

Work-Package 7: "Toolchain"

Event-B Model of Subset 026, Section 4.6

Matthias Güdemann

April 2013



Funded by:


 Federal Ministry
 of Education
 and Research

 Région de
 Bruxelles-
 Capitale

 GOBIERNO DE ESPAÑA
 MINISTERIO DE INDUSTRIA, ENERGÍA
 Y TURISMO

This page is intentionally left blank

Work-Package 7: “Toolchain”

**OETCS
April 2013**

Event-B Model of Subset 026, Section 4.6

Matthias Güdemann

Systemel
Les Portes de l'Arbois, Bâtiment A
1090 rue René Descartes
13857 Aix-en-Provence Cedex 3, France

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	Modeling Strategy	5
2	Model Overview	5
3	Model Benefits	6
4	Detailed Model Description.....	7
4.1	Machine 0 - Mode Transitions	7
4.2	Context 1 - Train	9
4.3	Machine 1 - ERTMS Level	9
4.4	Machine 2 - Train Data	11
4.5	Machine 3 - Driver	12

Figures and Tables

Figures

Figure 1. Machine Refinements and Contexts 6

Figure 2. Statemachine for Machine 0 7

Figure 3. Statemachine for Machine 1 10

Figure 4. Statemachine for Machine 3 13

Tables

Table 1. Glossary 5

This document describes a formal model of the requirements of section 4.6 of the subset 026 of the ETCS specification 3.3.0 [2]. This section describes the transition table between the different train modes. This includes the transition conditions and their respective priorities.

The model is expressed in the formal language Event-B [3] and developed within the Rodin tool [4]. 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 transitions from stand-by to shunting, from stand-by to full supervision and from stand-by to isolation. The transition table is expressed as a graphical model of a state machine which is automatically translated into Event-B code.

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

SB	stand-by
FS	full supervision
IS	isolation
SH	shunting
MA	movement authority
SRS	system requirements specification

Table 1. Glossary

1 Modeling Strategy

The description of the section 4.6. of the SRS consists of a large table which describes the possible transitions from one mode to another and the necessary preconditions of a transition. The table also specifies different priorities and the relative priorities of the transitions, i.e., to ensure an explicit precedence if more than one condition is enabled.

The model view consists of a state machine representing the current state, external events can trigger the prerequisites for the conditions. Priorities are modeled by negating a condition with a higher priority for a conditions with a lower priority. For example, let q and p be preconditions that can be fulfilled at the same time, q for event evt_q and p for event evt_p . Let q have a higher priority than p . In Event-B this precedence can be modeled by using q as condition for evt_q , and $\neg q \wedge p$ for evt_p , i.e., the preconditions for the two events cannot be fulfilled at the same time.

The state machine itself is modeled using the iUML statemachine plugin¹ which allows graphical modeling of state machines. The model start with the basic possibilities for the transitions between the modes. The conditions are iteratively refined and external events added to the model.

¹http://wiki.event-b.org/index.php/Event-B_Statemachines

2 Model Overview

Figure 1 shows the model overview. The left column shows the different state machines and the right column the context. An arrow from one machine to another represents a refinement relation, an arrow from a machine to a context represents a sees relation.

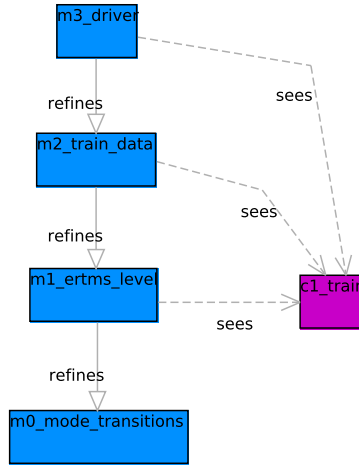


Figure 1. Machine Refinements and Contexts

The model starts with the principal possibilities for the transitions from mode SB to FS, SB to IS and SB to SH. This is realized in the machine *m0*. The refinement in *m1* implements the dependency on the train behavior and ETCS levels which are introduced in the context *c1*. The refined machine *m2* allows for storing abstract train data and gradient information for a movement authority. The refinement in *m3* finally introduces the external driver who can trigger events.

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.

- **Refinement** The Event-B approach allows iterative development based on refinement. This allows starting modeling with a very abstract machine and then step-wise adding more detailed behavior. Rodin generates all the necessary proof obligations which are required to assure correct refinement.
- **Requirements Tracing** Rodin provides an extensible EMF model, therefore it is easily possible to trace requirements using the requirements modeling framework of Eclipse (RMF) via the ProR plugin. This allows the usage of requirement documents in the OMG standardized Requirements Interchange Format (ReqIF).
- **Model Animation** The Event-B model can be animated via different plugins, e.g., ProB or AnimB. This allows the simulation of the model, by clicking on the activated events and tracking the resulting state of the variables. This technique allows to examine the run-time behavior of the model, e.g., for testing purposes. There is also ongoing development for a model-based testing plugin in Rodin, which will allow storing and replaying of event sequences.

- **Graphical Modeling** Rodin supports graphical modeling via its plugin mechanism. The iUML statemachine plugin allows for graphical state machine modeling and animation via ProB. The graphical model is automatically translated into the Event-B language.

4 Detailed Model Description

This section describes the formal model in more detail. For each refinement the new state variables are introduced and their meaning is explained. The machines are not fully presented, only the relevant changes done in the refinement.

The modes and transitions between the modes are modeled graphically using the iUML statemachine plugin. In this notation, transitions are labeled with events, i.e., the transitions are active if the preconditions of the respective events are fulfilled.

In the graphical model, it is possible to have multiple events triggering a transition, the semantics of this is the disjunction of the conditions, i.e., any single event can trigger the transition.

4.1 Machine 0 - Mode Transitions

The first machine $m0$ implements the transition possibilities for the states SB , SH , FS and IS . It is shown in Figure 2. The transition labels represent events of the Event-B machine. The initial state is SB , this is represented by the *INITIALISATION* event from the UML initial state.

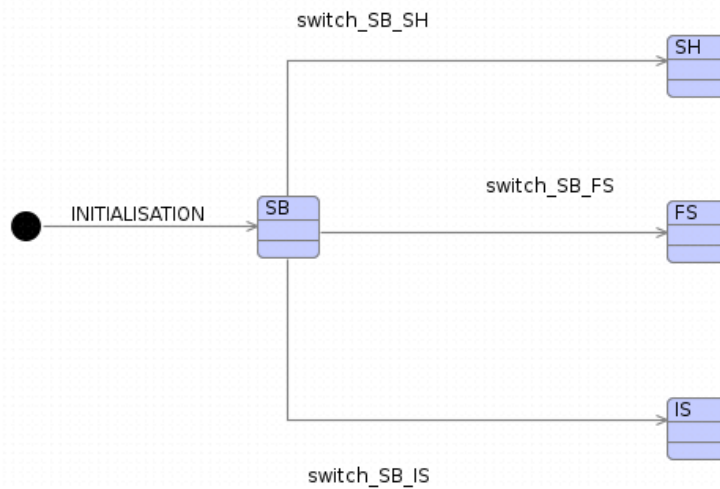


Figure 2. Statemachine for Machine 0

The state machine is translated into Event-B code. Every state is represented as a Boolean variable. An invariant ensures that there is always only a single variable with the value “TRUE”, i.e., the state machine is always in exactly one state.

A transition will use these Boolean variables to analyze whether its source state is active. A state change is encoded by changing the values of the variables corresponding to the source and target state of the transition. For more details on the possible encoding see ².

²http://wiki.event-b.org/index.php/Event-B_Statemachines

Implemented Requirements

- §4.6.2 transition from *SB* to *SH*
- §4.6.2 transition from *SB* to *FS*
- §4.6.2 transition from *SB* to *IS*

VARIABLES

SB

SH

FS

IS

INVARIANTS

typeof_SB : *SB* ∈ *BOOL*

typeof_SH : *SH* ∈ *BOOL*

typeof_FS : *FS* ∈ *BOOL*

typeof_IS : *IS* ∈ *BOOL*

distinct_states_in_mode_changes1 : $\text{partition}(\{TRUE\}, \{SB\} \cap \{TRUE\}, \{SH\} \cap \{TRUE\}, \{FS\} \cap \{TRUE\}, \{IS\} \cap \{TRUE\})$

EVENTS

Initialisation

begin

init_IS : *IS* := *FALSE*

init_SB : *SB* := *TRUE*

init_SH : *SH* := *FALSE*

init_FS : *FS* := *FALSE*

end

Event *switch_SB_SH* $\hat{=}$

when

then *isin_SB* : *SB* = *TRUE*

enter_SH : *SH* := *TRUE*

leave_SB : *SB* := *FALSE*

end

Event *switch_SB_FS* $\hat{=}$

when

then *isin_SB* : *SB* = *TRUE*

enter_FS : *FS* := *TRUE*

leave_SB : *SB* := *FALSE*

end

Event *switch_SB_IS* $\hat{=}$

when

then *isin_SB* : *SB* = *TRUE*

```

        enter_IS : IS := TRUE
        leave_SB : SB := FALSE
    end
END

```

4.2 Context 1 - Train

The first context introduces the notion of the different ETCS levels. It also introduces a very abstract notion of the behavior of a train, either moving or at standstill. At this model level, more detailed information is not necessary.

SETS

```

    train_behavior
    ERTMS_level

```

CONSTANTS

```

    L0
    L1
    L2
    L3
    NTC
    standstill
    moving

```

AXIOMS

```

    axm1 : partition(train_behavior, {standstill}, {moving})
    axm2 : partition(ERTMS_level, {NTC, L0, L1, L2, L3})
END

```

4.3 Machine 1 - ERTMS Level

The first refined machine uses the ETCS levels with the state variable *current_level* and the current train behavior with the state variable *current_behavior*. The ETCS level *NTC* is the initial level and the initial behavior is *standstill*. These variables are modified by their corresponding *change_*-events.

The level and the behavior are used to refine the transition from *SB* to *SH* into two different cases, dependent on the current ETCS level. This is shown in the transition label in Figure 3.

REFINES m0_mode_transitions

SEES c1_train

VARIABLES

```

    current_level
    current_behavior

```

INVARIANTS

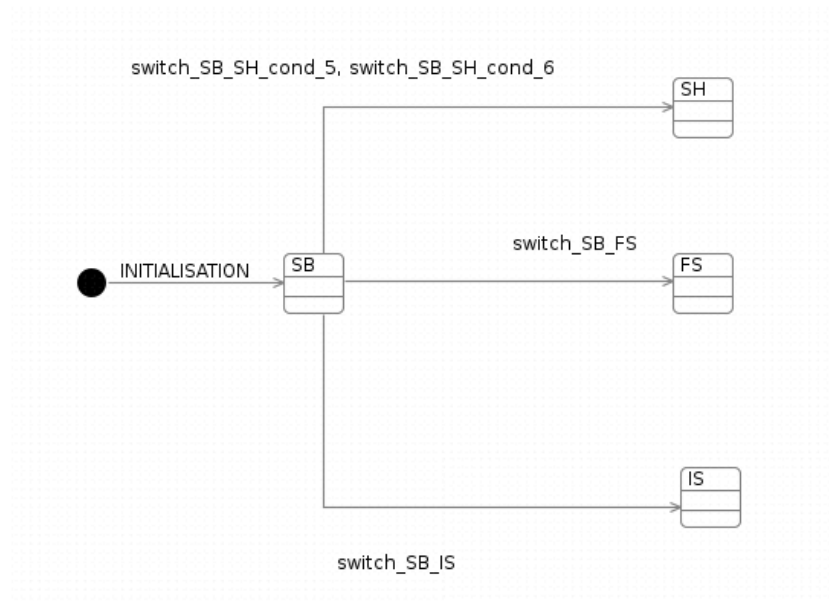


Figure 3. Statemachine for Machine 1

```

    inv1 : current_level ∈ ERTMS_level
    inv2 : current_behavior ∈ train_behavior
EVENTS
Initialisation
    extended
    begin
        act1 : current_level := NTC
        act2 : current_behavior := standstill
    end
Event change_level ≡
    any
    where
        l_level
    then
        grd1 : l_level ∈ ERTMS_level
        act1 : current_level := l_level
    end
Event change_behavior ≡
    any
    where
        l_behavior
    then
        grd1 : l_behavior ∈ train_behavior
        act1 : current_behavior := l_behavior
    end
Event switch_SB_SH_cond_5 ≡
extends switch_SB_SH
    when

```

```

    isin_SB : SB = TRUE
    grd2 : current_level ∈ {NTC, L0, L1}
    grd1 : current_behavior = standstill
  then

    enter_SH : SH := TRUE
    leave_SB : SB := FALSE
  end
Event switch_SB_SH_cond_6 ≡
extends switch_SB_SH
  when

    isin_SB : SB = TRUE
    grd1 : current_behavior = standstill
    grd2 : current_level ∈ {L2, L3}
  then

    enter_SH : SH := TRUE
    leave_SB : SB := FALSE
  end
END

```

4.4 Machine 2 - Train Data

The second machine refinement adds the notion of valid train data and movement authority gradient data. The validity of these two different kinds of data is represented by the Boolean variables *valid_train_data* and *MA_SSP_gradient_data*. Both variables can be modified by two events; one that stores valid data and one that deletes, i.e., invalidates the stored data.

Having valid MA and train data is the precondition for the transition from *SB* to *FS*. This transition is refined by guard strengthening.

```

REFINES m1_ertms_level
SEES c1_train
VARIABLES

  valid_train_data
  MA_SSP_gradient_data
INVARIANTS

  inv1 : valid_train_data ∈ BOOL
  inv2 : MA_SSP_gradient_data ∈ BOOL
EVENTS
Event store_valid_train_data ≡
  when

    grd1 : valid_train_data = FALSE
  then

    act1 : valid_train_data := TRUE
  end
Event remove_valid_train_data ≡

```

```

when

  then   grd1 : valid_train_data = TRUE

  end    act1 : valid_train_data := FALSE

Event store_MA_SSP_gradient_data  $\hat{=}$ 
  when

    then   grd1 : MA_SSP_gradient_data = FALSE

    end    act1 : MA_SSP_gradient_data := TRUE

Event remove_MA_SSP_gradient_data  $\hat{=}$ 
  when

    then   grd1 : MA_SSP_gradient_data = TRUE

    end    act1 : MA_SSP_gradient_data := FALSE

extends switch_SB_FS
  when

    isin_SB : SB = TRUE
    grd1 : MA_SSP_gradient_data = TRUE
    grd2 : valid_train_data = TRUE

  then

    enter_FS : FS := TRUE
    leave_SB : SB := FALSE

  end
END

```

4.5 Machine 3 - Driver

The next machine refinement adds the behavior of the driver and some abstract behavior of the RBC to the model, as well as the notion of a specific mode profile.

In this machine the transition from *SB* to *SH* is refined to a third possibility, the switch from *SB* to *FS* is refined with the required conditions of the driver and mode profile data and the high priority of the transition from *SB* to *IS* is taken into account. Figure 4 shows the corresponding state machine.

The RBC can grant a shunting request and display the information to acknowledge shunting. This is represented by the Boolean state variables *shunting_granted_RBC* and *display_shunting_ack*. The driver can isolate the ETCS system, he can manually select shunting, execute a shunting request and acknowledge when the train switches to shunting. This is represented by the Boolean state variables *driver_select_shunting*, *driver_request_shunting*, *driver_ack_shunting* and *driver_isolates_ETCS*.

These variables are set as active by events representing the respective actions by the RBC or the driver. Events which have one of these actions as precondition will reset the events, e.g., displaying the shunting acknowledge (*display_shunting_ok*) needs a previous request as precondition and will reset the variable *driver_request_shunting* when it is executed.

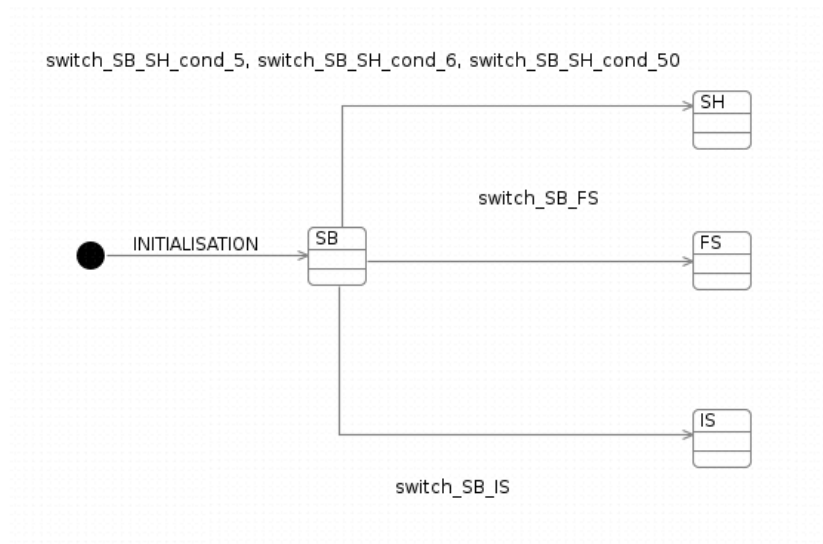


Figure 4. Statemachine for Machine 3

Implemented Requirements

- §4.6.2 condition [5]
- §4.6.2 condition [6]
- §4.6.2 condition [10]
- §4.6.2 condition [50]
- priority of *SB* to *IS* over *SB* to *SH* and *SB* to *FS*

NOTE: The invariant, that the preconditions of the transitions with the same priorities are mutually exclusive, does not hold, i.e., $([5] \vee [6] \vee [50]) \wedge [10] \neq FALSE$. Either this is an error in the specification or there is an underlying functional dependency which is not captured if these 3 transitions are modeled in isolation.

REFINES m2_train_data

SEES c1_train

VARIABLES

driver_select_shunting
driver_request_shunting
driver_ack_shunting
display_shunting_ack
shunting_granted_RBC
driver_isolates_ETCS
specific_mode_profile

INVARIANTS

inv1 : *driver_select_shunting* ∈ *BOOL*
inv2 : *driver_request_shunting* ∈ *BOOL*

```

inv3 : driver_ack_shunting ∈ BOOL
inv4 : display_shunting_ack ∈ BOOL
inv5 : shunting_granted_RBC ∈ BOOL
inv6 : driver_isolates_ETCS ∈ BOOL
inv7 : SB = TRUE ⇒
    ¬((current_level ∈ {NTC, L0, L1} ∧ current_behavior = standstill ∧
    driver_select_shunting = TRUE)
    ∧ (MA_SSP_gradient_data = TRUE ∧ valid_train_data = TRUE ∧
    specific_mode_profile = TRUE))
    not cond5 and cond10
inv8 : SB = TRUE ⇒
    ¬((current_behavior = standstill ∧ current_level ∈ {L2, L3})
    ∧ (MA_SSP_gradient_data = TRUE ∧ valid_train_data = TRUE ∧
    specific_mode_profile = TRUE))
    not cond6 and cond10
inv9 : SB = TRUE ⇒
    ¬((display_shunting_ack = TRUE ∧ driver_ack_shunting = TRUE)
    ∧ (MA_SSP_gradient_data = TRUE ∧ valid_train_data = TRUE ∧
    specific_mode_profile = TRUE))
    not cond50 and cond10
inv10 : specific_mode_profile ∈ BOOL

```

EVENTS

Initialisation

extended

begin

```

init_IS : IS := FALSE
init_SB : SB := TRUE
init_SH : SH := FALSE
init_FS : FS := FALSE
act1 : current_level := NTC
act2 : current_behavior := standstill
act3 : valid_train_data := FALSE
act4 : MA_SSP_gradient_data := FALSE
act7 : driver_ack_shunting := FALSE
act9 : shunting_granted_RBC := FALSE
act6 : driver_request_shunting := FALSE
act10 : driver_isolates_ETCS := FALSE
act5 : driver_select_shunting := FALSE
act11 : specific_mode_profile := FALSE
act8 : display_shunting_ack := FALSE

```

end

Event *change_specific_mode_profile* ≡

any

l_flag

where

grd1 : *l_flag* ∈ *BOOL*

then

act1 : specific_mode_profile := *l_flag*

end


```

Event driver_isolates_ETCS  $\widehat{=}$ 
  when

  then    grd1 : driver_isolates_ETCS = FALSE

  end      act1 : driver_isolates_ETCS := TRUE
Event driver_select_shunting  $\widehat{=}$ 
  when

  then    grd1 : current_level  $\in$  {NTC, L0, L1}

  end      act1 : driver_select_shunting := TRUE
Event driver_request_shunting  $\widehat{=}$ 
  when

  then    grd1 : current_level  $\in$  {L2, L3}

  end      act1 : driver_request_shunting := TRUE
Event display_shunting_ack  $\widehat{=}$ 
                                     train acknowledges shunting -> specific mode required?
  when

  then    grd1 : driver_request_shunting = TRUE
          grd2 : display_shunting_ack = FALSE

  end      act1 : display_shunting_ack := TRUE
          act2 : driver_request_shunting := FALSE
          act3 : shunting_granted_RBC := TRUE
Event driver_ack_shunting  $\widehat{=}$ 
  when

  then    grd1 : display_shunting_ack = TRUE

  end      act1 : driver_ack_shunting := TRUE
Event store_valid_train_data  $\widehat{=}$ 
extends store_valid_train_data
  when

  then    grd1 : valid_train_data = FALSE

  end      act1 : valid_train_data := TRUE
Event remove_valid_train_data  $\widehat{=}$ 
extends remove_valid_train_data
  when

  then    grd1 : valid_train_data = TRUE

```

```

        act1 : valid_train_data := FALSE
    end
Event store_MA_SSP_gradient_data ≡
extends store_MA_SSP_gradient_data
    when

        grd1 : MA_SSP_gradient_data = FALSE

    then

        act1 : MA_SSP_gradient_data := TRUE

    end
Event remove_MA_SSP_gradient_data ≡
extends remove_MA_SSP_gradient_data
    when

        grd1 : MA_SSP_gradient_data = TRUE

    then

        act1 : MA_SSP_gradient_data := FALSE

    end
Event change_level ≡
extends change_level
    any

        l_level

    where

        grd1 : l_level ∈ ERTMS_level

    then

        act1 : current_level := l_level

    end
Event change_behavior ≡
extends change_behavior
    any

        l_behavior

    where

        grd1 : l_behavior ∈ train_behavior

    then

        act1 : current_behavior := l_behavior

    end
Event switch_SB_SH_cond_5 ≡
extends switch_SB_SH_cond_5
    when

        isin_SB : SB = TRUE
        grd2 : current_level ∈ {NTC, L0, L1}
        grd1 : current_behavior = standstill
        grd4 : driver_isolates_ETCS = FALSE
        grd3 : driver_select_shunting = TRUE

    then

        enter_SH : SH := TRUE
        leave_SB : SB := FALSE
        act1 : driver_select_shunting := FALSE

    end

```

```

Event switch_SB_SH_cond_6  $\widehat{=}$ 
extends switch_SB_SH_cond_6
  when
    isin_SB : SB = TRUE
    grd1 : current_behavior = standstill
    grd2 : current_level  $\in$  {L2, L3}
    grd3 : driver_isolates_ETCS = FALSE
  then
    enter_SH : SH := TRUE
    leave_SB : SB := FALSE
    act1 : shunting_granted_RBC := FALSE
    act2 : display_shunting_ack := FALSE
  end
Event switch_SB_SH_cond_50  $\widehat{=}$ 
extends switch_SB_SH
  when
    isin_SB : SB = TRUE
    grd2 : driver_ack_shunting = TRUE
    grd1 : display_shunting_ack = TRUE
    grd3 : driver_isolates_ETCS = FALSE
  then
    enter_SH : SH := TRUE
    leave_SB : SB := FALSE
    act2 : driver_ack_shunting := FALSE
    act1 : display_shunting_ack := FALSE
  end
Event switch_SB_FS  $\widehat{=}$ 
extends switch_SB_FS
  when
    isin_SB : SB = TRUE
    grd1 : MA_SSP_gradient_data = TRUE
    grd2 : valid_train_data = TRUE
    grd4 : specific_mode_profile = FALSE
    grd3 : driver_isolates_ETCS = FALSE
  then
    enter_FS : FS := TRUE
    leave_SB : SB := FALSE
  end
Event switch_SB_IS  $\widehat{=}$ 
extends switch_SB_IS
  when
    isin_SB : SB = TRUE
    grd1 : driver_isolates_ETCS = TRUE
  then
    enter_IS : IS := TRUE
    leave_SB : SB := FALSE
    act1 : driver_isolates_ETCS := FALSE
  end
END

```