OETCS TK-01-01

openETCS

ITEA2
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

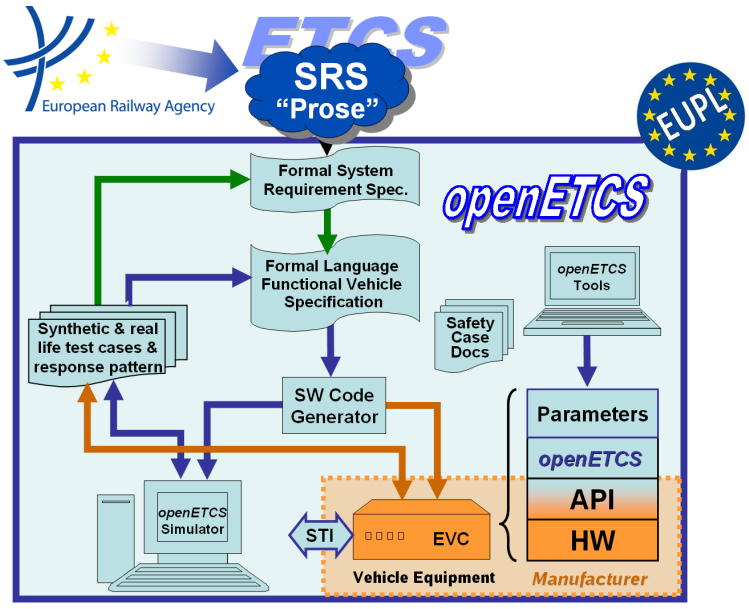
ITEA2 Project
Call 6 11025
2012 – 2015

Work-Package 3: “Modeling”

Delivery of second iteration: D 3.5.2 System Specification Model and D 3.6.2 Functional Model

Baseliyos Jacob, 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

April 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 TK-01-01
April 2015

Delivery of second iteration: D 3.5.2 System Specification Model and D 3.6.2 Functional Model

Document approbation

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

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

DB Netz AG
Völckerstrasse 5
D-80959 München Freimann, Germany

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 Functional Specification

Prepared for openETCS@ITEA2 Project

Abstract: This document gives an introduction to the architecture of openETCS. 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 the structure	Baseliyos Jacob	
0.2	Document	Workshop Results included and some pretty-printing	Bernd Hekele	
0.3	Document	Update of Design work, introduction and architecture	Baseliyos Jacob and WP 3 Team	

Table of Contents

Figures and Tables

Figures

Tables

1 Introduction

1.1 Motivation

The openETCS work package 3 (WP3) aims to provide – amongst others - the software architecture for the openETCS kernel in order to eventually build the software itself.

The WP3 partners had by the end of 2014 put great effort in the openETCS software design, thus far without making definite choices on the software architecture itself respective of functional breakdown and data structures of the openETCS kernel. Since the project planning foresees in the production of a reference software to be used as a demonstrator by June 2015, it was of paramount importance that a first design delivarable of the openETCS kernel architecture be finalized shortly but no later than November 2014.

This document shows a snapshot of the software and will also give an outlook to the scope of the next iteration and the final objectives.

1.2 Objectives

The prime objective of WP3 is to produce a fully formal prototype for the openETCS reference system that can function as a demonstrator in collaboration with WP 4 and WP 5 for the openETCS approach and will be used as such in the final phase of the project. That phase is the first half of 2015. This objective is defined as . . .

High level Objectives of this work: «any further general statements on the ITEA2 objectives, like. . . »

- Work on a model bases approach and process for effective collaborative work within an international ETCS developer team as stated above, the project needs a definite architecture design by the end of 2014.

  This document targets:

- Defining the general design and conditions of the openETCS architecture, functional breakdown, data structures, behaviour and interfaces.

- Providing the guidelines for discussion during the workshops that are planned in October and November 2014 that will result in the final and decisive version of this document

- Being the ‘platform’ for finalization i.e. whatever be the products or results of the workshops shall be integrated in this document.

  Apart from these general objectives, the document means to provide for the materials that will enable WP3 partners to improve the efficiency of the Work Package activities:

- The comprehensive architecture design shall enable splitting the work load according to the building blocks defined by the architecture and allocate strictly compartmented work parcels or activities to WP3 partners.
- Doing so will enable WP3 to avoid any double work
- Compartmenting the work load according to the functional building blocks as defined by the architecture will enable efficient planning of activities, be it individually or the integrated WP3 planning for the coming period, aiming at a just in time delivery of all results and products
- Compartment the work load according to the functional building blocks as defined by the architecture will enable efficient planning of activities, be it individually or the integrated WP3 planning for the coming period, aiming at a just in time delivery of all results and products

1.3 Scope of deliverables

The prime original objective of WP3 was to produce a rapid prototype for the openETCS reference system that can function as a demonstrator in collaboration with WP 4 and WP 5 for the openETCS approach and will be used as such in the final phase of the project. That phase is the first half of 2015.

We have to see this original objective in the context of the overall innovation promise of openETCS and the related exploitation potential. Taking into account the current situation of the project (as of Nov. 2014) we have to make choices that make sure that the remaining budget and resources are wisely used, which necessitates that the WP3 deliverables become more "multi- purpose":

• provide a functional formal specification of openETCS, based on functional analysis, and traceable to the SRS.

• Iteratively develop the software architecture and design description document (ADD) while using the formal specification as an executable functional model

• Utilise real- world use-case scenarios as early as possible in the development process

If we take this approach, we can use the iterative development method, while applying a pragmatic view of SCRUM (which has been defined as the high- level project management method for this project) in order to build the following deliverables:

• An ADD document, providing a functional description of software architecture and design.

• A RFC (Request For Comment) process that provides a cross- reference to the SRS, listing all requirements and design conflicts that have been identified during our work

• A formal, executable model of the openETCS kernel, which can directly be used as entry point into a (future) EN5012x SIL4 compliant software design process, as well as functional reference during simulation and on the planned WP5 demonstrator.

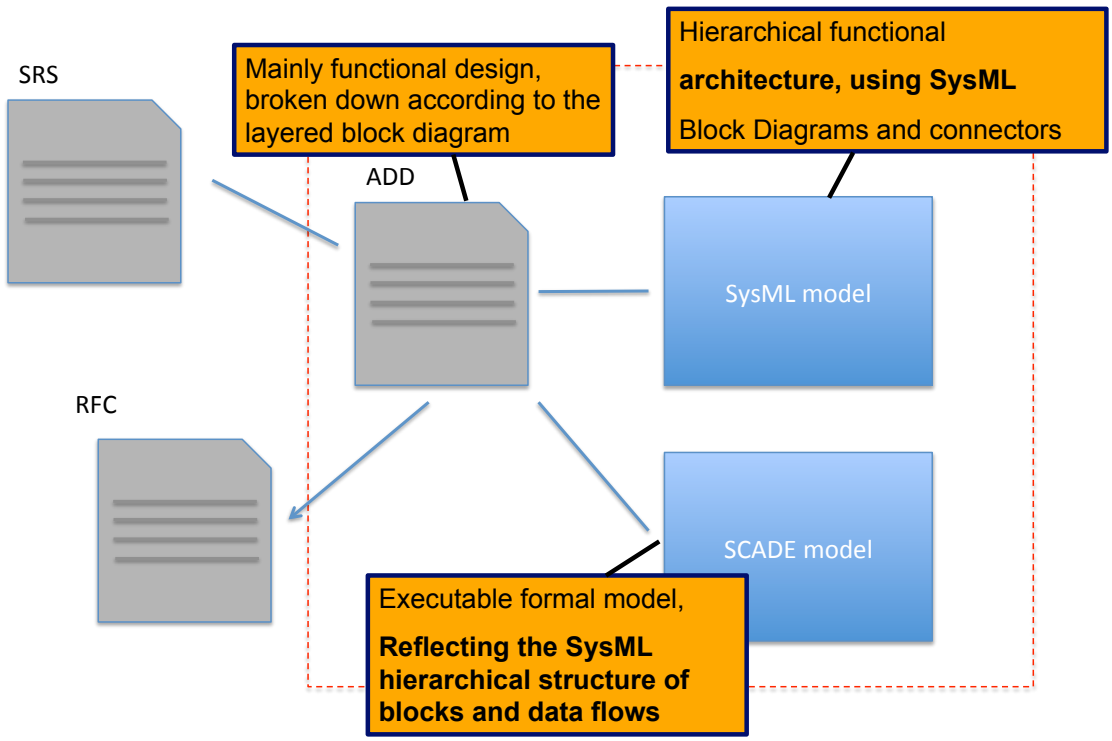


Figure 1. Document relationships in WP3

Each main WP3 collateral has it's distinct role in the overall project structure. For a schematic view of the dependencies, see figure 1.

Scope of the ADD document (this document)

The ADD document is intended to provide the following main information:

• Internal guidance for the project: Provide authoritative guidance on process, responsibilties, process and workflow.

• Architectural design description: Provide complete information on the overall architecture of the openETCS kernel functions.

• Functional design description: Provide comprehensive information on the design of the system, which includes QoS aspects, interfaces, data types, algorithms and finally the full software design.

The nature of the development process implies that this document will be a "living document", meaning that it will be constantly updated, with new iterations being planned as follows:

• Intermediate work iterations (0.x): as required until the ITEA2 intermediate review (March 2015)

• First main release (1.0): as a deliverable for the ITEA2 intermediate review (March 2015)

• Intermediate work iterations (1.x): as required until the demonstrator milestone (June 2015)

• Second main release (2.0): for the demonstrator milestone (June 2015)

• Intermediate work iterations (2.x) for the rest of the project duration (until Oct 2015)

• Final release (3.0) for the final review.

Intermediate work iteration shall be managed using the normal day-to-day pull/ review process, while main releases require a formal document sign-off.

Scope of the User Validation Scenarios

The work of WP3 has now been aligned to the milestones that are relevant for the overall project:

• (1) ITEA2 intermediate review (March 2015)

• (2) Demonstrator milestone (June 2015)

• (3) Final review (End of 2015)

In order to support the functional approach, we use operational scenarios that are described the the User Validation Scenario Document.

These are based on the following data:

• Rules from NS for the operation of trains

• Infrastructure data from the Utrecht- Amsterdam line

In order to support the functional analysis, which is aligned with the project's milestones, this document will define use cases along the following principles:

• (1) Selected operational scenarios on selected sections of the Utrecht- Amsterdam track

Based on nominal scenarios, our simulated train will drive across actual (simulated) infrastructure, selecting only a few, typical balise groups. This will allow WP3 to implement the relevant mode/level/message (balise and RBC) functionality first,

The openETCS kernel model has been complemented by a simple DMI representation, in order to validate actual behaviour.

In march 2015, only basic scenarios will be demonstrated, which have been organised as use cases, based on a plant/ controller co-simulation concept.

The future

• (2) Selected operational scenarios on the full Utrecht- Amsterdam track

Based on nominal scenarios, our simulated train will drive across actual (simulated) infrastructure, covering the entire Utrecht- Amsterdam track. This will allow WP3 to implement the relevant mode/level/message (balise and RBC) functionality first, fo full functional coverage of the test track.

Compared to the fist step in March, we will also cover more operational scenarios according to the NS operational rules.

A generic, interactive simulation concept will be created for this purpose.

The entire track with its 488 balise groups and known (real) RBC messages as well as specific validation scenarios will be modelled.

A presentation of this concept can be found in the section "Dynamic Simulation".

• (3) Full coverage of Utrecht- Amsterdam

For the final review, we will prepare a openETCS kernel software that can run defined scenarios, accordiing to the NS operation regulation, across the entire Utrecht- Amsterdam line, either replaying actual train rides or simulating various events, nominal and non- nominal.

In cooperation with WP4 and WP5 this will provide additional input for demonstration and validation.

Scope of the RFC process

The RFC document is a deliverable that was added to the WP3 set of documents during the Nov 3-7, 2014 Munich workshop.

Most (industrial) ETCS OBU systems have been built based on pure SRS analysis. at least in theory. In practise, each implementation is derived from the SRS, but with a mindset that is partially driven by the company culture and local operational regulation prior to ERTMS introduction.
This has lead to a plethora of ETCS systems, which exhibit subtle differences which lead to incomplete compatibility between OBUs and track equipment of different suppliers. Each OBU works best on a track which was built by the same supplier in the same country.
One of the objectives of openETCS is to provide a reference design for an ETCS UBU that reduces these variations.
The experts of WP3 and WP4 agree that a functional approach will lead to a better understanding of the system. As the SRS remains the formal reference, we have to provide formal feedback which is traceable to the SRS.
This is the objective of the RFC document.

The RFC document shall be structured following the paragraphs in the SRS and shall provide full traceability to the openETCS ADD. Each design decision that can be traced to a deviation from the SRS or that highlights inconsistencies inside the SRS shall be documented in the RFC. The RFC is hence a deliverable that provides formal feedback to the relevant ERTMS stakehold-

ers.

(we still have to discuss the precise interaction between the SRS Analysis effort, WP4 and this RFC document)

The nature of the development process implies that this document will be a "living document", meaning that it will be constantly updated, with new iterations being planned as follows:

• Intermediate work iterations (0.x): as required until the ITEA2 intermediate review (March 2015)

• First main release (1.0): as a deliverable for the ITEA2 intermediate review (March 2015)

• Intermediate work iterations (1.x): as required until the demonstrator milestone (June 2015)

• Second main release (2.0): for the demonstrator milestone (June 2015)

• Intermediate work iterations (2.x) for the rest of the project duration (until Oct 2015)

• Final release (3.0) for the final review.

Intermediate work iteration shall be managed using the normal day-to-day pull/ review process, while main releases require a formal document sign-off.

Objectives and scope related to formal executable model

During the previous iterations of this document, it was proposed to use a traditional waterfall model in order to define the architecture and subsequently the functional and software design of the openETCS kernel software. We have now shifted the design paradigm to a more agile process, where we will use iterations (functional analysis, implementation of prototype formal executable model, refinement) in order to design the software modules. (bottom- up design of modules)
These will then be merged into a top- down architecture by the chief architect, using best practises that are well established for data- flow oriented software designs.

The result will be a functional, formal model, from which code can be generated that implements the functionality fin an hardware- independent way.
This code is hardware- and platform agnostic and can be integrated with the openETCS runtime environment, which is adaptable to different system architectures and APIs.

1.4 Roles, responsibilities and tasks

In this section, the roles and responsibilities of the WP3 partners are confirmed, especially where they divert from what has been agreed upon at the start of WP3:

- **Responsibilities** First of all, in the last WP 3 meeting in Brussels on 10.09.2014 DB proposed to take over the lead of the architecture design and functional breakdown. At the subsequent weekly scrum meeting on 12.09.2014, it was agreed upon by all participants that DB will take over the lead (see Appendix ...);

- **Planning:** Alstom as WP 3 leader will remain to be responsible for the planning and the allocation of the defined tasks to the different partners

- **Roles:** Alstom will also coordinate the work and safeguard that the defined results will be delivered according to the quality requirements that are agreed within the ITEA2 project and the schedule and the milestones that will be agreed upon during the coming workshops;

- All WP3 partners will deliver the results or products according to planning as will be agreed upon during the said workshops.

  In the interest of a swift production of the critical documentation of which this version is a draft, specific tasks will be defined in terms of concrete results to be delivered, the timeframes in which these results must be produced and the partner who shall be responsible for that specific result and the planning. This is to safeguard the timely delivery. The process will be described in the next sections.

1.5 openETCS Design Process

1.5.1 SRS- Driven Approach vs. Functional Approach

Most attempts to formalise the ETCS onboard software have been focusing on creating models that were directly mirroring the SRS specification. While this gives a full picture of the status quo of the SRS, it is not sufficient in order to fully understand the main issues that stem from the approach by which the SRS was conceived.
The SRS is aiming in what could be called the quadrature of the circle:

- Define a common specification of all aspects of the requirements of the ETCS system (Basic System Description, Principles, Functionality, Procedures and Application Level Communication Protocols, amongst others)
- Create a framework that is compatible with all the local ways of driving trains
- Ensure a framework for simple interoperability
- Give a full understanding of the wayside and onboard views of the system

Instead of building yet another copy-paste direct formalisation of the ETCS SRS, we are proposing a different approach:

- Functional analysis of the SRS - onboard point-of view - SRS driven - Focused on the reference track (ETCS L2 Utrecht- Amsterdam)

- Formalisation - Executable formal model (using SCADE tool suite) - Dynamic simulation
- Validation - interface to WP4

Together with the document structure described above, this will for the first time lead to a OBU-centric formalisation of the SRS, with systematic Change Request process (through our RFC approach) and a functional reference that can be used to validate ETCS OBU software as well as wayside implementations once it is completed.

1.5.2 Top- down vs. Bottom- Up

The functional approach, together with other considerations mandated a combined top-down/ bottom up design approach. While the basic architecture was derived from existing architectures, with basic concepts and functional breakdown derived from the existing implementations of the WP3 leader, the actual functional analysis and the related software design was built bottom- up. Thanks to the model- based formal design approach, the components could then be seamlessly merged with the top- down architecture. Using this approach, the system and software architecture was at a relatively immature approach when we split the work in order to separately analyse and design the different functional blocks of the system. The openETCS agile approach allows to iteratively integrate and align the top- down architectural view with the separately developed main functional blocks.

1.5.3 Working groups

While the basic architectural concept has been driven jointly by Alstom, DB and NS, the analysis and design of the functional blocks was distributed into working groups. Each working group consists of a combination of application/ analysis specialists and software/ SCADE specialists.

1.5.4 Use- case driven setting of priorities

In order to focalise the work and to ensure our functional approach, DB has taken the role of product owner. As a reminder: in an agile/ SCRUM project management approach, the product owner has the responsibility to represent the interests of the customer. The goal being to prove the openETCS concept and model on the ETCS L2 Utrecht- Amsterdam line, DB is formulating a set of at first very basic, then advanced and towards project completion comprehensive and complete scenarios and requirements for demonstration.
In- line with agile project management methodology, these requirements are presented to the WP3 team in the form of "user stories" which help to drive the project priorities.

1.5.5 integration

Integration on openETCS context is focusing on several aspects:

- Integration of SCADE functional blocks with overall SCADE architectural model In a SCADE context, model-to-model integration is straightforward. As each functional block has a clear interface and all blocks have identical semantics we could follow an iterative, constructive and agile approach. Integration was mainly driven by alignment of interfaces and validated by interactive simulation.

- Integration of openETCS model with WP3 simulation environment For the March 2015 ITEA review, a simple dynamic simulation environment was set up using SCADE tools (SCADE Display to build a first version of an openETCS DMI software, SCADE Rapid Prototyper to interact with the simulator). Integration of the openETCS OBU software model followed the same principles as model-to-model integration. The same concept will be extended for a full dynamic simulation of the Utrecht-Amsterdam line in interaction with the openETCS reference model.

- Integration of openETCS model with openETCS runtime Currently, the demonstration is purely on model/ simulation level. However, since the openETCS toolchain allows for automatic code generation, we will integrate the generated code with the openETCS runtime. As the generated code is strictly target agnostic, integration can be adapted very flexibly to different hosts and execution models. Details of this integration phase will be discussed in the final version of this document, a first overview is given in the section discussing the APIs.

1.5.6 Towards EN50128 SIL4 objectives, interface to WP4

One of the objectives of openETCS is to demonstrate feasibility of the approach and concept in a CENELEC context. While we are focusing on analysis and design for now, we will also formulate a certification strategy.

An actual EN50128 certification of the openETCS software is beyond the scope of this project.

1.6 openETCS history and iterations

The openETCS Architecture and Design is beingimplemented in iterations. The current step (third iteration) is implementing essential kernel functions of the ETCS system. For a better understanding of the scope the Iteration is described in the following.

Third Iteration Functional Scope: The OBU functions for Szenarios defined in chapter 3

The openETCS third iteration model is defining the architecture and design of the openETCS OBU software. It is referencing [1] UNISIG Subset_026 version_3.3.0.

The appropriate functionality has been divided into a number of functional blocks that are being analysed and designed by work groups (as described in the previous section of this document).

The third iteration of the openETCS modelling effort is focusing on demonstrating some simple scenarios on the actual (for now simulated) Utrecht- Amsterdam infrastructure.

=======

1.7 Motivation

The openETCS work package 3 (WP3) aims to provide – amongst others - the software architecture for the openETCS kernel in order to eventually build the software itself. WP3 partner has put great effort in the openETCS software design, thus far without making definite choices on the software architecture itself respective of functional breakdown and data structures of the openETCS kernel. Since the project planning foresees in the production of a reference software to be used as a demonstrator by June 2015, it is of paramount importance that a first design deli-

varable of the openETCS kernel architecture be finalized shortly but no later than November 2014.

In compliance with the agreements made during the last WP 3 meeting at the 10.09.2014 in Brussels, DB has taken the initiative to design the aforesaid architecture including of functional breakdown and data structures in order to safeguard a timely delivery of these products. Furthermore, DB has ensured that these developments are focused on including end user requirements so as to develop a design in conformity with the needs and requirements of the operators. Specialists of DB and NS have cooperated together with other partners in WP3 to produce this document.

As referred to above, the architecture description has to be finalized in the month of November 2014. This version of the document is a draft version, demonstrating the general directions and philosophy of the architectural design, the functional breakdown of the software and the data structures. The design is focused on maximum efficiency in order to maximize on RAMS performance of the end product.

1.8 Objectives

The prime objective of WP3 is to produce a fully formal prototype for the openETCS reference system that can function as a demonstrator in collaboration with WP 4 and WP 5 for the openETCS approach and will be used as such in the final phase of the project. That phase is the first half of 2015. This objective is defined as ...

High level Objectives of this work: «any further general statements on the ITEA2 objectives, like...»

- Work on a model bases approach and process for effective collaborative work within an international ETCS developer team as stated above, the project needs a definite architecture design by the end of 2014.

  This document targets:

- Defining the general design and conditions of the openETCS architecture, functional breakdown and data structures
- Providing the guidelines for discussion during the workshops that are planned in October and November 2014 that will result in the final and decisive version of this document
- Being the 'platform' for finalization i.e. whatever be the products or results of the workshops shall be integrated in this document.

  Apart from these general objectives, the document means to provide for the materials that will enable WP3 partners to improve the efficiency of the Work Package activities:

- The comprehensive architecture design shall enable splitting the work load according to the building blocks defined by the architecture and allocate strictly compartmented work parcels or activities to WP3 partners.
- Doing so will enable WP3 to avoid any double work

- Compartmenting the work load according to the functional building blocks as defined by the architecture will enable efficient planning of activities, be it individually or the integrated WP3 planning for the coming period, aiming at a just in time delivery of all results and products
- Compartment the work load according to the functional building blocks as defined by the architecture will enable efficient planning of activities, be it individually or the integrated WP3 planning for the coming period, aiming at a just in time delivery of all results and products

1.9 Roles, responsibilities and tasks

In this section, the roles and responsibilities of the WP3 partners are confirmed, especially where they divert from what has been agreed upon at the start of WP3:

- **Responsibilities** First of all, in the last WP 3 meeting in Brussels on 10.09.2014 DB proposed to take over the lead of the architecture design and functional breakdown. At the subsequent weekly scrum meeting on 12.09.2014, it was agreed upon by all participants that DB will take over the lead (see Appendix ...);
- **Planning:** Alstom as WP 3 leader will remain to be responsible for the planning and the allocation of the defined tasks to the different partners
- **Roles:** Alstom will also coordinate the work and safeguard that the defined results will be delivered according to the quality requirements that are agreed within the ITEA2 project and the schedule and the milestones that will be agreed upon during the coming workshops;
- All WP3 partners will deliver the results or products according to planning as will be agreed upon during the said workshops.

  In the interest of a swift production of the critical documentation of which this version is a draft, specific tasks will be defined in terms of concrete results to be delivered, the timeframes in which these results must be produced and the partner who shall be responsible for that specific result and the planning. This is to safeguard the timely delivery. The process will be described in the next sections.

1.10 Assumptions and preconditions

- All future contributions shall be fully aligned an compliant the finalized and approved document
- Alls documents produced by the partners are requested to be compliant and merge to this document; other contributions will be discarded **The workshops are all about working as swift, as efficient and as productive as possible and make full use of the potential made available for these workshops by the partners. It is expected that the partners in the workshops will have the express intention to:**

- Contribute to the workshops with the intention to finalize the openETCS architecture;
- Provide resources according to the agreements made prior to the Workshops;
- Focus primarily on getting concrete results regardless of methodological issues that might arise. Where necessary or opportune, classical project management methodology will be applied;
- Provide full transparency with respect to experience, knowledge base and information touching the subjects to be treated in the workshops;
- Document on paper or electronically all output of the workshops and integrate these with the underlying document;
- Restrict discussions only to topics that have an immediate impact on the content or the quality of the end product: the improved version of this document.

1.11 openETCS history and iterations

The openETCS Architecture and Design willb de documented and implemented in iterations.

- **first iteration can be downloaded under the following link:**
- **second iteration has not been published due some technical issues and will fully substituted by the third iteration.**
- **first iteration can be downloaded under the following link:**

The current step (third iteration) is based on a step to implement the kernel functions of the ETCS system. For a better understanding of the scope the Iteration is described in the following.

Third Iteration Functional Scope: The OBU functions for ccenarios defined in chapter 3

The openETCS third iteration architecture and the design of the openETCS OBU software are derived from [1] UNISIG Subset_026 version_3.3.0.

All these functions are object of the openETCS project and have to be analysed from their requirements and subsequently modelled and implemented. With the given manpower in WP 3, a reasonable selection and order of these functions is required for the practical work that allows the distribution of the workload, more openETCS participants to join and leads to an executable—limited—kernel function as soon as possible.

Glossary

Notation	Description
application programming interface	an abstraction that is defined by the description of an interface and the behaviour of the interface.
balise group	One or more balises which are treated as having the same reference location on the track.
balise group message	
balise telegram	A telegram contains one header and an identified and coherent set of packets. A message maybe comprised of one or several telegrams.
Balise Transmission Module	On board equipment for intermittent transmission between track and train. It shall be able to receive telegrams from a balise.
Driver Machine Interface	ERTMS train-borne device to enable communication between ETCS and/or GSM-R and the train driver.
European Vital Computer	Computer device for the onboard ETCS.
EURORADIO	The functions required of a radio network coupled with the message protocols that provide an acceptably safe communications channel between track side and train borne equipment's
Juridical Recording Unit	Device to record all actions and exchanges relating to the movement of trains sufficient for off line analysis of all events leading to an incident.
Last Relevant Balise Group	It is the first balise group met and correctly read, when the linking information is not known by the train borne equipment. It is the last linked balise group found at the expected location and correctly read when the linking information is known by the train borne equipment. The LRBG is used as a common reference between the train borne and track side equipments in levels 2 and 3
linking information	Data defining the distance between groups of balises and the action to be taken if a balise group is not detected within given limits.

Notation	Description
location	Location describes a position in terms of topological relations.
Loop Transmission Module	Train borne equipment that reads the track mounted loop data.
odometry	The process of measuring the train?s movement along the track. Used for speed measurement and distance measurement.
on-board unit	on-board equipment for ETCS and the ETCS-related GSM-R.
orientation	
radio message	The Radio Block Centre (RBC) sends electronic messages to, and receives electronic messages from, ETCS onboard equipment on trains within the area which the RBC is controlling. These messages are transmitted via GSM-R data radio
service brake	Train stopping, from a given speed, at such a deceleration that the passengers do not suffer discomfort or alarm or at an equivalent deceleration in the case of non-passenger trains.
Specific Transmission Module	The train borne equipment of the ERTMS / ETCS must be able to be interfaced with the train borne equipment of an existing train supervision system. The Specific Transmission Module shall perform a translation function between these systems and the ERTMS / ETCS.
system requirement specification	Specification describing the technical properties of a piece of equipment based on a corresponding functional requirement specification.
Systems Modeling Language	The Systems Modeling Language (SysML) is general purpose visual modeling language for systems engineering applications. SysML is defined as a dialect of the Unified Modeling Language (UML) standard, and supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities.
Train Interface Unit	The unit that provides the interface between the train borne equipment and the train. It is likely to be unique to a class of train.

Notation	Description
train position	information related to the position of a train on the railway infrastructure.

2 Input documents

This paragraphs gives an overview about the input documents used for the:

- Analysis of the OBU Functions
- Functional decomposition and allocation of functional blocks, functions and libraries.
- Design of the OBU Functions
- Determination of "Use Cases" and Scenarios for the different iteration of the Architecture and Design Document

List of main documents that are being used as reference or input for analysis and design:

- **ERA TSI CCS Documents**
- **openETCS API Spec**
- **openETCS Requirements WP 2**
- **Railway Operator Documents**
- **Industry Documents**
- **ERSA Simulator Documents**
- **Other Project Partner Data information and documents**
- **Utrecht - Amsterdam Track Documents**

Furthermore relevant ETCS Know How from Industry and Operator is used for the design and the analysis

While the list above serves as high- level reference, detailed information, links to the actual documents and additional remarks are being maintained at: `https://github.com/openETCS/modeling/wiki/Input-Documents-Repository` while the documents describing the standard are referenced here: `https://github.com/openETCS/SSRS/wiki/SSRS-Documents`

The figure below illustrates the relationships among the input used documents:

For a detailed discussion of the actual work process, please refer to the previous chapters.

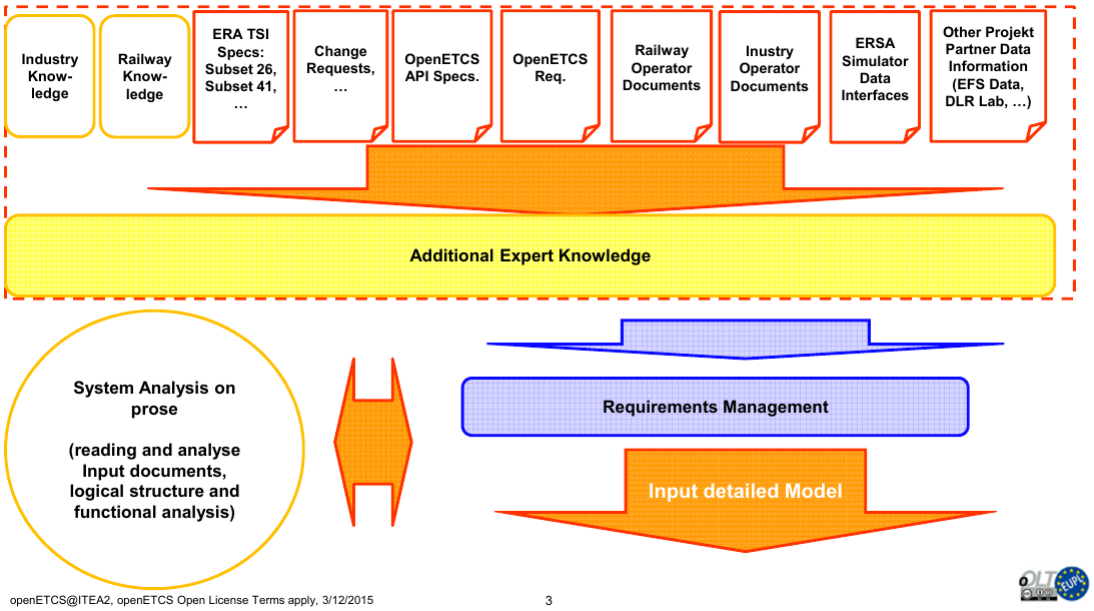


Figure 2. Analysing of input document

3 Use case description - proof of concept Utrecht - Amsterdam

3.1 Proof of concept on the Track Utrecht Amsterdam User Stories 1 - 4

The goal of the openETCS@ITEA2 project is to deliver at the and a proof of concept in a lab on a real ETCS Track. Since the Level 2 Utrecht - Amsterdam track was evaluated as the most approriate reference track for this concept due the maturity and representative of the track, it will be use for the mentioned simulation.

To start with the realisation of the concept in a iterative way in the same pattern the industry is proceeding and an regarding the "classical" state of the art of sytemanalisiys we started in this third iteration with the following Use Cases and Scenarios:

3.1.1 Use Case and Scenario 1

Start of Mission - Awakening of the Train: This use case according to the procedure in chapter 5 will demonstrate the start of a train from a no power modus to the state that train will be ready for level and mode change according to the chapter 5.
Link on Git-Hub `https://github.com/openETCS/modeling/issues/66`

The following Subsystems needs to be realised for Scenario 1:

- Procedure
- TIU Management
- DMI Management and Controller
- Position Report
- Management of Radio Communication
- Manage Track Data
- Manage Mode and Level
- Train Supervision

3.1.2 Use Case and Scenario 2

Start of Mission - Start in Level 2 Mode FS: This use case according to the procedure in chapter 5 will demonstrate the start of a train from the awakening of the train in mode stand by to the state that train will receive a movement authority in level 2 and change into the mode full

supervision to start running under real supervision according to the chapter 5.
link on Git-Hub `https://github.com/openETCS/modeling/issues/67`

The following Subsystems needs to be realised for Scenario 2:

- Procedure
- TIU Management
- DMI Management and Controller
- Position Report
- Management of Radio Communication
- Manage Track Data
- Manage Mode and Level
- Train Supervision

3.1.3 Use Case and Scenario 3

Brake intervention - Revocation of a Movement Authority and Overrun Permitted Speed:
This use case according to the subset 26 chapter 3 principles will demonstrate the brake intervention that will cause by a revocation of a movement authority due to a occupied section or track an due simple overrun of a permitted speed according to the chapter 3.
Link on Git-Hub `https://github.com/openETCS/modeling/issues/68`

The following Subsystems needs to be realised for Scenario 3:

- Procedure
- TIU Management
- DMI Management and Controller
- Position Report
- Management of Radio Communication
- Manage Track Data
- Manage Mode and Level
- Train Supervision

3.1.4 Use Case and Scenario 4

ETCS Onboard Unit is reading and sending track information: This use case according to the subset 26 chapter 3 principles will demonstrate the full completeness and checking the reading and sending of track information in interaction with the ETCS Onboard Unit and the track that will be separated in radio and balise messages. Messages and packages are defined in chapter 7 and 8 of the subset 26.
Link on Git-Hub `https://github.com/openETCS/modeling/issues/69`

The following Subsystems needs to be realised for Scenario 4:

- Procedure
- TIU Management
- DMI Management and Controller
- Position Report
- Management of Radio Communication
- Manage Track Data
- Manage Mode and Level
- Train Supervision
- Building of coordinate system

3.2 Environment model for the use case demonstrations

In order to dynamically explore and demonstrate the openETCS OBU kernel software, a dynamic simulation and demonstration environmental model is being created. During Iteration 3, this comprises the following features and functionalities:
• openETCS OBU formal model
• openETCS DMI formal model with Display specification model
• Simplified version, not all features are implemented yet; following our use- case driven approach we are only implementing features that are essential to show the use cases selected by our "internal customer".
• Environment model with
• - simplified track model (balise locations, speed profile)
• - simplified model of Movement Authority
• - interactive widgets to manipulate the simulation environment

(see Figure 3.)

Figure 3. Environment for use case demonstration

3.3 Dynamic track model of the ETCS Level 2 line Amsterdam- Utrecht

This environment model will be fourthly enhanced during the last project period, in order to:

- Allow full dynamic simulation of the Utrecht- Amsterdam line
- Form a basis for a (future) dynamic track simulator, which models the balise locations, balise messages and RBC messages for any given line, and which can automatically be generated from engineering datasets.
- Provide a full track model for the purposes of openETCS

The principle of this model is shown in Figure **??**. The idea is to split the model in two parts:
- Generic functions, that represent the behaviour of the trackside installations (for example balise groups, RBC reception, RBC sending)
- Data that instantiate these generic functions (for example balise locations, balise message/ packet data, radio message/ packet data)

This simulation approach is designed to be adaptable to very different kinds of scenarios:

- Proof of concept in openETCS: Amsterdam- Utrecht line

In that case we have a partially known line (balise locations and track topology are known, messages and packets are known "as-is" from JRU data owned by the partners. We have been using SCADE graphical notation for the development of the concept in order to disseminate it more easily. The models will however be instantiated by automatic conversion of engineering and JRU data to simulation scenarios for "nominal" operations validation. Specific test cases will either be transformed by script from test data bases (in WP4) or can also be graphically modelled (for user validation scenarios or demonstration purposes).

• OBU validation for existing tracks
Once the concept and simulation environment have been validated in openETCS context, the tooling can be used to import track data from other tracks, such as the Corridor A, in order to develop interoperable ETCS specifications.
• Track validation for existing OBUs (or against the TSI) With a validated OBU software model, the system can also be used to validate new track layouts.
Dynamic track simulation opens new dimensions of attacking the interoperability issues that Europe is facing.

3.3.1 Model concept

Traditionally, most ETCS simulators use predefined simulation scenarios, which define the inputs to the model at predefined steps. They can either be time triggered (update every n time units), distance- oriented (update every n distance units) or cycle- oriented (update every simulation cycle)

The disadvantage of such an approach is clear: all parameters have to be predefined and pre-validated, for example to simply make sure that the time/ distance ratio is consistent. In such a setting it is very difficult to simulate events in just one dimension (independently of the other parameters).

By creating functional models of the track elements and combining them with data we are now able to explore much more complex scenarios with for example interactive driver interaction, thus not only exploring the implementation of the OBU or the engineering / layout of the track, but also the impact on operational procedures, human factors.

As all events can be recorded and replayed and are based on a formal model (also for the track model and engineering data), the entire environment can be validated and potentially even be certified. The same EN50128 certifiable toolchain is used that serves the consortium for modelling of the OBU itself.

The basic concept is very simple, but can be developed into a full dynamic track and RBC model. One functional block represents the balises, one functional block represents the RBC (see Figure 4.) When train movement is simulated, the functional blocks representing the balises and the RBC, respectively, receive the steadily updated train position (via the input "Loc") and react accordingly. When a balise group detects a passing train, it reacts accordingly by emitting the defined messages and packets.

In an analog way, the RBC block reacts as defined when it receives a train to track message from the simulated OBU by emitting the appropriate set of messages and packets. While the functionality of the balise groups is modelled in order to reflect their behaviour, they are instantiated using parameter sets that are defined as SCADE constants. One dataset each defines:

• Balise positions (from engineering data)
(see Figure 5.)

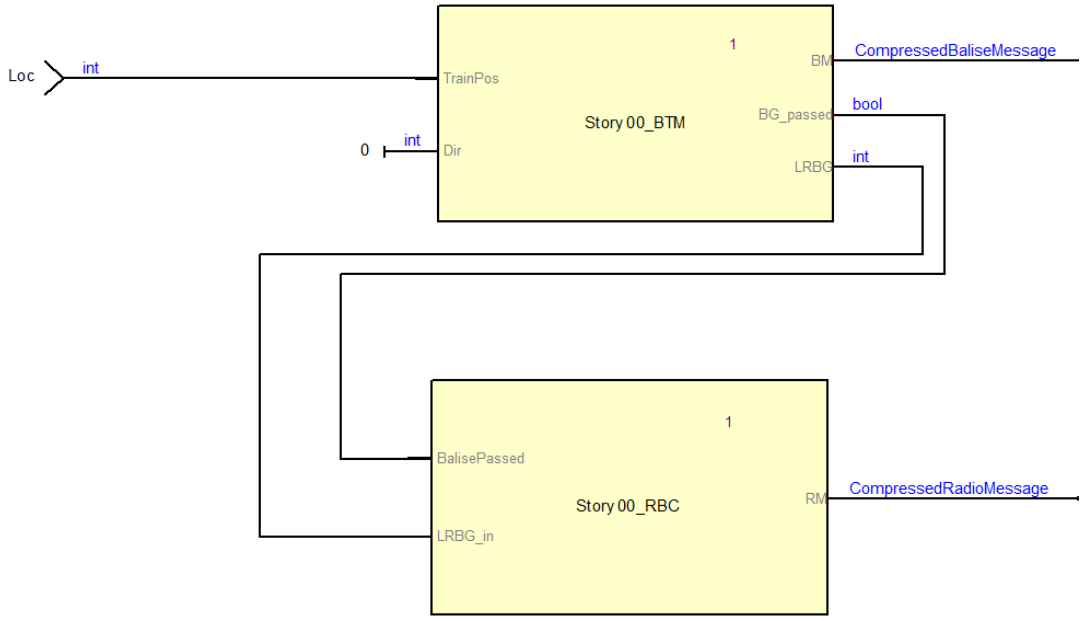


Figure 4. High- level view of dynamic model with Balises and Radio Block Center

BG139	Basics::BaliseGroupData	vspa-dvaz DM
NID_C	int	426
NID_BG	int	139
Pos	int	20155
Or_BG	OrBG	Utrecht
Or_Line	OrLine	N

Figure 5. Example of balise position data

In this example, the balise's engineering data have a local coordinate system (km on track, nominal orientation in direction of Amsterdam or Utrecht, respectively, and orientation of the line (Z= zuid, Dutch for southbound, N= northbound)

• Balise messages and packets (from JRU data)
(see Figure 6.)

Actual data from the Utrecht- Amsterdam line can be seen in this example. Typical for a Level 2 line is that most balise groups are only used as positional reference for the train. The balise groups are then put together to a "track" which emits telegrams as the train "passes" over it. This concept is scaleable and the balises can equally instantiated using textual Scade models which can automatically from (for example) JRU data. (see Figure 7.)

• RBC messages and packets (from JRU data). The modelling of the RBC follows a heavily simplified concept at the moment: The simulated RBC emits telegrams/messages/packets in predefined situations. For example, the reception of a specific train-to-track message such as a specific position report triggers emission of a set of messages and packets, such as a complete Movement Authority message with packets for international static speed profile, national values, gradient profile, MA, etc. The functional block may seamlessly be replaced with a full RBC simulation. 8. shows a dataset representing RBC packets in SCADE constant format.

Constant	Type	Value
B139_H1	BaliseTelegramHeader_int_T	
q_updown	int	1
m_version	int	16
q_media	int	0
n_pig	int	0
n_total	int	1
m_dup	int	1
m_mcount	int	255
nid_c	int	426
nid_bg	int	139
q_link	int	1
B139_H2	BaliseTelegramHeader_int_T	
q_updown	int	1
m_version	int	16
q_media	int	0
n_pig	int	1
n_total	int	1
m_dup	int	2
m_mcount	int	255
nid_c	int	426
nid_bg	int	139
q_link	int	1

Figure 6. Example of Balise Message Header Data

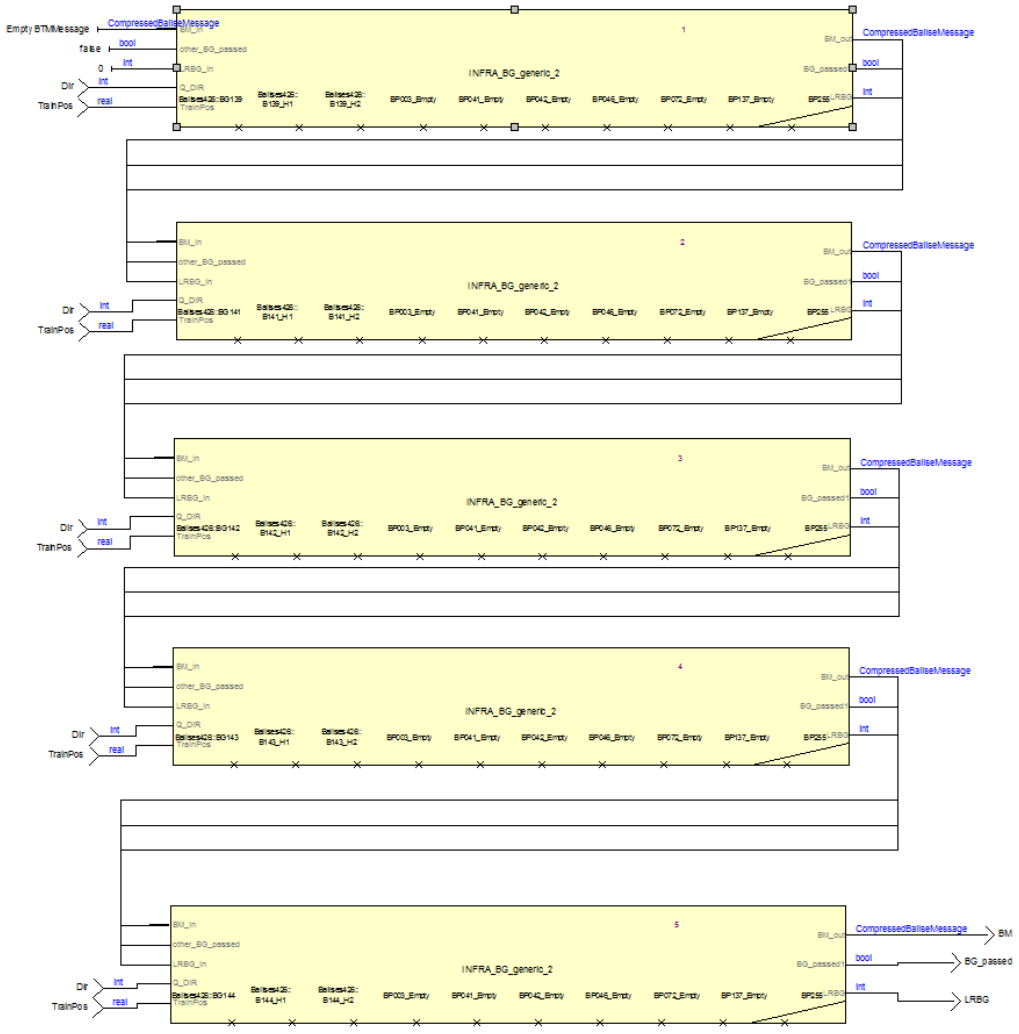


Figure 7. Graphical representation of five balise groups

Constant	Type	Value
RM139_00Header	Radio_TrackTrain_Header...	
radioDevice	int	0
receivedSyste...	int	0
nid_message	int	3
t_train	int	4048087
m_ack	int	1
nid_lrbg	int	140
t_train_reference	int	0
nid_em	int	0
q_scale	int	0
d_sr	int	0
t_sh_rqst	int	0
d_ref	int	0
q_dir	int	0
d_emergencyst...	int	0
m_version	int	0
RM139_03	RMP03	
valid	bool	true
Q_DIR	int	2
L_PACKET	int	MaxElementsRPacket03
Q_SCALE	int	1
D_VALIDNV	int	0
N_ITER	int	1
NID_C	int	426
V_NVSHUNT	int	40
V_NVSTFF	int	40
V_NVONSIGHT	int	40
V_NVUNFIT	int	10
V_NVREL	int	15
D_NVROLL	int	5
Q_NVSRBKTRG	int	0
Q_NVEMRRLS	int	0
V_NVALLOWO...	int	0
V_NVSUPOVT...	int	40
D_NVOVTRP	int	200
T_NVOVTRP	int	60
D_NVOTRP	int	60
M_NVCONTACT	int	0
T_NVCONTACT	int	35
M_DVDERUN	int	1
D_NVSTFF	int	-1
Q_NVDRIVER	int	1
RM139_05	RMP05	
valid	bool	true
Q_DIR	int	1
L_PACKET	int	MaxElementsRPacket05
Q_SCALE	int	1
N_ITER	int	4
SECTIONS	RMP05Es_T	
0	RMP05E_T	{valid : true, D_LINK : 435, Q_NEWCOUNTRY : 0, NID_BG : 139, Q_LINKORIENTATION : 1, Q_LINKREACTION : 0, Q_LOCACC : 2}
1	RMP05E_T	{valid : true, D_LINK : 54, Q_NEWCOUNTRY : 0, NID_BG : 141, Q_LINKORIENTATION : 0, Q_LINKREACTION : 0, Q_LOCACC : 2}
2	RMP05E_T	{valid : true, D_LINK : 973, Q_NEWCOUNTRY : 0, NID_BG : 142, Q_LINKORIENTATION : 0, Q_LINKREACTION : 0, Q_LOCACC : 2}
3	RMP05E_T	{valid : true, D_LINK : 54, Q_NEWCOUNTRY : 0, NID_BG : 143, Q_LINKORIENTATION : 0, Q_LINKREACTION : 0, Q_LOCACC : 2}
4	RMP05E_T	{valid : true, D_LINK : 167, Q_NEWCOUNTRY : 0, NID_BG : 144, Q_LINKORIENTATION : 1, Q_LINKREACTION : 0, Q_LOCACC : 2}

Figure 8. Example of Radio Message and Packet data

• Potentially, any required data set can be defined on both the input side (static data or scenarios) and on the output side (expected outcome for validation)

4 Architecture Description

4.1 System Architecture view in ERA TSI Subset 25 Chapter 2 "Basic System Description"

The system architecture is an outcome on the functional decomposition according to the analysis of the document in § 2 Input documents. A starting point for the first analysis is the System Strcuture in ERA TSI Subset 26 chapter 2 the subchapter 2.5 and 2.4.

Since we decided in the openETCS project not to consider all subsystem due to our using scope for the proof of concept on the ETCS Level 2 Utrecht - Amsterdam line, we analysed the necessary subsystem as seen here:

4.1.1 System Structure from the subchapter 2.4. of ERA TSI Subset 26 chapter 2

- 2.4.1.1 Due to the nature of the required functions, the ERTMS/ETCS system will have to be partly on the trackside and partly on board the trains.
- 2.4.1.2 This defines two sub-systems, the on-board sub-system and the trackside sub-system. 2.4.1.3 The environment of ERTMS/ETCS system is composed of:
- a) the train, which will then be considered in the train interface specification;
- b) the driver, which will then be considered via the driver interface specification;
- c) other onboard interfaces (see architecture drawing in 2.5.3),
- d) external trackside systems (interlockings, control centres, etc.), for which no
- interoperability requirement will be established.

4.1.2 Sub System from the subchapter 2.5. of ERA TSI Subset 26 chapter 2

4.1.2.1 2.5.1 Trackside subsystem

§ 2.5.1.1 Depending of the application level (see further sections), the trackside subsystem can be composed of:

- a) balise
- b) lineside electronic unit ***- not in the scope of this project***
- c) the radio communication network (GSM-R)
- d) the Radio Block Centre (RBC)
- e) Euroloop ***- not in the scope of this project***

- f) Radio infill unit ***- not in the scope of this project***
- g) Key Management Centre (KMC) ***- not in the scope of this project***

2.5.1.2 Balise

- 2.5.1.2.1 The balise is a transmission device that can send telegrams to the on-board sub-system.
- 2.5.1.2.2 The balise is based on the existing Eurobalise specifications. These documents are included in the frame of the ERTMS/ETCS specifications.
- 2.5.1.2.3 The balises provides the up-link, i. e. the possibility to send messages from trackside to the on-board sub-system.
- 2.5.1.2.4 The balises can provide fixed messages or, when connected to a lineside electronic unit, messages that can be changed.
- 2.5.1.2.5 The balises will be organised in groups, each balise transmitting a telegram and the combination of all telegrams defining the message sent by the balise group.

2.5.1.3 Lineside electronic unit *- not in the scope of this project*

- 2.5.1.3.1 The lineside electronic units are electronic devices, that generate telegrams to be sent by balises, on basis of information received from external trackside systems.

2.5.1.4 Trackside radio communication network (GSM-R)

- 2.5.1.4.1 The GSM-R radio communication network is used for the bi-directional exchange of messages between on-board sub-systems and RBC or radio infill units.
- 2.5.1.4.2 Intentionally deleted

2.5.1.5 RBC

- 2.5.1.5.1 The RBC is a computer-based system that elaborates messages to be sent to the train on basis of information received from external trackside systems and on basis of information exchanged with the on-board sub-systems.
- 2.5.1.5.2 The main objective of these messages is to provide movement authorities to allow the safe movement of trains on the Railway infrastructure area under the responsibility of the RBC.
- 2.5.1.5.3 The interoperability requirements for the RBC are mainly related to the data exchange between the RBC and the on-board sub-system.

2.5.1.6 Euroloop *- not in the scope of this project*

- 2.5.1.6.1 The Euroloop subsystem operates on Level 1 lines, providing signalling information in advance as regard to the next main signal in the train running direction.
- 2.5.1.6.2 The Euroloop subsystem is composed of an on-board functionality and by one or more trackside parts.

2.5.1.7 Radio infill Unit *- not in the scope of this project*

- 2.5.1.7.1 The RADIO INFILL subsystem operates on Level 1 lines, providing signalling information in advance as regard to the next main signal in the train running direction.
- 2.5.1.7.2 The RADIO INFILL subsystem is composed of an on-board functionality and by one or more trackside parts (named RADIO INFILL Unit).

2.5.1.8 KMC *- not in the scope of this project*

- 2.5.1.8.1 The role of the KMC is to manage the cryptographic keys, which are used to secure the EURORADIO communications between the ERTMS/ETCS entities (ERTMS/ETCS on-board equipments, RBCs and RIUs).

2.5.2 On-board sub-system *kernel part of the project*

2.5.2.1 Depending of the application level (see further sections), the on-board sub-system can be composed of:

- a) the ERTMS/ETCS on-board equipment;
- b) the on-board part of the GSM-R radio system;

2.5.2.2 ERTMS/ETCS on-board equipment

- 2.5.2.2.1 The ERTMS/ETCS on-board equipment is a computer-based system that supervises the movement of the train to which it belongs, on basis of information exchanged with the trackside sub-system.
- 2.5.2.2.2 The interoperability requirements for the ERTMS/ETCS on-board equipment are related to the functionality and the data exchange between the trackside sub-systems and the on-board sub-system and to the functional data exchange between the on-board sub-system and:
- a) the driver;
- b) the train;
- c) the onboard part of the existing national train control system(s).

2.5.2.3 Onboard radio communication system (GSM-R)

- 2.5.2.3.1 The GSM-R on-board radio system is used for the bi-directional exchange of messages between on-board sub-system and RBC or radio infill unit.
- 2.5.2.3.2 Intentionally deleted.

2.5.3. ERTMS/ETCS Reference Architecture

The figure below shows the system scope for design of the ETCS Subsystem regarding the openETCS Onboard Unit and the Test Environment within the scope of the openETCS project proof of conept on ETCS Level 2 Utrecht - Amsterdam.

4.1.3 System Structure from the subchapter 2.4. of ERA TSI Subset 26 chapter 2

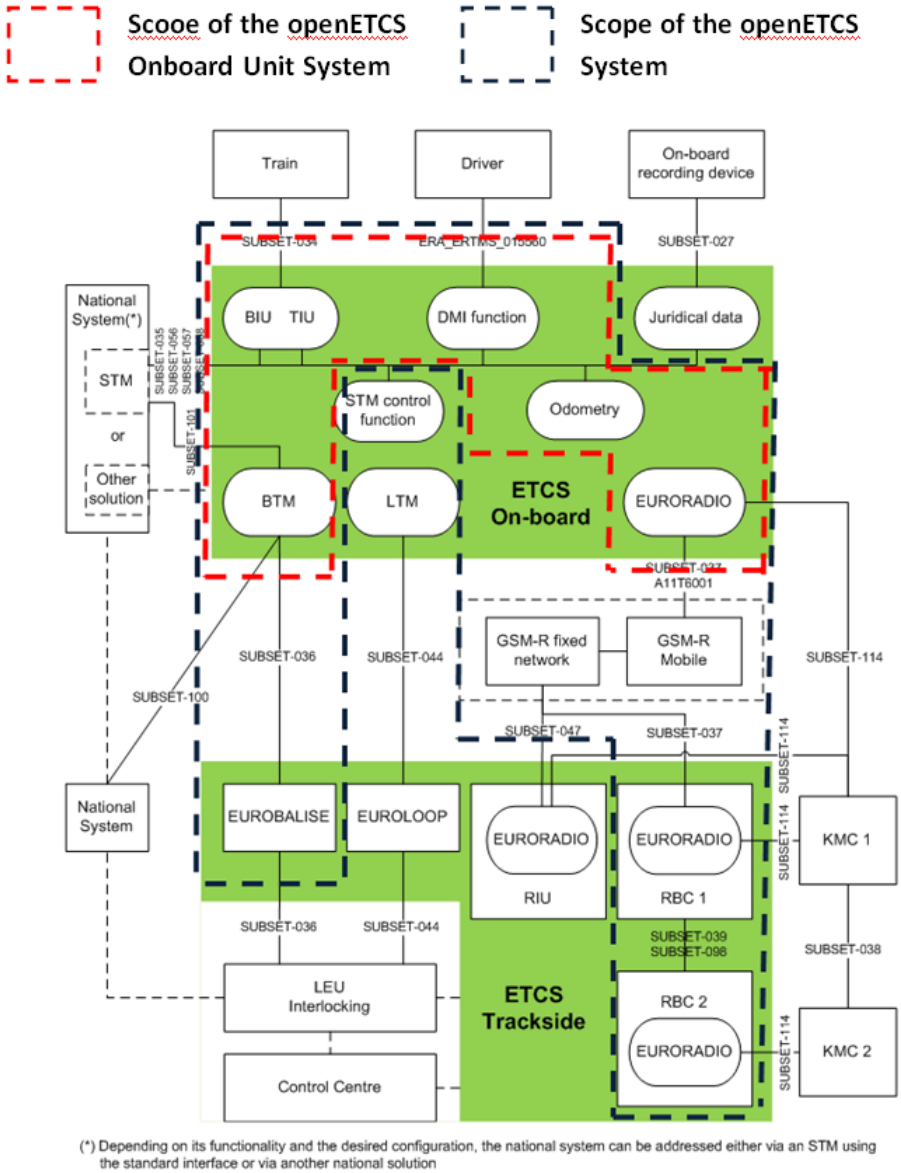


Figure 9. Scope of System according to ERA TSI Chapter 2

4.2 System Architecture SysML View

The SysML System view of the architecture will reflect the scope accorgin to 4.1 and is a top down breakdown to the design layer. The functional breakdown has been done in Scade System and is part of the design model. Furthermore it will reflect all the external and internal interface that will will be described in 4.3. Another goal of the System Architecture SysML view is to explain and set the boundaries for the ETCS Kernel development "F2 Kernel" as the main design part of the openETCS@ITEA2 project.

4.2.1 1st level System Architecture view

All subystem of the ETCS/ERTMS Basic Sytem according in the scope of the openETCS@ITEA2 project will be reflected in this 1st level view. Furthermore the interlocking as part of a full Rail Signalling System, but not part of the openETCS scope, will be highlighted in this view.

Interlocking = interlocking is an arrangement of signal apparatus that prevents conflicting movements through an arrangement of tracks such as junctions or crossings. The signalling appliances and tracks are sometimes collectively referred to as an interlocking plant. An interlocking is designed so that it is impossible to display a signal to proceed unless the route to be used is proven safe.

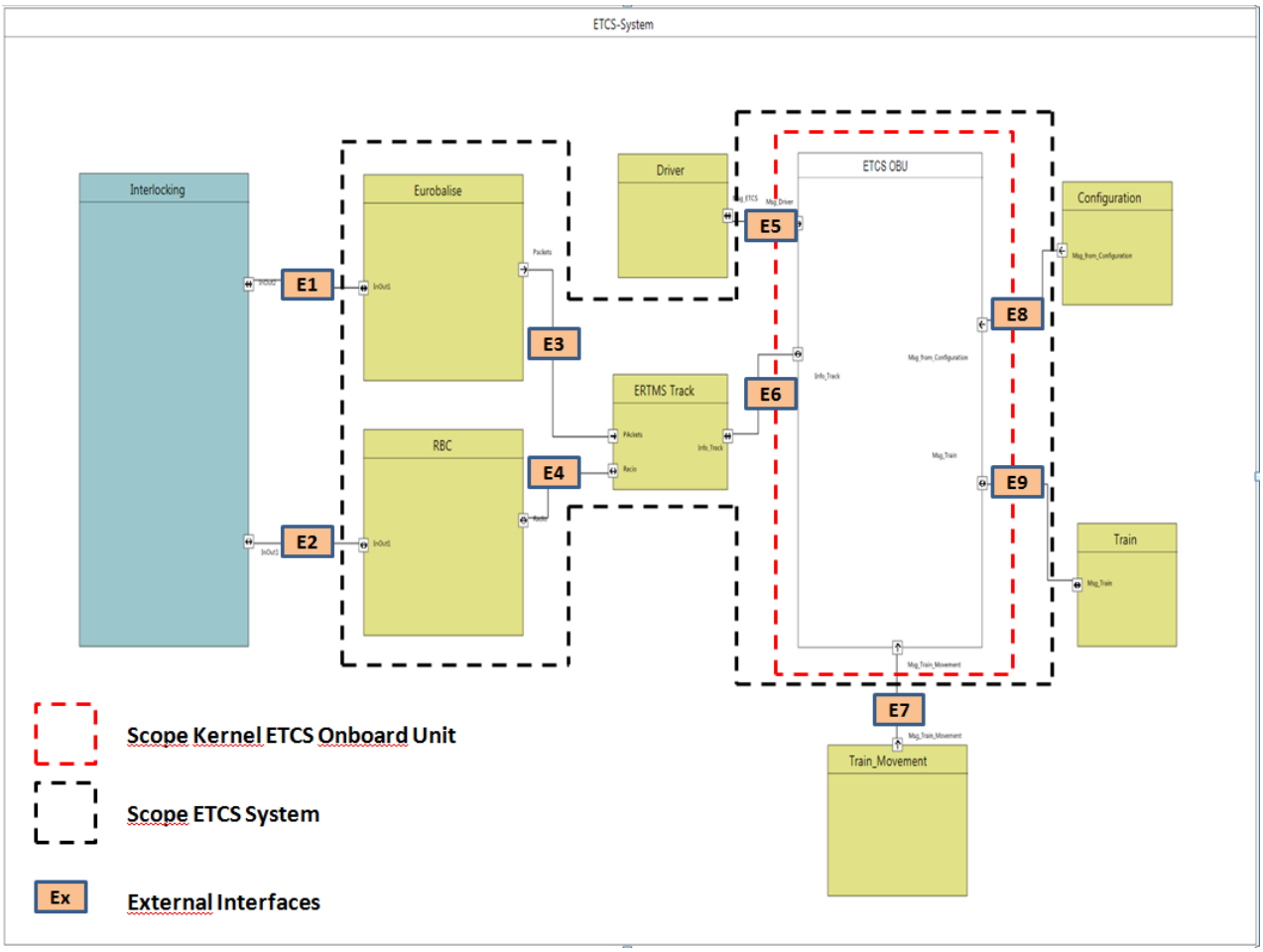


Figure 10. 1st level System Architecture view

4.2.2 2st level System Architecture view

The 2nd level system view will provide a decopmosition of the ETCS Onboard Unit systems and the Kernerl of the ETCS. The Kernel is the main part of the ETCS Onboard Unit system and reflects the functions specified in the ERA TSI Subset 26. Therefore the boundaries and interfaces to the other subasystems of the ETCS Onbard Unit needs to be fully desribed and formal. At least the formalisation kernel functions and boundaries should be realized in the openETCS project.

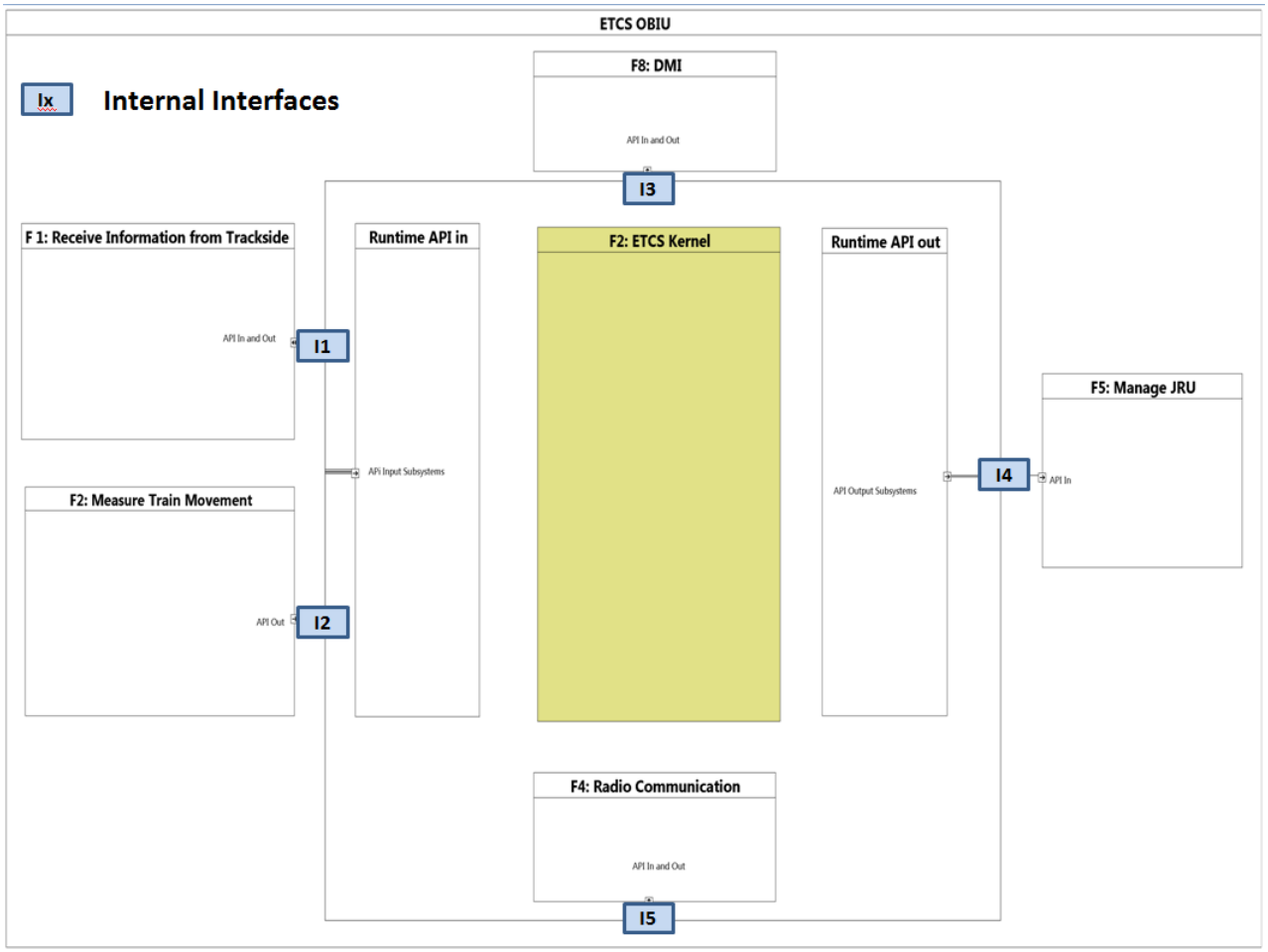


Figure 11. 2nd level System Architecture view

4.2.3 3rd level System Architecture view

The 3rd level system view will provide a decopmosition of the ETCS Kernel of the ETCS Onboard Unit Systems. The decomposition and further design of the subfunctions of the kernel are part of the chapter 6 in this document. In chapter 6 we will consider the design description that will be completed by every designer itself. The designer can decided in this layer about the decomposition and boundaries of his subsystem, but need to describe the design choices.

4.3 Interfaces

This section will consider the External and Internal interfaces as descibed in the system decomposition figures in 4.2.1 and 4.2.2

4.3.1 External Interfaces

External interfaces will describe the data flow between the Systems outside of the scope of the openETCS Project and the ETCS Onboard Unit System

E1:

In- and Out flow between the Interlocking an Eurobalise. There will be 2 kind of balises

- Fixed Balise: no interaction to the interlocking
- Balise Controlled: interaction to the interlocking trough LEU

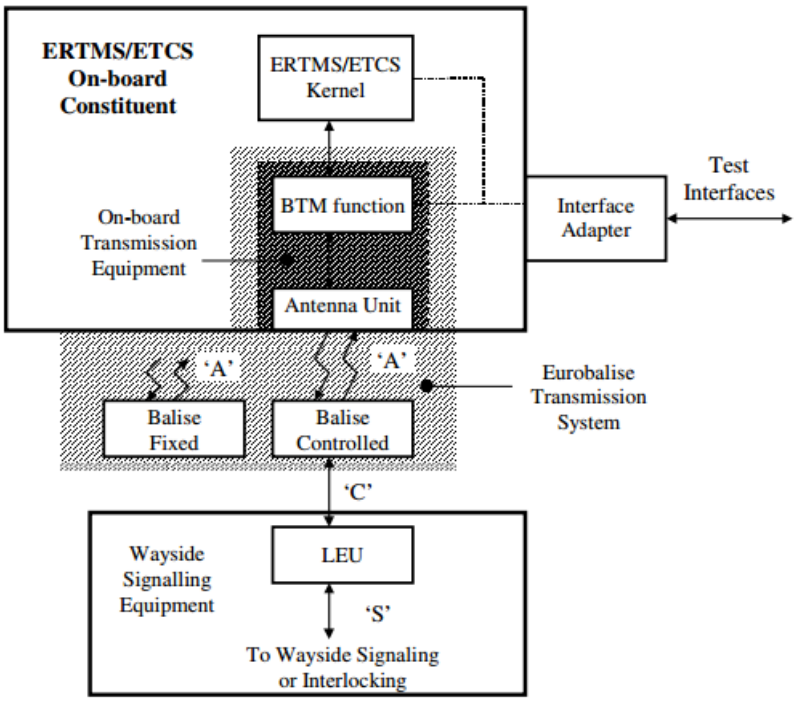


Figure 1: Eurobalise Transmission System, Interfaces

Figure 12. Eurobalise

E2:

In- and Out flow between the Interlocking and Radio Block Control. This External interface will ensure the states or logics directly to the Radio Block Control and the other way back from the train to the interlocking.

E3:

Input flow from the Eurobalise to the Balise Transmission Module or Antenna Unit (BTM) into the ETCS Onboard Unit. As already described on the figure in E1.

E4:

In- and Out flow between the Radio Block Controll and the Euroradio Modul into the ETCS Onboard Unit. This interface is not in Level 0 or 1 active since there is no necessary for ETCS Radio interaction between track and train.

E5:

This interface will describe the interaction between the Human and Display (Human Machine Interface or Driver Machine Interface). See description on the figure below

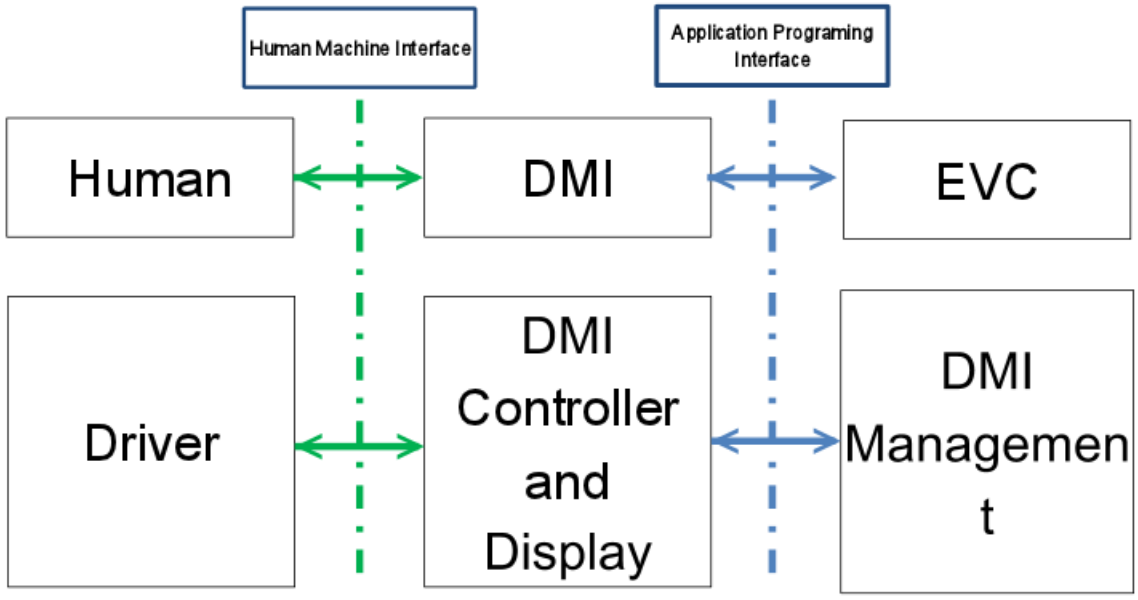


Figure 13. DMI Interfaces

E6:

This interface is composite the interfaces E3 and E4.

E7:

Input interface to the Odometrie Subsystem of the ETCS Onboard System. Will send information to train if there is any movemement outside the ETCS System is leading such as "cold movement".

E8:

Input interface to the ETCS Onboard Unit system to set configuration data such as fixed values, system values, national values and train configuration.

E9:

In- and Out flow between the ETCS Onboard Unit System and the Train. This interface will describe the itneraction between the Train and the ETCS Onboard Unit System such as brake controll, traction controll, door controll, ...

4.3.2 Internal Interfaces

Internal interfaces will describe the data flow between the ETCS Onboard Unit Kernel and ETCS Onboard Unit Subsystems within the ETCS Onboard Unit System.

I1:

In flow from the Balise Transmission Module (BTM or Antenna) to the "F2 ETCS Kernel" trough Runtime API in. Transmitted data are information from the Eurobalise.

I2:

In flow from the Odometrie (ODO) to the "F2 ETCS Kernel" trough Runtime API in. Transmitted data are information from the Movement of the train.

I3:

In- and Out flow between the DMI Controller and the "F2 ETCS Kernel" trough Runtime API in and out. Transmitted data are information of driver action and display. See description in figure of "External Interface E5".

I4:

Out flow from "F2 ETCS Kernel" to the JRU Manager trough Runtime API out. Transmitted data are all necessary information for a juridical recorder unit "black box".

I5:

In- and Out flow between the Euroradio and "F2 ETCS Kernel" trough Runtime API in and out. Transmitted data are radio track information (RBC) and information to the track (RBC).

5 Runtime API

5.1 Introduction to the Architecture

5.1.1 Abstract Hardware Architecture

For proper understanding of openETCS API and of constraints imposed on both sides of the API, we need to define a *reference abstract hardware architecture*. This hardware architecture is “abstract” is the sense that the actual vendor specific hardware architecture might be totally different of the abstract architecture described in this chapter. For example, several units might be grouped together on the same processor.

However the actual vendor specific architecture shall fulfil all the requirements and constraints of this reference abstract hardware architecture and shall not request additional constraints.

5.1.2 Definition of the reference abstract hardware architecture

The reference abstract hardware architecture is shown in figure 14.

The reference abstract hardware architecture is made of a bus on which are connected *units* defining the on-board unit (OBU):

- European Vital Computer (EVC);
- Train Interface Unit (TIU);
- odometry (ODO);
- Driver Machine Interface (DMI);
- Specific Transmission Module (STM);
- Balise Transmission Module (BTM);
- Loop Transmission Module (LTM): Not part of this openETCS implementation;
- EURORADIO;

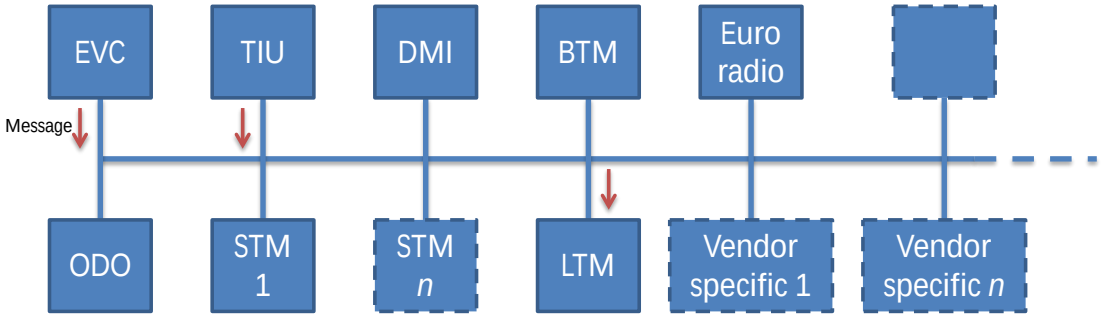


Figure 14. Reference abstract hardware architecture

- Juridical Recording Unit (JRU): Not part of this openETCS implementation;

Elements not being part of this implementation are marked.

Those units shall working concurrently. They shall exchange information with other units through asynchronous message passing.

5.1.3 Reference abstract software architecture

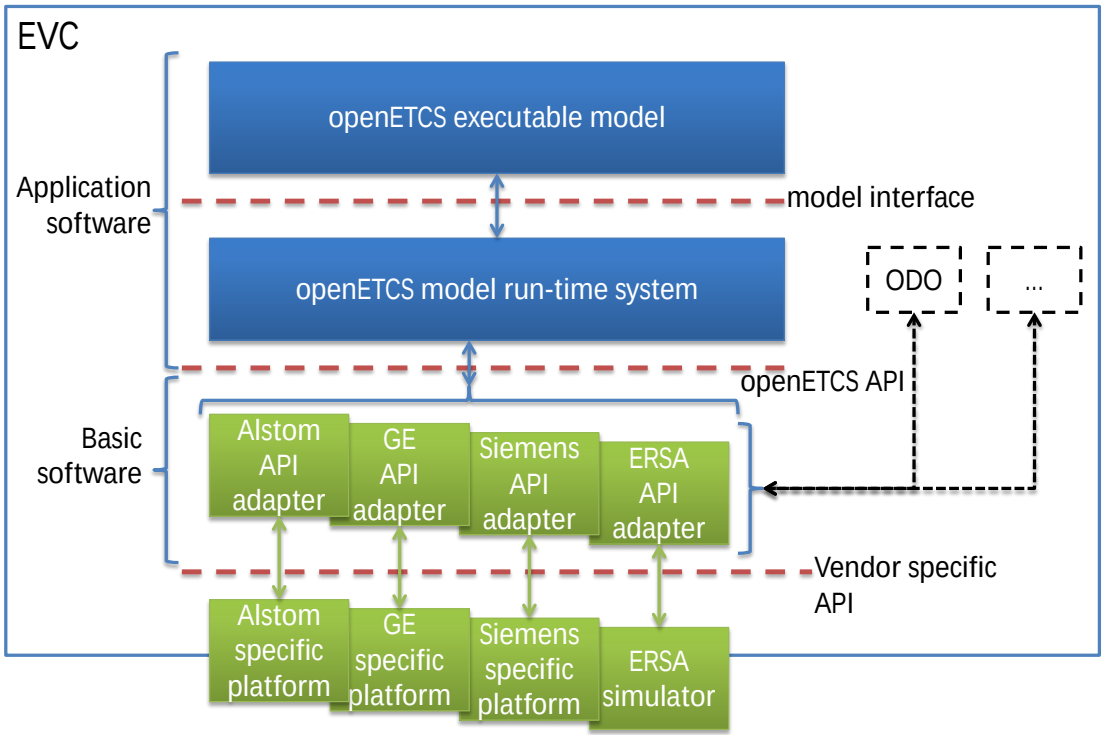


Figure 15. Reference abstract software architecture

The *reference abstract software architecture* is shown in figure 15. This architecture is made of following elements:

- *openETCS executable model* produced by the [2] Scade Model. It shall contain the program implementing core ETCS functions;
- *openETCS model run-time system* shall help the execution of the openETCS executable model by providing additional functions like encode/decode messages, proper execution of the model through appropriate scheduling, re-order or prioritize messages, etc.
- *Vendor specific API adapter* shall make the link between the Vendor specific platform and the openETCS model run-time system. It can buffer message parts, encode/decode messages, route messages to other EVC components, etc.
- All above three elements shall be included in the EVC;
- *Vendor specific platform* shall be all other elements of the system, bus and other units, as shown in figure 14.

We have thus three interfaces:

- *model interface* is the interface between openETCS executable model and openETCS model run-time system.
- *openETCS API* is the interface between openETCS model run-time system and Vendor specific API adapter.
- *Vendor specific API* is the interface between Vendor specific API adapter and Vendor specific platform. This interface is not publicly described for all vendors. You can find the Alstom imnplementation as an example.

The two blocks openETCS executable model and openETCS model run-time system are making the *Application software* part. This Application software might be either openETCS reference software or vendor specific software.

The Vendor specific API adapter is making the *Basic software* part.

5.2 Functional breakdown

5.2.1 F1: openETCS API Runtime System and Input to the EVC)

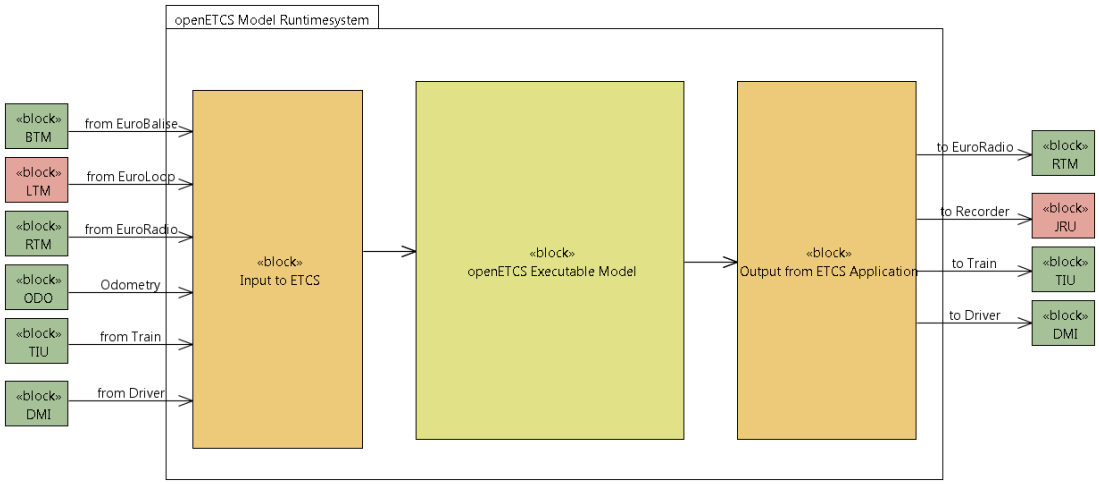


Figure 16. openETCS API Highlevel View

Figure 16 shows the structure of API with respect of the software architecture. Input boxes and output boxes not implemented in this stage are marked as red, other interfaces are marked as green. 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.

5.2.1.1 Principles for Interfaces (openETCS API)

Information is exchanged *messages* in an asynchronous way. A message is a set of information corresponding to an event of a particular unit, e.g. a balise received from the BTM. The possible kind of messages are described in chapter **??**.

The information is passed to the executable model as parmeters to the snychronous 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 arraysquite 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.

5.2.1.2 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 system requirement specification (SRS) and passed to the buffers to the units.

5.2.1.3 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.

5.2.1.4 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.

5.2.1.5 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.

5.2.1.6 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.

5.2.1.7 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.

5.2.1.8 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

6 Design Description

6.1 F1: Receive information from Trackside

6.2 F2: ETCS Kernel

6.2.1 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.

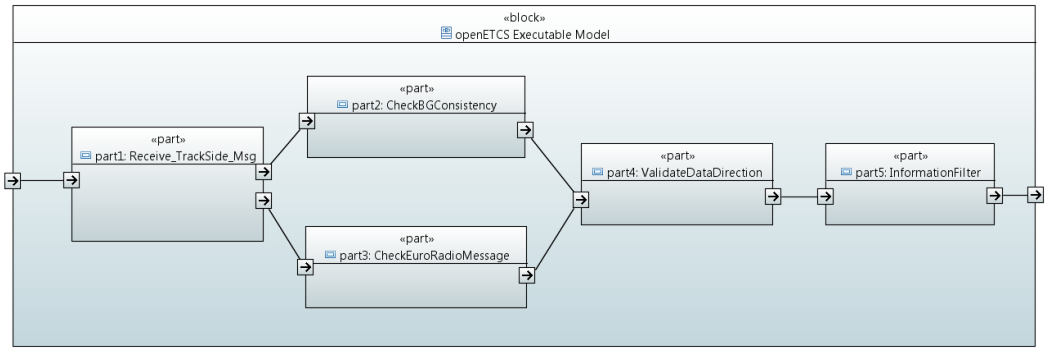


Figure 17. Structure of the Manage_TrackSideInformation_Integration module with submodules

6.2.1.1 Input

For providing the output, the module needs different input data flows. An overview is provided in table 2

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 2. Overview over input

Input 0: fullChecks

The boolean indicates, if all checks on the message should be performed. The possible values are given in table 3.

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

Table 3. Possible values for the input fullChecks

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.

Input 2: ActualOdometry

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

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 4. Possible values for the input reset

Input 4: trainPosition

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

Input 5: modeAndLevel

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

Input 6: tNvContact

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

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.

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 5. Possible values for the input connectionStatus

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.

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.

Input 11: q_nvlocacc

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

6.2.1.2 Output

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 6.

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 6. Dataflow at output

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 7. 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.

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.

Output 2: BadBaliseMessageToDMI Information to be passed to the DMI to indicate the reception of a “bad balise” to the driver.

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 8. Possible values for the input errorLinkedBG

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 9. Possible values for the input errorUnlinkedBG

Output 5: passedBG The output passedBG provides the received balise group message in a special format needed by the module “Calculate train position”.

Output 6: outPositionParams The output outPositionParams provides the parameters for the position report in a special format needed by the module “Provide Position Report”.

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”.

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 10. Possible values for the input radioSequenceError

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 11. Possible values for the input radioMessageConsistencyError

6.2.1.3 Receive_TrackSide_Msg in Manage_TrackSideInformation_Integration

Reference to the SRS (or other requirements)

- [1, Chapt. 7 and 8]: Definition of the Balise Telegram
- [3, 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

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.

Interface

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.

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/ManageLocationRelatedInformation/BaliseGroup/Receive_TrackSide_Msg

6.2.1.4 CheckBGConsistency in Manage_TrackSideInformation_Integration

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

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.

Interface

An input of the operator is a list of recived 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.

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" .

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/ManageLocationRelatedInformation/BaliseGroup/CheckBGConsistency

6.2.1.5 CheckEuroradioMessage in Manage_TrackSideInformation_Integration

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

- [1, Chapt. 8.4.4]: Rules for Euroradio messages
- [1, Chapt. 3.16]: Data consistency

Short description of the functionality

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.

Interface

The operator expects a radio message, information about the timestamp of the last relevant event, the national value for timeout (`T_NVCONTACT`) and the status of the radio connection as input. The operator provides as output the radio message marked as valid or invalid in the message header and updated metadata in the message. Errors are indicated through boolean outputs.

Functional Design Description

The operator performs the following checks:

- Content checks
  - The whole message must be complete and contains all necessary fields. [1, 3.16.1.1]
  - The message must respect the ETCS language. [1, 3.16.1.1] The operator checks, if all necessary variables and packets for the corresponding message are part of the message and if no packets and variables are included in the message, which are not allowed.
- Timing checks
  - Check if the timestamp of a message is greater than the timestamp of the message received before. [1, 3.16.3.3.3]

  - If a message contains the timestamp “Unknown”, check if this message is part of the initiation of the communication session. [1, 3.16.3.3.4]
  - Perform the check with the current message n: $T\_TRAIN_n <= T\_TRAIN_{n-1} + T\_NVCONTACT$ [1, 3.16.1.1]. This ensures, that the message was received in due time.

For inconsistent messages, the following actions are performed by the operator:

- If a message is not consistent, it shall be rejected. [1, 3.16.3.1.1.1] For this purpose, the message is marked as invalid, so it can be rejected by the operators using the output of the CheckEuroradioMessage-operator.
- The RBC shall be informed, when a message was rejected. [1, 3.16.3.1.1.2] Therefore the necessary information for creating an error report is provided as boolean flags.
- This operator will not trigger the reaction for an interrupted radio connection to the RBC. The reaction sepcified by `M_NVCONTACT` will be triggered by the RBC session management module.

The check by the Euroradio-protocol [1, 3.16.3.1.1] will not be performed by the operator, but on a lower level (RTM or openETCS-API).

The operator is not responsible for checking the integrity of the message in the bitstream format. The bitwalker, which is converting the bitstream into the internally used datatypes, is performing the checks, which are only possible on the raw data. This includes a CRC check on the whole message and a value range check for each variable. If a consistency error is detected by the bitwalker, it is signalled to the model. If the bitwalker marks a packet as valid, all variables are expected to contain a valid value. The bitwalker is not part of the ETCS kernel.

Safe connection supervision is not in the scope of this operator. This functionality will be implemented by the “Management of Radio communication” module.

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/ManageLocationRelatedInformation/BaliseGroup/CheckEuroRadioMessage`

6.2.1.6 ValidateDataDirection in Manage_TrackSideInformation_Integration

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

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

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.

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.

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.

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/ManageLocationRelatedInformation/BaliseGroup/ValidateDataDirection

6.2.1.7 InformationFilter

Reference to the SRS and other requirements

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

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.

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 12.

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 12. Overview of the InformationFilter interface

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 18 show the highlevel decomposition of the functionality. The filter functionality can be decomposed into a FirstFilter, SecondFilter, ThirdFilter an 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 FirstFilter, 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 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.

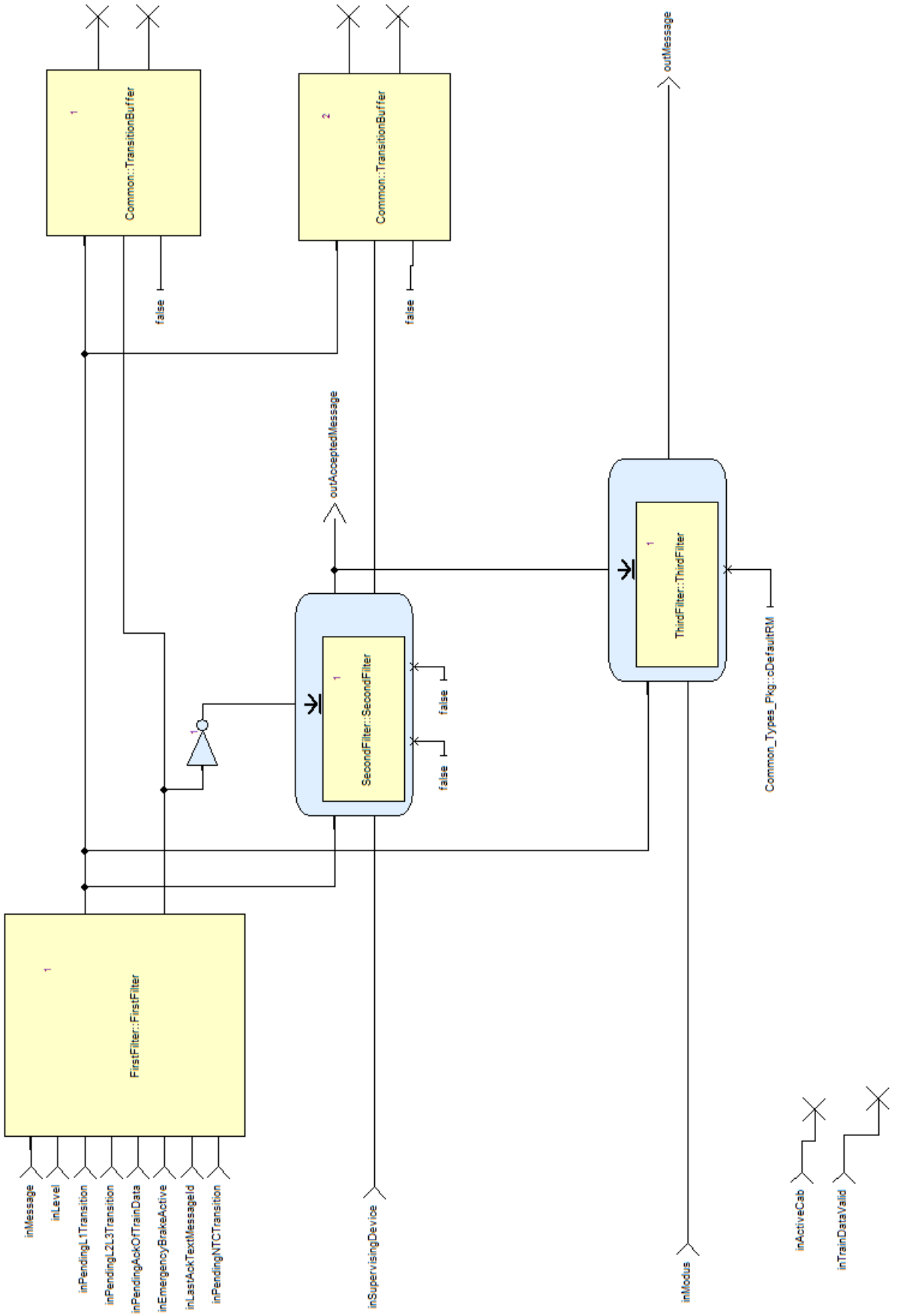


Figure 18. High level overview of the InformationFilter components.

A detailed list of packages and their handling depending of ETCS level or mode can be seen in table 19 and 20.

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
	Information from National System X through STM interface	Onboard operating level					
		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 19. 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	A	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	A	A	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	A	R	R	R	NR	NR	A	R
49	Message Radio 15	Conditional Emergency Stop	NR	R	R	R	A	A	R	A	R	A	R	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	A	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	A	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	A	R	A	R	A	NR	NR	A	R
67	Package 52	Permitted Braking Distance Information	NR	A[2][4]	R	R	A	A	A	A	A	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	A	R	A	R	A[1]	NR	NR	A	R
69	Package 60()	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	Modes																	
			NP	SB	PS	SH	FS	LS	SR	OS	SL	NL	UN	TR	PT	SF	IS	SN		RV
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	AD:[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	AD:[4]	R	R	A	A	A	A	R	R	A	R	A:[1]	NR	NR	A		R
4	Packet 12, 15	Movement Authority	NR	AD:[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	AD:[4]	R	R	A	A	A	A	R	R	A	R	A:[1]	NR	NR	A		R
9	Packet 27	International SSP	NR	AD:[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	AD:[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+ (optional) List of Balises in SR mode	NR	AD:[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	AD:[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	AD:[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	AD:[4]	R	R	A	A	A	A	R	R	A	A	A:[1]	NR	NR	A		R
26	Packet 70	Route Suitability Data	NR	AD:[4]	R	R	A	A	A	A	R	R	A	R	A:[1]	NR	NR	A		R
27	Packet 71	Adhesion Factor	NR	AD:[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	AD:[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	A	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 (non-)equipped RBC	NR	R	R	R	A	A	A	A	R	R	R	R	R	NR	NR	R		R
36	Packet 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	AD:[4]	R	R	A	A	A	A	R	A	A	A	A:[1]	NR	NR	A		R

Figure 20. Lists of packages and their handling depending on train modes

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/ManageLocationRelatedInformation/BaliseGroup/InformationFilter

6.2.2 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 21, 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`.

6.2.2.1 Input

For providing the output, the module needs different input data flows. Table 13 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 13. Overview of input

Input 0: `NationalValues`

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

Input 1: `TrainPosition`

This input is the current train position.

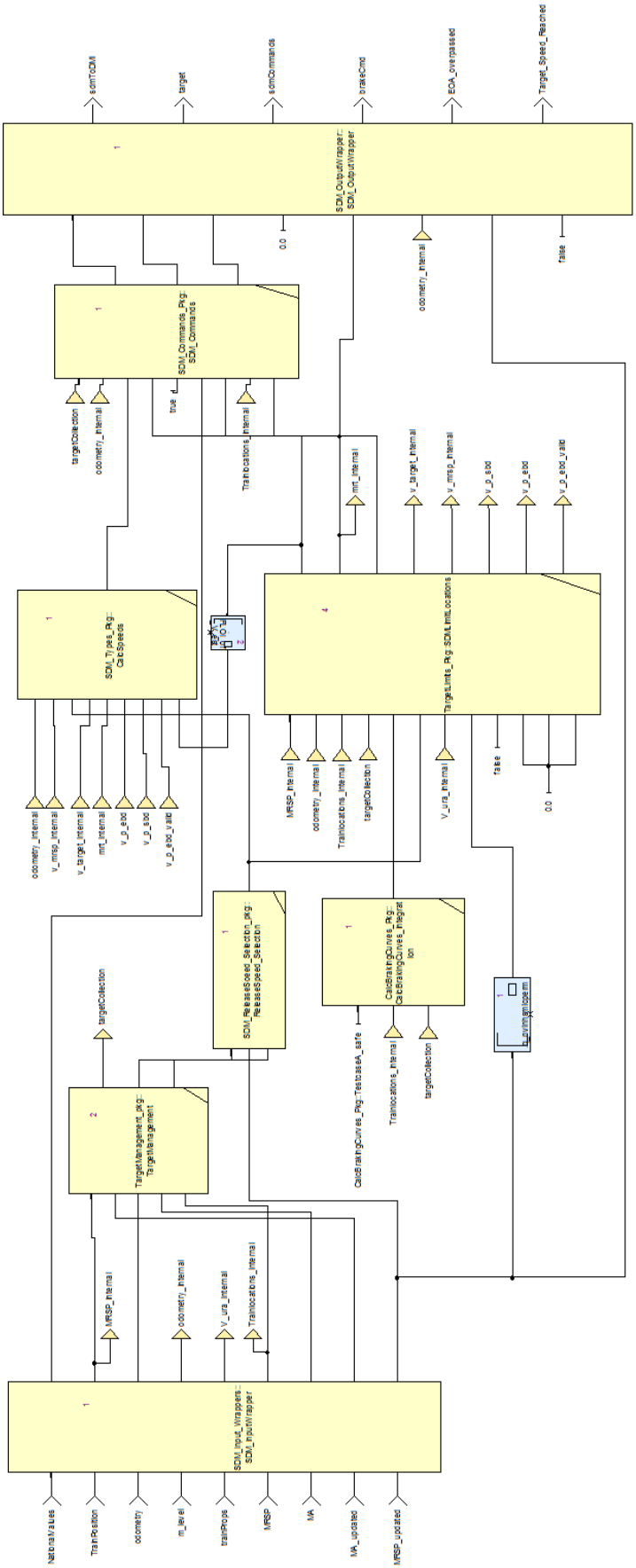


Figure 21. Structure of component ProvidePositionReport

Input 2: odometry

This input is the odometry data.

Input 3: m_level

This input is the current level of the train.

Input 4: trainProps

This input is a set of train related properties.

Input 5: MRSP

This input is the most restrictive speed profile.

Input 6: MA

This input is a movement authority.

Input 7: MA_updated

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

Input 8: MRSP_updated

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

6.2.2.2 Output

Based on the input the block produces the following output. Table 14 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 14. Overview of output

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.

Output 1: target

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

Output 2: `sdmCommands`

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

Output 3: `brakeCmd`

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

Output 4: `EOA_overpassed`

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

Output 5: `Target_Speed_Reached`

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

6.2.2.3 SDM_InputWrapper in Train Supervision

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

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

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.

Interface

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.

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.4 TargetManagement in Train Supervision

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

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

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.

Interface

Functional Design Description

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.

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.

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.

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.5 CalcBrakingCurves_Integration in Train Supervision

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)

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.

Interface

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. 22.

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

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.6 SDMLimitLocations in Train Supervision

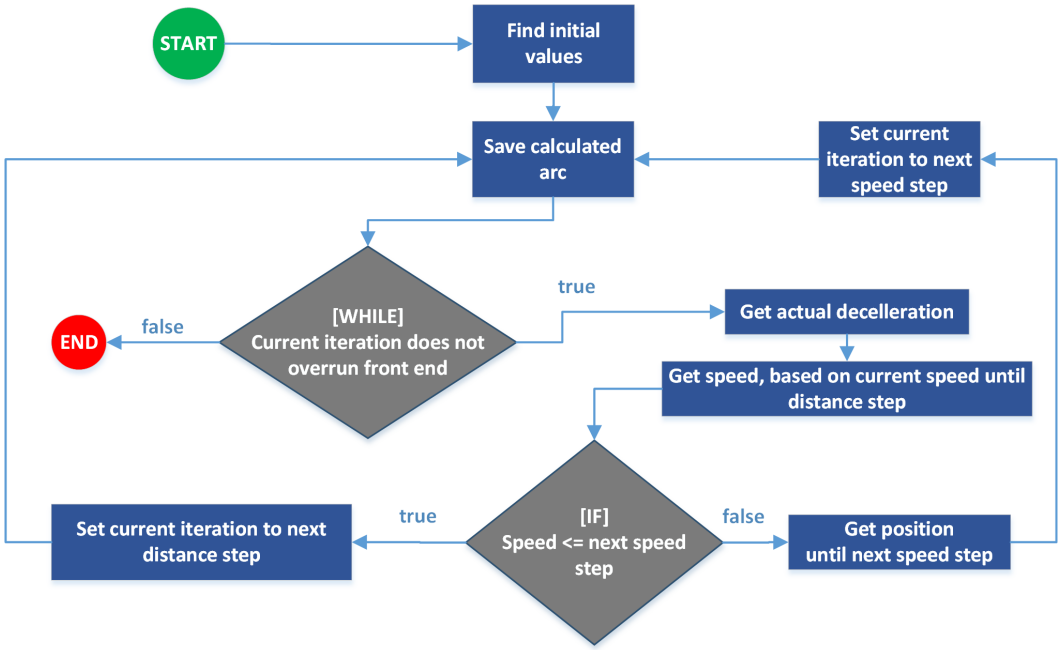


Figure 22. Calculation of Braking Curves

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

- [1, Chapt. 3.13.9]: Supervision Limits
- [4, 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

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.

Interface

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.

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.

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.

6.2.2.7 CalcSpeeds in Train Supervision

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

- [1, Chapt. 3.13.9]: Supervision Limits

Short description of the functionality

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

Interface

Functional Design Description

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

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.8 ReleaseSpeed_Selection in Train Supervision

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

- [1, Chapt. 3.8]: Movement authority

Short description of the functionality

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

Interface

Functional Design Description

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

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.9 SDM_Commands in Train Supervision

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

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

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.

Interface

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.

Reference to the Scade Model

only in special case or link to the Scade model

6.2.2.10 SDM_OutputWrapper in Train Supervision

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

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

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.

Interface

Functional Design Description

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

Reference to the Scade Model

only in special case or link to the Scade model

6.2.3 Manage ETCS Procedures

6.2.3.1 Start of Mission - Awakness of Train

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

Chapter 5, § 5.4

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.

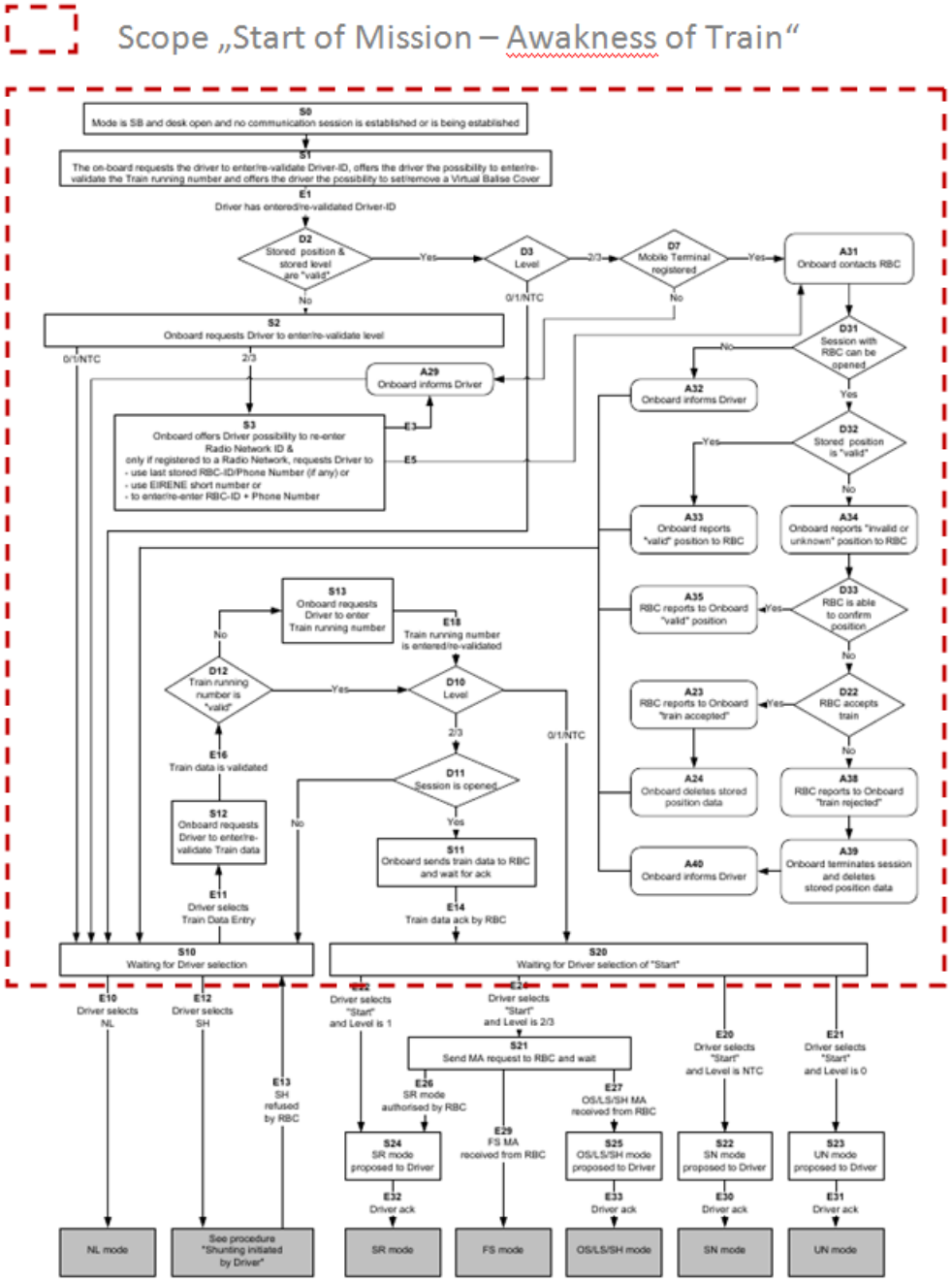


Figure 23. Start of Mission - Awakness of Train

Interface

Input Flow

- Information from TIU
- Action from Driver (DMI)
- Information from Position Calculation
- Information from Persistent Data
- Information from Management of Radio Communication
- Information from Mode and Level Management (Level and Mode Status)
- Information from Radio Block Control

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)

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.

Refernce to the Scade Model

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

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

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

Chapter 5, § 5.4

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.

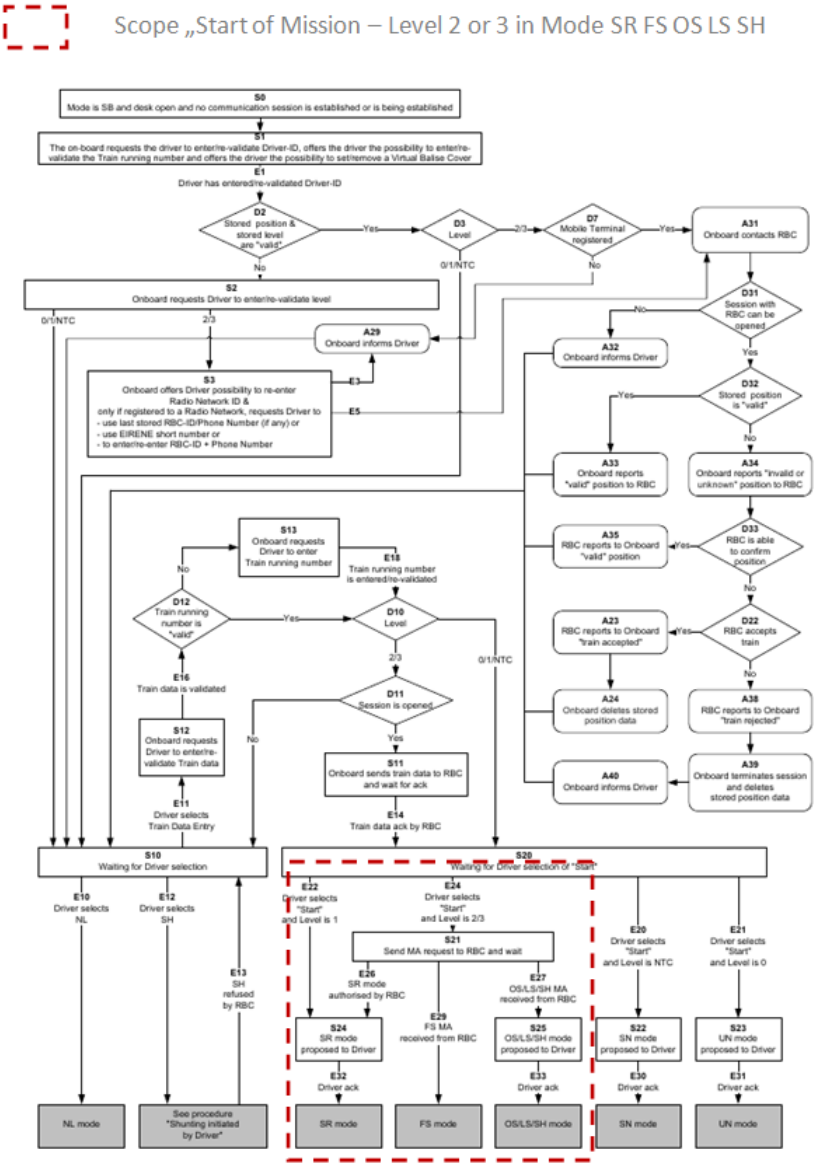


Figure 24. Start of Mission - Start of Mission in Level 2 or 3 and Mode SR FS OS LS SH

Interface

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)

Output Flow

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

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.

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`

6.2.4 Manage Track Data

6.2.4.1 F.2.2 Calculate Train Position

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

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` .

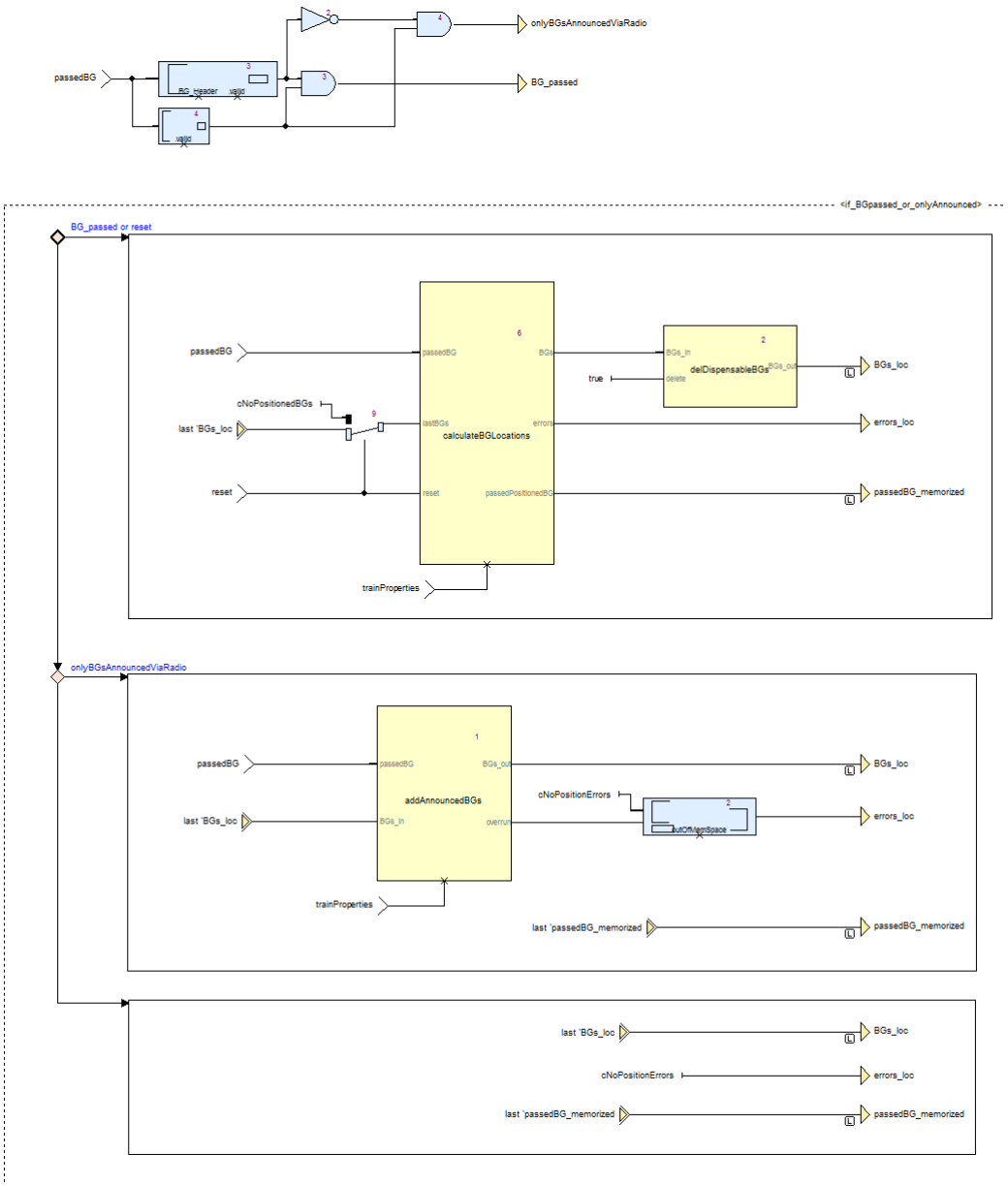


Figure 25. Calculating the balise group locations

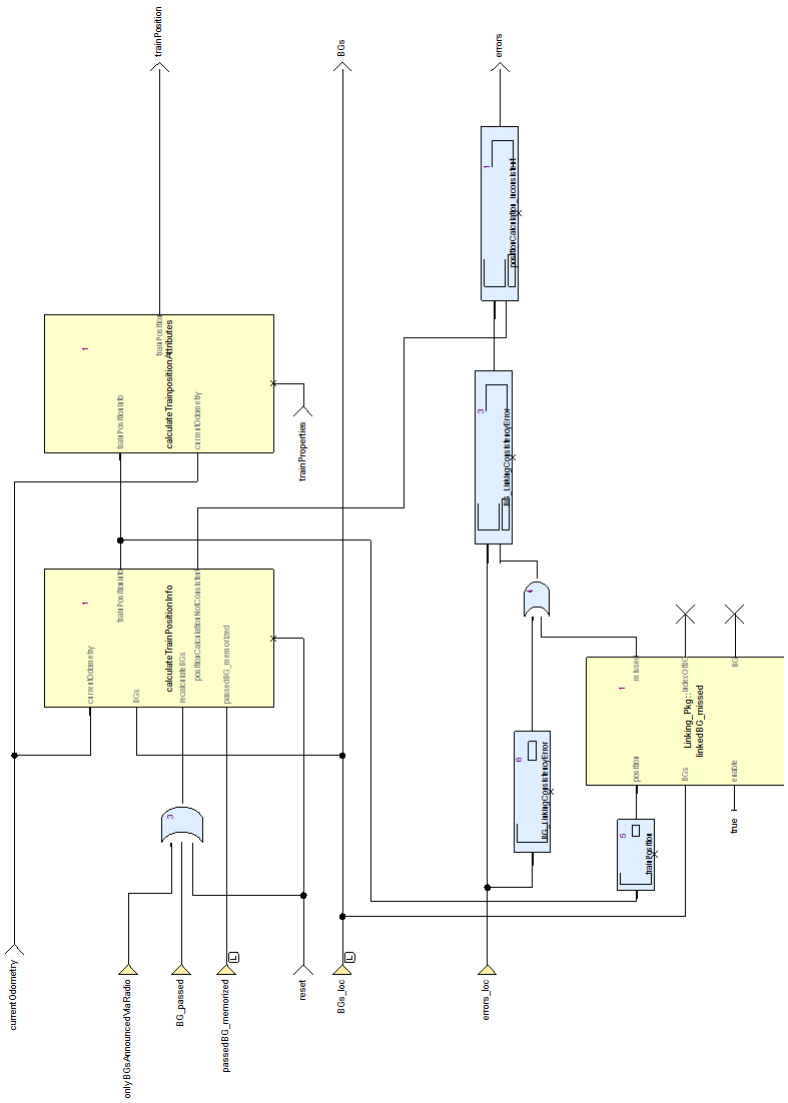


Figure 26. Calculating the current train position and attributes

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.

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.

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].

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.

6.2.4.2 Provide Position Report

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,

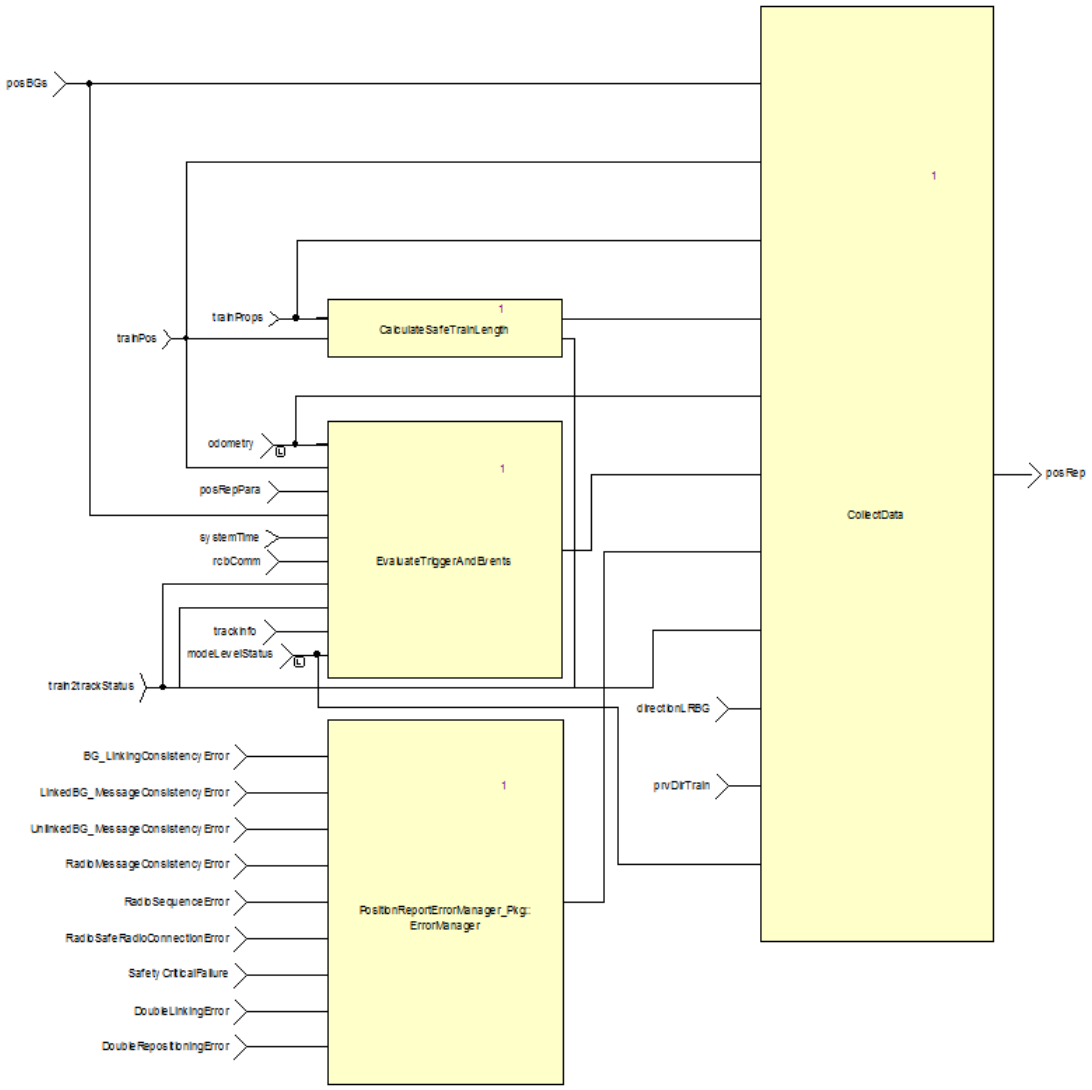


Figure 27. Structure of component ProvidePositionReport

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 27, 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`.

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.

Reference to the SRS (or other requirements

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

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.
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.

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.

6.2.5 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.

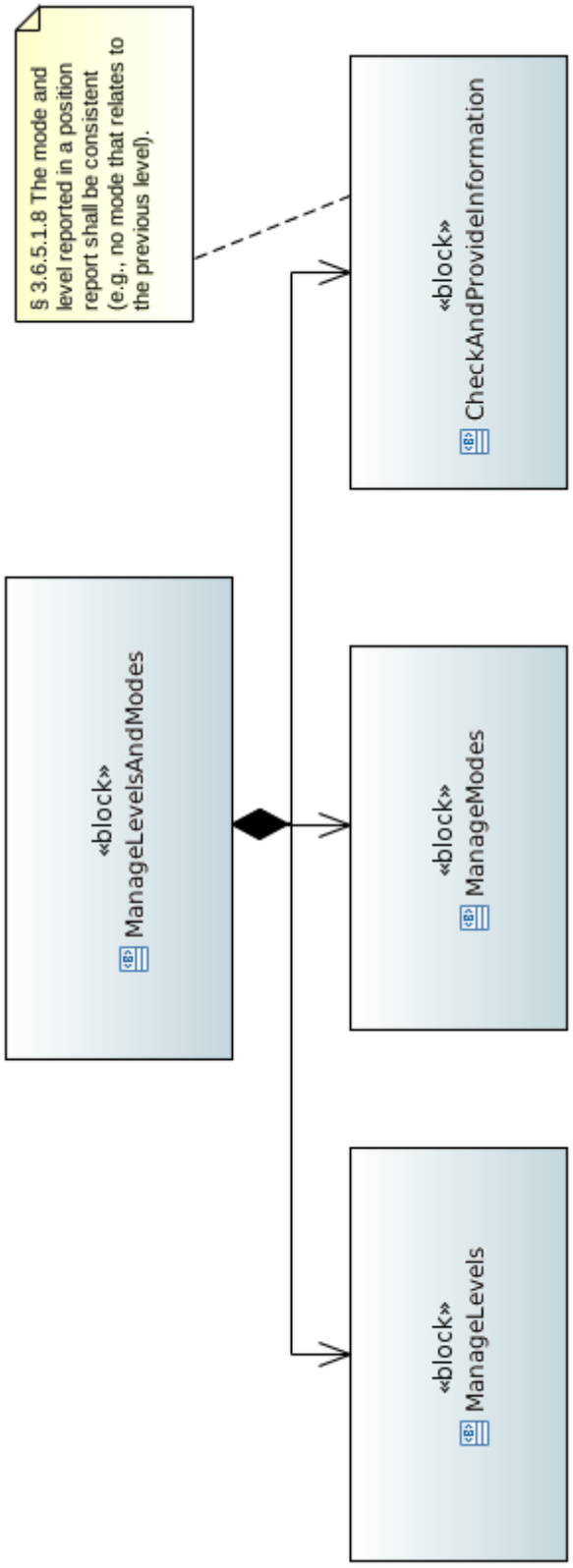


Figure 28. High level Architecture

6.2.5.1 Function Level Management

Reference to the SRS or other Requirements

see [1] section 5.10

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.

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

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.

Reference to the Scade Model

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

6.2.5.2 Function Mode Management

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

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.

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

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 paquet of outputs.

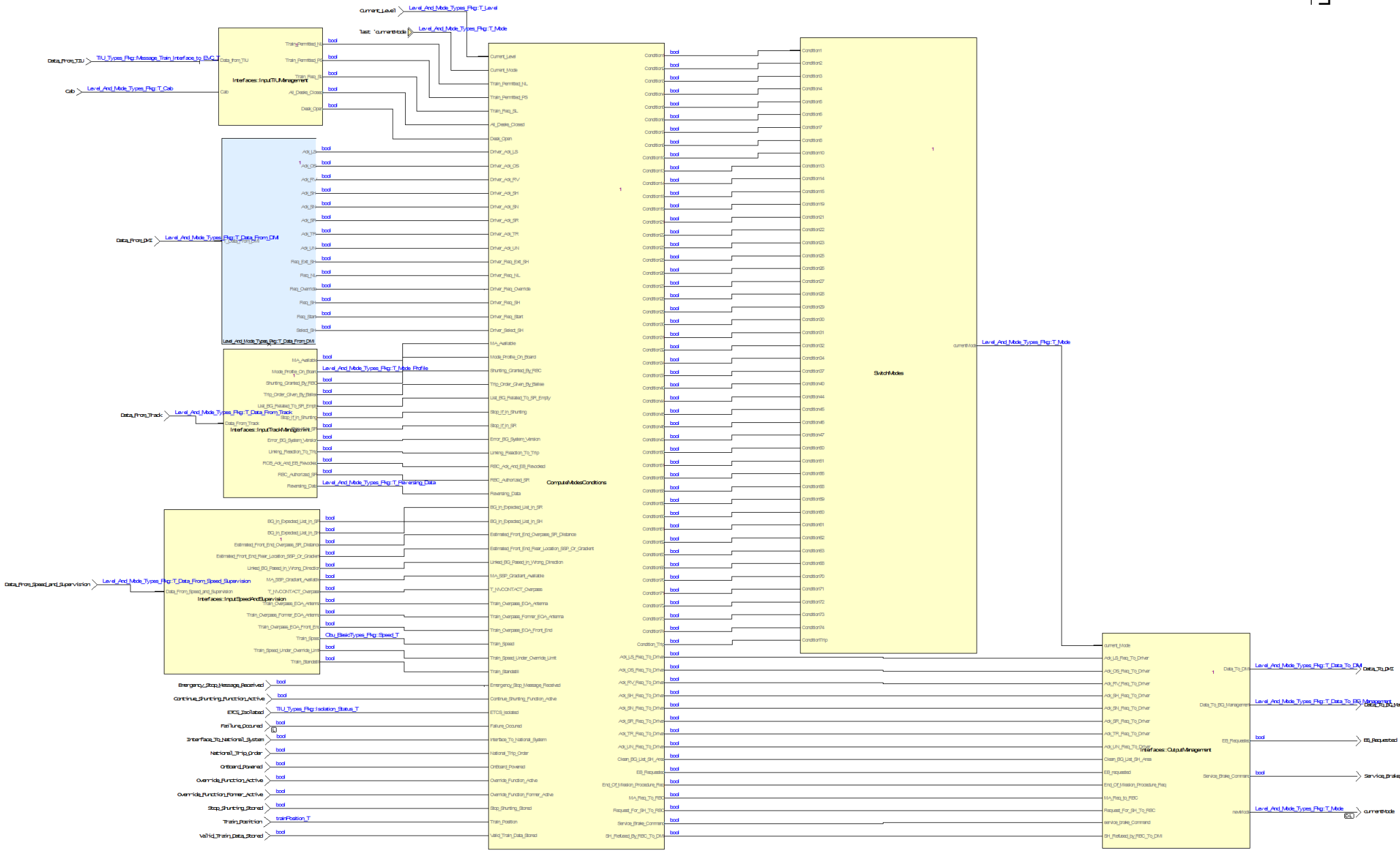


Figure 29. Modes subfubction architecture

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`

6.2.5.3 Function Check and Provide Level and Mode

Reference to the SRS or other Requirements

see [1] section 3.6.5

Short description of the functionality

checks compatibility between mode and level and provides outputs

Interface

To design

Functional Design Description

To design

Reference to the Scade Model

To design

6.2.6 Manage Radio Communication

6.2.6.1 Management of Radio Communication (*MoRC*)

Reference to the SRS

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

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.

Interface

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

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

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 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.

Reference to the Scade Model

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

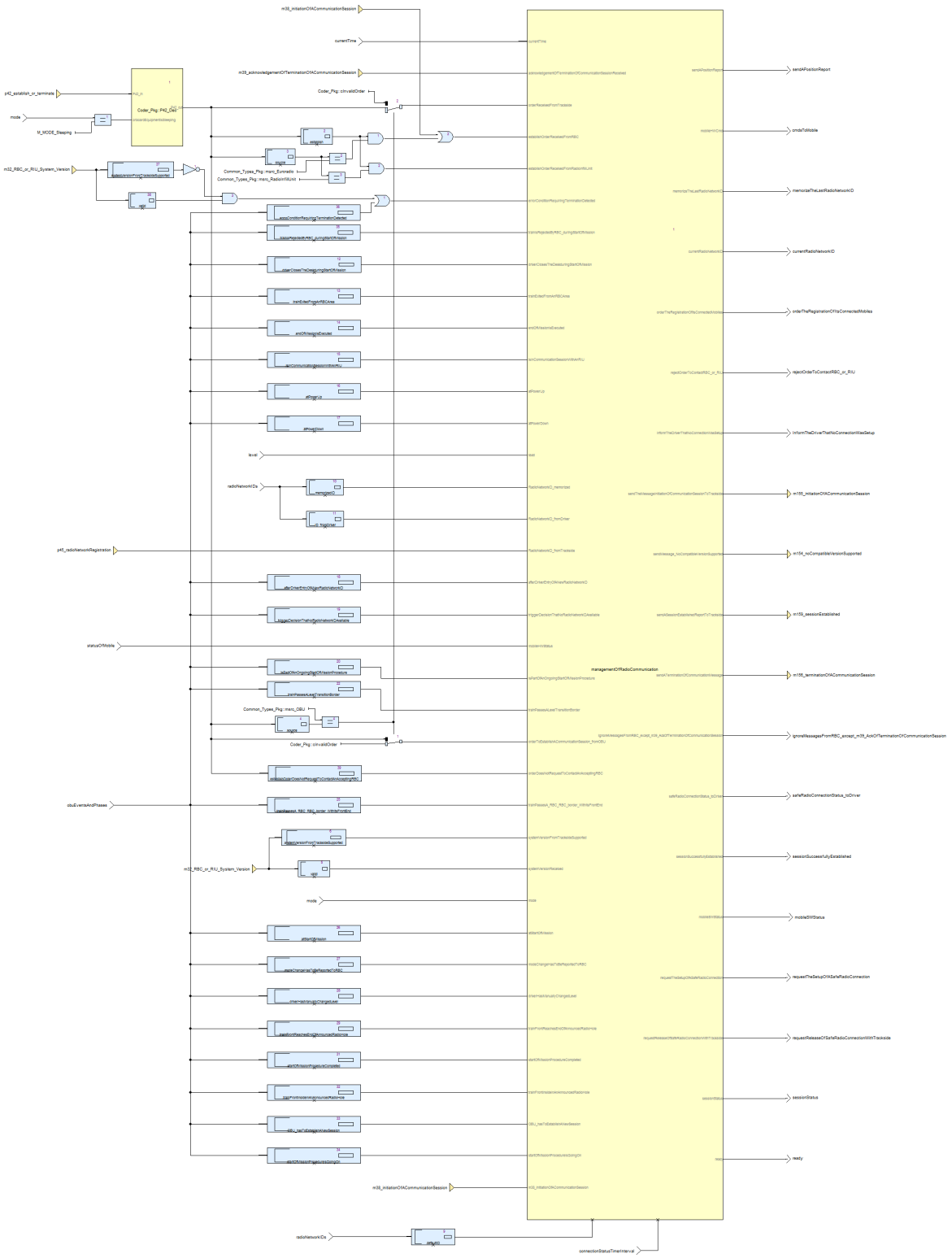


Figure 30. Main function of *MoRC*

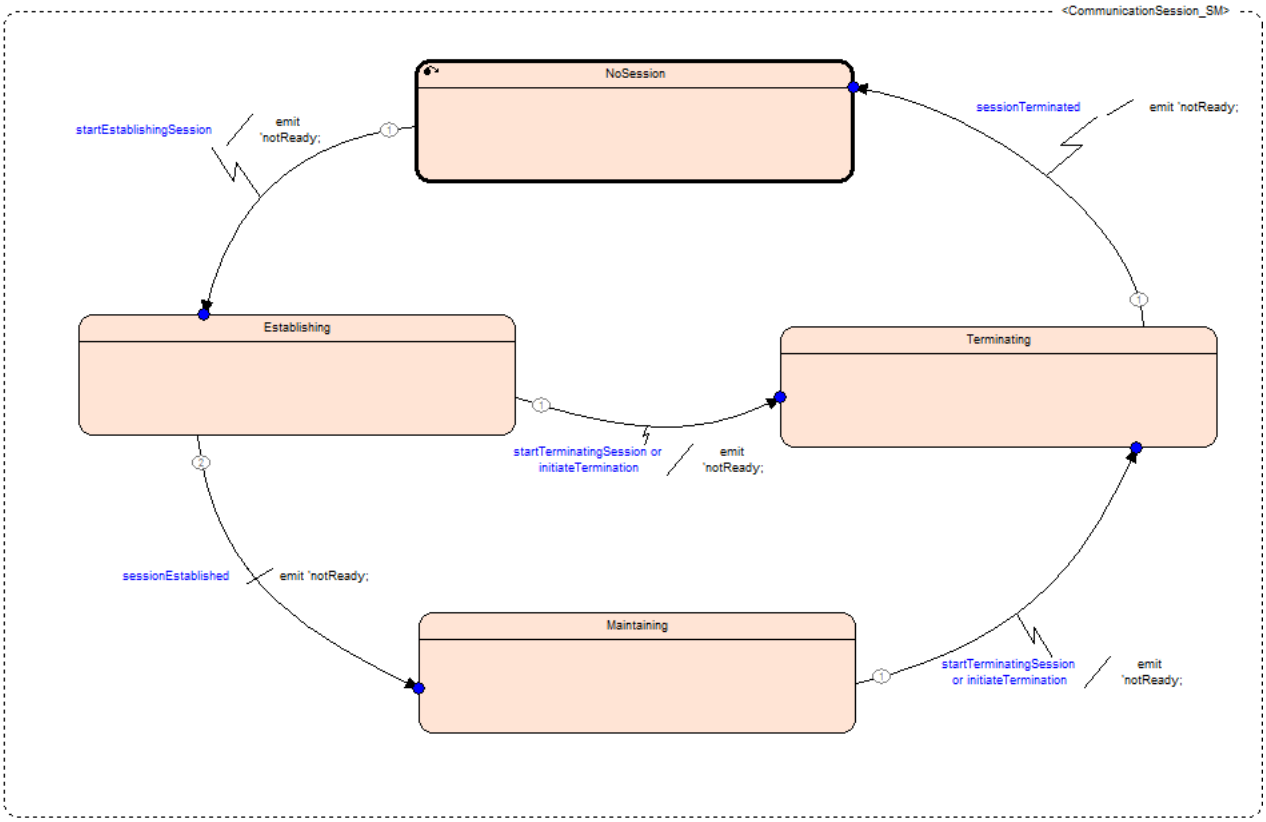


Figure 31. Implementation of session states

6.3 F3: Measure Train Movement

6.4 F4: Manage Radio Communication

6.5 F5: Manage JRU

6.6 F6: DMI Controller

6.6.1 DMI Controller

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

ERA_ERTMS_015560

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.

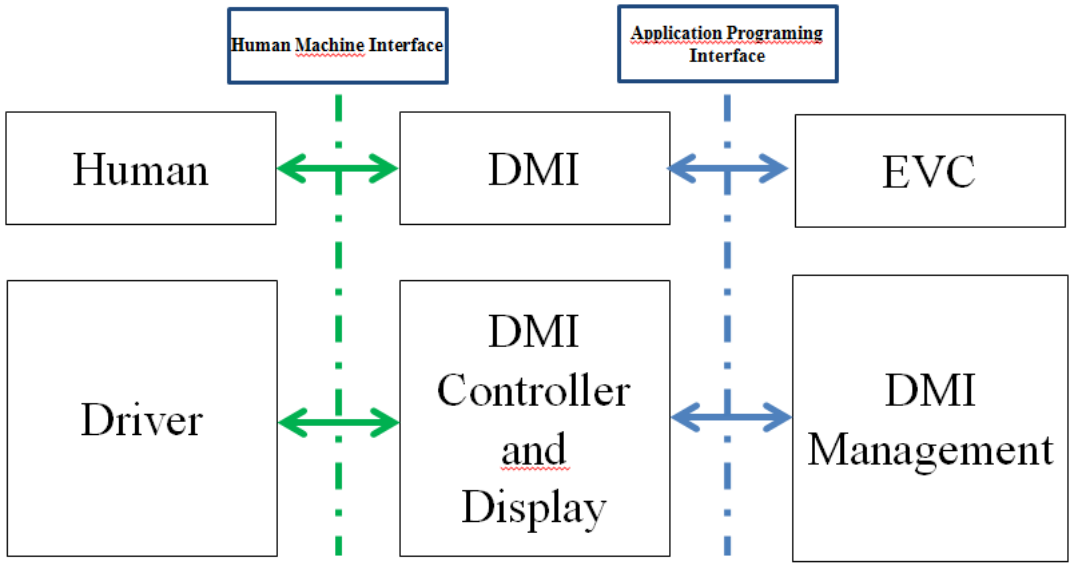


Figure 32. DMI Interfaces

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:

- 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)

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.

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

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

Functional Design Description

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. 33) triggered by a *OpenDesk* signal (from TIU). Inside the *DeskIsOpen* state there are other two state machines :*HandshakeSM* and *DynamicInfoSM* (Fig. 34).
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.

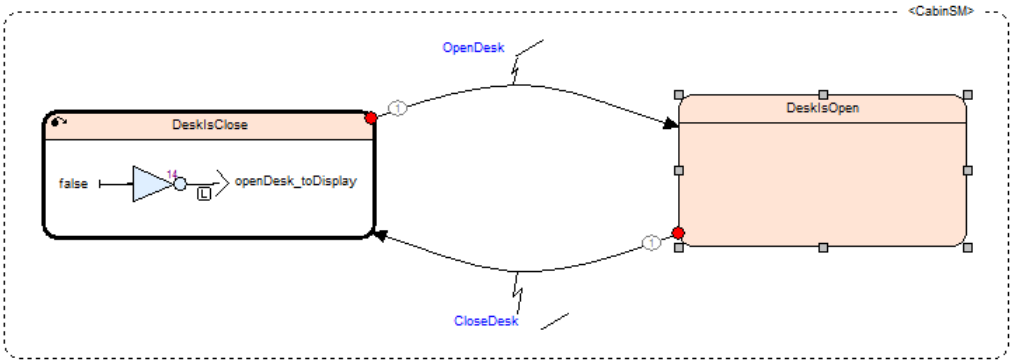


Figure 33. Cabin State Machine

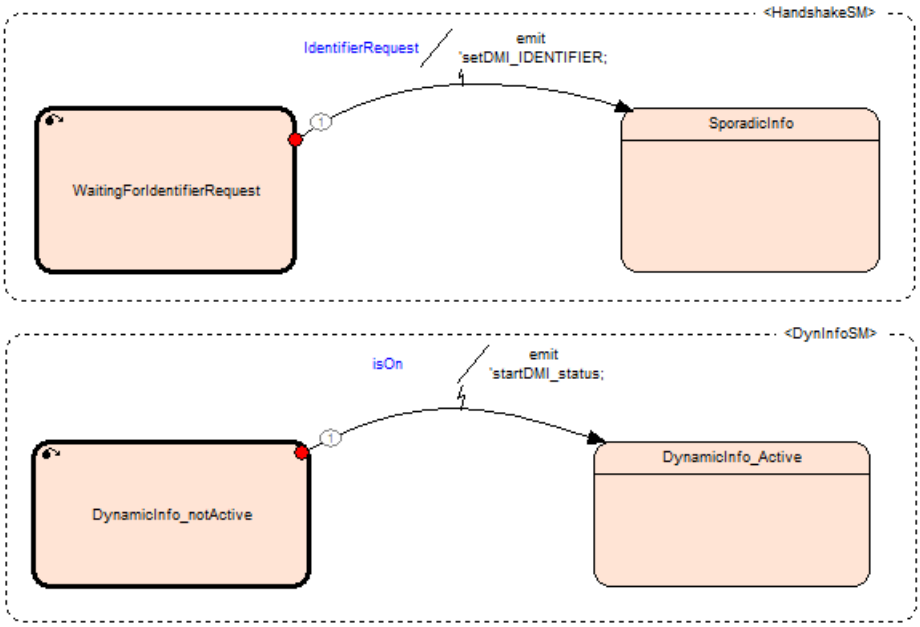


Figure 34. HandshakeSM and DynamicInfoSM State Machines

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. 35).
- **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.
- **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).

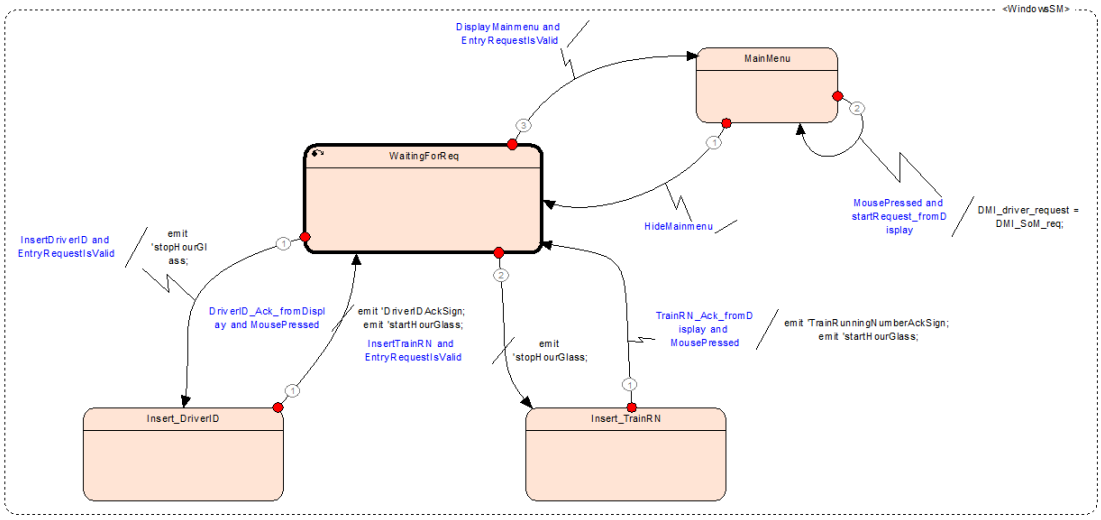


Figure 35. Windows state machine

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.

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. 36).

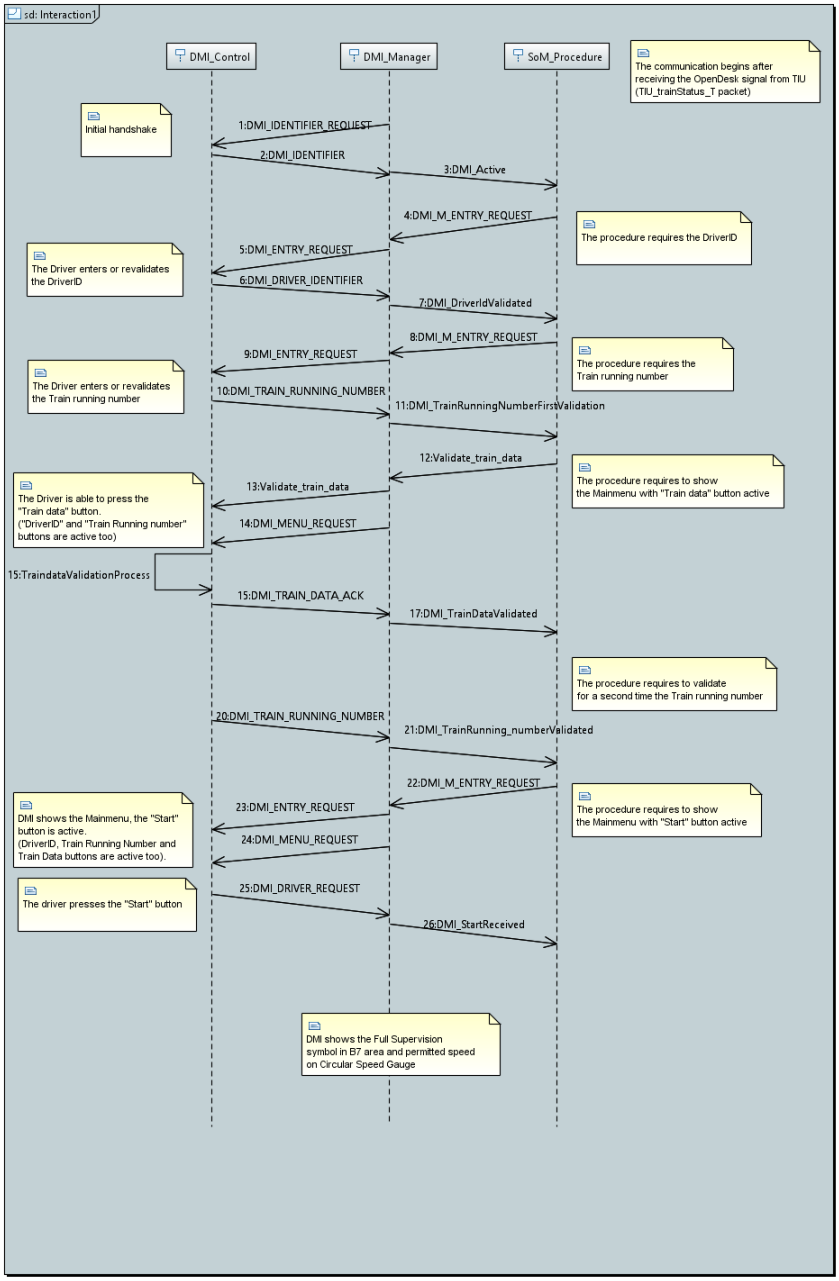


Figure 36. Sequence Diagram of start of mission scenario

Cyclic Exchange of messages 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.

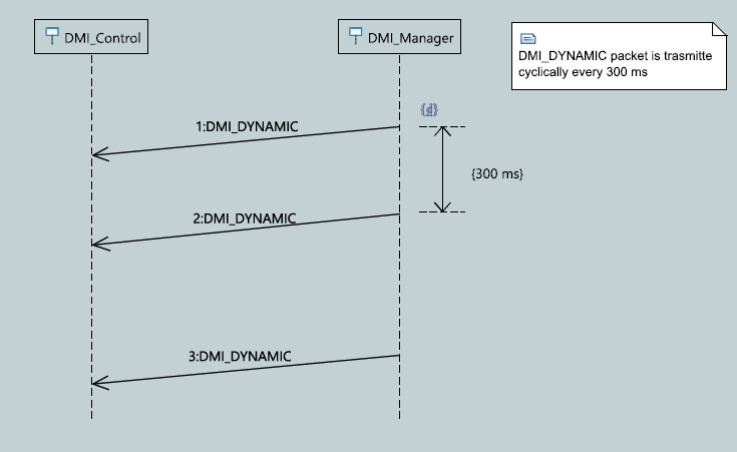


Figure 37. Sequence diagram of Dynamic data

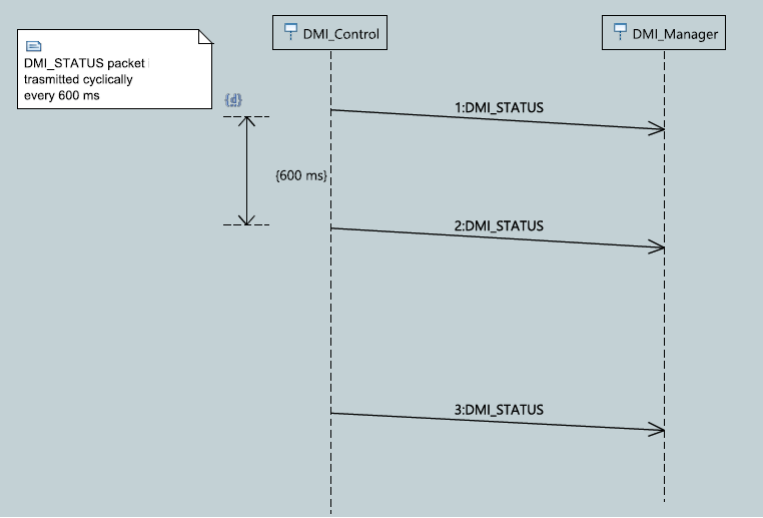


Figure 38. Sequence Diagram of DMI status

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/DMI_Control`

References

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

[2] openETCS. *openETCS SCADE model*, 2014. `https://github.com/openETCS/modeling/tree/master/model/Scade/System`.

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

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