OETCS

openETCS

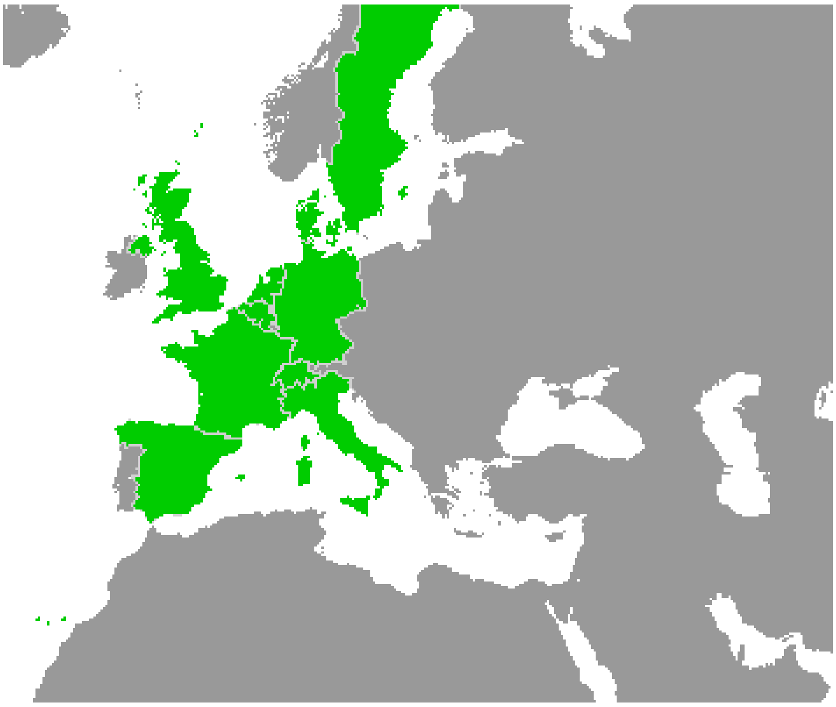
ITEA2 Project
2012 – 2015

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

Event-B Model of Subset 026, Section 5.9

Matthias Güdemann
Systerel, France

Systerel

Model Description

Prepared for ITEA2 openETCS consortium
Europa

Table of Contents

Figures and Tables **Figures**

Tables

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.

Table 1. Glossary

1 Short Introduction to Event-B

The formal language Event-B is based on a set-theoretic approach. It is a variant of the B language, with a focus on system level modeling [Abr10]. An Event-B model is separated into a static and a dynamic part.

The dynamic part of an Event-B model describes abstract state machines. The state is represented by a set of state variables. A transition from one state to another is represented by parametrized events which assign new values to the state variables. Event-B allows unbounded state spaces. They are constrained by invariants expressed in first order logic with equality which must be fulfilled in any case. The initial state is created by a special initialization event.

The static part of an Event-B model is represented by contexts. These consist of carrier sets, constants and axioms. The type system of a model is described by means of carrier sets and constraints expressed by axioms.

Event-B is not only comprised of descriptions of abstract state machines and contexts, but also includes a development approach. This approach consists of iterative refinement of the machines until the desired level of detail is reached. In the Rodin tool, proof obligations are automatically created which ensure correct refinement.

Together with the machine invariants, the proof obligations for the refinement are formally proven, creating proof trees. To accomplish this, there are different options: many proof obligations can be discharged by automated provers (e.g., AtelierB, NewPP, Rodin's SMT-plugin), but as the underlying logic is in general undecidable, it is sometimes necessary to use the interactive proof support of Rodin.

Any external actions, e.g., mode changes by the driver or train level changes are modeled via parametrized events. Only events can modify the variables of a machine. An Event-B model is on the system level, events are assumed to be called from a software system into which the functional model is embedded. The guards of the events assure that any event can only be called when appropriate.

2 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.

3 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.

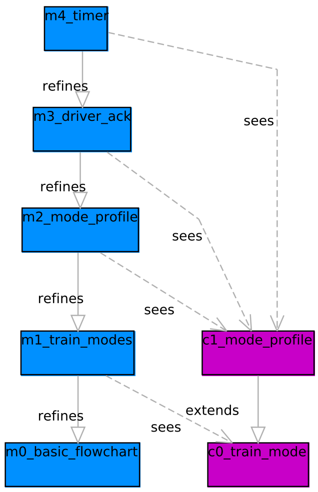


Figure 1. Overview on State Machine and Context Hierarchy

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.

4 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.

- **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 **??**).

5 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.

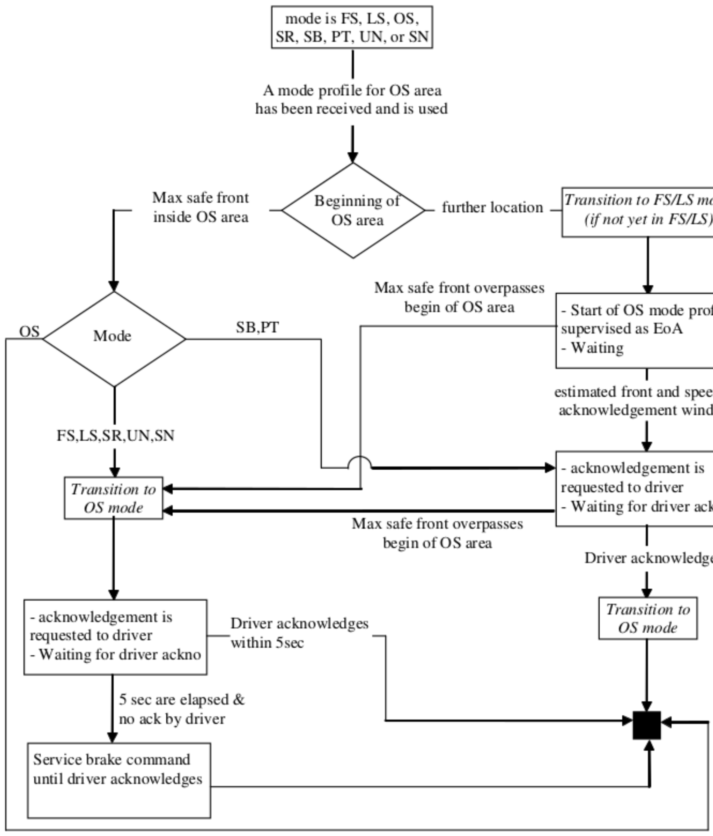


Figure 2. Flowchart for "On-Sight" Procedure [Eur12]

5.1 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.

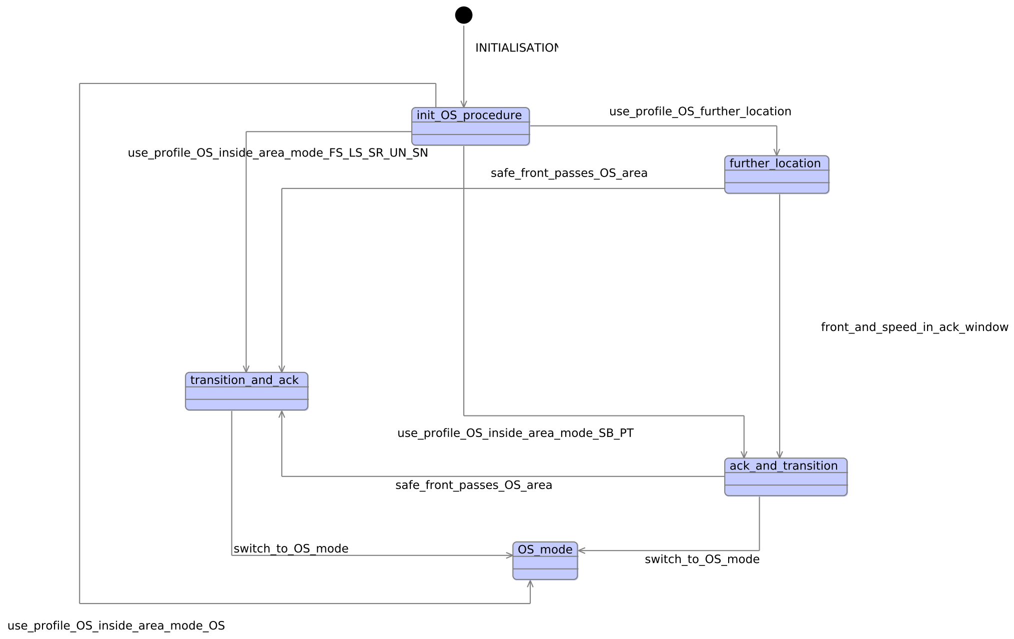


Figure 3. Basic Flowchart Representation

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.

5.2 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
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

5.3 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.

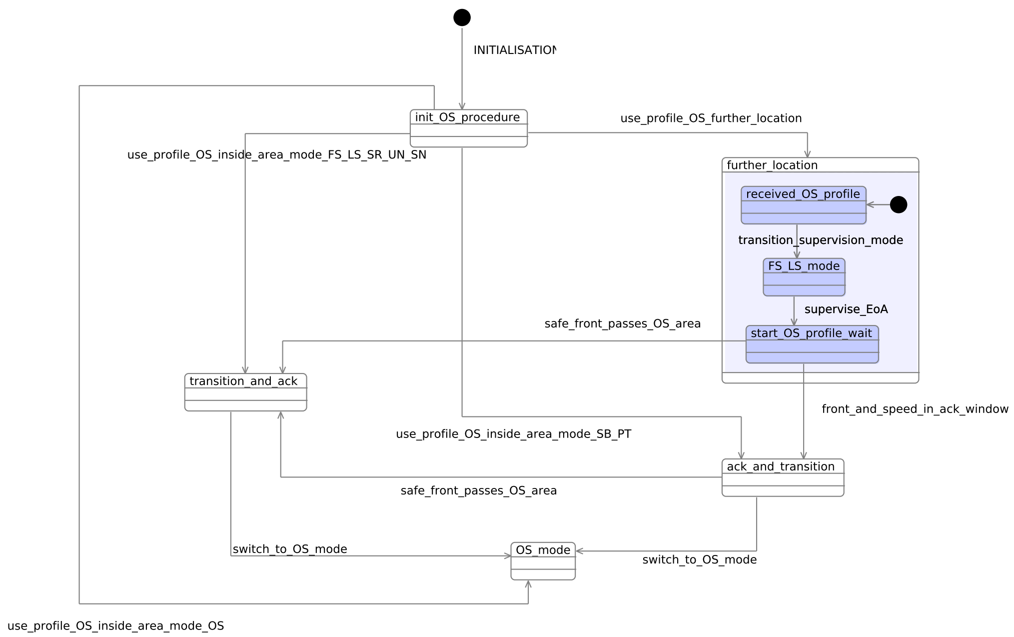


Figure 4. First Refinement with Train Modes

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.

MACHINE m1_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* $\widehat{=}$

extends *safe_front_passes_OS_area*

when

isin_ack_and_transition_or_isin_further_location : $ack\_and\_transition = TRUE \lor further\_location = TRUE$

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* $\widehat{=}$

extends *switch_to_OS_mode*

when

isin_ack_and_transition_or_isin_transition_and_ack : $ack\_and\_transition = TRUE \lor transition\_and\_ack = TRUE$

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* $\widehat{=}$

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* $\widehat{=}$

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* $\widehat{=}$

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* $\widehat{=}$
extends *use_profile_OS_inside_area_mode_SB_PT*
when
isin_init_OS_procedure : $init\_OS\_procedure = TRUE$
grd1 : $current\_mode \in \{c\_SB, c\_PT\}$
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* $\widehat{=}$
extends *use_profile_OS_inside_area_mode_FS_LS_SR_UN_SN*
when
isin_init_OS_procedure : $init\_OS\_procedure = TRUE$
grd1 : $current\_mode \in \{c\_FS, c\_LS, c\_SR, c\_UN, c\_SN\}$
then
leave_init_OS_procedure : $init\_OS\_procedure := FALSE$
enter_transition_and_ack : $transition\_and\_ack := TRUE$
end
Event *transition_supervision_mode* $\widehat{=}$
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* $\widehat{=}$
begin
act1 : $current\_mode := c\_OS$
end
END

5.4 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 : $c\_front0 \in t\_train\_fronts$
axm13 : $\forall front, profile \cdot front \in t\_train\_fronts \wedge profile \in t\_mode\_profile \Rightarrow (f\_safe\_train\_front\_overpasses(front \mapsto profile) = TRUE \Rightarrow (\forall speed \cdot speed \in t\_speed \Rightarrow f\_estimated\_train\_front\_speed\_in\_window(front \mapsto profile \mapsto speed) = FALSE))$
axm14 : $\forall front, profile \cdot front \in t\_train\_fronts \wedge profile \in t\_mode\_profile \Rightarrow (f\_safe\_train\_front\_overpasses(front \mapsto profile) = FALSE \Rightarrow (\exists speed \cdot speed \in t\_speed \Rightarrow f\_estimated\_train\_front\_speed\_in\_window(front \mapsto profile \mapsto speed) = TRUE))$

axm15 : $\forall profile \cdot profile \in t\_mode\_profile \Rightarrow (\exists front1, front2 \cdot front1 \in t\_train\_fronts \wedge front2 \in t\_train\_fronts \Rightarrow (f\_safe\_front\_in\_OS\_area(front1 \mapsto profile) \neq f\_safe\_front\_in\_OS\_area(front2 \mapsto profile)))$

for each profile there are fronts before and inside the OS area

END

5.5 Machine 2 - Mode Profiles

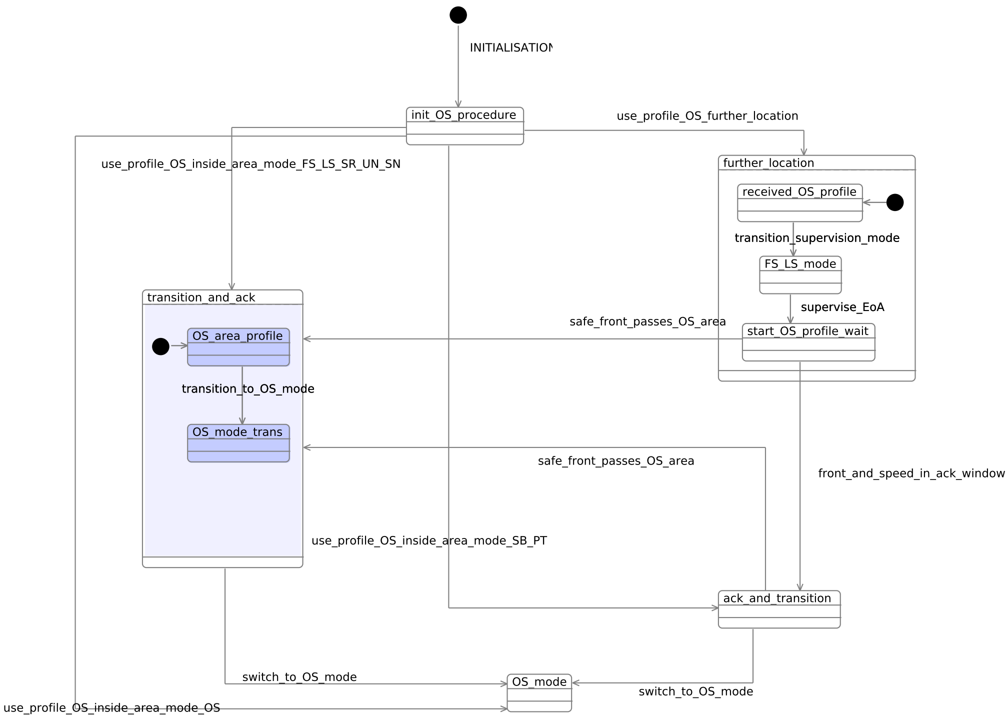


Figure 5. Second Refinement

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.