS area and ii) the train mode is *BS* or *PT*. The complete model is automatically generated from this state machine. Note however, that in this abstraction level, there is no concrete notion of train modes, these appear in the first refinement.

The transitions *switch_to_OS_mode* signal the completion of the on-sight procedure, the internal switch to OS mode in the train happens elsewhere. The state *OS_mode* signals the final state.

4.2 Context 0 - Train Modes

The first context *c0* specifies the possible modes of the train, these are of type *t_train_modes*. There is one Event-B constant for each possible mode. The constant *c_initial_mode* represents the initial mode of the train when the procedure on-sight is started. The constant *c_supervision_mode* is one mode from the supervision modes.

SETS

t_train_modes

CONSTANTS

c_FS full supervision
c_LS limited supervision
c_OS on sight
c_SR staff responsible

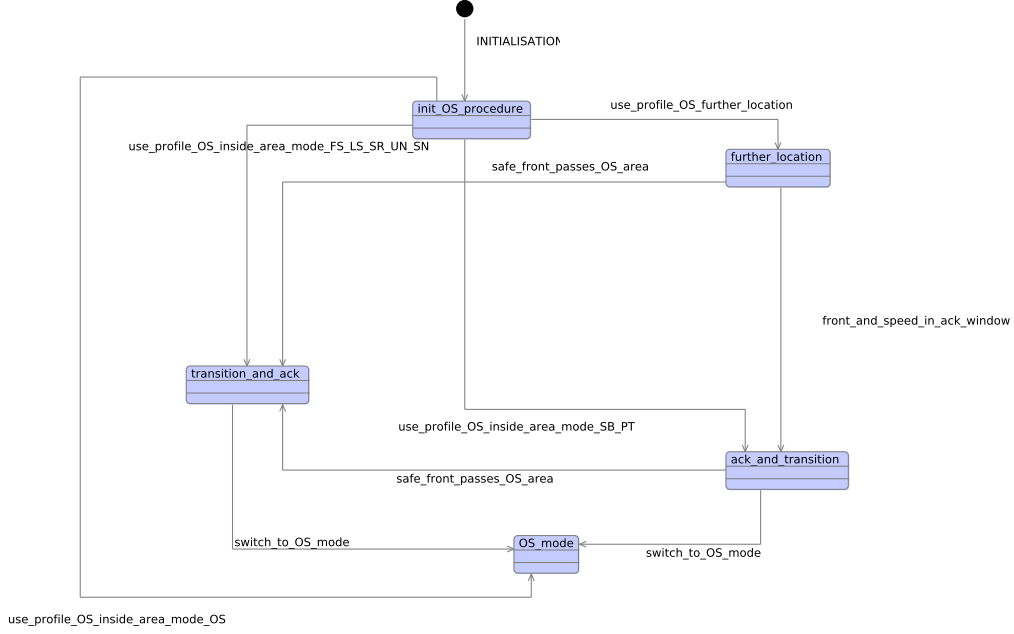


Figure 3. Basic Flowchart Representation

c_{SB} stand-by
 c_{PT} post-trip
 c_{UN} unfitted
 c_{SN} national system
 $c_{initial_mode}$
 $c_{supervision_mode}$

AXIOMS

$axm1 : partition(t_train_modes, \{c_{FS}\}, \{c_{LS}\}, \{c_{OS}\}, \{c_{SR}\}, \{c_{SB}\}, \{c_{PT}\}, \{c_{UN}\}, \{c_{SN}\})$
 $axm2 : c_{initial_mode} \in \{c_{FS}, c_{OS}, c_{PT}\}$
 $axm3 : c_{supervision_mode} \in \{c_{LS}, c_{FS}\}$

END**4.3 Machine 1 - Train Modes**

The first machine refinement adds the variable *current_mode* which tracks the current mode of the train. This variable is initialized with the value of $c_{initial_mode}$.

The state of this variable is used to constrain the guards of the events that depend on the train modes, i.e., corresponding to those that lead from the “Mode” diamond in the flowchart (see Fig. 2). Its state is changed in the *transition_to_supervision_mode* event which assigns the value of $c_{supervision_mode}$ or in the *transition_to_OS_mode* event which assigns the on-sight mode.

The refined state chart is shown in Fig. 4. The state *further_location* is refined to contain three sub-states and two events. This switches the train to supervision mode, and starts EoA supervision. The train stays in this state until either the maximal safe front passes the OS area or the estimated front and speed leave the acknowledge window. The exiting transitions are changed to originate from the *start_OS_profile_wait* state instead of its super-state. This is possible as the sub-states correctly refine the super-state.

MACHINE m1_train_modes

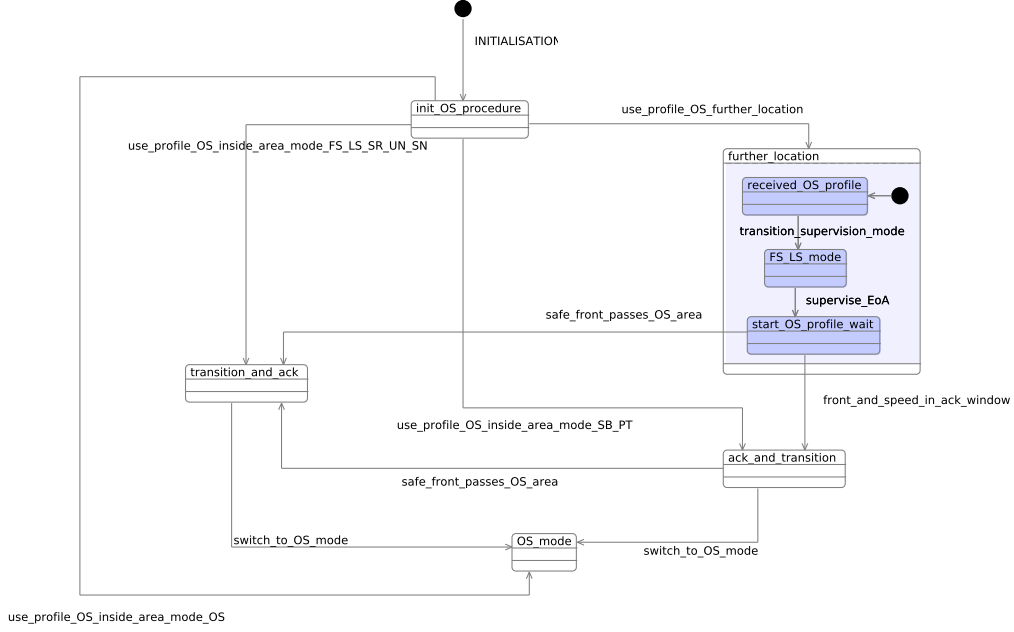


Figure 4. First Refinement with Train Modes

REFINES m0_basic_flowchart

SEES c0_train_mode

VARIABLES

current_mode

INVARIANTS

inv1 : $current_mode \in t_train_modes$

EVENTS

Event *safe_front_passes_OS_area* $\hat{=}$

extends *safe_front_passes_OS_area*

when

isin_ack_and_transition_or_isin_further_location : $ack_and_transition = TRUE \vee$
further_location = TRUE
isin_start_OS_profile_wait : $start_OS_profile_wait = TRUE$

then

enter_transition_and_ack : $transition_and_ack := TRUE$
leave_ack_and_transition : $ack_and_transition := FALSE$
leave_further_location : $further_location := FALSE$
leave_start_OS_profile_wait : $start_OS_profile_wait := FALSE$

end

Event *switch_to_OS_mode* $\hat{=}$

extends *switch_to_OS_mode*

when

isin_ack_and_transition_or_isin_transition_and_ack : $ack_and_transition = TRUE \vee$
transition_and_ack = TRUE

then

leave_ack_and_transition : $ack_and_transition := FALSE$
enter_OS_mode : $OS_mode := TRUE$
leave_transition_and_ack : $transition_and_ack := FALSE$

end

Event *front_and_speed_in_ack_window* $\hat{=}$

extends *front_and_speed_in_ack_window*

when

isin_further_location : $further_location = TRUE$
isin_start_OS_profile_wait : $start_OS_profile_wait = TRUE$

```

    then
        enter_ack_and_transition : ack_and_transition := TRUE
        leave_further_location : further_location := FALSE
        leave_start_OS_profile_wait : start_OS_profile_wait := FALSE
    end
Event use_profile_OS_further_location ≡
extends use_profile_OS_further_location
    when
        isin_init_OS_procedure : init_OS_procedure = TRUE
    then
        leave_init_OS_procedure : init_OS_procedure := FALSE
        enter_further_location : further_location := TRUE
        enter_received_OS_profile : received_OS_profile := TRUE
    end
Event use_profile_OS_inside_area_mode_OS ≡
extends use_profile_OS_inside_area_mode_OS
    when
        isin_init_OS_procedure : init_OS_procedure = TRUE
        grd1 : current_mode = c_OS
    then
        enter_OS_mode : OS_mode := TRUE
        leave_init_OS_procedure : init_OS_procedure := FALSE
    end
Event use_profile_OS_inside_area_mode_SB_PT ≡
extends use_profile_OS_inside_area_mode_SB_PT
    when
        isin_init_OS_procedure : init_OS_procedure = TRUE
        grd1 : current_mode ∈ {c_SB, c_PT}
    then
        enter_ack_and_transition : ack_and_transition := TRUE
        leave_init_OS_procedure : init_OS_procedure := FALSE
    end
Event use_profile_OS_inside_area_mode_FS_LS_SR_UN_SN ≡
extends use_profile_OS_inside_area_mode_FS_LS_SR_UN_SN
    when
        isin_init_OS_procedure : init_OS_procedure = TRUE
        grd1 : current_mode ∈ {c_FS, c_LS, c_SR, c_UN, c_SN}
    then
        leave_init_OS_procedure : init_OS_procedure := FALSE
        enter_transition_and_ack : transition_and_ack := TRUE
    end
Event transition_supervision_mode ≡
    when
        isin_received_OS_profile : received_OS_profile = TRUE
    then
        leave_received_OS_profile : received_OS_profile := FALSE
        act1 : current_mode := c_supervision_mode
        enter_FS_LS_mode : FS_LS_mode := TRUE
    end
Event transition_to_OS_mode ≡
    begin
        act1 : current_mode := c_OS
    end
END

```

4.4 Context 1 - Mode Profiles

This context extension introduces the type $t_mode_profile$ for mode profiles, t_train_fronts for train fronts (e.g., max safe front, estimated front), t_speed for train speed and $t_locations$ for on track locations.

The context also defines several functions, notably one which signals whether a mode profile specifies an OS area, one which signals whether a given train front overpasses the OS area for a specific mode profile, one that signals whether a train front and train speed are in the acknowledge window for a specific mode profile, one that signals whether a given train front is in the OS area of a given mode profile and finally a function that returns the EoA from a given profile.

CONTEXT $c1_mode_profile$

EXTENDS $c0_train_mode$

SETS

$t_mode_profile$

t_train_fronts

t_speed

$t_locations$

CONSTANTS

$f_mode_profile_OS_mode$ indicates whether mode profile demands OS mode

$f_safe_train_front_overpasses$

$f_estimated_train_front_speed_in_window$

$c_profile0$

$f_safe_front_in_OS_area$

$f_EoA_from_profile$

c_loc0

c_front0

AXIOMS

axm1 : $f_mode_profile_OS_mode \in t_mode_profile \rightarrow BOOL$

axm2 : $f_safe_train_front_overpasses \in t_train_fronts \times t_mode_profile \rightarrow BOOL$

train front overpasses begin OS area

axm3 : $f_estimated_train_front_speed_in_window \in t_train_fronts \times t_mode_profile \times t_speed \rightarrow BOOL$

est. train front and speed in ack window

axm4 : $c_profile0 \in t_mode_profile$

axm5 : $f_safe_front_in_OS_area \in t_train_fronts \times t_mode_profile \rightarrow BOOL$

axm6 : $f_EoA_from_profile \in t_mode_profile \rightarrow t_locations$

axm7 : $c_loc0 \in t_locations$

axm10 : $c_front0 \in t_train_fronts$

END

4.5 Machine 2 - Mode Profiles

The second refinement of the machine introduces the notion of mode profiles, train fronts (max safe and estimated) and the end of authority (EoA) location into the model. These are represented by the variables EoA_loc , $mode_profile_OS$, $safe_train_front$ and $estimated_train_front$.

The train fronts can be changed by the events $update_estimated_front$ and $update_safe_front$. The current values of the fronts, the current mode profile and its corresponding EoA are used as parameters for the Boolean functions that guard the events, e.g., for the event $safe_front_passes_OS_area$ or for the event $front_and_speed_in_ack_window$.

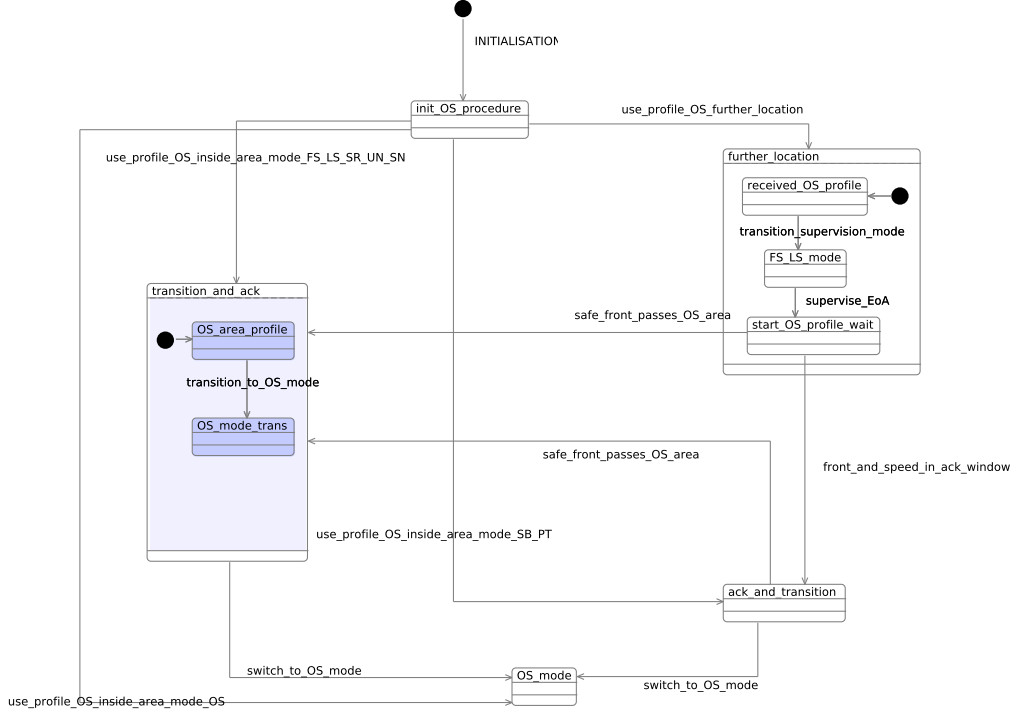


Figure 5. Second Refinement

The second refinement of the state machine is shown in Fig. 5. Here the state *transition_and_ack* is refined with two sub-states. The transition between the two new sub-states is *transition_to_OS_mode* which sets the current train mode to on-sight.

MACHINE m2_mode_profile

REFINES m1_train_modes

SEES c1_mode_profile

VARIABLES

EoA_loc

mode_profile_OS

safe_train_front

estimated_train_front

INVARIANTS

inv1 : *EoA_loc* ∈ *t_locations*

inv2 : *mode_profile_OS* ∈ *t_mode_profile*

inv3 : *safe_train_front* ∈ *t_train_fronts*

inv4 : *estimated_train_front* ∈ *t_train_fronts*

EVENTS

Event *safe_front_passes_OS_area* ≡

extends *safe_front_passes_OS_area*

where

isin_ack_and_transition_or_isin_further_location : *ack_and_transition* = TRUE ∨ *further_location* = TRUE

isin_start_OS_profile_wait : *start_OS_profile_wait* = TRUE

grd1 : *f_safe_train_front_overpasses*(*safe_train_front* ↦ *mode_profile_OS*) = TRUE

grd2 : *l_safe_train_front* ∈ *t_train_fronts*

then

enter_transition_and_ack : *transition_and_ack* := TRUE

leave_ack_and_transition : *ack_and_transition* := FALSE

leave_further_location : *further_location* := FALSE

```

        leave_start_OS_profile_wait : start_OS_profile_wait := FALSE
        enter_OS_area_profile : OS_area_profile := TRUE
    end
Event front_and_speed_in_ack_window ≡
extends front_and_speed_in_ack_window
    any
        l_train_speed
    where
        isin_further_location : further_location = TRUE
        isin_start_OS_profile_wait : start_OS_profile_wait = TRUE
        grd3 : l_train_speed ∈ t_speed
        grd1 : f_estimated_train_front_speed_in_window(estimated_train_front ↦ mode_profile_OS ↦
l_train_speed) = TRUE
    then
        enter_ack_and_transition : ack_and_transition := TRUE
        leave_further_location : further_location := FALSE
        leave_start_OS_profile_wait : start_OS_profile_wait := FALSE
    end
Event update_estimated_front ≡
    any
        l_front
    where
        grd1 : l_front ∈ t_train_fronts
    then
        act1 : estimated_train_front := l_front
    end
Event update_safe_front ≡
    any
        l_front
    where
        grd1 : l_front ∈ t_train_fronts
    then
        act1 : safe_train_front := l_front
    end
END

```

4.6 Machine 3 - Driver Acknowledge

The third machine refinement introduces the driver acknowledgment. In two cases, the driver is asked to acknowledge. This is modeled by additional Boolean variables, two for acknowledging OS mode and two for acknowledging the service brake. Each time, one variable signals that the driver has been informed that he has to acknowledge, e.g., for the service brake this is the *currently_asking_driver_brake_ack* variable, and to signal the completed acknowledge there is the *driver_responded_brake_ack* variable. There is also the Boolean variable *service_brake* which signals the active service brake of the train.

The third refinement of the state machine is shown in Fig. 6. Here the two states *ack_and_transition* and *OS_mode_trans* are refined.

For *ack_and_transition* there are four new sub-states defined. There is first an event to ask the driver to acknowledge the switch to on-sight mode, then the OBU waits for the driver acknowledgment. If the max safe train front passes the OS area without driver acknowledge, then the *transition_and_ack* state is entered, else the train mode is switched to on-sight and the

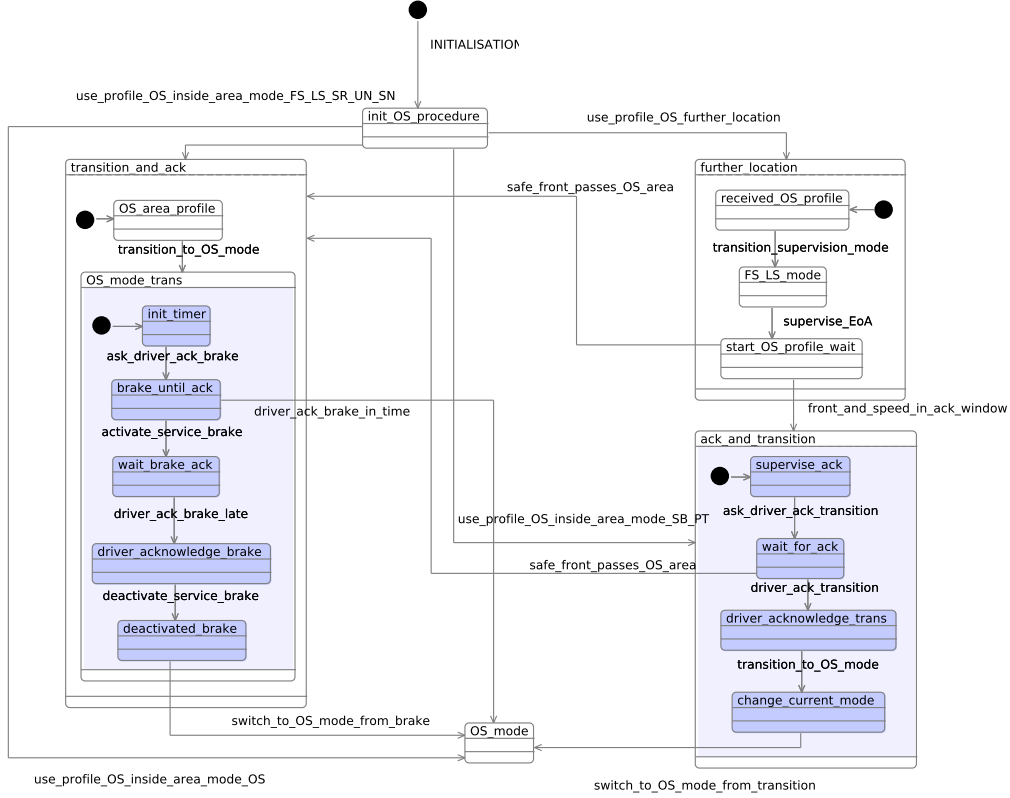


Figure 6. Third Refinement with Driver Acknowledge

final state is entered. Here the outgoing transitions from the abstract *ack_and_transition* state are change to originate from *wait_for_ack* or *change_current_mode* respectively. Again this is possible as the sub-states correctly refine the abstract behavior of the super state.

The abstract *OS_mode_trans* state is refined by five sub-states. First the driver is asked to acknowledge the imminent service brake. If he does so in time, then the procedure finishes and the final state is entered, else the service brake is activated and stays activated until the driver acknowledges and the final state is entered.

At this refinement stage, it is possible to prove that whenever the final state is reached, the mode of the train is on-sight (*inv6*).

MACHINE m3_driver_ack

REFINES m2_mode_profile

SEES c1_mode_profile

VARIABLES

currently_asking_driver_OS_ack
driver_responded_OS_ack
service_brake_active
currently_asking_driver_brake_ack
driver_responded_brake_ack

INVARIANTS

inv6 : $OS_mode = TRUE \Rightarrow current_mode = c_OS$

EVENTS

Event *ask_driver_ack_brake* $\hat{=}$

when


```

    isin_init_timer : init_timer = TRUE
    grd1 : currently_asking_driver_brake_ack = FALSE
  then

    act1 : currently_asking_driver_brake_ack := TRUE
    enter_brake_until_ack : brake_until_ack := TRUE
    act2 : driver_responded_brake_ack := FALSE
    leave_init_timer : init_timer := FALSE
  end
Event ask_driver_ack_transition ≡
  when

    isin_supervise_ack : supervise_ack = TRUE
    grd1 : currently_asking_driver_OS_ack = FALSE
  then

    act1 : currently_asking_driver_OS_ack := TRUE
    leave_supervise_ack : supervise_ack := FALSE
    enter_wait_for_ack : wait_for_ack := TRUE
    act2 : driver_responded_OS_ack := FALSE
  end
Event driver_ack_brake_in_time ≡
  extends switch_to_OS_mode
  when

    isin_ack_and_transition_or_isin_transition_and_ack : ack_and_transition = TRUE ∨
    transition_and_ack = TRUE
    isin_brake_until_ack : brake_until_ack = TRUE
    grd1 : currently_asking_driver_brake_ack = TRUE
  then

    leave_ack_and_transition : ack_and_transition := FALSE
    enter_OS_mode : OS_mode := TRUE
    leave_transition_and_ack : transition_and_ack := FALSE
    leave_OS_mode_trans : OS_mode_trans := FALSE
    leave_OS_area_profile : OS_area_profile := FALSE
    act2 : driver_responded_brake_ack := TRUE
    leave_brake_until_ack : brake_until_ack := FALSE
    act1 : currently_asking_driver_brake_ack := FALSE
  end
Event driver_ack_brake_late ≡
  when

    grd1 : currently_asking_driver_brake_ack = TRUE
    isin_wait_brake_ack : wait_brake_ack = TRUE
  then

    act1 : currently_asking_driver_brake_ack := FALSE
    enter_driver_acknowledge_brake : driver_acknowledge_brake := TRUE
    act2 : driver_responded_brake_ack := TRUE
    leave_wait_brake_ack : wait_brake_ack := FALSE
  end
Event driver_ack_transition ≡
  when

    grd1 : currently_asking_driver_OS_ack = TRUE
    isin_wait_for_ack : wait_for_ack = TRUE
  then

    act2 : driver_responded_OS_ack := TRUE
    enter_driver_acknowledge_trans : driver_acknowledge_trans := TRUE
    leave_wait_for_ack : wait_for_ack := FALSE
    act1 : currently_asking_driver_OS_ack := FALSE
  end
Event activate_service_brake ≡
  when

    grd1 : service_brake_active = FALSE

```

```

    then
        isin_brake_until_ack : brake_until_ack = TRUE
    then
        act1 : service_brake_active := TRUE
        leave_brake_until_ack : brake_until_ack := FALSE
        enter_wait_brake_ack : wait_brake_ack := TRUE
    end
Event deactivate_service_brake ≡
    when
        grd1 : service_brake_active = TRUE
        isin_driver_acknowledge_brake : driver_acknowledge_brake = TRUE
    then
        enter_deactivated_brake : deactivated_brake := TRUE
        act2 : service_brake_active := FALSE
        act3 : driver_responded_brake_ack := FALSE
        leave_driver_acknowledge_brake : driver_acknowledge_brake := FALSE
    end
END

```

4.7 Machine 4 - Timeout

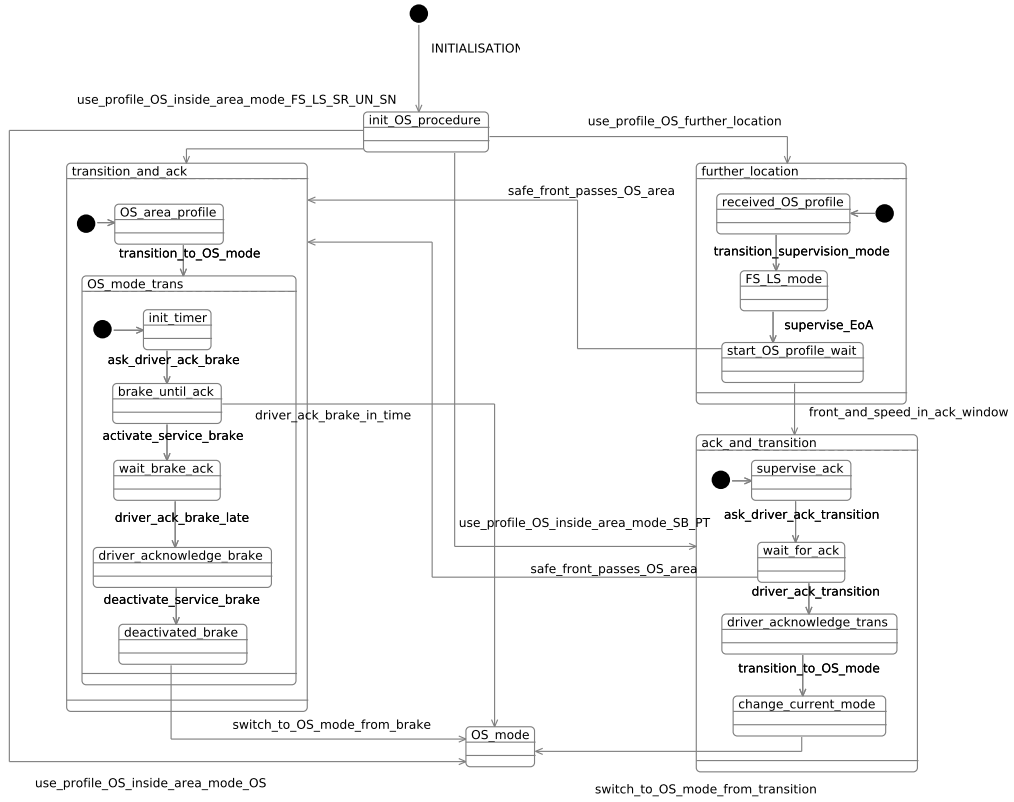


Figure 7. Fourth Refinement State Machine

MACHINE m4_timeout
REFINES m3_driver_ack
SEES c1_mode_profile
VARIABLES

timer_expired
INVARIANTS

```

inv2 :  $timer\_expired = TRUE \wedge OS\_mode\_trans = TRUE \Rightarrow$ 
 $\neg((ack\_and\_transition = TRUE \vee transition\_and\_ack = TRUE) \wedge$ 
 $brake\_until\_ack = TRUE \wedge$ 
 $currently\_asking\_driver\_brake\_ack = TRUE \wedge$ 
 $timer\_expired = FALSE)$ 
    PROPERTY_5.9_02, timer expired and in OS_mode_trans state disables
    other transition than service brake

EVENTS
Event driver_ack_brake_in_time  $\hat{=}$ 
extends driver_ack_brake_in_time
    when
        isin_ack_and_transition_or_isin_transition_and_ack :  $ack\_and\_transition = TRUE \vee$ 
 $transition\_and\_ack = TRUE$ 
        isin_brake_until_ack :  $brake\_until\_ack = TRUE$ 
        grd1 :  $currently\_asking\_driver\_brake\_ack = TRUE$ 
        grd2 :  $timer\_expired = FALSE$ 
    then
        leave_ack_and_transition :  $ack\_and\_transition := FALSE$ 
        enter_OS_mode :  $OS\_mode := TRUE$ 
        leave_transition_and_ack :  $transition\_and\_ack := FALSE$ 
        leave_OS_mode_trans :  $OS\_mode\_trans := FALSE$ 
        leave_OS_area_profile :  $OS\_area\_profile := FALSE$ 
        act2 :  $driver\_responded\_brake\_ack := TRUE$ 
        leave_brake_until_ack :  $brake\_until\_ack := FALSE$ 
        act1 :  $currently\_asking\_driver\_brake\_ack := FALSE$ 
    end
Event driver_ack_brake_late  $\hat{=}$ 
extends driver_ack_brake_late
    when
        grd1 :  $currently\_asking\_driver\_brake\_ack = TRUE$ 
        isin_wait_brake_ack :  $wait\_brake\_ack = TRUE$ 
        grd2 :  $timer\_expired = TRUE$ 
    then
        act1 :  $currently\_asking\_driver\_brake\_ack := FALSE$ 
        enter_driver_acknowledge_brake :  $driver\_acknowledge\_brake := TRUE$ 
        act2 :  $driver\_responded\_brake\_ack := TRUE$ 
        leave_wait_brake_ack :  $wait\_brake\_ack := FALSE$ 
    end
Event activate_service_brake  $\hat{=}$ 
extends activate_service_brake
    when
        grd1 :  $service\_brake\_active = FALSE$ 
        isin_brake_until_ack :  $brake\_until\_ack = TRUE$ 
        grd2 :  $timer\_expired = TRUE$ 
    then
        act1 :  $service\_brake\_active := TRUE$ 
        leave_brake_until_ack :  $brake\_until\_ack := FALSE$ 
        enter_wait_brake_ack :  $wait\_brake\_ack := TRUE$ 
    end
Event expire_timer  $\hat{=}$ 
    when
        grd1 :  $timer\_expired = FALSE$ 
    then
        act1 :  $timer\_expired := TRUE$ 
    end
END

```

4.8 Machine 5 - Speed Supervision

MACHINE m5_speed_supervision

REFINES m4_timeout

SEES c2_speed_limit

VARIABLES

current_speed

INVARIANTS

inv1 : $current_speed \in t_speed$

inv2 : $(driver_acknowledge_brake = TRUE \wedge$
 $f_speed_exceeds(current_speed \mapsto c_OS_speed_limit) = TRUE \wedge$
 $driver_responded_brake_ack = TRUE) \Rightarrow$
 $service_brake_active = TRUE$

PROPERTY_5.9_03 driver acknowledge does not deactivate
 service brake if train speed exceeds OS speed limit

inv10 : $transition_and_ack = TRUE \Rightarrow$
 $((f_speed_exceeds(current_speed \mapsto c_OS_speed_limit) = TRUE \wedge$
 $current_mode = c_OS) \Rightarrow$
 $service_brake_active = TRUE)$
 PROPERTY_5.9_01, brake is activated if mode is OS and current speed exceeds limit

EVENTS

Event deactivate_service_brake $\hat{=}$

extends deactivate_service_brake

when

grd1 : $service_brake_active = TRUE$

isin_driver_acknowledge_brake : $driver_acknowledge_brake = TRUE$

grd2 : $f_speed_exceeds(current_speed \mapsto c_OS_speed_limit) = FALSE$

then

enter_deactivated_brake : $deactivated_brake := TRUE$

act2 : $service_brake_active := FALSE$

act3 : $driver_responded_brake_ack := FALSE$

leave_driver_acknowledge_brake : $driver_acknowledge_brake := FALSE$

end

Event update_train_speed_brake $\hat{=}$

if brake is on new speed cannot exceed current speed

any

l_speed

where

grd1 : $l_speed \in t_speed$

grd2 : $service_brake_active = TRUE$

grd3 : $f_speed_exceeds(l_speed \mapsto current_speed) = FALSE$

grd4 : $init_OS_procedure = TRUE \vee OS_mode = TRUE$

then

act1 : $current_speed := l_speed$

end

Event update_train_speed_no_brake $\hat{=}$

any

l_speed

where

grd1 : $service_brake_active = FALSE$

grd2 : $l_speed \in t_speed$

grd3 : $driver_acknowledge_brake = FALSE$

grd4 : $init_OS_procedure = TRUE \vee OS_mode = TRUE$

then

act1 : $current_speed := l_speed$

end

END

References

- [Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.
- [Eur12] European Railway Agency (ERA). System Requirements Specification - ETCS Subset 026. <http://www.era.europa.eu/Document-Register/Documents/Index00426.zip>, 2012.
- [Jas12] Michael Jastram, editor. *Rodin User's Handbook*. DEPLOY Project, 2012.
- [SBS09] Mar Yah Said, Michael Butler, and Colin Snook. Language and tool support for class and state machine refinement in uml-b. In *FM 2009: Formal Methods*, pages 579–595. Springer, 2009.