OETCS/WP3/D3.5.2

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

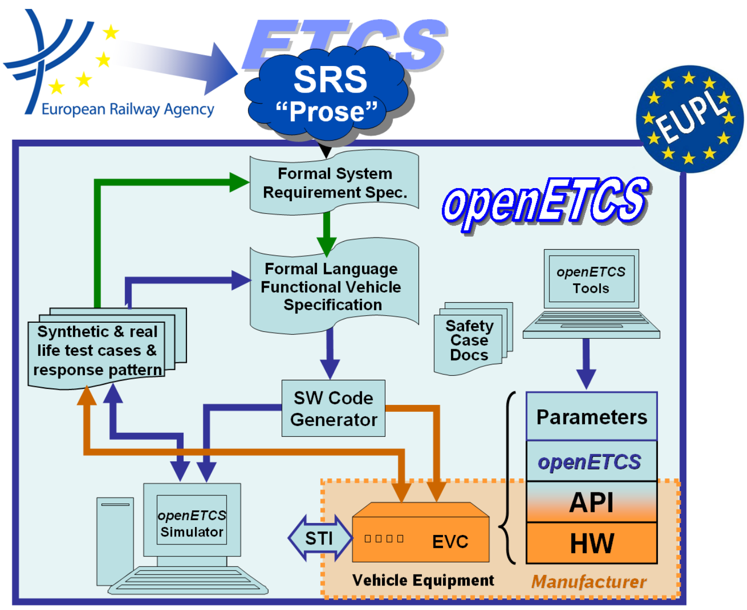
Work Package 3: “Modeling”

openETCS Design Specification

Software Component Design and Internal Interface Specification

Baseliyos Jacob, Peter Mahlmann, Bernd Hekele, Peyman Farhangi, Stefan Karg, Valerio D´Angelo, Uwe Steinke, Christian Stahl, Jakob Gärtner, Jos Holtzer, Jan Welvaarts, Vincent Nuhaan, Benjamin Beichler, Thorsten Schulz, Marielle Petit-Doche, Matthias Gudemann, Vernique Gontier, Ghristian Giraud, Fausto Cochetti and Alexander Stante

June 2015



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 3: “Modeling” **OETCS/WP3/D3.5.2**
June 2015

openETCS Design Specification

Software Component Design and Internal Interface Specification

Document approbation

Lead author:	Technical assessor:	Quality assessor:	Project lead:
location / date	location / date	location / date	location / date
signature Peter Mahlmann (DB Netz AG)	signature Jan Welte (Technische Universität Braunschweig)	signature Izaskun de la Torre (SQS)	signature Klaus-Rüdiger Hase (DB Netz)

Baseliyos Jacob, Peter Mahlmann, Bernd Hekele, Peyman Farhangi, Stefan Karg, Valerio D´Angelo

DB Netz AG

Uwe Steinke

Siemens AG

Christian Stahl

TWT-GmbH

Jakob Gärtner

LEA Railergy

Jos Holtzer, Jan Welvaarts, Vincent Nuhaan

Nederlands Spoorwegen

Benjamin Beichler, Thorsten Schulz

University of Rostock

Marielle Petit-Doche, Matthias Gudemann

Systerel

Vernique Gontier

All4Tec

Ghristian Giraud, Fausto Cochetti

Alstom

Alexander Stante

Fraunhofer ESK

Architecture and Design Specification

Prepared for openETCS@ITEA2 Project

Abstract: This document gives an introduction to the software and component design of the openETCS OBU model. The functional scope is tailored to cover the functionality required for the openETCS demonstration as an objective of the ITEA2 project. The goal is to develop a formal model and to demonstrate the functionality during a proof of concept on the ETCS Level 2 Utrecht Amsterdam track with real scenarios. It has to be read as a complement to the models in SysML and Scade languages.

Modification History

Version	Section	Modification / Description	Author	Date
0.1	Document	Initial document providing structure	Peter Mahlmann	27.05.2015
0.2	2	New template for design descriptions	Peter Mahlmann	10.06.2015

Table of Contents

Figures and Tables

Figures

Tables

Part I

Functional Breakdown

1 openETCS API Runtime System and Input to the EVC)

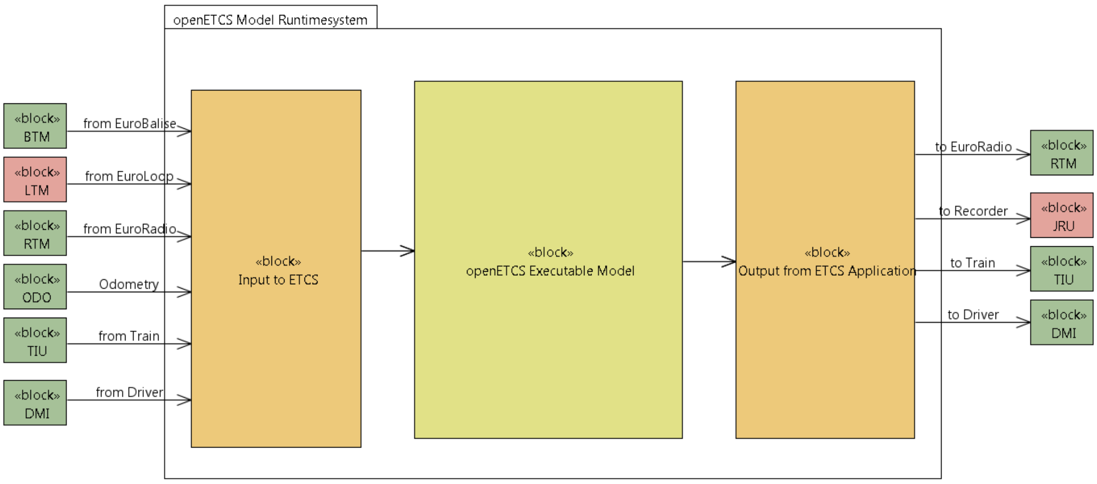


Figure 1. openETCS API Highlevel View

Figure 1 shows the structure of API with respect to the software architecture. Note that red input and output modules are were not yet implemented and thus are not part of the openETCS OBU model. The system covers functions for processing inputs from other units, functions for processing outputs to other functions and a basic runtime system. Inputs are used to feed the input to the executable model before calling it, outputs are used for collecting information provided by the executable model to be passed to the relevant interfaces after the execution cycle has finished.

1.1 Principles for Interfaces (openETCS API)

Information is exchanged via asynchronous *messages*. A message is a set of information corresponding to an event of a particular unit, e.g. a balise message received from the BTM. For possible types of messages please refer to Chapter **??**.

The information is passed to the executable model as parameters to the synchronous call of a procedure (Interface to the executable model). Since the availability of input messages to the application is not guaranteed the parts of the interfaces are defined with a "present" flag. In addition, fields of input arrays quite often is of variable size. Implementation in the concrete interface in this use-case is the use of a "size" parameter and a "valid"-flag.

1.2 openETCS Model Runtime System

The openETCS model runtime system also provides:

Input Functions From other Units In this entity messages from other connected units are received.

Output Functions to other Units The entity writes messages to other connected units.

Conversation Functions for Messages (Bitwalker) The conversion function are triggered by Input and Ouput Functions. The main task is to convert input messages from an bit-packed format into logical ETCS messages (the ETCS language) and Output messages from Logical into a bit-packed format. The logical format of the messages is defined for all used types in the openETCS data dictonary.

Variable size elements in the Messages are converted to fixed length arrays with an used elements indicator. Optional elements are indicated with an valid flag.

The conversion routines are responsible for checking the data received is valid. If faults are detected the information is passed to the openETCS executable model for further reaction.

Model Cycle The version management function is part of the message handling. This implies, conversions from other physical or logical layouts of messages are mapped onto a generic format used in the EVC. Information about the origin version of the message is part of the messages.

The executable model is called in cycles. In the cycle

- First the received input messages are decoded
- The input data is passed to the executable model in a predefined order. **(Details for the interface to be defined)**.
- Output is encoded according to the SRS and passed to the buffers to the units.

1.3 Input Interfaces of the openETCS API From other Units of the OBU

Interfaces are defined in the Scade project APITypes (package API_Msg_Pkg.xscade).

In the interfaces the following principles for indicating the quality of the information is used:

Indicator	Type	Purpose
present	bool	True indicates the component has been changed compared to the previous call of the routine
valid	bool	True indicates the component is valid to be used.

In the next table we can see the interfaces being used in the openETCS system. Details on the interfaces are defined further down.

Unit	Name	Processing Function
BTM	Balise Telegram	Receive Messages
DMI	Driver Machine Interface	DMI Manager
EURORADIO	Communication Management	Communication Management
EURORADIO	Radio Messages	Receive Messages
ODO	Odometer	All Parts
System TIME	Time system of the OBU	All Parts
TIU	Train Data	All Parts

Information in the following sections gives an more detailed overview of the structure of the interfaces.

1.4 Message based interface (BTM, RTM)

Balise Message (Track to Train)

Message Name	Optional Packets	Restrictions in the current scope
Balise Telegram	3: National Values 41: Level Transition Order 42: Session Management 45: Radio Network registration 46: Conditional Level Transition Order 65: Temporary Speed Restriction 66: Revoke Temporary Speed Restriction 72: Packet for sending plain text messages 137: Stop if in Staff Responsible 255: End of Information	Used in Scenario
Balise Telegram	0, 2, 3, 5, 6, 12, 16, 21, 27, 39, 40, 41, 42, 44, 45, 46, 49, 51, 52, 65, 66, 67, 68, 69, 70, 71, 72, 76, 79, 80, 88, 90, 131, 132, 133, 134, 135, 136, 137, 138, 139, 141, 145, 180, 181, 254	Not Used in Scenario

Radio Messages (Track to Train)

Message Name	Optional Packets	Restrictions in the current scope
2: SR Authorisation	63: List of Balises in SR Authority	Message Not Supported
3: Movement Authority	21: Gradient Profile 27: International Static Speed Profile 49: List of balises for SH Area 80: Mode profile plus common optional packets	a
9: Request To Shorten MA	49: List of balises for SH Area 80: Mode profile	
24: General Message	From RBC: 21: Gradient Profile 27: International Static Speed Profile plus common optional packets From RIU: 44, 45, 143, 180, 254	Messages from RIU are not supported

28: SH authorised	3, 44, 49	
33: MA with Shifted Location Reference	21: Gradient Profile 27: International Static Speed Profile 49: List of balises for SH Area 80: Mode profile plus common optional packets	
37: Infill MA	5, 21, 27, 39, 40, 41, 44, 49, 51, 52, 65, 66, 68, 69, 70, 71, 80, 88, 138, 139	Message Not Supported
List of common optional parameters	3, 5, 39, 40, 51, 41, 42, 44, 45, 52, 57, 58, 64, 65, 66, 68, 69, 70, 71, 72, 76, 79, 88, 131, 138, 139, 140, 180	

The runtime system is in charge to transfer the messages from its stream mode first to compressed message format.

1.5 Interfaces to the Time System

The interface types are defined in the OBU_Basic_Types_Pkg Package. The system time is defined in the basic software.

The system TIME is provided to the executable model at the begin of the cycle. It is not refreshed during the cycle. The time provided to the application is equal to 0 at power-up of the EVC (it is not a “UTC time” nor a “Local Time”), then must increase at each cycle (unit = 1 msec), until it reaches its maximum value (i.e current EVC limitation = 24 hours)

- TIME (T_internal_Type, 32-bit INT)
  Standardized system time type used for all internal time calculations: in ms. The time is defined as a cyclic counter: When the maximum is exceeded the time starts from 0 again.
- CLOCK (to be implemented)
  The clocking system is provided by the JRU. A GPS based clock is assumed to provide the local time.

1.6 Interfaces to the Odometry System

The interface types are defined in the OBU_Basic_Types_Pkg Package. The odometer gives the current information of the positing system of the train. In this section the structure of the interfaces are only highlighted. Details, including the internal definitions for distances, locations speed and time are implemented in the package.

- Odometer (odometry_T)
  - valid (bool)
    valid flag, i.e., the information is provided by the ODO system and can be used.
  - timestamp (T_internal_Type)
    of the system when the odometer information was collected. Please, see also general remarks on the time system.

- Coordinate (odometryLocation_T)
  * nominal (L_internal_Type) [cm]
  * min (L_internal_Type) [cm]
  * max (L_internal_Type) [cm]

  The type used for length values is a 32 bit integer. Min and max value give the interval where the train is to be expected. The bounderies are determined by the inaccuracy of the positioning system. All values are set to 0 when the train starts.
- speed (OdometrySpeeds_T) [km/h]
  * v_safeNominal (speed internal type) [km/h]
    The safe nominal estimation of the speed which will be bounded between 98% and 100% of the upper estimation
  * v_rawNominal (speed internal type) [km/h]
    The raw nominal estimation of the speed which will be bounded between the lower and the upper estimations
  * v_lower (speed internal type) [km/h]
    The lower estimation of the speed
  * v_upper (speed internal type) [km/h]
    The upper estimation of the speed

  The type used for speed values is a 32 bit integer. Min and max value give the interval where the train is to be expected. The bounderies are determined by the inaccuracy of the positioning system. All values are set to 0 when the train starts.
- acceleration (A_internal_Type)[0.01 m/s2],
  Standardized acceleration type for all internal calculations : in
- motionState (Enumeration)
  indicates whether the train is in motion or in no motion
- motionDirection (Enumeration)
  indicates the direction of the train, i.e., CAB-A first, CAB-B first or unknown.

1.7 Interfaces to the Train Interfaces (TIU)

The following infomration is based on the implementation of the Alstom API. The interface is organised in packets. The packets of the Alstom implementation are listed in the appendix to this document.

The description of interfaces needed for the current scope will be added according to the use.

1.8 Output Interfaces of the openETCS API TO other Units of the OBU

From Function	Name	To Unit	Description
	Radio Output Message	EURORADIO	
	Communication Management	EURORADIO	
	Driver Information	DMI	
	Train Data	TIU	

Packets: to be completed

Radio Messages to be completed

Part II

Design Description

2 F1: Receive information from Trackside

3 F2: ETCS Kernel

3.1 Manage_TrackSideInformation_Integration

125 ### 3.1.1 Component Requirements

Component name	Manage_TrackSideInformation_Integration
Link to SCADE model	???
SCADE designer	[Name, affiliation]
Description	The block “Manage_TrackSideInformation_Integration” is responsible for receiving Eurobalise telegrams and Euroradio messages from the API and performs several consistency checks on the inputs. The block collects the telegrams of balises in order to build balise group messages. Euroradio messages are always delivered as a whole message. On each message, a consistency check is performed, before the data is validated according to the driving direction of the train. In general, messages not designated for the current driving direction of the train are not forwarded to the further processing. After applying consistency checks, the data direction is validated.
Input documents	See sub-components.
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

3.1.2 Interface

An overview of the interface of component [component name] is shown in Figure 2. The inputs and outputs are described in detail in Section 3.1.2.1 respectively 3.1.2.2.

3.1.2.1 Inputs

130 ##### 3.1.2.1.1 [Input 1 name]

Input name	[Name of the input]
Description	[Brief description of the input]
Source	[Name of the source component]
Type	[Type of the input]

Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when input value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when input value is out of valid range]

3.1.2.1.2 [Input 2 name]

Input name	[Name of the input]
Description	[Brief description of the input]
Source	[Name of the source component]
Type	[Type of the input]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when input value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when input value is out of valid range]

3.1.2.2 Outputs

3.1.2.2.1 [Output 1 name]

Output name	[Name of the output]
Description	[Brief description of the output]
Destination	[Name of the destination component(s)]
Type	[Type of the output]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when output value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when output value is out of valid range]

3.1.2.2.2 [Output 2 name]

Output name	[Name of the output]

Description	[Brief description of the output]
Destination	[Name of the destination component(s)]
Type	[Type of the output]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when output value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when output value is out of valid range]

3.1.3 Sub Components

3.1.3.1 Receive_TrackSide_Msg

3.1.3.1.1 Component Requirements

Component name	Receive_TrackSide_Msg
Link to SCADE model	`https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/Receive_TrackSide_Msg`
SCADE designer	[Name, affiliation]

Description	This function defines the interface of the OBU model to the openETCS generic API for Eurobalise and Euroradio messages. On the interface, either a valid telegram/message is provided or a telegram/message is indicated which could not be received correct when passing the balise or receiving the radio message. The function passes a balise telegram without major changes of the information to the next entity for collecting the balise group information. This entity collects telegrams received via the interface into Balise Group Information. In case of a radio message, the message is converted to an internal format for further processing and passed without changing the information contained. • The decoding of balises is done at the API. Also, packets received via the interface are already transformed into a usable shape. • Only packets used inside the current model are passed via the interface. • Treatment of Packet 5: Linking Information. Linking Information is added to the linking array starting from index 0 without gaps. Used elements are marked as valid. Elements are sorted according to the order given by the telegram sequence. • Telegrams received as invalid are passed to the "Check-Function" to process errors in communication with the track side according to the requirements and in a single place. Telegrams are added to the telegram array starting from index 0 without gaps. Used elements are marked as valid. Elements are stored according to the order given by the telegram sequence. • This function does not process information from the packets. The information is passed to the check without further processing of the values.
Input documents	Subset-026, Chapter 7 and 8: Definition of the Balise Telegram Subset-026, Chapter 4.2.2, 4.2.4, 4.2.9: Interface to the BTM Subset-026, Chapter 3.4.1 - 3.4.3, 3.16.2: Handling of Balise Telegrams Subset-026, Chapter 3.16.2: Check of the balise group Subset-026, Chapter 3.4.2: Determining the orientation Subset-026, Chapter 4.5.2 Active Functions Table Subset-026, Chapter 8.4.4: Rules for Euroradio messages
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

3.1.3.1.2 Interface

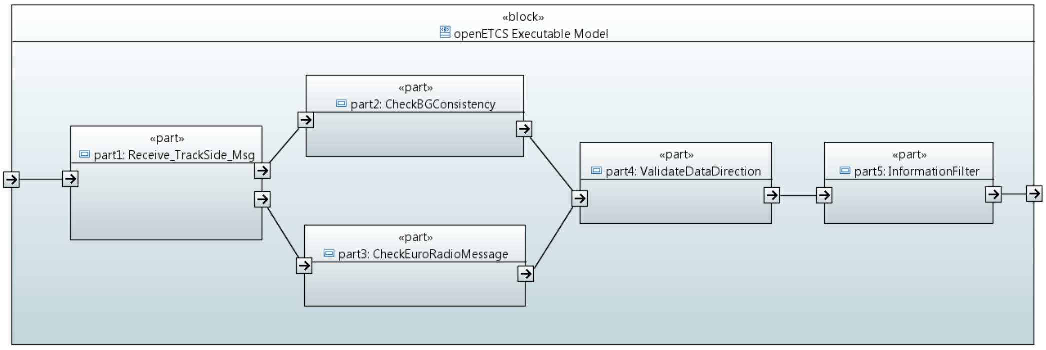


Figure 2. SysML diagram of Manage_TrackSideInformation_Integration module.

For an overview of the interface of this internal component we refer to the SCADE model
140 (c.f. link above) respectively the SCADE generated documentation.

3.1.3.2 CheckBGConsistency

3.1.3.2.1 Component Requirements

Component name	CheckBGConsistency
Link to SCADE model	`https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/CheckBGConsistency`
SCADE designer	[Name, affiliation]
Description	This function verifies the completeness and correctness of the received messages from balise groups. A message consists of at least a telegram and a maximum of 8 telegrams. • A message is still complete and correct, if a telegram is missing (or not decoded or incomplete decoded), and this telegram is duplicated within the balise group and the duplicating one is correctly read. • By more than one telegram, the order of the telegrams must be either ascending (nominal) or descending(reverse). • A message is correct, if all message counters (M MCUNT) do not equal 254 (that means: The telegram never fits any message of the group). A message counter can be equal 255 (that means: The telegram fits with all telegrams of the same balise group) and all other values must be the same. The orientation of the BG will also be calculated in this block. The check, if the message has been received in due time and the right at the right expected location, will be performed in "Calculate Train Position". The checks on the validity of the data in the packets and the validity with respect to the direction of motion will be performed in other modules, e.g. "Validate Data Direction".

Input documents	Subset-026, Chapter 7 and 8: Definition of the Balise Telegram Subset-026, Chapter 3.4.1-3, 3.16.2: Handling of Balise Telegrams Subset-026, Chapter 3.16.2: Check of the balise group Subset-026, Chapter 4.5.2: Active Functions Table
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

3.1.3.2.2 Interface

For an overview of the interface of this internal component we refer to the SCADE model (c.f. link above) respectively the SCADE generated documentation.

3.1.3.3 CheckEuroradioMessage

3.1.3.3.1 Component Requirements

Component name	CheckEuroradioMessage
Link to SCADE model	`https://github.com/openETCS/modeling/tree/b9c31ce6fdf702b412bbeab3032a8a4dc7c92e5c/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/CheckEuroRadioMessage`
SCADE designer	Stefan Karg, DB Netz AG
Description	The component "CheckEuroradioMessage" performs several checks on the received radio message. These checks include checking of the message sequence, completeness of messages. Invalid messages are marked as invalid in the message header.
Input documents	Subset-026, Chapter 3.16 Subset-026, Chapter 8.4.4
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

3.1.3.3.2 Interface

For an overview of the interface of this internal component we refer to the SCADE model (c.f. link above) respectively the SCADE generated documentation.

3.1.3.4 ValidateDataDirection

3.1.3.4.1 Component Requirements

Component name	CheckEuroradioMessage
Link to SCADE model	`https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/ValidateDataDirection`
SCADE designer	???
Description	The component filters an input message in order to mark all elements as invalid, which are not designated for the current driving direction of the train. • The operator contains two processing paths for different message types. Radio messages and balise group messages are handeled in a different way. For validating the data direction of a radio message, the check is performed using the balise group referenced in the radio message header as relevant balise group. For balise group message, the LRBG is used. • The metadata of packets, which are recognized as not valid for the current driving direction, is invalidated.
Input documents	Subset-026, Chapter 3.6.3
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

3.1.3.4.2 Interface

For an overview of the interface of this internal component we refer to the SCADE model (c.f. link above) respectively the SCADE generated documentation.

3.1.3.5 InformationFilter

3.1.3.5.1 Component Requirements

Component name	CheckEuroradioMessage
Link to SCADE model	`https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/InformationFilter`
SCADE designer	Alexander Stante, FhG

Description	This function filters incoming information received from Eurobalise, Euroradio, and Euroloop. The information is received via messages and filtering is done depending on the criteria described in [1, Chapter 4.8]. Messages are only allowed to pass the filter if specified critera are met like for example the correct mode of the train (e.g. Full Supervision, Shunting, etc.) or the ETCS level. Some messages have to be stored in a transition buffer to be later reevaluated by the filter again. The filter receives track information (balise and radio) and filters them depending of the mode, level and further information. Only messages that pass the filter are valid and should be considered by other ETCS subsystems. Figure 7 shows the highlevel decomposition of the functionality. The filter functionality can be decomposed into a FirstFilter, SecondFilter, ThirdFilter and TransitionBuffer. **FirstFilter** This filter performs filtering of messages based on the current ETCS level. The decisions taken process is described via a big decision table which contains rows for every packet and columns for every ETCS level. This table encodes also if certain additional information is necessary to filter a message like pending ETCS Level transitions. Based on this filter packets of an incoming message is either rejected, accepted or the whole message is put in the TransitionBuffer. Messages are put in the TransitionBuffer if there is an announced level transition and the received message is only valid for the upcoming level. **SecondFilter** The SecondFilter mainly considers messages that are received via Euroradio. Certain messages are directly rejected while other may be stored in the TransitionBuffer. The buffer is used to store messages that are received from non supervising RBCs, but will be reevaluated after a RBC transition. **ThirdFilter** The last filter is functionally very similiar the the First-Filter, however it filters depending on the mode. It also contains a decision table with rows for every packet but the columns are modes. **TransitionBuffer** The InformationFilter uses two Transition-Buffers. One is used to store up to three messages for the ETCS level transition and the other buffer is used for RBC transitions. The buffer is designed as a ring buffer and message are read in FIFO order.
Input documents	Subset-026, Chapter 4.8
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

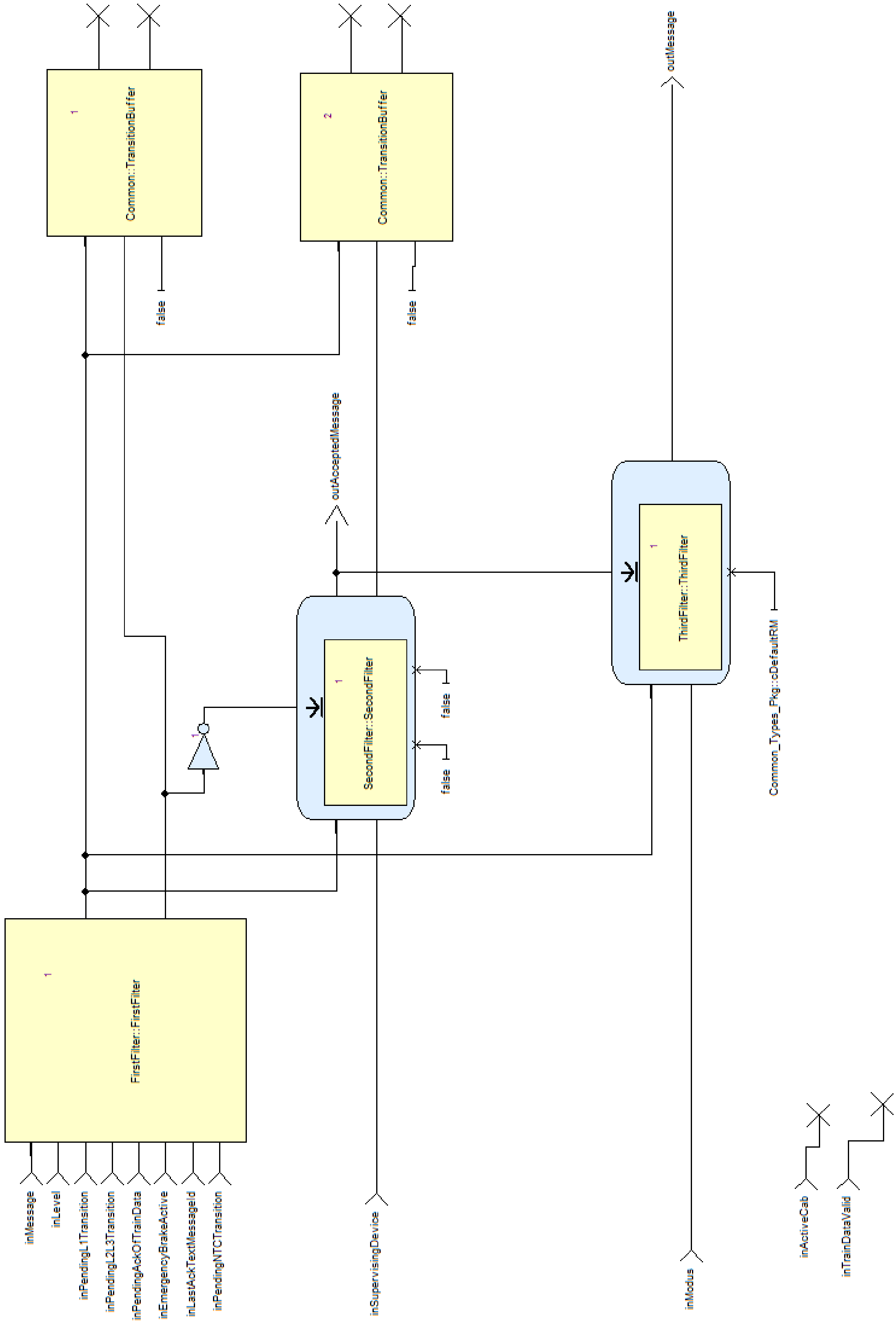


Figure 3. High level overview of the InformationFilter components.

3.1.3.5.2 Interface

For an overview of the interface of this internal component we refer to the SCADE model (c.f. link above) respectively the SCADE generated documentation.

Part III

Deprecated Design Description

4 Component Design Template

4.1 Component Requirements

Component name	[Component name]
Link to SCADE model	`https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/Receive_TrackSide_Msg`
SCADE designer	[Name, affiliation]
Description	[Brief description of the components functionality]
Input documents	Subset-026, version, chapter ?.? Subset-026, version, chapter ?.? Subset-026, version, chapter ?.?.?
Safety integrity level	4
Time constraints	[If applicable description of time constraints, otherwise n/a]
API requirements	[If applicable description of API requirements, otherwise n/a]

4.2 Interface

An overview of the interface of component [component name] is shown in Figure 4. The inputs and outputs are described in detail in Section 4.2.1 respectively 4.2.2.

4.2.1 Inputs

4.2.1.1 [Input 1 name]

Input name	[Name of the input]
Description	[Brief description of the input]
Source	[Name of the source component]
Type	[Type of the input]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when input value is at boundary]

[Put SysML diagram of component here]

Figure 4. Component SysML diagram

Behaviour for values out of valid range	[Description of components behaviour when input value is out of valid range]

4.2.1.2 [Input 2 name]

Input name	[Name of the input]
Description	[Brief description of the input]
Source	[Name of the source component]
Type	[Type of the input]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when input value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when input value is out of valid range]

4.2.2 Outputs

4.2.2.1 [Output 1 name]

Output name	[Name of the output]
Description	[Brief description of the output]
Destination	[Name of the destination component(s)]
Type	[Type of the output]
Valid range of values	[Complete list of valid values]
Behaviour when value is at boundary	[Description of components behaviour when output value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when output value is out of valid range]

4.2.2.2 [Output 2 name]

Output name	[Name of the output]
Description	[Brief description of the output]
Destination	[Name of the destination component(s)]
Type	[Type of the output]
Valid range of values	[Complete list of valid values]

Behaviour when value is at boundary	[Description of components behaviour when output value is at boundary]
Behaviour for values out of valid range	[Description of components behaviour when output value is out of valid range]

5 F1: Receive information from Trackside

6 F2: ETCS Kernel

6.1 Manage_TrackSideInformation_Integration

The block “Manage_TrackSideInformation_Integration” is responsible for receiving Eurobalise telegrams and Euroradio messages from the API and perform several consistency checks on the input.

The block collects the telegrams of balises in order to build balise group messages. Euroradio messages are always delivered as a whole message. On each message, a consistency check is performed, before the data is validated according to the driving direction of the train. In general, messages not designated for the current driving direction of the train are not forwarded to the further processing. After applying consistency checks, the data direction is validated.

6.1.1 Inputs

For providing the output, the module needs different input data flows. An overview is provided in table 16.

Index	Input name	Input type	Source
0	`fullChecks`	`bool`	Configuration
1	`API_trackSide_Message`	`API_Msg_Pkg::API_TrackSideInput_T`	API
2	`ActualOdometry`	`Obu_BasicTypes_Pkg::odometry_T`	Odometer
3	`reset`	`bool`	Environment
4	`trainPosition`	`TrainPosition_Types_Pck::trainPosition_T`	Calculate Train Position
5	`modeAndLevel`	`BG_Types_Pkg::ModeAndLevelStatus_T`	Mode and Level
6	`tNvContact`	`Obu_BasicTypes_Pkg::T_internal_Type`	Database
7	`lastRelevantEventTimestamp`	`Obu_BasicTypes_Pkg::T_internal_Type`	Database
8	`connectionStatus`	`Radio_Types_Pkg::sessionStatus_Type`	Manage Radio Communication
9	`inSupervisingRbcId`	`int`	Database
10	`inAnnouncedBGs`	`TrainPosition_Types_Pck::positionedBGs_T`	Calculate Train Position
11	`q_nvlocacc`	`Q_NVLOCACC`	Database

Table 16. Overview over input

6.1.1.1 Input 0: `fullChecks`

The boolean indicates, if all checks on the message should be performed. The possible values are given in table 17.

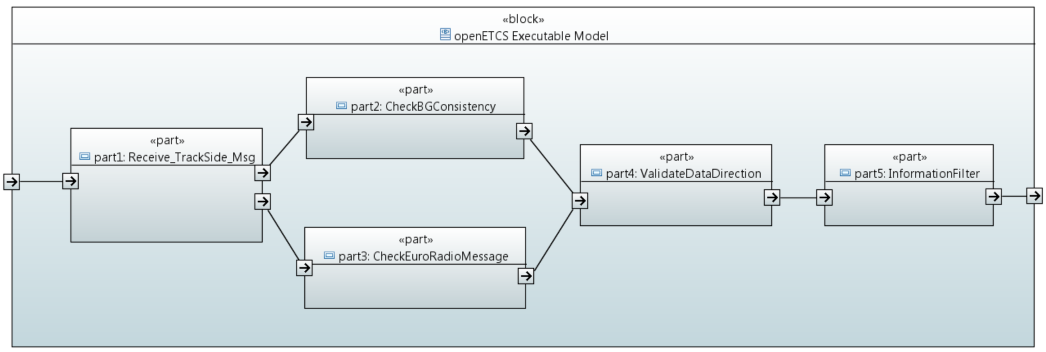


Figure 5. Structure of the Manage_TrackSideInformation_Integration module with submodules.

Value	Interpretation
true	All checks are performed.
false	The module `Information Filter` is deactivated.

Table 17. Possible values for the input `fullChecks`

6.1.1.2 Input 1: `API_trackSide_Message`

The `API_trackSide_Message` is the message received from the API. The API performs pre-processing of RTM and BTM messages and deliveres a maximum of a single message per cycle to the SCADE model.

6.1.1.3 Input 2: `ActualOdometry`

The input `ActualOdometry` is provided by the external odometry module of the train. It contains location information with inaccuracies.

6.1.1.4 Input 3: `reset`

To delete all data stored in the module (e.g. collected balise telegrams, which do not yet form a complete message), a reset input can be used. If the input is set to `true`, all data kept in the module is deleted and no input is accepted.

Value	Interpretation
true	All data kept in the module is deleted and no input is accepted.
false	No action. Data at input is accepted.

Table 18. Possible values for the input `reset`.

6.1.1.5 Input 4: `trainPosition`

The input `trainPosition` is generated by the "Calculate Train Position" module and contains the current position of the train.

6.1.1.6 Input 5: `modeAndLevel`

The input is generated by the "Mode and level management" module. It provides the current level and mode of the EVC.

6.1.1.7 Input 6: `tNvContact`

For monitoring the safe radio connection, the national value `T_NVCONTACT` is needed as an input.

6.1.1.8 Input 7: `lastRelevantEventTimestamp`

For monitoring the safe radio connection, it's necessary, that the time between two packets is less than the value of `T_NVCONTACT`.

In situations like level-changes or announced radioholes, not the timestamp of the last message is relevant for comparison, but the timestamp of the last relevant event. This can be e.g. the timestamp of the level change or the timestamp of the timestamp of the moment, when the train was passing the end of the radiohole.

For performing this check, the timestamp of the last relevant event is provided to the model as an `T_internal_Type`-type.

6.1.1.9 Input 8: connectionStatus

The input `connectionStatus` will give information about the radio connection. This input is delivered by the session management module, not from the API. The information is needed to perform the timing check, which is depending on the connection state.

Value	Interpretation
DISCONNECTED	The OBU is currently not connected to a RBC.
CONNECTING	The OBU is currently connecting to the RBC. Received messages belong to the process of establishing a connection.
CONNECTION_ESTABLISHED	The connection to RBC is established.

Table 19. Possible values for the input connectionStatus.

6.1.1.10 Input 9: inSupervisingRbcId

For the submodule “Information Filter”, the information is needed, which radio messages are sent by the supervising RBC. To recognize these messages, the identifier of the supervising RBC is needed.

6.1.1.11 Input 10: inAnnouncedBGs

This input provides information about balise groups which will be passed by the train soon. This information is generated by “Calculate Train Position” based on the linking information received from trackside.

6.1.1.12 Input 11: q_nvlocacc

The national value determines the location accuracy and is delivered by the database.

6.1.2 Outputs

The output of the module provides the received and processed Euroradio and Eurobalise messages. The module combines messages both from Eurobalises and from Euroradio to one common dataflow.

An overview over the output dataflows is provided in table 20.

Index	Output name	Output type
0	outputMessage	Common_Types_Pkg::ReceivedMessage_T
1	ApplyServiceBrake	bool
2	BadBAliseMessageToDMI	bool
3	errorLinkedBG	bool
4	errorUnlinkedBG	bool
5	passedBG	BG_Types_Pkg::passedBG_T
6	outPositionParams	Common_Types_Pkg::PositionReportParameter_T
7	outRadioManagement	Common_Types_Pkg::radioManagementMessage_T
8	radioSequenceError	bool
9	radioMessageConsistencyError	bool

Table 20. Dataflow at output

6.1.2.1 Output 0: outputMessage

The element outputMessage consists of the type ReceivedMessage_T combines both balise and radio messages to one common datatype. This datatype contains all variables and packets, which are possible for the given scenario.

Name	Datatype	Description
valid	bool	true, if no consistency errors were detected.
source	Common_Types_Pkg::MsgSource_T	Defines, if this is a Euroradio or Eurobalise message.
packetMetadata	Common_Types_Pkg::Metadata_T	contains the metadata of the packets
radioMetadata	Common_Types_Pkg::RadioMetadata_T	contains the metadata of the radio specific header variables
BG_Common_Header	BG_Types_Pkg::BG_Header_T	Header of Eurobalise message
Radio_Common_Header	Radio_Types_Pkg::Radio_TrackTrain_Header_T	Header of Euroradio message
packets	Common_Types_Pkg::Packets_T	Structure of packets in messages

Table 21. Structure of ReceivedMessage_T

The Eurobalise-common-header BG_Header_T consists of the fields visible in the SCADE-declaration. The structure corresponds to the structure defined in the SRS chapter 8.4.2.1. Some fields were removed since they are not needed anymore for further processing after building messages from separate telegrams.

The Euroradio-common-header Radio_TrackTrain_Header_T consists of the fields visible in the SCADE declaration. The structure corresponds to the structure defined in the SRS chapter 8.4.4.6.1. The structure contains all variables required by possible NID_MESSAGE values for the given scenario. Which values are valid is defined in the field radioMetadata.

6.1.2.2 Output 1: ApplyServiceBreak

The flag indicates the balise group the train just passed could not be processed correctly. The check results in the request for a service break.

6.1.2.3 Output 2: BadBaliseMessageToDMI

Information to be passed to the DMI to indicate the reception of a “bad balise” to the driver.

6.1.2.4 Output 3: errorLinkedBG

Value	Interpretation
true	A error in a linked balise group was detected.
false	No error in a linked balise group was detected.

Table 22. Possible values for the input `errorLinkedBG`

6.1.2.5 Output 4: `errorUnlinkedBG`

Value	Interpretation
true	A error in an unlinked balise group was detected.
false	No error in an unlinked balise group was detected.

Table 23. Possible values for the input `errorUnlinkedBG`

6.1.2.6 Output 5: `passedBG`

The output `passedBG` provides the received balise group message in a special format needed by the module “Calculate train position”.

6.1.2.7 Output 6: `outPositionParams`

The output `outPositionParams` provides the parameters for the position report in a special format needed by the module “Provide Position Report”.

6.1.2.8 Output 7: `outRadioManagement`

The output `outRadioManagement` provides the messages for radio session management in a special format needed by the module “Management of Radio Communication”.

6.1.2.9 Output 8: `radioSequenceError`

Value	Interpretation
true	A sequence error or a timeout has been detected in the radio message.
false	No error in the radio message sequence was detected.

Table 24. Possible values for the input `radioSequenceError`

6.1.2.10 Output 9: `radioMessageConsistencyError`

Value	Interpretation
true	A consistency error has been detected in the radio message.
false	No consistency error in the radio message was detected.

Table 25. Possible values for the input `radioMessageConsistencyError`

6.1.3 Receive_TrackSide_Msg in Manage_TrackSideInformation_Integration

6.1.3.1 Reference to the SRS (or other requirements)

- [1, Chapt. 7 and 8]: Definition of the Balise Telegram

- [2, Chapt. 4.2.2, 4.2.4, 4.2.9]: Interface to the BTM
- [1, Chapt. 3.4.1 - 3.4.3, 3.16.2]: Handling of Balise Telegrams
- [1, Chapt. 3.16.2]: Check of the balise group
- [1, Chapt. 3.4.2]: Determining the orientation
- [1, Chapt. 4.5.2]: Active Functions Table
- [1, Chapt. 8.4.4]: Rules for Euroradio messages

6.1.3.2 Short description of the functionality

This function defines the interface of the OBU model to the openETCS generic API for Eurobalise and Euroradio messages. On the interface, either a valid telegram/message is provided or a telegram/message is indicated which could not be received correct when passing the balise or receiving the radio message. The function passes a balise telegram without major changes of the information to the next entity for collecting the balise group information. This entity collects telegrams received via the interface into Balise Group Information. In case of a radio message, the message is converted to an internal format for further processing and passed without changing the information contained.

6.1.3.3 Interface

6.1.3.4 Functional Design Description

Design Constraints and Choices

1. The decoding of balises is done at the API. Also, packets received via the interface are already transformed into a usable shape.
2. Only packets used inside the current model are passed via the interface.
3. Treatment of Packet 5: Linking Information. Linking Information is added to the linking array starting from index 0 without gaps. Used elements are marked as valid. Elements are sorted according to the order given by the telegram sequence.
4. Telegrams received as invalid are passed to the “Check-Function” to process errors in communication with the track side according to the requirements and in a single place. Telegrams are added to the telegram array starting from index 0 without gaps. Used elements are marked as valid. Elements are stored according to the order given by the telegram sequence.
5. This function does not process information from the packets. The information is passed to the check without further processing of the values.

6.1.3.5 Reference to the Scade Model

The SCADE model can be found on GitHub under the following path: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInfor` `BaliseGroup/Receive_TrackSide_Msg`

6.1.4 CheckBGConsistency in Manage_TrackSideInformation_Integration

6.1.4.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 7 and 8]: Definition of the Balise Telegram
- [1, Chapt. 3.4.1 - 3.4.3, 3.16.2]: Handling of Balise Telegrams
- [1, Chapt. 3.16.2]: Check of the balise group
- [1, Chapt. 4.5.2]: Active Functions Table

6.1.4.2 Short description of the functionality

This function has the task to verify the completeness and correctness of the received messages from balise groups. A message consists of at least a telegram and a maximum of 8 telegrams.

- A message is still complete and correct, if a telegram is missing (or not decoded or incomplete decoded), and this telegram is duplicated within the balise group and the duplicating one is correctly read.
- By more than one telegram, the order of the telegrams must be either ascending (nominal) or descending(reverse).
- A message is correct, if all message counters (M MCUNT) do not equal 254 (that means: The telegram never fits any message of the group). A message counter can be equal 255 (that means: The telegram fits with all telegrams of the same balise group) and all other values must be the same.

6.1.4.3 Interface

An input of the operator is a list of received telegrams. After consistency check of the telegrams' list, the function generates the balise-group-message. This function is active in certain modes and the output and reactions are dependent on if the linking information is used.

6.1.4.4 Functional Design Description

The orientation of the BG will also be calculated in this block. The check, if the message has been received in due time and the right at the right expected location, will be performed in "Calculate Train Position". The checks on the validity of the data in the packets and the validity with respect to the direction of motion will be performed in other modules, e.g. "Validate Data Direction" .

6.1.4.5 Reference to the Scade Model

The SCADE model can be found on github under the following path: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInfor BaliseGroup/CheckBGConsistency`

6.1.5 CheckEuroradioMessage in Manage_TrackSideInformation_Integration

6.1.5.1 Component Requirements

[Put SysML diagram of component here]

Figure 6. Component SysML diagram

Component name	`CheckEuroradioMessage`
Link to SCADE model	`https://github.com/openETCS/modeling/tree/b9c31ce6fdf702b412bbeab3032a8a4dc7c92e5c/model/Scade/System/ObuFunctions/ManageLocationRelatedInformation/BaliseGroup/CheckEuroRadioMessage`
SCADE designer	Stefan Karg, DB Netz AG
Description	The operator “CheckEuroradioMessage” performs several checks on the received radio message. These checks include checking of the message sequence, completeness of messages. Invalid messages are marked as invalid in the message header.
Requirements realized	Subset-026, Chapter 3.16 Subset-026, Chapter 8.4.4
Safety integrity level	4
Time constraints	n/a
API requirements	n/a

6.1.5.2 Interface

An overview of the interface of component CheckEuroradioMessage is shown in Figure 6. The inputs and outputs are described in detail in section 6.1.5.2.1 respectively 6.1.5.2.2.

6.1.5.2.1 Inputs

rtmMessage

Input name	`rtmMessage`
Description	Input message from API
Source	`Receive\_TrackSide\_Msg\_Pkg::Receive\_TrackSide\_Msg`
Type	`Common_Types_Pkg::TrackSide_ForCheck_T`
Valid range of values	n/a
Behaviour when value is at boundary	n/a
Behaviour for values out of valid range	n/a

tNvContact

Input name	tNvContact

Description	National value which defines the maximum time difference between a radio message or some defined special conditions.
Source	Current set of national values
Type	`Obu_BasicTypes_Pkg::T_internal_Type`
Valid range of values	0-255
Behaviour when value is at boundary	Message is always accepted (tNvContactError = false), see 7.5.1.148
Behaviour for values out of valid range	All messages are rejected (tNvContactError = true)

lastRelevantEventTimestamp

Input name	`lastRelevantEventTimestamp`
Description	The input "lastRelevantTimestamp" is needed in order to check timeouts. In most of the cases, the timestamp of the last relevant event is the timestamp of the last message received. But the SRS defines special cases like driving through a defined radio hole. In this case, the timestamp of the last relevant event is the point of time, when the EVC has a radio connection again.
Source	Database
Type	`Obu_BasicTypes_Pkg::T_internal_Type`
Valid range of values	dependent on hardware
Behaviour when value is at boundary	n/a
Behaviour for values out of valid range	n/a

355 connectionStatus

Input name	`connectionStatus`
Description	Status of the secure Euroradio session
Source	Management of Radio Communication
Type	`Radio_Types_Pkg::sessionStatus_Type`
Valid range of values	`{morc_st_inactive, morc_st_establishing, morc_st_maintaining, morc_st_terminating}`
Behaviour when value is at boundary	n/a

Behaviour for values out of valid range	n/a

activateCheck

Input name	`activateCheck`
Description	If true, the Euroradio messages are checked, if false, the message is just passed without any modifications
Source	Debug settings
Type	`bool`
Valid range of values	`{true, false}`
Behaviour when value is at boundary	n/a
Behaviour for values out of valid range	n/a

6.1.5.2.2 Outputs

checkedMessage

Output name	`checkedMessage`
Description	The message, which was checked with an updated valid flag.
Destination	ValidateDataDirection
Type	`Common_Types_Pkg::ReceivedMessage_T`
Valid range of values	n/a

radioSequenceError

Output name	`radioSequenceError`
Description	Indicates if the current message violated the sequence of radio messages.
Destination	`ProvidePositionReport`
Type	`bool`
Valid range of values	`{true, false}`

tNvContactError

Output name	`tNvContactError`
Description	Indicates if a timeout on the radio channel occured.
Destination	`ProvidePositionReport`
Type	`bool`
Valid range of values	`{true, false}`

otherTimingError

Output name	`otherTimingError`
Description	Indicates if other errors in the timing were detected, e.g. no last valid timestamp is available.
Destination	`ModeAndLevelManagement`
Type	`bool`
Valid range of values	`{true, false}`

radioMessageConsistencyError

Output name	`radioMessageConsistencyError`
Description	Indicates, if a consistency error was detected in the Euroradio message.
Destination	EVC internal
Type	`bool`
Valid range of values	`{true, false}`

6.1.6 ValidateDataDirection in Manage_TrackSideInformation_Integration

6.1.6.1 Reference to the SRS or other Requirements (or other requirements)

- The functionality is mainly described in [1, Chapter 3.6.3].

6.1.6.2 Short description of the functionality

The operator will filter an input message in order to mark all elements as invalid, which are not designated for the current driving direction of the train.

6.1.6.3 Interface

The operator expects a message and information about the LRBG, passed balises and the current train position. As output, the message with packets valid for the current direction of driving is provided.

6.1.6.4 Functional Design Description

- The operator contains two processing paths for different message types. Radio messages and balise group messages are handeled in a different way. For validating the data direction of a radio message, the check is performed using the balise group referenced in the radio message header as relevant balise group. For balise group message, the LRBG is used.
- The metadata of packets, which are recognized as not valid for the current driving direction, is invalidated.

6.1.6.5 Reference to the Scade Model

The SCADE model can be found on github under the following path: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInfor BaliseGroup/ValidateDataDirection`

6.1.7 InformationFilter

6.1.7.1 Reference to the SRS and other requirements

- The functionality of the InformationFilter is described in [1, Chapter 4.8].

6.1.7.2 Short description of the functionality

The function InformationFilter filters incoming information received from Eurobalise, Euroradio, and Euroloop. The information is received via messages and filtering is done depending on the criteria described in [1, Chapter 4.8]. Messages are only allowed to pass the filter if specified critera are met like for example the correct mode of the train (e.g. Full Supervision, Shunting, etc.) or the ETCS level. Some messages have to be stored in a TransitionBuffer to be later reevaluated by the filter again.

6.1.7.3 Interface

The interface of the information filter contains mainly the incoming message and inputs to check for the conditions described in the SRS. The complete interface is shown in table 37.

Name	Direction	Description
inMessage	IN	Received message that is valid for the train direction
inLevel	IN	The current ETCS Level
inMode	IN	The current train mode
inSupervisingDevice	IN	The device id which communicates with the current supervising RBC
inPendingL1Transition	IN	Information if an ETCS Level 1 transition is pending
inPendingL1L2Transition	IN	Information if an ETCS Level 2/3 transition is pending
inPendingNTCTransition	IN	Information if a NTC transition is pending
inPendingAckOfTrainData	IN	Information if the acknowledgement of train data is pending
inEmergencyBrakeActive	IN	Information if the emergency brake is active
inLastAckTextMessageId	IN	The id of the last acknowledged message ID
inActiveCab	IN	Information if the cab is active
inTrainDataValid	IN	Information if the train data is valid
outMessage	OUT	The filtered input message

Table 37. Overview of the InformationFilter interface

6.1.7.4 Functional Design Description

The filter receives track information (balise and radio) and filter them depending of the mode, level and further information. Only messages that pass the filter are valid and should be considered by other ETCS subsystems. The figure 7 show the highlevel decomposition of the functionality. The filter functionality can be decomposed into a FirstFilter, SecondFilter, ThirdFilter an TransitionBuffer.

6.1.7.4.1 FirstFilter

This filter performs filtering of messages based on the current ETCS level. The decisions taken process is described via a big decision table which contains rows for every packet and columns for every ETCS level. This table encodes also if certain additional information is necessary to filter a message like pending ETCS Level transitions. Based on this filter packets of an incoming message is either rejected, accepted or the whole message is put in the TransitionBuffer. Messages are put in the TransitionBuffer if there is an announced level transition and the received message is only valid for the upcoming level.

6.1.7.4.2 SecondFilter

The SecondFilter mainly considers messages that are received via Euroradio. Certain messages are directly rejected while other may be stored in the TransitionBuffer. The buffer is used to store messages that are received from non supervising RBCs, but will be reevaluated after a RBC transition.

6.1.7.4.3 ThirdFilter

The last filter is functionally very similiar the the FirstFilter, however it filters depending on the mode. It also contains a decision table with rows for every packet but the columns are modes.

6.1.7.4.4 TransitionBuffer

The InformationFilter uses two TransitionBuffers. One is used to store up to three messages for the ETCS level transition and the other buffer is used for RBC transitions. The buffer is designed as a ring buffer and message are read in FIFO order.

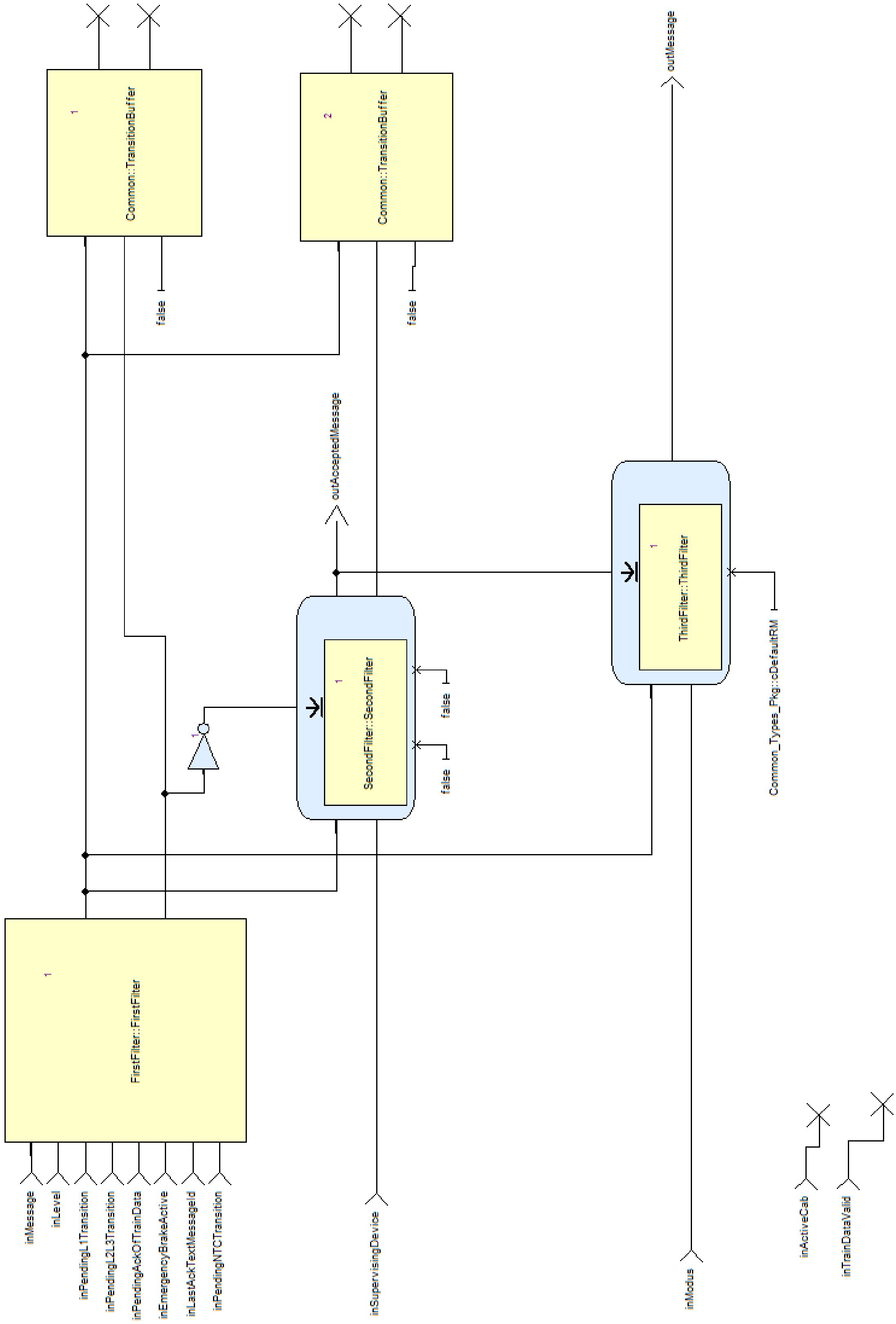


Figure 7. High level overview of the InformationFilter components.

A detailed list of packages and their handling depending of ETCS level or mode can be seen in table 8 and 9.

6.1.7.5 Reference to the Scade Model

The SCADE model can be found on github under the following path: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLocationRelatedInfor` `BaliseGroup/InformationFilter`

6.2 Train Supervision

The task of block “Train Supervision” is to monitor the speed of the train and the train location and as such to ensure that the speed remains within the given speed and distance limits. This block is mainly based on [1, Chapt. 3.13].

The block “Train Supervision” takes as input (1) movement related information such as train speed, train position and acceleration, (2) train related information such as brake information and train length, and (3) track related information such as speed and distance limits and national values.

Based on this information a speed profile is calculated. Speed restrictions create target speeds (targets) that have to be followed. For each such target braking curves are generated to supervise at which location of the track the train must perform the brake. In case of no target restrictions the train may accelerate to the supervised maximum speed of the speed profile. These calculations lead to commands being sent to the driver and the brake system.

The functionality is modeled using eight operators, as shown in Figure 10, which are explained below.

The current status of the analysis of “Train Supervision” and a functional breakdown can be found in a separate document, `SpeedSupervision_analysis.pdf`.

6.2.1 Input

For providing the output, the module needs different input data flows. Table 38 gives an overview.

Index	Input name	Input type	Source
0	`NationalValues`	`P3_NationalValues_T`	???
1	`TrainPosition`	`trainPosition_T`	Manage Track Data
2	`odometry`	`odometry_T`	Odometry
3	`m_level`	`M_LEVEL`	Mode and Level
4	`trainProps`	`trainProperties_T`	Database
5	`MRSP`	`MRSP_Profile_t`	??
6	`MA`	`MAs_t`	??
7	`MA_updated`	`bool`	internal
8	`MRSP_updated`	`bool`	internal

Table 38. Overview of inputs

Package/Variables	Information	From RBC	Onboard operating level				
			0	NTC	1	2	3
Packet 3	National Values	No	A	A	A	A	A
		Yes	R [2]	R [2]	R [2]	A	A
Packet 5	Linking	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
V_Main Packet 12	Signalling Related Speed Restriction	No	R [1]	R [1]	A	R [1]	R [1]
		Yes					
Packet 12, 15 Packet 80 Packet 49	Movement Authority + (optional) Mode Profile + (optional) List of Balises for SH area	No	R [1]	R [1]	A [4]	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3] [4] [5]	A [3] [4] [5]
Packet 16	Repositioning Information	No	R	R	A	R	R
		Yes					
Packet 21	Gradient Profile	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 27	International SSP	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 51	Axle Load speed profile	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 41	Level Transition Order	No	A	A	A	A	A
		Yes	A	A	A	A	A
Packet 46	Conditional Level Transition Order	No	A [11]	A [11]	A [11]	A [11]	A [11]
		Yes					
Packet 42	Session Management	No	A	A	A	A	A
		Yes	A	A	A	A	A
Packet 45	Radio Network registration	No	A	A	A	A	A
		Yes	A	A	A	A	A
Packet 57	MA Request Parameters	No					
		Yes	A	A	A	A	A
Packet 58	Position Report parameters	No					
		Yes	A	A	A	A	A
Package 63 + Message Radio 2 + (optional) Packet 49	SR Authorisation + (optional) List of Balises in SR mode	No					
		Yes	R	R	R	A [3]	A [3]
Packet 137	Stop if in SR mode	No	R	R	A	A	A
		Yes					
D_SR in Packet 13	SR distance information from loop	No	R	R	A	R	R
		Yes					

Packet 65	Temporary Speed Restriction	No	A	R [1] [2]	A	A [8]	A [8]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 66	Temporary Speed Restriction Revocation	No	A	R [1] [2]	A	A	A
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Package 64	Inhibition of revocable TSRs from balises in L2/3	No					
		Yes	R [2]	R [2]	R [2]	A	A
Packet 141	Default Gradient for TSR	No	A	R [1] [2]	A	A	A
		Yes					
Packet 70	Route Suitability Data	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 71	Adhesion Factor	No	R[1]	R[1]	A	R	R
		Yes	R[2]	R[2]	R[2]	A	A
Packet 72	Plain Text Information	No	A	R [1] [2]	A	A	A
		Yes	R [2]	R [2]	R [2]	A [12]	A [12]
Packet 76	Fixed Text Information	No	A	R [1] [2]	A	A	A
		Yes	R [2]	R [2]	R [2]	A [12]	A [12]
Packet 79	Geographical Position	No	A	R [1] [2]	A	A	A
		Yes	R [2]	R [2]	R [2]	A	A
Packet 131	RBC Transition Order	No	R	R	R	A	A
		Yes	R	R	R	A [3]	A [3]
Packet 132	Danger for SH information	No	A [13]	A [13]	A	A	A
		Yes					
Package 135	Stop Shunting on desk opening	No	A	A	A	A	A
		Yes					
Packet 133	Radio Infill Area information	No	R	R	A	R [1]	R [1]
		Yes					
Package 42	Session Management with neighbouring RIU	No	R	R	A	R	R
		Yes					
Packet 134	EOLM information	No	A	A	A	A	A
		Yes					
Messager 45	Assignment of Co-ordinate system	No					
		Yes	A [10]	A [10]	A [10]	A [10]	A [10]
Packet 136	Infill Location Reference	No	R	R	A	R [1]	R [1]
		Yes					
Packet 39, Packet 68	Track Conditions excluding big metal masses	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 67	Track condition big metal masses	No	A	A	A	A	A
		Yes					

Header Balise	Location Identity (NID_C + NID_BG transmitted in the balise telegram)	No	A	A	A	A	A
		Yes					
Radio Message 6	Recognition of exit from TRIP mode	No					
		Yes	R	R	R	A	A
Message Radio 8	Acknowledgement of Train Data	No					
		Yes	A	A	A	A	A
Message 9 Packet 80 Packet 49	Co-operative shortening of MA + (optional) Mode Profile + (optional) List of Balises for SH area	No					
		Yes	R	R	R	A [3] [4] [5]	A [3] [4] [5]
Message Radio 16	Unconditional Emergency Stop	No					
		Yes	R [2]	R [2]	R [2]	A	A
Message Radio 15	Conditional Emergency Stop	No					
		Yes	R [2]	R [2]	R [2]	A	A
Message Radio 18	Revocation of Emergency Stop (Conditional or Unconditional)	No					
		Yes	R	R	R	A	A
Message Radio 27	SH refused	No					
		Yes	R	R	R	A [3]	A [3]
Message Radio 28 + (optional) Packet 49	SH authorised + (optional) List of Balises for SH area	No					
		Yes	R	R	R	A [3]	A [3]
??	Trackside constituent System Version	No	A	A	A	A	A
		Yes	A	A	A	A	A
Packet 2	System Version order	No	A	A	A	A	A
		Yes					
Message Radio 34	Track Ahead Free Request	No					
		Yes	R	R	R	A [3]	A [3]
Packet 140 Track to train, Packet 40 Train to track	Train Running Number	No					
		Yes	R	R	R	A	A
Message Radio 38	Initiation of session	No					
		Yes	R	R	R	A	A
Message 39	Acknowledgement of session termination	No	A	A	A	A	A
		Yes	A	A	A	A	A

Message 40	Train Rejected	No					
		Yes	R	R	R	A	A
Message 41	Train Accepted	No					
		Yes	R	R	R	A	A
Message Radio 43	SoM Position Report Confirmed by RBC	No					
		Yes	R	R	R	A	A
Packet 138	Reversing Area Information	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 139	Reversing Supervision Information	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Packet 254	Default Balise/Loop/RIU Information	No	A	A	A	A	A
		Yes					
Packet 90	Track Ahead Free up to level 2/3 transition location	No	A [9]	A [9]	A [9]	R	R
		Yes					
Package 52	Permitted Braking Distance Information	No	R [1]	R [1]	A	R [1]	R [1]
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Package 88	Level Crossing information	No	R [1] [2]	R [1] [2]	A	A	A
		Yes	R [2]	R [2]	R [2]	A [3]	A [3]
Package 6(0)	Virtual Balise Cover order	No	A	A	A	A	A
		Yes					
Package 44	Data to be used by applications outside ERTMS/ETCS	No	A	A	A	A	A
		Yes	A	A	A	A	A
		Onboard operating level					
	Information from National System X through STM interface	0	NTC X	NTC Y	1	2	3
??	STM max speed	A [7]	R	R [6]	A [7]	A [7]	A [7]
??	STM system speed/distance	A [7]	R	R	A [7]	A [7]	A [7]

Figure 8. Lists of packages and their handling depending on train modes

40	Packet 39, 68	Track conditions sound horn, non stopping areas, tunnel stopping areas	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
41	Packet 67	Track condition big metal masses	NR	A[2][4]	A	A	A	A	A	A	A	A	A	A	A[1]	NR	NR	A	R
42	Header Balise	Location Identity (NID_C +NID_BG)	NR	A[2]	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
43	Message Radio 6	Recognition of exit from TRIP mode	NR	R	R	R	R	R	R	R	R	R	R	R	A	NR	NR	R	R
44	Message Radio 8	Acknowledgement of Train Data	NR	A[2]	R	R	A	A	A	A	R	R	A	A	A	NR	NR	A	A
45	Message 9	Co-operative shortening of MA	NR	R	R	R	A	A	R	A	R	R	R	R	R	NR	NR	R	R
46	Packet 80	+ (optional) Mode Profile																	
47	Packet 49	+ (optional) List of Balises for SH area																	
48	Message Radio 16	Unconditional Emergency Stop	NR	A[2]	R	R	A	A	A	A	R	R	A	R	R	NR	NR	A	R
49	Message Radio 15	Conditional Emergency Stop	NR	R	R	R	A	A	R	A	R	R	A	R	R	NR	NR	A	R
50	Message Radio 18	Revocation of Emergency Stop	NR	R	R	R	A	A	R	A	R	R	R	R	A[1]	NR	NR	R	R
51		(Conditional or Unconditional)																	
52	Message Radio 27	SH refused	NR	A[2]	R	R	A	A	A	A	R	R	R	R	A[1]	NR	NR	R	R
53	Message Radio 28 + (optional) Packet 49	SH authorised + (optional) List of Balises for SH area	NR	A[2]	R	R	A	A	A	A	R	R	R	R	A[1]	NR	NR	R	R
54	??	Trackside constituent System Version	NR	A	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
55	Packet 2	System Version order	NR	A	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
56	Message Radio 34	Track Ahead Free Request	NR	A[2]	R	R	R	A	A	A	R	R	R	R	A[1]	NR	NR	R	R
57	Packet 140 Track to train, Packet 40 Train to track	Train Running Number	NR	A[2]	R	R	A	A	A	A	R	A	R	A	A	NR	NR	R	A
58	Message Radio 38	Initiation of session	NR	A	R	R	A	A	A	A	A	A	R	A	A	NR	NR	R	A
59	Message 39	Acknowledgement of session termination	NR	A	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
60	Message 40	Train Rejected	NR	A[2]	R	R	R	R	R	R	R	R	R	R	R	NR	NR	R	R
61	Message 41	Train Accepted	NR	A[2]	R	R	R	R	R	R	R	R	R	R	R	NR	NR	R	R
62	Message Radio 43	SoM Position Report Confirmed by RBC	NR	A[2]	R	R	R	R	R	R	R	R	R	R	R	NR	NR	R	R
63	Packet 138	Reversing Area Information	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	A
64	Packet 139	Reversing Supervision Information	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	A
65	Packet 254	Default Balise/Loop/RIU Information	NR	A[2]	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
66	Packet 90	Track Ahead Free up to level 2/3 transition location	NR	A[2]	R	R	A	A	A	A	R	R	A	A	A	NR	NR	A	R
67	Package 52	Permitted Braking Distance Information	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
68	Package 88	Level Crossing information	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
69	Package 6(0)	Virtual Balise Cover order	NR	A	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A
70	Package 44	Data to be used by applications outside ERTMS/ETCS	NR	A	A	A	A	A	A	A	A	A	A	A	A	NR	NR	A	A

	Packet, Message	Information	NP	SB	PS	SH	FS	LS	SR	OS	SL	NL	UN	TR	PT	SF	IS	SN	RV
			Modes																
1	Packet 3	National Values	NR	A[2]	A	A	A	A	A	A	A	A	A	A	A[1]	NR	NR	A	A
2	Packet 5	Linking	NR	A[2][4]	R	R	A	A	A	A	R	A	A	R	A[1]	NR	NR	A	R
3	V_Main in Packet 12	Signalling Related Speed Restriction	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
4	Packet 12, 15	Movement Authority	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
5	(optional) Packet 80	+ (optional) Mode Profile																	
6	(optional) Packet 49	+ (optional) List of Balises for SH area																	
7	Packet 16	Repositioning Information	NR	R	R	R	A	A	R	A	R	R	R	R	R	NR	NR	R	R
8	Packet 21	Gradient Profile	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
9	Packet 27	International SSP	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
10	Packet 51	Axle load speed profile	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
11	??	STM max speed	NR	A[2]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	A	R
12	??	STM system speed/distance	NR	A[2]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	R	R
13	Packet 41	Level Transition Order and Conditional Level Transition Order	NR	A[2]	A[7]	A[7]	A	A	A	A	A	A	A	A	A[1][5]	NR	NR	A	R
14	Packet 42	Session Management	NR	A	A[3]	A[3]	A	A	A	A	A	A	A	A	A[1]	NR	NR	A	A
15	Packet 49	Radio Network registration	NR	A[2]	A	A	A	A	A	A	A	A	A	A	A[1]	NR	NR	A	A
16	Packet 57	MA Request Parameters	NR	A[2]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
17	Packet 58	Position Report parameters	NR	A[2]	R	R	A	A	A	A	R	A	A	R	A[1]	NR	NR	A	A
18	Package Radio 63 + Message Radio 2 + (optional) Packet 49	SR Authorisation+																	
19		(optional) List of Balises in SR mode	NR	A[2][4]	R	R	R	R	A	R	R	R	R	R	A[1]	NR	NR	R	R
20	Packet 137	Stop if in SR mode	NR	R	R	R	R	R	A	R	R	R	R	R	R	NR	NR	R	R
21	D_SR in Packet 13	SR distance information from loop	NR	R	R	R	R	R	A[6]	R	R	R	R	R	R	NR	NR	R	R
22	Packet 65	Temporary Speed Restriction	NR	A[2][4]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	A	R
23	Packet 66	Temporary Speed Restriction Revocation	NR	A[2][4]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	A	R
24	Package 64	Inhibition of revocable TSRs from balises in L2/3	NR	A[2]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	A	R
25	Packet 141	Default Gradient for TSR	NR	A[2][4]	R	R	A	A	A	A	R	R	A	A	A[1]	NR	NR	A	R
26	Packet 70	Route Suitability Data	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
27	Packet 71	Adhesion Factor	NR	A[2][4]	R	R	A	A	A	A	R	R	A	R	A[1]	NR	NR	A	R
28	Packet 72	Plain Text Information	NR	A[2]	R	R	A	A	A	A	R	A	A	A	A[1]	NR	NR	A	A
29	Packet 76	Fixed Text Information	NR	A[2]	R	R	A	A	A	A	R	A	A	A	A[1]	NR	NR	A	A
30	Packet 79	Geographical Position	NR	A[2]	R	R	A	A	A	A	R	A	A	A	A[1]	NR	NR	A	R
31	Packet 131	RBC Transition Order	NR	A[2][4]	A[8]	A[8]	A	A	A	A	A	A	R	A	A[1]	NR	NR	R	R
32	Packet 132	Danger for SH information	NR	R	R	A	R	R	R	R	R	R	R	R	R	NR	NR	R	R
33	Package 135	Stop Shunting on desk opening	NR	R	A	R	R	R	R	R	R	R	R	R	R	NR	NR	R	R
34	Package 133	Radio Infill Area information	NR	R	R	R	A	A	A	A	R	R	R	R	R	NR	NR	R	R
35	Package 42	Session Management with neighbouring RIU	NR	R	R	R	A	A	A	A	R	R	R	R	R	NR	NR	R	R
36	Package 134	EOLM information	NR	R	R	A	A	A	A	A	A	A	A	A	R	NR	NR	A	A
37	Message Radio 45	Assignment of Co-ordinate system	NR	A[2]	R	R	R	R	A	R	R	A	A	R	A[1]	NR	NR	A	R
38	Package 136	Infill Location Reference	NR	R	R	R	A	A	R	R	R	R	R	R	R	NR	NR	R	R
39	Packet 39, 68	Track Conditions excluding sound horn, non stopping areas, tunnel stopping areas and big metal masses	NR	A[2][4]	R	R	A	A	A	A	R	A	A	A	A[1]	NR	NR	A	R

Figure 9. Lists of packages and their handling depending on train modes

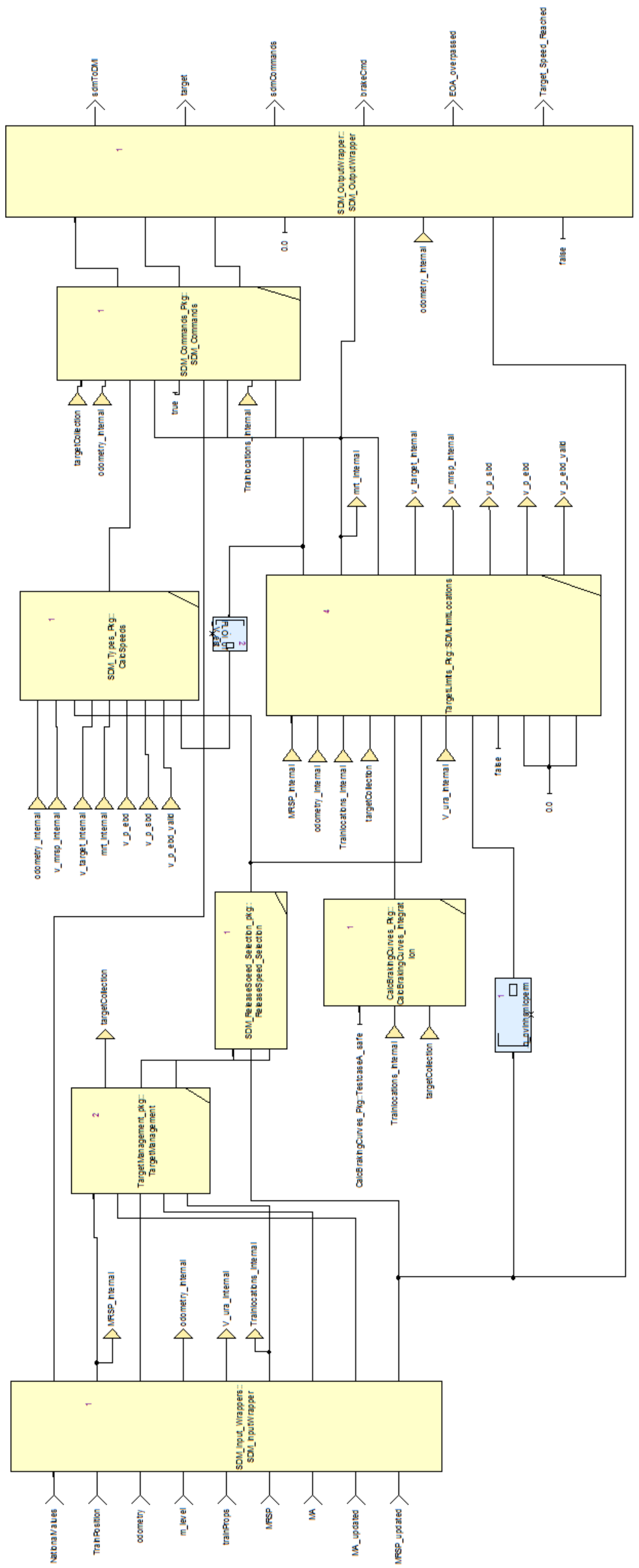


Figure 10. Structure of component ProvidePositionReport

6.2.1.1 Input 0: `NationalValues`

This input is packet 3 of [1, Chapt. 8], describing the national values.

6.2.1.2 Input 1: `TrainPosition`

This input is the current train position.

6.2.1.3 Input 2: `odometry`

This input is the odometry data.

6.2.1.4 Input 3: `m_level`

This input is the current level of the train.

6.2.1.5 Input 4: `trainProps`

This input is a set of train related properties.

6.2.1.6 Input 5: `MRSP`

This input is the most restrictive speed profile.

6.2.1.7 Input 6: `MA`

This input is a movement authority.

6.2.1.8 Input 7: `MA_updated`

This flag is true if the movement authority has been updated in this clock cycle and false otherwise.

6.2.1.9 Input 8: `MRSP_updated`

This flag is true if the most restrictive speed profile has been updated in this clock cycle and false otherwise.

6.2.2 Output

Based on the input the block produces the following output. Table 39 gives an overview.

Index	Output name	Output type
0	`sdmToDMI`	`speedSupervisionForDMI_T`
1	`target`	`Target_T`
2	`sdmCommands`	`SDM_Commands_T`
3	`brakeCmd`	`Brake_command_T`
4	`EOA_overpassed`	`bool`
5	`Target_Speed_Reached`	`bool`

Table 39. Overview of outputs

6.2.2.1 Output 0: `sdmToDMI`

This output contains information about different speeds and positions, on the one hand and the current supervision status, on the other hand. This information shall be displayed to the driver.

6.2.2.2 Output 1: `target`

This output is the most restrictive displayed target (MRDT).

6.2.2.3 Output 2: `sdmCommands`

This output gives some intermediate results of operator SDM_Commands. It is currently used for test purposes only.

6.2.2.4 Output 3: `brakeCmd`

This output is the brake command, indicating whether performing the service brake or the emergency brake have been commanded.

6.2.2.5 Output 4: `EOA_overpassed`

This output is true if the end of authority has been overpassed and false otherwise.

6.2.2.6 Output 5: `Target_Speed_Reached`

This output is true if the current speed is greater than or equal the target speed and false otherwise.

6.2.3 SDM_InputWrapper in Train Supervision

6.2.3.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13]: Speed and distance monitoring

6.2.3.2 Short description of the functionality

The motivation for this operator is to convert all inputs of block “Speed Supervision” that contain information about length, speed, distance, and acceleration defined as integer into `real` to allow automatically the highest precision in the calculations by the meaning of floating point operations. In addition, to ease the modeling, inside block “Speed Supervision” only units meters ($[m]$), seconds($[s]$), meters per second($[\frac{m}{s}]$), and meters per square second($[\frac{m}{s^2}]$) are used.

6.2.3.3 Interface

6.2.3.4 Functional Design Description

This operator forwards input messages, takes data from complex data types or transforms inputs messages into an internal type thereby converting int to real.

6.2.3.5 Reference to the Scade Model

6.2.4 TargetManagement in Train Supervision

6.2.4.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13.8.2]: Determination of the supervised targets

6.2.4.2 Short description of the functionality

This operator calculates/updates the list of targets to be supervised by the block “Train Supervision”. Taking the current movement authority, the most restrictive speed profile and the current maximum safe front end position as an input, the operator outputs a single End of Authority target, a list of all MRSP-Targets and a list of all LoA-Targets.

6.2.4.3 Interface

6.2.4.4 Functional Design Description

6.2.4.4.1 Derivation of Targets from Movement Authority Sections

The sections of the *Movement Authority* could cause two types of targets:

End Of Authority(EoA) only one could exist and this is only in the *end section* of the *MA*

Limit of Authority (LoA) is possibly in every section of the *MA* except the end section

In every cycle in which the MA is updated, the operator iterates through the entire MA and puts all speed limitations by *LoA*s into a list of targets. The end section is used to derived the *EoA* target. All LoA targets are sorted by location.

6.2.4.4.2 Derivation of Targets from MRSP

According to [1, Chapt. 3.13.8.2], every speed decrease of the MRSP is used to derive a target. Therefore in every cycle in which the MRSP is updated, the operator iterates through the entire MRSP searching for all MRSP targets. For this purpose, every element of the MRSP is compared with its successor.

6.2.4.4.3 Update of Targets

In every cycle the operator monitors whether all targets are already passed. To this end, it iterates over the list of targets comparing the current max safe front end position with the target position.

6.2.4.5 Reference to the Scade Model

6.2.5 CalcBrakingCurves_Integration in Train Supervision

6.2.5.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13.8.3]: Emergency Brake Deceleration curves (EBD)

- [1, Chapt. 3.13.8.4]: Service Brake Deceleration curves (SBD)
- [1, Chapt. 3.13.8.5]: Guidance curves (GUI)

6.2.5.2 Short description of the functionality

For each type of target a certain braking curve has to be calculated. This curve enables proactive monitoring of the train's speed. A reverse lookup on this braking curve indicates, where the train has to start braking given the current speed. The braking curve does not depend on the actual train status. As a consequence the braking curve stays constant over time. As a legitimate simplification the calculation of the braking curve is not extended after the estimated front end position of the train has been passed.

6.2.5.3 Interface

6.2.5.4 Functional Design Description

The calculation of the braking curve takes the complex function A_{safe} as an input, which describes the overall braking performance of the train in a speed and position dependent meaning. This two-dimensional function needs to be simplified for every target to get a function of position to speed. Each individual target has a position and a speed (a point in the distance speed plane) and is the starting point of the braking curve. The first step is to get the deceleration of the train at the target point from the A_{safe}-function. Afterward the calculation iterates through the A_{safe}-function until the current estimated front end position is reached. Two cases can be distinguished:

- a new distance step of A_{safe} is reached
- a new speed step of A_{safe} is reached

Both cases are checked and the applicable one is used to calculate a new arc. Every arc of the braking curve consists of:

- the distance where the arc begins,
- the speed at the point where the arc begins,
- the deceleration for the whole arc.

An abstract overview of the calculation could be seen in Fig. 11.

Currently, the model supports the calculation of the following braking curves:

- the Emergency Brake Deceleration curve for the most restrictive speed profile,
- the Emergency Brake Deceleration curve for the limit of authority,
- the Emergency Brake Deceleration curve for the end of authority, and

6.2.5.5 Reference to the Scade Model

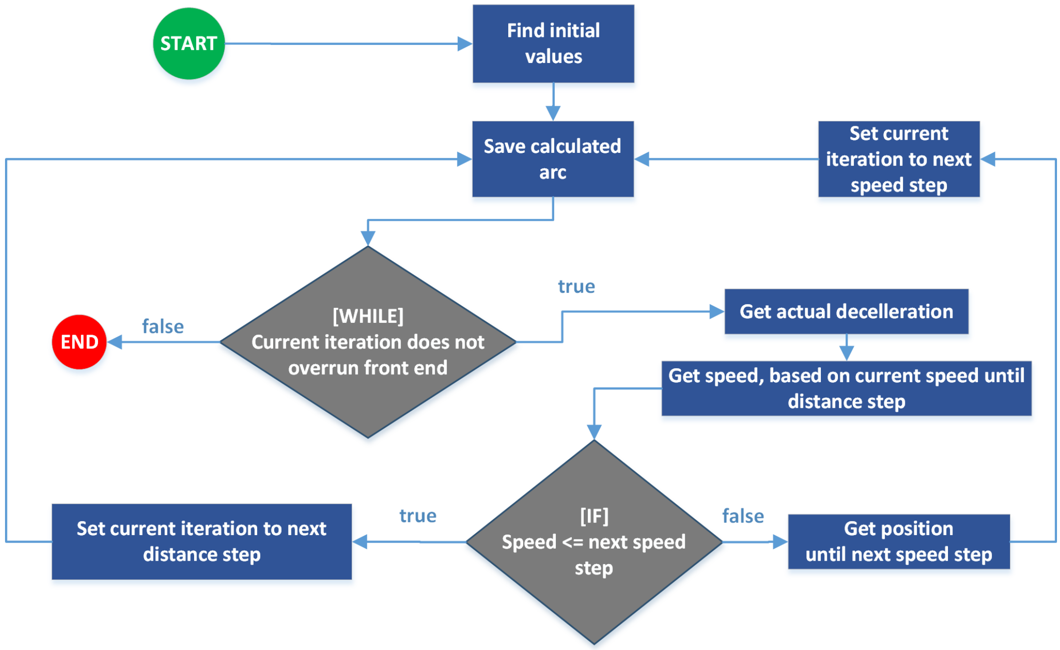


Figure 11. Calculation of Braking Curves

6.2.6 SDMLimitLocations in Train Supervision

6.2.6.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13.9]: Supervision Limits
- [3, Chapt. 5.3.1.2]: f_{41} – accuracy of speed known on-board
- [1, Chapt. 3.13.10]: Monitoring Commands as reference for required outputs of this module

6.2.6.2 Short description of the functionality

This operator calculates the various locations needed to determine the speed and distance monitoring commands. The current implementation of functionality is stateless and requires a complete recalculation each cycle.

6.2.6.3 Interface

6.2.6.3.1 Input

1. `MRSPProfile` Speed profile related to current track under train.
2. `odometry`, `trainLocations` External state of train provided by odometry.
3. `targetCollection` The different target (list) types wrapped in a structure.
4. `curveCollection` The related braking curves correlated to above targets.
5. $v_{release}$ Release speed as defined by external sources.
6. $v_{mathitura}$ Speed under reading amount.

7. `inhibitUnderReadingCompensation` A flag defined by National Value, relating to above item.

8. T_{bs}, T_{be}, $T_{tractionCutOff}$ Time constants defined externally or in other modules.

6.2.6.3.2 Output

1. `locations` Internal type to wrap the locations calculated herein and pass it on directly to SDM-Commands-Operator.

2. `MostRestrictiveTarget` An internal structure to contain the information the target based locations are linked to.

3. `FLOIisSBI1` Flag is true if First Line of Intervention uses the service brake curve (SBI1) or false if it uses deceleration values based on the emergency brake curve (SBI2).

4. v_{target} The designated speed of the Most Restrictive Target. This is a convenience reference into the above data structure.

5. $v_{mathitMRSP}$ The current Most Restrictive Speed at the Max Safe Frontend of the train.

6.2.6.4 Functional Design Description

This operator gathers all necessary input values and computes some frequently used intermediate values in the operators `surplusTractionDeltas` and v_{bec}. The other input preparation operator is the `TargetSelector` whose main task is to dissect the list of targets to find the Most Restrictive Target. The accompanying braking curves are extracted and promoted to trailing location calculations. Also the special values of the EOA are exposed.

The operator creates the requested values for the commands package. These are in particular the preindication locations for EBD and SBD based targets, the release speed monitoring start locations, the locations for target speed monitoring of the I-, W-, P- and FLOI-curve, the related FLOI speed and the location of the permitted speed supervision limit. Included in the output are also certain flags for the validity of linked values.

6.2.7 CalcSpeeds in Train Supervision

6.2.7.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13.9]: Supervision Limits

6.2.7.2 Short description of the functionality

This operator calculates the various speeds needed to determine the speed and distance monitoring commands.

6.2.7.3 Interface

6.2.7.4 Functional Design Description

This operator will be integrated into other operators in the next iteration.

only in special case or link to the Scade model

6.2.8 ReleaseSpeed_Selection in Train Supervision

6.2.8.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.8]: Movement authority

6.2.8.2 Short description of the functionality

This operator outputs the release speed which can be given either by the national values or the movement authority.

6.2.8.3 Interface

6.2.8.4 Functional Design Description

This operator will be integrated into other operators in the next iteration.

6.2.8.4.1 Reference to the Scade Model

6.2.9 SDM_Commands in Train Supervision

6.2.9.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13.10]: Speed and distance monitoring commands

6.2.9.2 Short description of the functionality

This operator models the speed and distance monitoring commands. More precisely, it triggers the service or emergency brake and outputs the current supervision status of the OBU together with information on speeds and locations to the driver.

6.2.9.3 Interface

6.2.9.4 Functional Design Description

The OBU can be in any of three types of speed and distance monitoring modes: ceiling speed monitoring, release speed monitoring and target speed monitoring. We use a state machine to model the switching between the three modes: each state models a mode and a transition between to states is enabled if the condition two switch between the two corresponding modes is evaluated to true. In each mode, the OBU can be in up to five different supervision stati. The behavior of changing from one status to another is also modeled as a state machine. As a result, the model is a hierarchical state machine.

6.2.9.5 Reference to the Scade Model

6.2.10 SDM_OutputWrapper in Train Supervision

6.2.10.1 Reference to the SRS or other Requirements (or other requirements)

- [1, Chapt. 3.13]: Speed and distance monitoring

6.2.10.2 Short description of the functionality

This operator is the counterpart to operator SDM_OutputWrapper—that is, it converts all internal outputs of block “Speed Supervision” that contain information about length, speed, distance, and acceleration defined as real into int, such that all other blocks can stick to their types and also performs the calculation into units used by the environment.

6.2.10.3 Interface

6.2.10.4 Functional Design Description

This operator forwards input messages and transforms inputs messages into an internal type thereby converting real to int.

6.2.10.5 Reference to the Scade Model

6.3 Manage ETCS Procedures

6.3.1 Start of Mission - Awakness of Train

6.3.1.0.1 Reference to the SRS or other Requirements (or other requirements)

Chapter 5, § 5.4

6.3.1.1 Short descriptoiin of the functionality

This functionality describes the Start of Mission procedure of the train until the status of the awakness. From this point of the awakness the train will be able to start different modes, levels and further procedure. See scope of the Start of Mission - Awakness of train in the figure below.

6.3.1.2 Interface

6.3.1.2.1 Input Flow

- Information from TIU
- Action from Driver (DMI)
- Information from Position Calculation
- Information from Persistent Data
- Information from Management of Radio Communication

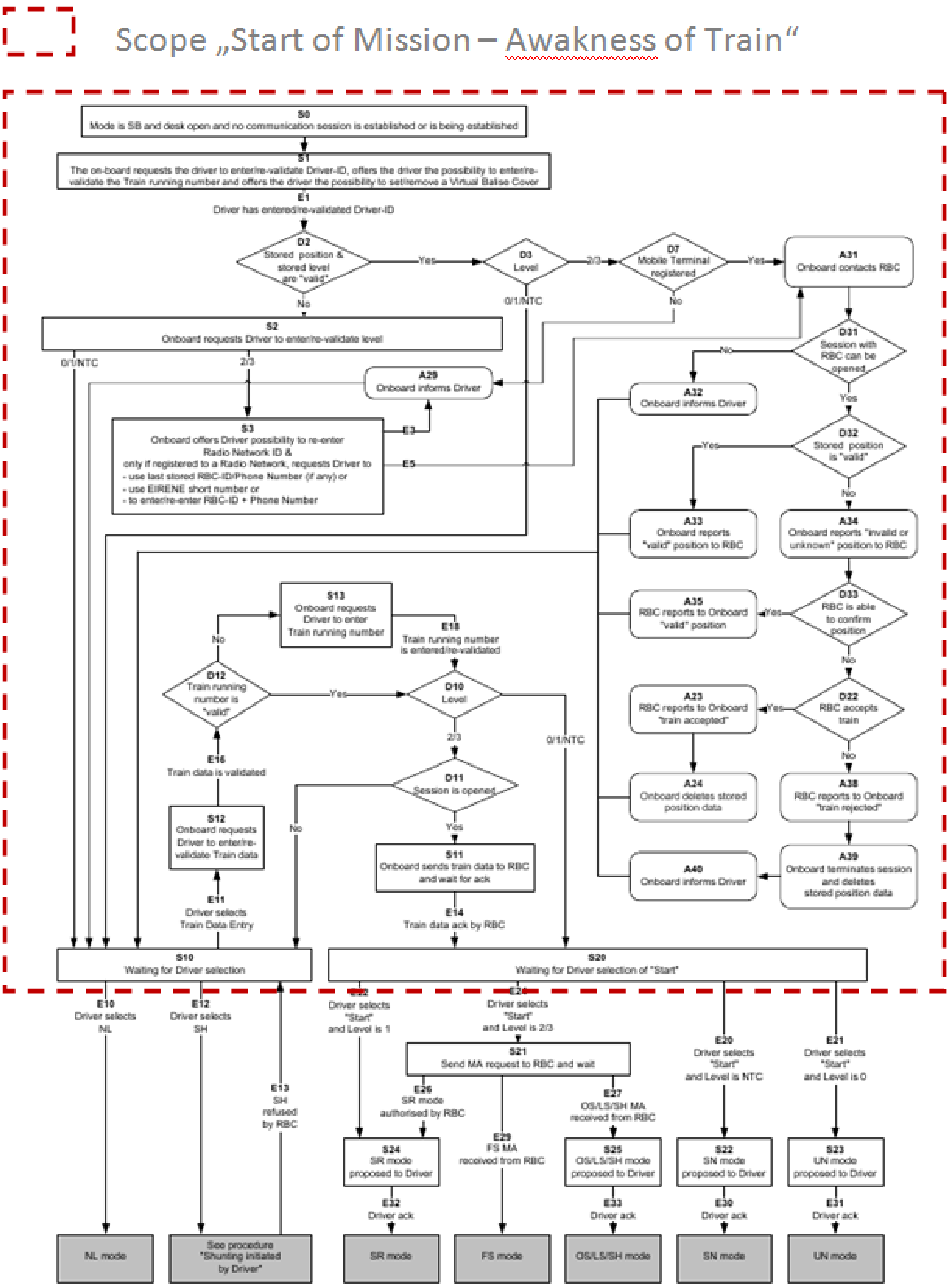


Figure 12. Start of Mission - Awakness of Train

- Information from Mode and Level Management (Level and Mode Status)
- Information from Radio Block Control

6.3.1.2.2 Output Flow

- Information to Management of Radio Communication
- Request to Management of Radio Communication
- Request to Driver (DMI)
- Request to Mode and Level Management (Request Mode Change)
- Request to Radio Block Control (Validation of Train Data)

6.3.1.3 Functional Design Description

For the third iteration just a part of the Scope has been design. To complete the scenario in the third iteration the ideal path to the awakness of train until the state "waiting for Driver selection of "Start"" have been realized. Furthermore the initial data from the persistend database such as Level, Driver ID, Train Number, Train Data, Radio Number, RBC ID hase been consider as constants.

6.3.1.4 Refernce to the Scade Model

`https://github.com/openETCS/modeling/blob/master/model/Scade/System/ObuFunctions/Procedures/ManageProcedure_Pkg.xscade`

6.3.2 Start of Mission in Level 2 or 3 Mode SR FS OS LS SH

6.3.2.1 Reference to the SRS or other Requirements (or other requirements)

Chapter 5, § 5.4

6.3.2.2 Short description of the functionality

This functionality describes the Start of Mission procedure of the train in Level 2 or 3 and the Modes SR FS OS LS SH where the train under the defined Mode Level supervision starts running. See scope of the Start of Mission - Level 2 or 3 and Modes SR FS OS LS SH in the figure below.

6.3.2.3 Interface

6.3.2.3.1 Input Flow

- Action from Driver (DMI)
- Information from Mode and Level Management (Level and Mode Status)
- Information from Movement Authority Managmement (Receiving of Movement Authority)

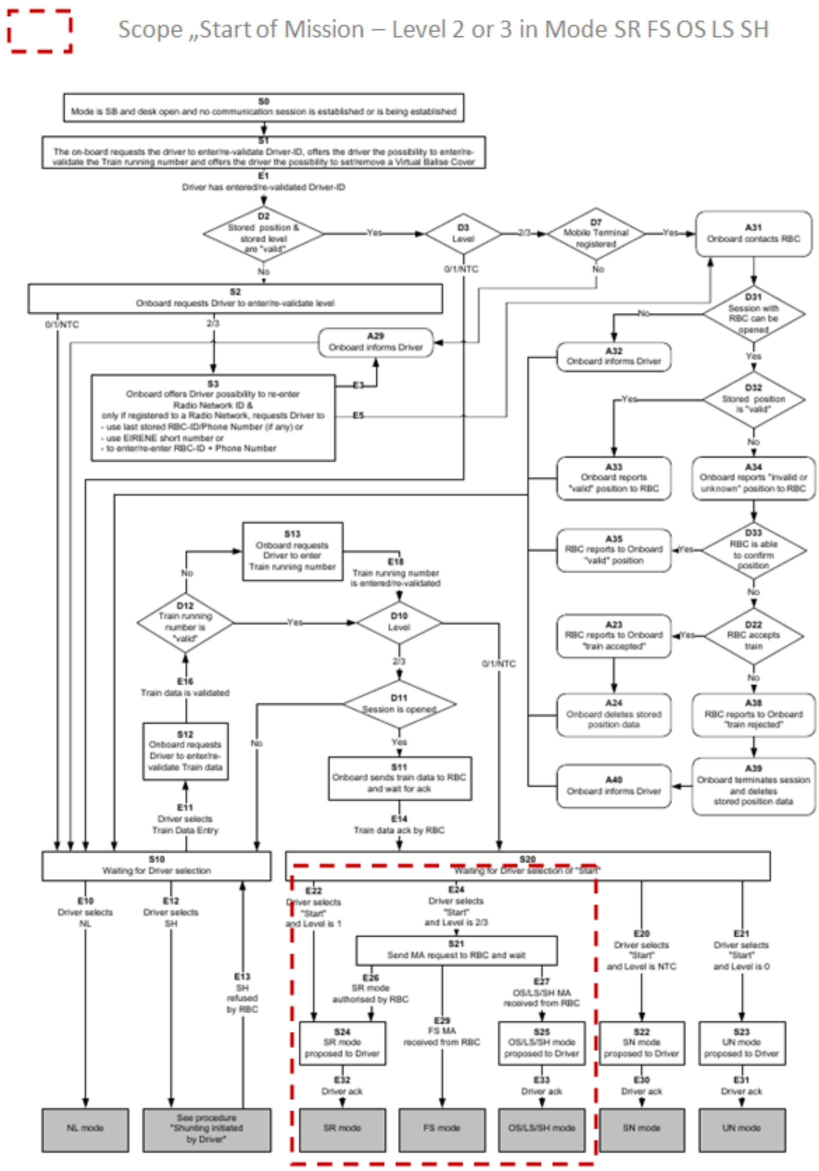


Figure 13. Start of Mission - Start of Mission in Level 2 or 3 and Mode SR FS OS LS SH

6.3.2.3.2 Output Flow

- Request to Driver (DMI)
- Request to Movement Authority Management (Request Movement Authority)
- Request to Mode and Level Management (Request Mode Change)

6.3.2.4 Functional Design Description

For the third iteration just a part of the Scope has been design. To complete the scenario in the third iteration the path "Full Supervision Movement Authority received from RBC" has been realized. The state will end after the train receives the Change Authority to FS and will be ready to run.

6.3.2.5 Refernce to the Scade Model

`https://github.com/openETCS/modeling/blob/master/model/Scade/System/ObuFunctions/Procedures/SoM_SR_FS_OS_LS_SH_SN_UN.xscade`

6.4 Manage Track Data

6.4.1 F.2.2 Calculate Train Position

6.4.1.1 Short Description of Functionality

The main purpose of the function is to calculate the locations of linked and unlinked balise groups (BGs) and the current train position while the train is running along the track.

In detail, the calculateTrainPosition function provides a couple of essential subfunctions for the onboard unit. These are mainly

- creating and maintaining an obu internal coordinate system for all types of location based data
- storing all linked and unlinked balise groups resulting from over passing or from announcements (linking information) from the track
- calculating and maintaining the locations of all stored balise groups during the train trip, based on odometry and linking information
- permanently calculating the current train position based on odometry and passed balise group information
- providing the last recently passed linked balise group as the LRBG
- providing additional position attribute information
- deleting stored balise groups, when appropriate
- detecting linking consistency errors
- determining, if linking is used on board

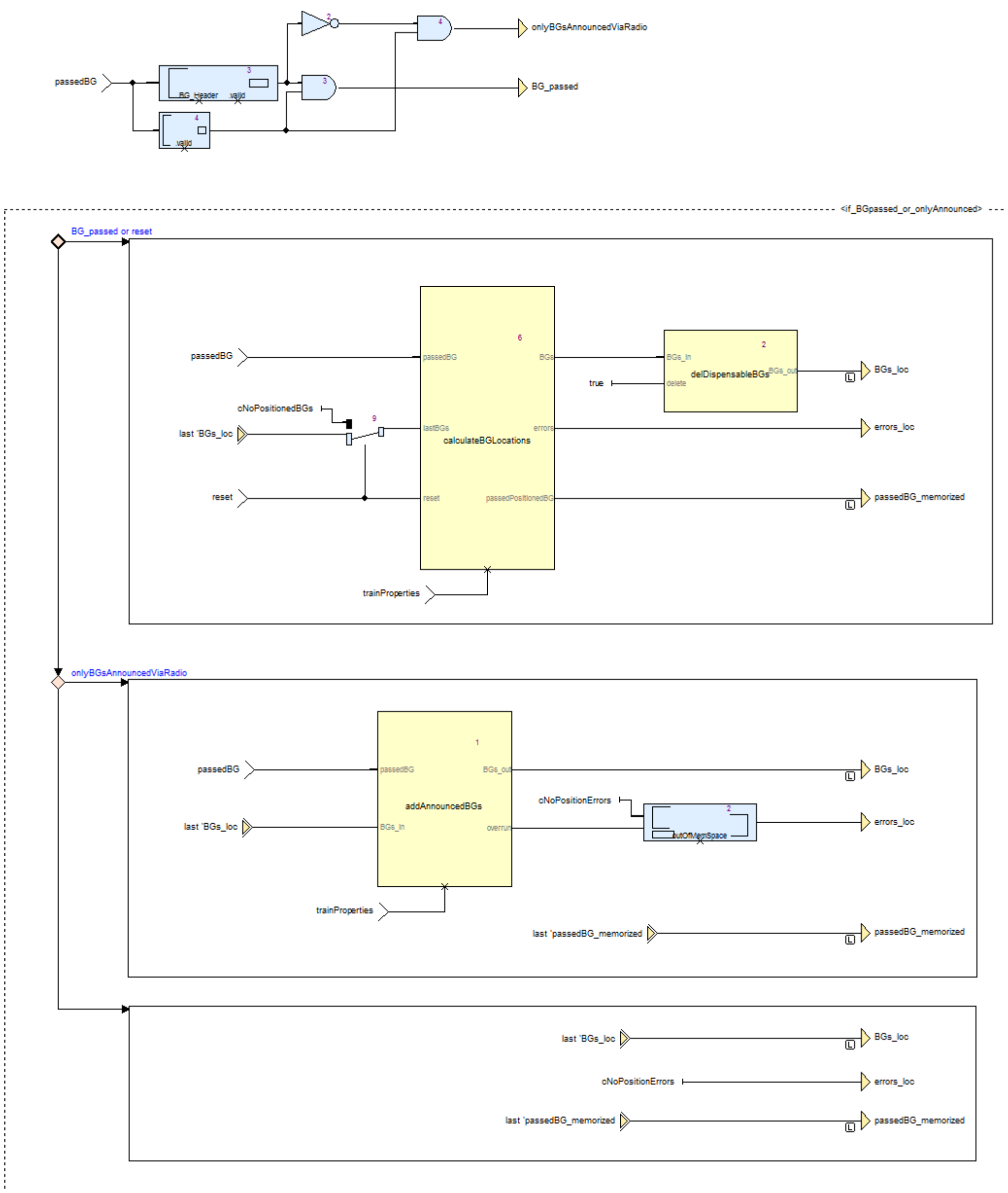


Figure 14. Calculating the balise group locations.

The calculation algorithms for locations and positions are implemented as specified in `https://github.com/openETCS/SRS-Analysis/blob/master/System%20Analysis/WorkingRepository/Group4/SUBSET_26_3-6/DetermineTrainLocationProcedures.pdf` .

6.4.1.2 Functional Structure in Stages

calculateTrainPositions receives its input information via its passedBG entry as an event. The first decicion to be made is, if an input event is available and if it origins from a balise group just over passed or from the RBC via radio.

If the passedBG input events is caused by over passing a balise group, the balise group gets an OBU coordinate system location assigned ("calculateBGLocations") and is stored internally. If the just passed balise group announces more balise groups ahead via linking, they are stored with their locations calculated.

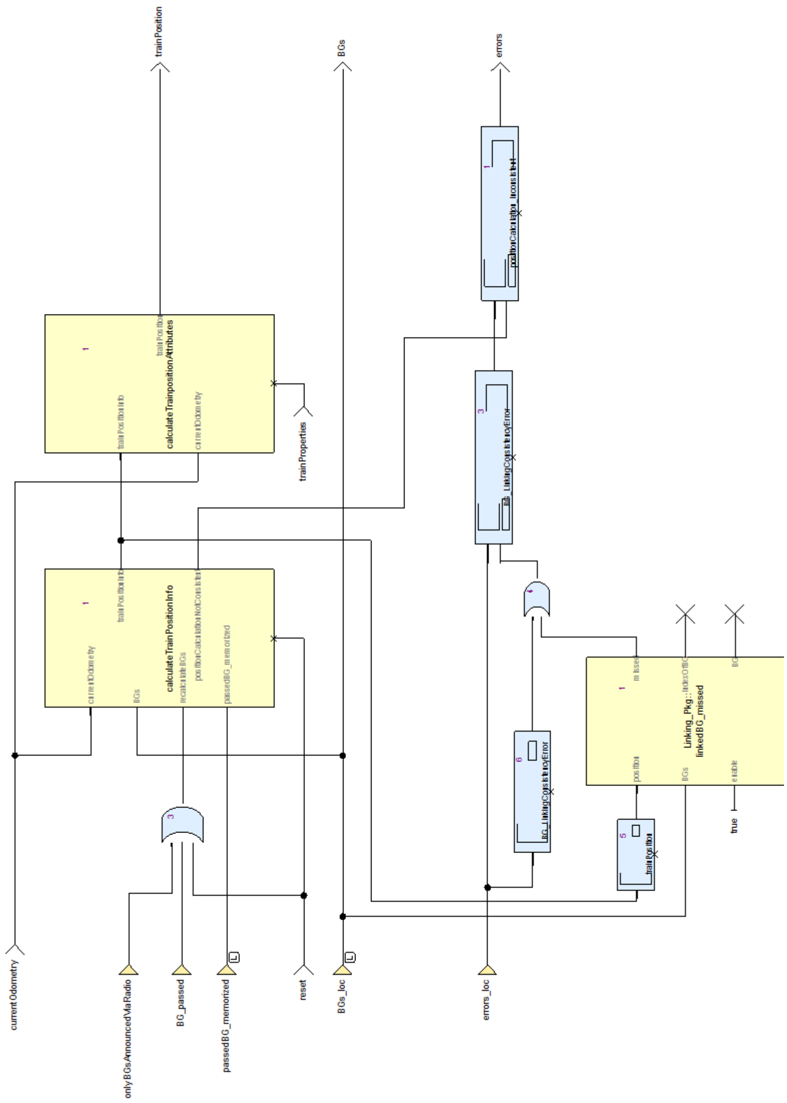


Figure 15. Calculating the current train position and attributes.

If the passedBG input event origins from RBC data received ("addAnnouncedBGs"), it only announces balise groups ahead. These announced balise groups get their locations calculated with reference to the LRBG and are stored as well.

"calculateBGLocations" and "addAnnouncedBGs" produce a list of known balise groups ("BGs").

The following stages "calculateTrainPositionInfo" and "calculateTrainpositionAttributes" all the time calculate the current train position by using the list of known balise groups (including the LRBG) and the current odometry information and determine additional position attributes.

In parallel "linkedBG_missed" is part of linking consistency supervision. It detects, when an announced balise group is not found within its expectation window.

In more detail, "calculateTrainPosition" is divided into a data flow of different stages, which are being performed sequentially:

1. ***calculateBGLocations***: Calculate the balise group locations

   The stage is triggered each time the train passes a balise group (input *passedBG*). It takes the balise group header with the BG identification, the linking information (Subset 26, packet 5) and the current odometry values as inputs and calculates the location of the passed balise group. If the passed BG has been announced via linking information previously, it takes into account the linking as well as the odometry information. If the passed BG does not meet the expectation window announced by linking, an error flag is set. If the passed BG is an unlinked BG, its location is determined by odometry only, but related to the next previously passed linked BG (LRBG), if there is one.

   Then, if the passed BG is a linked BG comprising linking information for BGs ahead, the linking information is evaluated by creating the announced BGs and computing their locations from the linking distances.

   The passed and the announced BGs are stored in a list *BGs* in the order they are passed and by their announced nominal location on the track.

   Afterwards the locations of all BGs are further improved by re-adjusting their locations with reference to the just passed BG. This optimizes the BG location inaccuries around the current train position (= location of the passed BG).

2. ***delDispensableBGs***: Delete dispensable balise groups

   The function removes balise groups supposed not to be needed any longer from the list of *BGs*. If the number of stored passed linked BGs exceeds the maximum number as specified in [1, Chapter 3.6.2.2.2 c], all BGs astern are deleted. If only (passed) unlinked BGs are in the list and exceed the number of *cNoOfAtLeast_x_unlinkedBGs*, all passed BGs astern to those are removed from the list.

3. ***addAnnouncedBGs***:

   This function is executed once each time balise groups ahead are announced by the RBC. The locations of the announced *BGs* are calculated with reference to the LRBG reported by the RBC.

4. ***calculateTrainPositionInfo***: Calculate train position information.

   This stage takes the list of stored BGs and the current odometry values as inputs and steadily provides the current train position. Additionally, it watches the list of announced *BGs* and provides the "Linking information is used" information as specified in chapt. 3.4.4.2.1.1 of the SRS.

5. ***calculateTrainpositionAttributes***: Calculate train position attribute information.

   This stage provides several additional position related attributes that might conveniently be used by subsequent consumers in the architecture. It in addition provides the current LRBG and the previous LRBG from the list *BGs*.

6. ***linkedBG_missed***:

   This function observes the list of *BGs* and the current train position. If an announced balise group is not found within its expectation window, an error flag will be raised.

6.4.1.3 Reference to the SRS (or other requirements)

The component calculateTrainPosition determines the location of linked and unlinked balise groups and the current train position during the train trip as specified mainly in [1, Chapter 3.6].

6.4.1.4 Design Constraints and Choices

The following constraints and prerequisites apply:

1. The input data received from the balises groups or via radio must have been checked and filtered for validity, consistency and the appropriate train orientation before delivering them to calculateTrainPosition.

2. The storage capacity for balise groups is finite. calculateTrainPosition will raise an error flag when a balise group cannot be stored due to capacity limitations.

3. calculateTrainPosition will raise an error flag if a just passed balise group is not found where announced by linking information or if an announced balise group is missed when the end of its expectation window is reached. It does not yet implement all conditions of linking consistency.

4. calculateTrainPosition is not yet prepared for train movement direction changes.

5. calculateTrainPosition does not yet consider repositioning information.

6.4.2 Provide Position Report

6.4.2.1 Short Description of Functionality

This function takes the current train position and generates a position report which is sent to the RBC. The point in time when such a report is sent is determined by events, on the one hand, and position report parameters—which are basically triggers—provided by the RBC or a balise group passed, on the other hand. The functionality is modeled using four operators, as shown in Figure 16, which are explained below.

CalculateSafeTrainLength Calculates the the safeTrainLength and the MinSafeRearEnd according to [1, Chapter 3.6.5.2.4/5].
`safeTrainLength = absolute(EstimatedFrontEndPosition - MinSafeRearEnd)`,
where `MinSafeRearEnd = minSafeFrontEndPosition - L_TRAIN`.

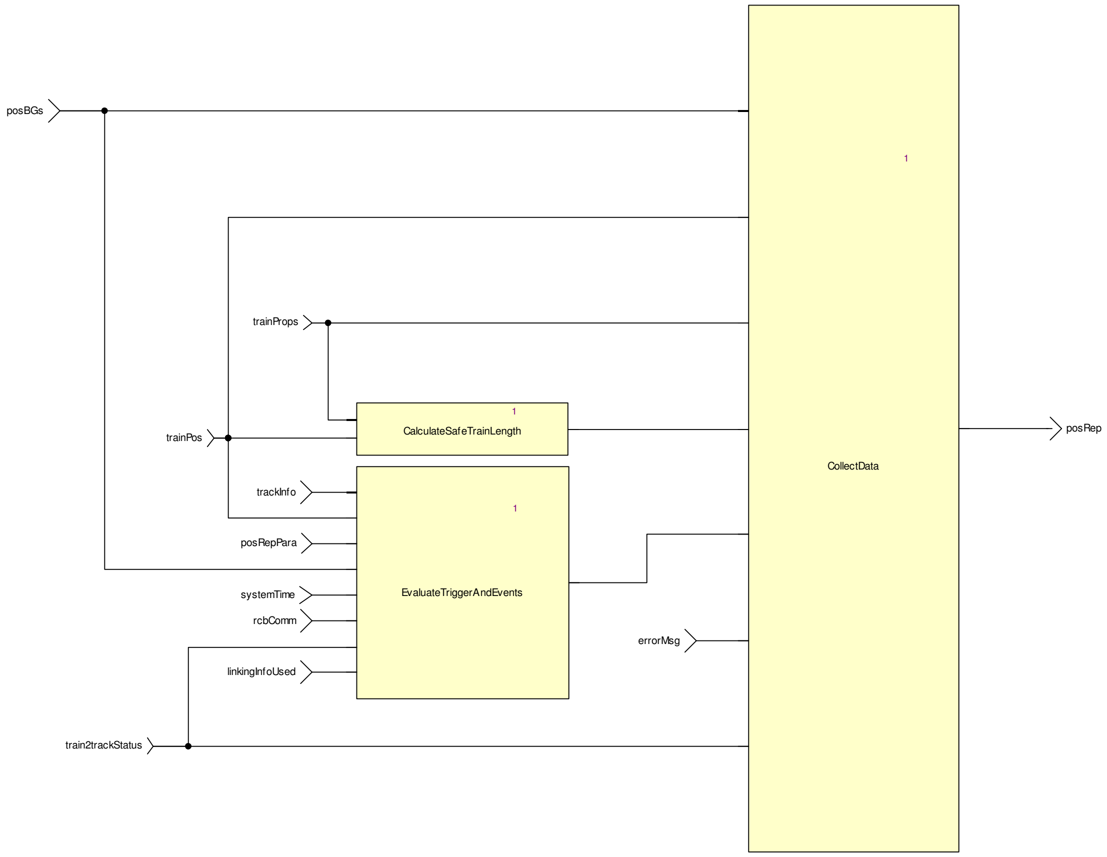


Figure 16. Structure of component ProvidePositionReport

EvaluateTriggerAndEvents Returns a Boolean modelling whether the sending of the next position report is triggered or not. This value is the conjunction of the evaluation of all triggers (PositionReportParameters, i.e., Packet 58) and events (see [1, Chapter 3.6.5.1.4]).

ErrorManager Takes a boolean flag for each possible error that has been occurred and outputs the respective error using type `M_ERROR`

CollectData This operation aggregates data of Packet 0, ..., Packet 5 and the header to a position report.

6.4.2.2 Reference to the SRS (or other requirements)

Most of the functionality is described in [1, Chapter 3.6.5].

6.4.2.3 Design Constraints and Choices

1. The message length (i.e., attribute `L_MESSAGE`) is by default set to 0; the actual value will be set by the Bitwalker/API.
2. The attribute `Q_SCALE` is assumed to be constant; that is, all operations using this attribute do not convert between different values of that attribute.
3. *PositionReportHeader*: The time stamp (i.e., attribute `T_TRAIN`) is not set; this should be done once the message is being sent by the API.

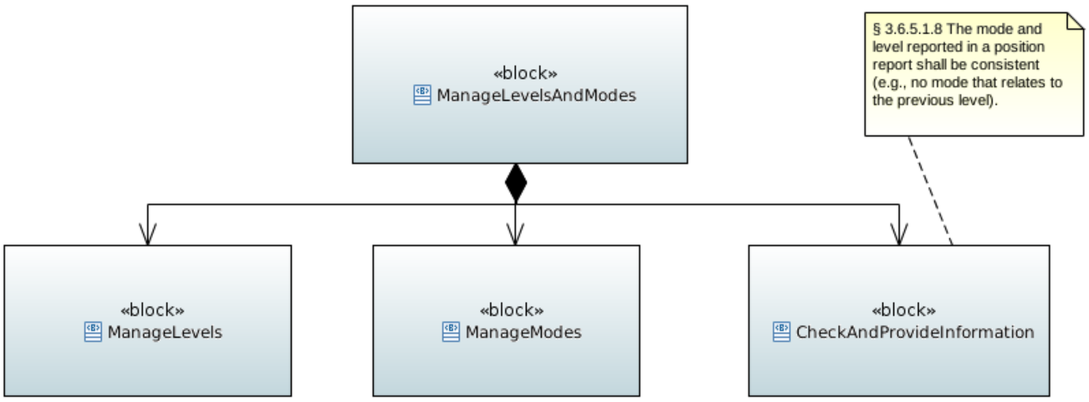


Figure 17. High level Architecture

4. *Packet 4*: When aggregating data for this packet, an error message might be overwritten by a succeeding error message. Because the specification allows only to sent one error in one position report, errors are not being stored in a queue, for instance.

5. *Packet 44*: This packet is currently not contained in a position report as it is not part of the kernel functions.

6. The usage of attributes `D_CYCLOC` and `T_CYCLOC` as part of the triggers specified by the position report parameters (i.e., Packet 58 sent by the RBC) may lead to unexpected results if a big clock cycle together with small values for the attributes is used. The cause is that at every clock cycle the current model increments the reference value for the distance and time by at most `D_CYCLOC` and `T_CYCLOC`, respectively and not a factor of it.

7. The *ErrorHandler* is currently restricted to deal with a single error. As a consequence, as for each error reported a report has to be sent to the RBC, the number of reports is limited to one.

6.4.2.4 Open Issues

1. The specification requires to store the last eight balise groups for which a position report has been sent (see [1, Chapter 3.6.2.2.2.c]).

2. For all reports that contain Packet 1 (i.e., report based on two balise groups), the RBC sends a coordinate system. It is unclear where this has to be stored (i.e., somehow the balise groups have to be stored in a database which has then to be updated), see [1, Chapter 3.4.2.3.3.6]. Moreover, such a coordination system can be invalid and then has to be rejected (see [1, Chapter 3.4.2.3.3.7-8]). On a more abstract level, we need to think about the interface between the RBC and the OBU or a proper abstraction thereof.

6.5 Mode and Level

The "Management of Modes and Levels" function is mainly described in chapter 4 and 5 of [1]. Modes and levels define the status of the ETCS regarding on-board functional status and track infrastructure.

6.5.1 Function Level Management

6.5.1.1 Reference to the SRS or other Requirements

See [1] section 5.10

6.5.1.2 Short description of the functionality

The level management subsystem receives level transition order tables and selects the order with the highest probability. It stores the information about the selected transition order and transits to the requested level once the train passes the location of the level transition.

If required, the driver is asked to acknowledge the transition, in case of no acknowledge or if conditions for the level transition are not fulfilled, the train gets tripped.

6.5.1.3 Interface

The interface consists of the following inputs:

- *conditional transitions:* a priority table containing the conditional level transition orders (from paquet 46)
- *level transition priority table:* a priority table containing the (non-conditional) level transition orders (from paquet 41)
- *train standstill:* a Boolean value indicating whether the train is at standstill (from odometry)
- *driver level transition:* a level transition order selected by the driver (from DMI)
- *ERTMS capabilities:* the ERTMS capabilities of the track
- *getAck:* Boolean input that signals the acknowledgment of the driver (from DMI)
- *resetIdle:* Boolean input to reset without acknowledge
- *currentDistance:* the current position of the train given with the same reference as the position of the level transition order (train position , from localisation)
- *ackDistance:* the maximal distance for driver acknowledge after the level transition (from paquet 41)
- *immediateAck:* a Boolean that signals that an immediate acknowledge is required
- *received L2 L3 MA:* a Boolean that indicates that a level 2 or level 3 movement authority for the track behind the level transition has been received (from paquet 15)
- *received L1 MA:* a Boolean that indicates that a level 1 movement authority for the track behind the level transition has been received (from paquet 12)
- *received target speed:* a Boolean indicating that a target speed for the track behind the level transition has been received (from paquet 27) ?

and the following outputs:

- *next level:* the next level after this computation cycle

- *Trip train:* a Boolean indicating whether the train should be tripped
- *previous level:* the previous level before this computation cycle
- *needsAckFromDriver:* a Boolean that indicates whether an acknowledgment from the driver is necessary

6.5.1.4 Functional Design Description

On the most abstract level the design consists of the *manage_priorities* function which takes the level transition order priority tables as inputs and computes the highest priority transition.

This transition order is the fed to the *computeLevelTransitions* operator. This operator consists of three main parts. The *ComputeTransitionConditions* operator that emits the fulfilled conditions to change from a given level to a new level, the *LevelStateMachine* that stores the current level and takes the computed change conditions as input for possible level transitions and finally the *driverAck* operator which contains a state machine that stores the information whether the system is currently waiting for a driver acknowledge and emits the train trip information if necessary.

6.5.1.5 Reference to the Scade Model

The Scade model is available on GitHub: `https://github.com/openETCS/modeling/tree/master/openETCSArchitectureAndDesign/WorkGroups/Group3/SCADE/LevelManagement/`

6.5.2 Function Mode Management

6.5.2.0.1 Reference to the SRS or other Requirements

see [1] sections 4.4, 4.6, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.11, 5.12, 5.13, 5.19

6.5.2.1 Short description of the functionality

This function is in charge of the computation of new mode to apply according to conditions from inputs (track information, driver interactions, train data,...) and other functions.

6.5.2.2 Interface

The inputs are the following:

- *Cab* identification of the current cabin (A or B)
- *Continue_shunting_Function_Active*: boolean to describe the activation state of the shunting function
- *Current_Level*: outputs of the Level management function
- *Data_From_DMI*: set of data received from the driver via the DMI interface, indeed:
  - *Ack_LS : bool* Driver acknoledges LS mode
  - *Ack_OS : bool*
  - *Ack_RV : bool*

  - *Ack_SH : bool*
  - *Ack_SN : bool*
  - *Ack_SR : bool*
  - *Ack_TR : bool*
  - *Ack_UN : bool*
  - *Req_Exit_SH : bool* driver selects exit of shunting
  - *Req_NL : bool* Driver requests NL mode
  - *Req_Override : bool* Driver requests override function
  - *Req_SH : bool* driver requests SH mode
  - *Req_Start : bool* Driver requests start of mission
  - *ETCS_Isolated: bool*: isolation status of the ETCS

- *Data_From_Localisstion*: set of data received from the function in charge of localistion of the train, indeed:

  - *BG_In_List_Expected_BG_In_SR : bool*: the identity of the overpass balise group is in the list of expected balises related to SR mode (from SR to trip mode condition 36)
  - *BG_In_List_Expected_BG_In_SH : bool*: the identity of the overpass balise group is in the list of expected balises related to SH mode (from SH to trip mode condition 52)
  - *Linked_BG_In_Wrong_Direction : bool* balise group contained in the linking information is passed in the unexpected direction (from FS, LS, OS to trip mode condition 66) *Localisaion function ?*
  - *Train_Position*: output provided by function in charge of computation of train possition (type TrainPosition_Types_Pck::trainPosition_T)
  - *Train_Speed : Obu_BasicTypes_Pkg::Speed_T* provided by odometry function
  - *Train_Standstill : bool* provided by odometry function

- *Data_From_Speed_and_Supervision*: set of data received from the function in charge of speed and supervision management, indeed:

  - *Estim_front_End_overpass_SR_Dist : bool*: the train overpass the SR distance with its estimated front end (from SR to trip mode condition 42)
  - *Estim_Front_End_Rear_SSP : bool*: estimated front end is rear of the start location of either SSP or gradient profile stored on-board (from FS, LS, OS to trip mode condition 69)
  - *Override_Function_Active*: boolean to indicate the state of the activation function
  - *EOA_Antenna_Overpass : bool*: the train overpasses the EOA with min safe antenna position Level 1 (from FS, LS, OS to trip mode condition 12)
  - *EOA_Front_End : bool* the train overpasses the EOA with min safe front end, Level 2 or 3 (from FS, LS, OS to trip mode condition 16)
  - *Train_Speed_Under_Overide_Limit : bool* supervision when override function is active (to SR mode condition 37)

- *Data_From_TIU : TIU_Types_Pkg::Message_Train_Interface_to_EVC_T*:message provided by TIU interface

- *Data_From_Track*: set of data received from track side (via RBC or Balises telegram), indeed:

  - *MA_SSP_Gradiant_Available : bool* MA, SSP and gradient have been received, checked and stored on-board from paquet 12, 15, 21 and 27 or message 3 or 33
  - *Mode_Profile_On_Board : Level_And_Mode_Types_Pkg::T_Mode_Profile* from packet 80
  - *Shunting_granted_By_RBC : bool* from message 27 and 28
  - *Trip_Order_Given_By_Balise : bool*
  - *List_Bg_Related_To_SR_Empty : bool* from packet 63
  - *Stop_If_In_shunting : bool* from packet 135
  - *Stop_If_In_SR : bool* from packet 137
  - *Error_BG_System_Version : bool*
  - *Linking_Reaction_To_Trip : bool*
  - *RBC_Ack_TR_EB_Revocked : bool* from message 6
  - *RBC_Authorized_SR : bool* from message 2
  - *Reversing_Data : Level_And_Mode_Types_Pkg::T_Reversing_Data* from packet 138/ 139
  - *T_NVCONTACT_Overpass : bool* Maximal time without new safe message overpass
  - *Emergency_Stop_Message_Received*: boolean to describe the reception of Emergency Stop message from message 15 or 16

- *Failure_Occured*: boolean to indicate safety failure occurence
- *Interface_To_National_System*: boolean to indicate existance of an interface to a national system
- *National_Trip_Order*: boolean to indicate reception of a trip order from a national system
- *OnBoard_Powered*: boolean to indicate the poxering state of the system
- *Stop_Shunting_Stored*: boolean to store the information in regards of shunting function
- *Valid_Train_Data_Stored*: boolean to indication train data are available and valid.

The outputs are the following:

- *currentMode* the new computed mode (typeis Level_And_Mode_Types_Pkg::T_Mode, default value is Level_And_Mode_Types_Pkg::SB)
- *EB_Requested* boolean to request triggering of emergency brake
- *Service_Brake_Command* boolean to request command of service brake
- *Data_To_DMI*: set of data provided to the DMI Level_And_Mode_Types_Pkg::T_Data_To_DMI :
  - *Ack_LS : bool* Driver acknoledges LS mode
  - *Ack_OS : bool*
  - *Ack_RV : bool*
  - *Ack_SH : bool*
  - *Ack_SN : bool*

- *Ack_SR : bool*
- *Ack_TR : bool*
- *Ack_UN : bool*
- *Req_Exit_SH : bool* driver selects exit of shunting
- *Req_NL : bool* Driver requests NL mode
- *Req_Override : bool* Driver requests override function
- *Req_SH : bool* driver requests SH mode
- *Req_Start : bool* Driver requests start of mission
- *ETCS_Isolated: bool*: isolation status of the ETCS

- *Data_To_BG_Management*: set of date to trackside Level_And_Mode_Types_Pkg::T_Data_To_BG_Management :
  - *EoM_Procedure_req : bool* request of end of mission procedure indeed end of the communication session for message 150
  - *Clean_BG_List_SH_Area : bool* request to clean the BG list when entering an SH area §5.6.2
  - *MA_Req : bool* for message 132
  - *Req_for_SH_from_driver : bool* for message 130

6.5.2.3 Functional Design Description

Three subfunctions are defined:

Inputs proceeds to inputs check and preparation.

ComputeModesCondition performs all specific procedure linked to mode management and defined in [1] sections 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.11, 5.12, 5.13, 5.19 and specifies the conditions to define a mode transition according condition table of section 4.6.3 of [1]

SwitchModes performs the mode selection according the conditions and priorities defined in transition table section 4.6.2 of [1]

Outputs prepares packet of outputs.

6.5.2.4 Reference to the Scade Model

The Scade model is available on github: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/ManageLevelsAndModes/Modes`

6.5.3 Function Check and Provide Level and Mode

6.5.3.1 Reference to the SRS or other Requirements

see [1] section 3.6.5

6.5.3.2 Short description of the functionality

checks compatibility between mode and level and provides outputs

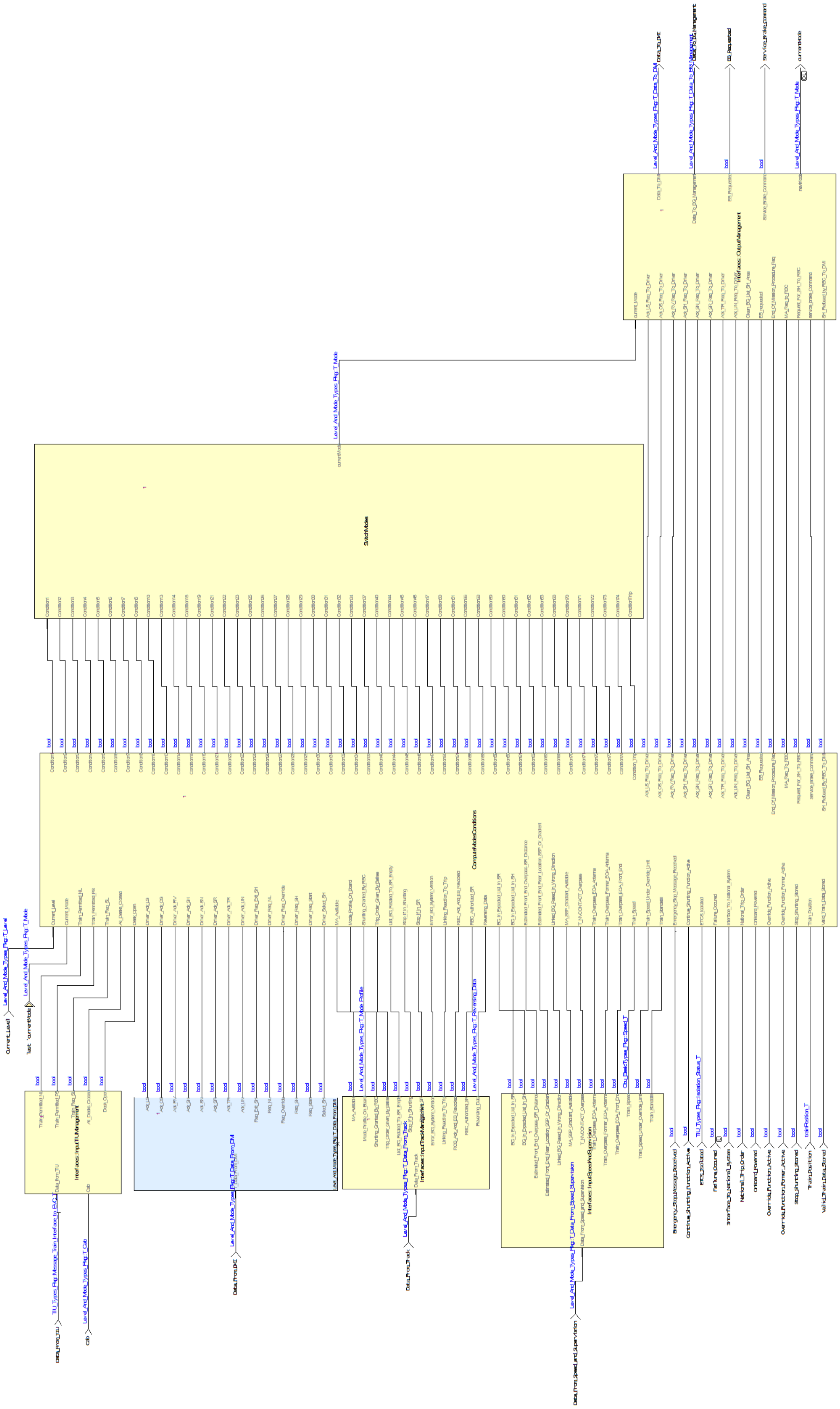


Figure 18. Modes subfunction architecture.

6.5.3.3 Interface

To design

6.5.3.4 Functional Design Description

To design

6.5.3.5 Reference to the Scade Model

To design

6.6 Manage Radio Communication

6.6.1 Management of Radio Communication (*MoRC*)

6.6.1.1 Reference to the SRS

The management of radio communication is specified in Subset-026, chap. 3.5.

6.6.1.2 Short description of the functionality

The management of radio communication *MoRC* implements the on board management part of a single communication session with the track, i.e. a single RBC. It controls the establishing, maintaining and termination process of a radio communication session and steers the underlying communication safety layer and the mobile device. Those and the data transfer itself are not part of the function.

6.6.1.3 Interface

6.6.1.3.1 Inputs

The MoRC function takes as inputs datagrams received from track, OBU internal phases and status information and configuration data:

- Datagrams received from track (*inMessage*):
  - Packet 42 (session management) received from balise group or RBC
  - Packet 45 (radio network registration) received from balise group or RBC
  - Message 32 (RBC/RIU System Version) received from RBC: *MoRC* only needs to know if the system version received from track side is supported by the OBU.
  - Message 38 (initiation of a communication session) received from RBC
  - Message 39 (acknowledgement of termination of a communication session)
- *obuEventsAndPhases*: information about OBU internal events and OBU internal phases:
  - *atPowerDown*
  - *atPowerUp*
  - *atStartOfMission*
  - *startOfMissionProcedureIsGoingOn*

  - *startOfMissionProcedureCompleted*
  - *trainIsRejectedByRBC_duringStartOfMission*
  - *endOfMissionIsExecuted*
  - *driverClosesTheDeskduringStartOfMission*
  - *driverHasManuallyChangedLevel*
  - *afterDriverEntryOfANewRadioNetworkID*
  - *triggerDecisionThatNoRadioNetworkIDAvailable*
  - *isPartOfAnOngoingStartOfMissionProcedure*
  - *trainPassesALevelTransitionBorder*
  - *trainPassesA_RBC_RBC_border_WithItsFrontEnd*
  - *trainExitedFromAnRBCArea*
  - *modeChangeHasToBeReportedToRBC*
  - *trainFrontInsideInAnAnnouncedRadioHole*
  - *trainFrontReachesEndOfAnnouncedRadioHole*
  - *OBU_hasToEstablishANewSession*
  - *isInCommunicationSessionWithAnRIU*
  - *errorConditionRequiringTerminationDetected*

- Current OBU internal states:
  - *currentTime*: current OBU system time
  - *t_train*: current trainborne clock (T_TRAIN) as specified in Subset-026, chap. 7
  - *mode*: current OBU mode
  - *level*: current OBU level
- *statusOfMobile*: status of the associated mobile device
- configuration parameters:
  - *onboardPhoneNumbers* (NID_RADIO)
  - *radioNetworkIDs*: Identities of radio networks (NID_MN): default, memorized or from driver
  - *nid_engine*: Onboard ETCS identity (NID_ENGINE)
  - *connectionStatusTimerInterval*: Connection status timer period

6.6.1.3.2 Outputs

MoRC generates a couple of outputs:

- *MessageToRBC*: messages to be sent to the RBC:
  - Message 155 (initiation of a communication session)
  - Message 156 (termination of a communication session)
  - Message 159 (session established)

  - Message 154 (no compatible version supported)

- Action triggers:

  - *sendAPositionReport*: triggers a position report to be sent to the RBC
  - *memorizeTheLastRadioNetworkID*: triggers to store the last radio network ID for later use
  - *orderTheRegistrationOfItsConnectedMobiles*
  - *rejectOrderToContactRBC_or_RIU*
  - *InformTheDriverThatNoConnectionWasSetup*
  - *requestTheSetupOfASafeRadioConnection*: initiate the setup of a safe radio connection
  - *requestReleaseOfSafeRadioConnectionWithTrackside*: initiate the release of a safe radio connection
  - *ignoreMessagesFromRBC_except_m39_AckOfTerminationOfCommunicationSession*
  - *sessionSuccessfullyEstablished*

- *cmdsToMobile*: control commands to the mobile device

- Status information:

  - *sessionStatus*: current session status
  - *mobileSWStatus*: connection status
  - *currentRadioNetworkID*: current radio network ID

6.6.1.4 Functional Design Description

The kernel function of the *MoRC* component is *managementOfRadioCommunication* (figure ???). The implementation is kept close to the prose of Subset-026, chap. 3.5. Since chap. 3.5 rarely refers to terms, variable types, packets and messages of the ETCS language as specified in Subset-026, chap. 7 and 8, *managementOfRadioCommunication* does neither.

To be capable of being integrated with other OBU software components, *MoRC* had to be wrapped with a transformer between the ETCS and the "chap. 3.5" language. This is the purpose of the main function of *MoRC*, *MoRC_Main*.

The function *managementOfRadioCommunication* implements the session states establishing, maintaining and termination as described in Subset-026, chap. 3.5. A SCADE state machine reflects this state model (figure ???) accurately. Within each of the states, the activities needed as long as the state is active, are performed. When there is no communication session (state *NoSession*) currently, the state machine waits for events that initiate a session (subfunction *initiate_a_Session*). When the appropriate conditions are fulfilled, the state machine moves to the *Establishing* state. Here in, it runs through the sequence required fore establishing a session (subfunction *establish_a_Session*. Dependent on the results, the state machine changes over to the *Maintaining* or *Terminating* state. While in *Maintaining*, the communication connection is monitored. When an event triggering the session termination occurs, the state machine switches to the state *Terminating* with the subfunction *terminating_a_CommunicationSession* and performs the session termination sequence.

In parallel to the main state machine, *managementOfRadioCommunication* monitors all the time whether the session has to be terminated (subfunction *initiateTerminatingASession*) or if the

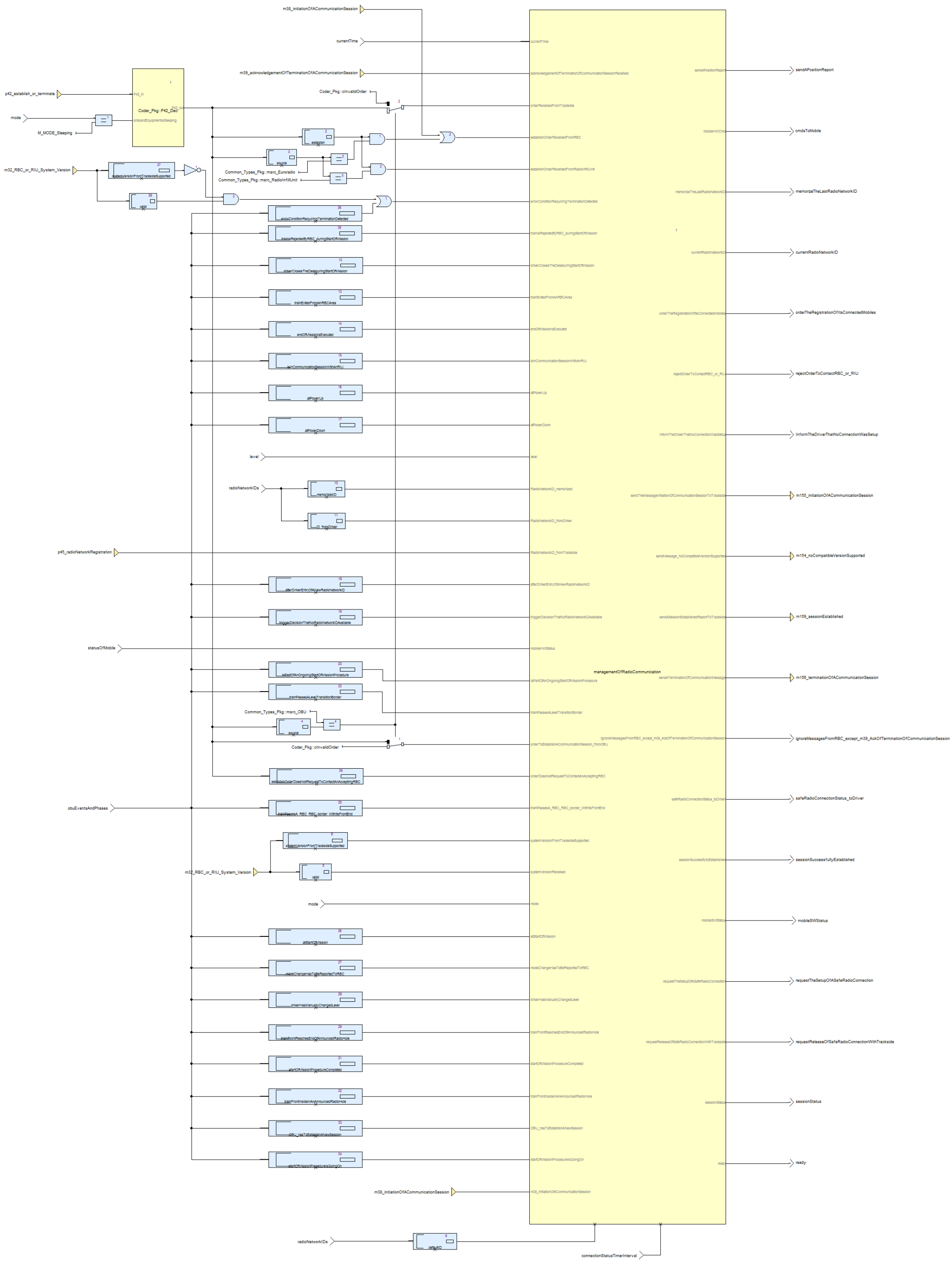


Figure 19. Main function of *MoRC*.

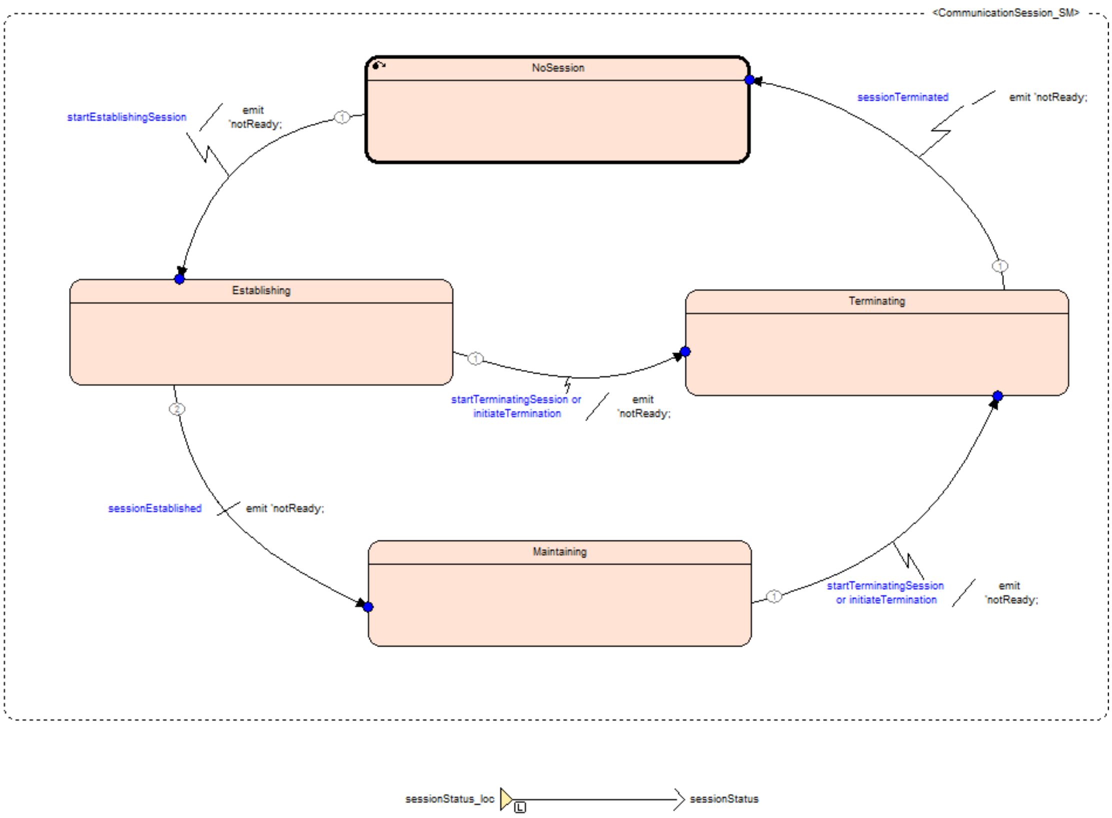


Figure 20. Implementation of session states.

session has the be terminated and subsequently established (subfunction *terminateAndEstablishSession*). *registeringToTheRadioNetwork* is responsible for connection to the radio network. *safeRadioConnectionIndication* controls the radio connection indication for the driver.

6.6.1.5 Reference to the Scade Model

The MoRC SCADE model resides at `https://github.com/openETCS/modeling/tree/master/model/Scade/System/ObuFunctions/Radio/MoRC`.

7 F3: Measure Train Movement

8 F4: Manage Radio Communication

9 F5: Manage JRU

10 F6: DMI Controller

10.1 DMI Controller

10.1.0.6 Reference to the SRS or other Requirements (or other requirements)

ERA_ERTMS_015560

10.1.0.7 Short description of the functionality

The DMI controller interact with the DMI display and is responsible for alls procedures between the DMI display and Driver. Furthermore, the DMI controller will interact with the DMI Management to compute the received information (e.g. driver number request, ...) and send, if necessary, data or reports to the DMI Management (acknowledge, text messages...). The DMI Controller is a passive module, this means that all the processing are performed EVC-side, therefore the DMI Controller simply responds to the requests of the EVC or Driver and performs some checks according with the information received from EVC.

10.1.0.8 Interface

The DMI Controller has two interfaces. One between DMI Controller and DMI Display and one between DMI Controller and DMI Management. The structure of the interface between DMI Controller and DMI Display is driven by the logic of SCADE Display therefore It doesn't follow any standard or constraints (It will not be described in this chapter). DMI Controller and DMI Management exchange packets. Each packet is a structured type with a valid flag (a boolean variable), the DMI controller takes into account the data inside the packet only when the valid flag is true.

The interface between DMI Controller and DMI Management consist of three parts according with the direction of the information:

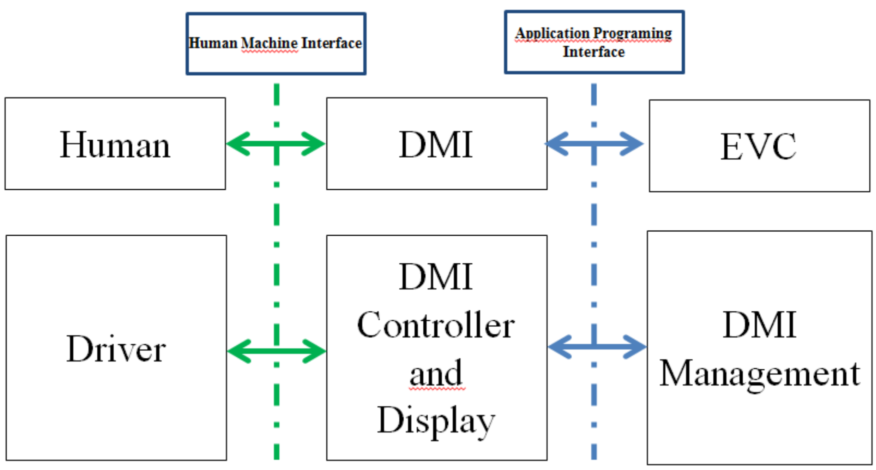


Figure 21. DMI Interfaces

- From DMI Management to DMI Controller
- From DMI Controller to DMI Management
- Both ways directions (You will find the same type both as input than as output)

10.1.0.8.1 From DMI Management to DMI Controller

In the following table are listed the inputs coming from DMI Management with a brief description:

NAME	DESCRIPTION
DMI_entry_request	Request to input data (e.g. driver id, Train running number etc.)
DMI_identifier_request	Request of the DMI informations
DMI_menu_request	Request to enable or disable buttons
DMI_dynamic	Contains informations about current speed, current mode etc.
DMI_text_message	Contains predefined or plain text messages
DMI_icons	Request to display one or more icons in any area

Please note: TIU_trainStatus input is missing in the above table. This is the only input coming directly from TIU and contains the open/close Desk signal.

10.1.0.8.2 From DMI Controller to DMI Management

In the following table are listed the outputs directed to DMI Management with a brief description:

NAME	DESCRIPTION
DMI_identifier	Information about DMI (e.g. version, cabin identifier etc.)
DMI_driver_request	Driver request or acknowledgement
DMI_train_data_ack	Train data acknowledgement
DMI_status_report	The actual status of DMI (keep alive)
DMI_text_message_ack	Text message acknowledgement
DMI_icons_ack	Icon acknowledgement

10.1.0.8.3 Both ways direction

In the following table are listed the outputs/inputs to/from DMI Management with a brief description:

NAME	DESCRIPTION
DMI_driver_identifier	Contains the default or entered driver identifier
DMI_train_running_number	Contains the default or entered train running number
DMI_train_data	Contains the default or entered train data

10.1.0.9 Functional Design Description

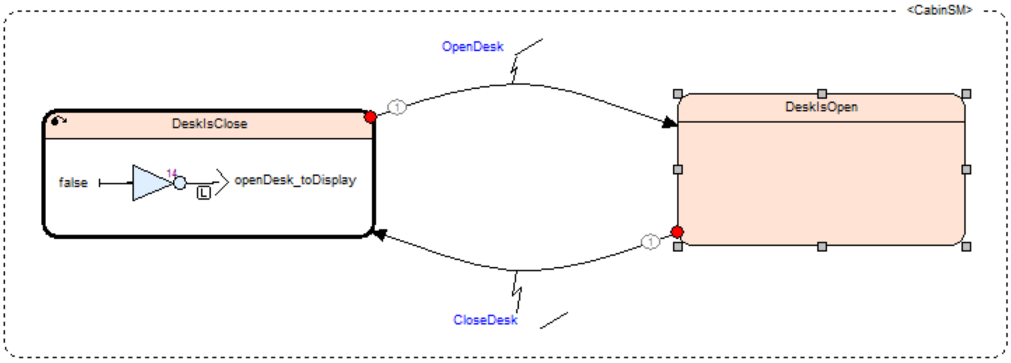


Figure 22. Cabin State Machine.

Please note: *DMI Controller is a project under construction, a lot of features and functionalities are missing, therefore the structure described below is a draft version and will be changing in the future.*

The informations (received and sent) could be divided in two groups: Sporadic and Periodic. The first one are received/sent aperiodically in any time instead the second one are received/sent periodically, with a fixed deadline. Are part of Periodic group the output DM_status_report and the input DMI_dynamic all other are Sporadic. Therefore, the structure of DMI Controller module consists of a first main state machine *CabinSM* (Fig. 22) triggered by a *OpenDesk* signal (from TIU). Inside the *DeskIsOpen* state there are other two state machines :*HandshakeSM* and *DynamicInfoSM* (Fig. 23).

HandshakeSM performs an initial handshake between DMI Controller and DMI Management. Before that, no data has to be sent or received to/from DMI Management. When the transition is fired a DMI_identifier packet is sent to DMI Management with informations about the DMI (e.g. DMI identifier, DMI name etc.). At this point the DMI Controler is ready to manage the sporadic information (e.g. Enter or revalidate DriverID, Enter or revalidate Train running number etc.).

The DynamicInfoSM state machine is triggered after the handshake, exactly when HandshakeSM reaches the DynInfo_Activated state. At the time when the transition is fired a signal is emitted (startDMI_status) and begins a periodic sending of DMI status information (keep alive) to DMI Management. Once reached DynamicInfo_Active, the DMI Controller is ready to receive and manage the dynamic informations.

With the aim to improve the readability and for a better management of complexity, all the functions (modules, state machines etc.) implemented in each state are divided several diagrams.

The *SporadicInfo* consist of:

- **diagram_SporadicInfo_Main**: Contains all the modules to manage the sporadic data like "Enter revalidate Driver ID", "Enter or revalidate train running number", enable buttons in menus. The WindowSM state machine manages the windows that should appear on the DMI(Fig. 24).
- **diagram_SporadicInfo_TrainData**: Contains all the logic to store and adapt the incoming train data to a correct visualization on DMI Display.
- **diagram_SporadicInfo_Icon_Management**: Contains the logic to show/hide one or several icons in area and manage the acknowledgement mechanism if It's required.

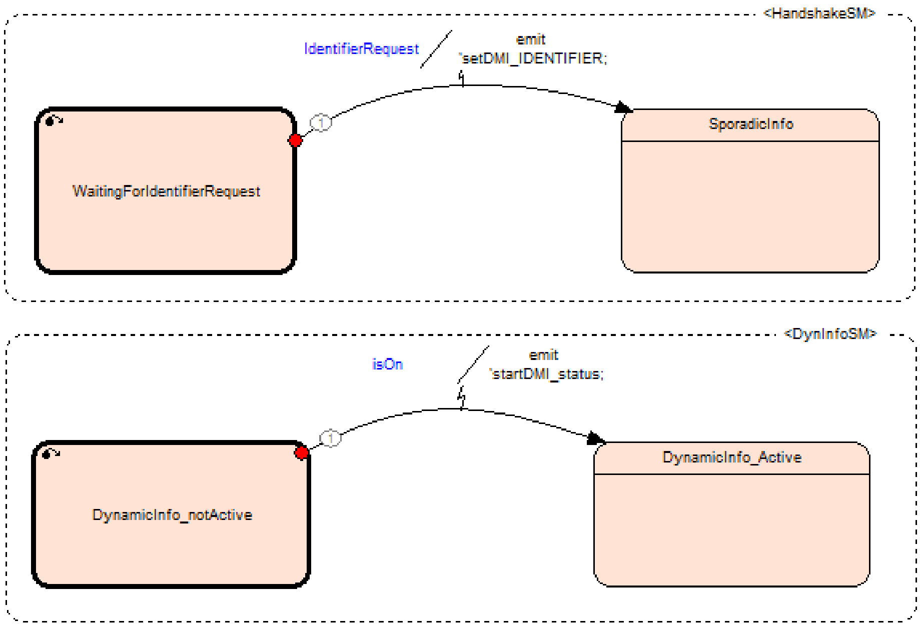


Figure 23. HandshakeSM and DynamicInfoSM State Machines.

- **diagram_SporadicInfo_DriverID_TRN**: Contains the logic to store and sent the Train running number and the Driver ID.
- **diagram_SporadicInfo_Text_Messages**: Contains the modules, state machines and all the logic to manage and display predefined and customized text messages.

The *DynamicInfo_Active* state consists of:

- **diagram_DynamicInfo_Main**: Contains modules to store and display the informations like the current mode, ETCS level, RBC connection status and location brake target.
- **diagram_SpeedSupervision**: Contains the module where are implemented the behaviour of the speed pointer and the circular speed gauge (informations about speed target, speed permitted and speed release).

10.1.0.10 Communication Protocol

This section explains which messages are exchanged among DMI Controller, DMI Management and Start of mission procedure. As mentioned previously the DMI Controller is a passive component, It simply responds to requests, therefore is able to cover different scenarios. Below are some examples.

10.1.0.10.1 Start Of Mission scenario

Are detailed, through a sequence diagram, all the activities (exchanged messages) that should be done to start. In this scenario we have three actors: DMI Controller, DMI Management and SoM procedure (the module where is implemented the start of mission procedure). It's assumed that a OpenDesk signal is received and the system starts in Stand By mode (Fig. 25).

10.1.0.10.2 Cyclic Exchange of messages

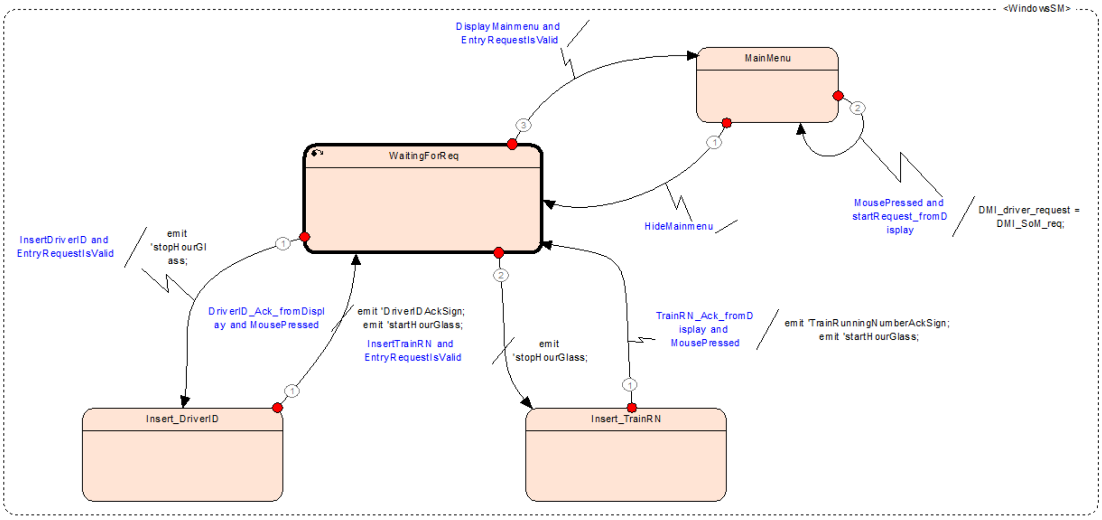


Figure 24. Windows state machine.

1240 The time between two messages has not yet been definitively established, It might change in the future. The DMI status packet implements a keep alive mechanism, this means, if the EVC does not receive any DMI status signal during the lapse time, It shall consider a failure in DMI. This check is not yet implemented.

10.1.0.11 Reference to the Scade Model

1245 The SCADE model can be found on github under the following path: `https://github.com/openETCS/modeling/tree/master/model/Scade/System/DMI_Control`

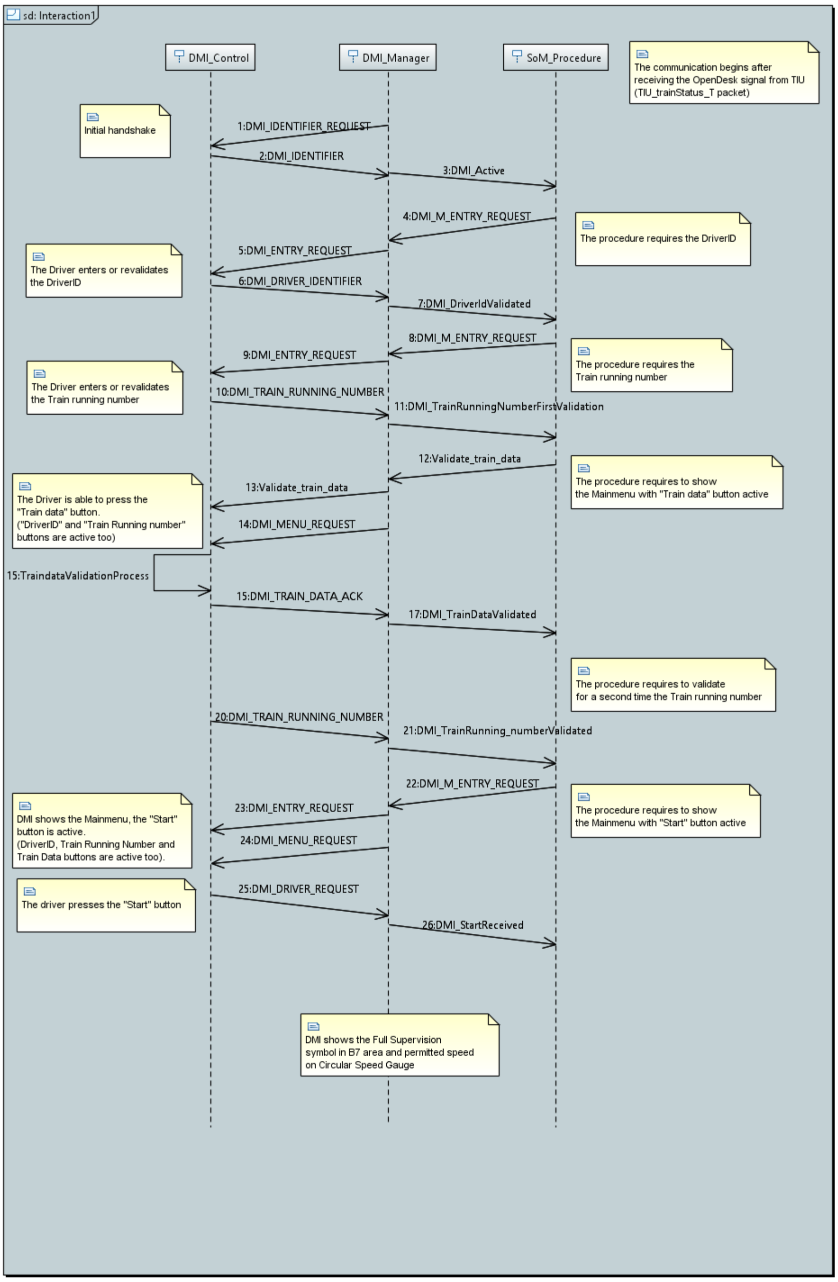


Figure 25. Sequence Diagram of start of mission scenario

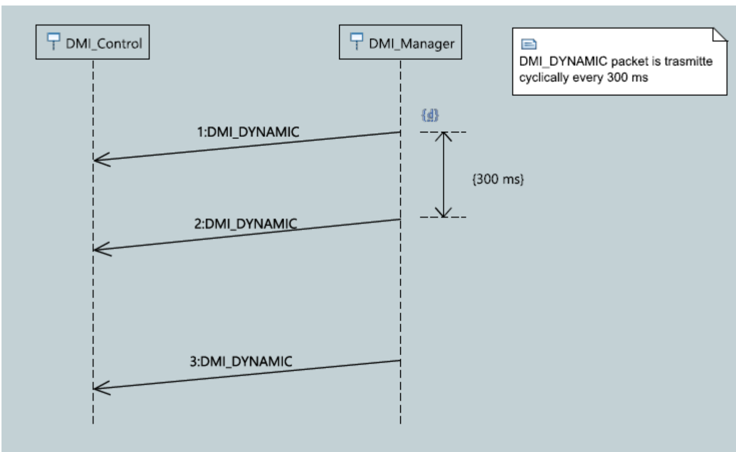


Figure 26. Sequence diagram of Dynamic data.

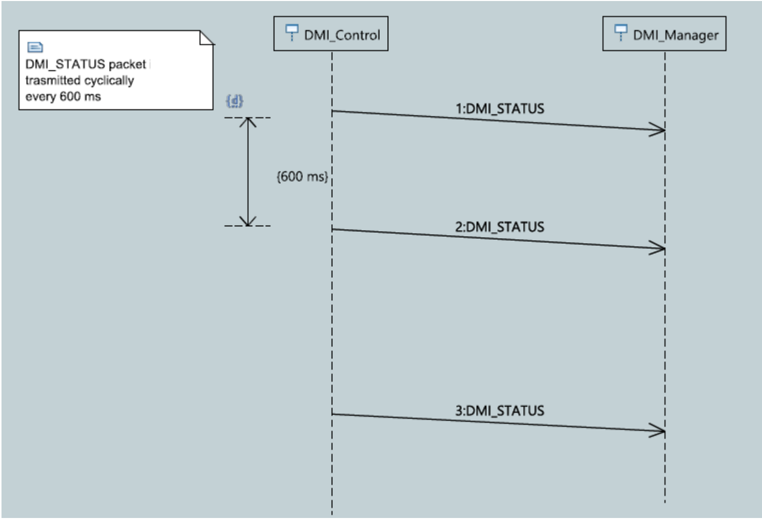


Figure 27. Sequence Diagram of DMI status.

References

[1] ERA. *System Requirements Specification, SUBSET-026*, v3.3.0 edition, March 2012.

[2] ERA. *FFFIS for Eurobalise, SUBSET-036*, v3.0.0 edition, February 2012.

[3] ERA. *Performance Requirements for Interoperability, SUBSET-041*, v3.1.0 edition, March 2012.