Deliverable D2.2-Moving Block Specification Development

S. Marrone, F. Flammini, B. Janssen, R. Saddem-Yagoubi, Julie Beugin, Mohamed Ghazel, C. Seceleanu, U. Sanwal, M. Benerecetti, S. Libutti, et al.

Deliverable D2.2
Moving Block Specification Development

Project acronym:	PERFORMINGRAIL
Starting date:	01/12/2020
Duration (in months):	31
Call (part) identifier:	S2R-OC-IP2-01-2020
Grant agreement no:	101015416
Due date of deliverable:	Month 16
Actual submission date:	13^{th} September 2023
Responsible/Author:	Stefano Marrone (CINI)
Dissemination level:	PU
Status:	Issued to EU

Reviewed: no

Document history		
Revision	*Date*	*Description*
0.1	May 16^{th} 2022	First issue for internal review
1.0	May 31^{st} 2022	Submitted to EU
1.1	September 13^{th} 2023	Second issue after EU comments

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Stefano Marrone	CINI	Coordination, Chapter 1, Chapter 4, Chapter 5, Section 6.1, Section 6.2, Section 6.3.6, Section 6.4.8, Section 6.5.3, Section 6.6, Section 7.1, Section 7.2, Chapter 9, Chapter 10.
Francesco Flammini	MDH	Contributor, Chapter 9, internal review and revision
Bob Janssen	Eulynx	Section 2.2, Subsection 6.3.1, Section 5.4
Rim Saddem Julie Beugin Mohamed Ghazel	Univ Eiffel	Section 2 (Modelling phases), Section 2.3 (Adopted Languages), Subsection 6.3.3 (On Sight Movement), Subsection 6.3.4 (Loss of Train Integrity), Subsection 6.3.7 (Sweeping), Subsection 6.4.6 (TTD Management), Subsection 6.5.1 (Train Position Reporting), Subsection 6.5.2 (Integrity Information Management), Section 8.2 (IIM UPPAAL model), Section 8.3 (TPR UPPAAL model), Section 8.4 (TTD UPPAAL model), Annex A (Used Formal Languages and Notations), contribution to the “architectural specification” in Section 6.1, contribution to the “data model” in Section 6.2
Cristina Seceleanu Usman Sanwal	MDH	Subsection 6.3.6 (Points Control), Subsection 6.4.8 (Points Management), Section 8.5 (Points Management UPPAAL model)
Massimo Benerecetti Simone Libutti Elena Napolitano Fabio Mogavero Roberto Nardone Adriano Peron Luigi Starace Valeria Vittorini	CINI	Subsection 2.3.2 (Stochastic Activity Networks), Chapter 3 (ETCS-L3 modeling state-of-the-art), Subsection 6.3.2 (Normal Train Movement), Subsection 6.3.5 (Staff Responsible), Subsection 6.3.8 (Loss of Communication), Subsection 6.4.1 (Track Status Management), Subsection 6.4.2 (Reserved Status Management), Subsection 6.4.3 (Trains Management), Subsection 6.4.4 (Movement Authority Management), Subsection 6.4.5 (Route Management), contribution to the Preliminary Activity Template in subsection 7.2, Subsection 6.4.9 (Communication Management), Section 7.1 Section 8.1 (Communication Management and Trains Management UPPAAL model), Section 8.6 Movement SAN model, Annex A. Section ”Stochastic Activity Networks” , integration with WP4.

Reviewers	
Name	*Company or Institution*
Arne Borläv	Prover Technology
Miquel Garcia Fernandez	Rokubun
Rob M.P. Goverde	TU Delft, Department of Transport & Planning

Funding

This project has received funding from the Shift2Rail Joint Undertaking (JU) under grant agreement No 101015416. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the Shift2Rail JU members other than the Union.

Disclaimer

Contents

Executive Summary 7
Abbreviations and acronyms 8
1. Introduction 11
1.1 Objectives and Scope 11
1.2 About modelling 12
1.3 Relationships with other PERFORMINGRAIL deliverables 12
2. Background 14
2.1 Modelling Phases 14
2.2 The EULYNX Approach 15
2.3 Adopted Languages 16
2.3.1 UPPAAL timed automata 17
2.3.2 Stochastic Activity Networks 17
3. ETCS-L3 Modelling 19
3.1 Semi-Formal and Formal Methods in S2R IP2 projects 19
3.2 Review of the Scientific Literature 21
3.3 Comparison with PERFORMINGRAIL contribution 24
4. The Overall Modelling Process 25
5. The Specification Approach and the SysML Structure 29
5.1 The Modelling Scope 29
5.2 Specification Process Description 31
5.3 Building the SysML model 33
5.4 Integrating EULYNX DP 39
5.5 Choice of the Functional Elements 39
5.6 Tooling 40
6. The Detailed SysML Model 42
6.1 The Architectural Specification 42
6.2 The Data Model 50

6.3 The ERTMS Use Cases 55
6.3.1 Trackside Initialisation 55
6.3.2 Normal Train Movement 59
6.3.3 On Sight Movement 65
6.3.4 Loss of Train Integrity 69
6.3.5 Staff Responsible 73
6.3.6 Points Control 76
6.3.7 Sweeping 80
6.3.8 Loss of Communication 83
6.4 The Trackside Behaviour 86
6.4.1 Track Status Management 86
6.4.2 Reserved Status Management 88
6.4.3 Trains Management 90
6.4.4 Movement Authority Management 92
6.4.5 Route Management 93
6.4.6 TTD Management 94
6.4.7 Manage Temporary Speed Restrictions 100
6.4.8 Points Management 101
6.4.9 Communication Management 105
6.5 The Onboard Behaviour 106
6.5.1 Train Position Reporting 106
6.5.2 Integrity Information Management 110
6.5.3 Speed and Distance Supervision 113
6.6 The Requirement Allocation Table 117
7. The Followed Modelling Approach 121
7.1 The Formal Modelling Process 121
7.2 The Preliminary Activity Template 123
7.3 Description of the Preliminary Activities for EUCs 124
7.4 Description of the Preliminary Activities for Internal Functions 130

8. Moving Block Formal Models 141
8.1 Communication Management & Trains Management UPPAAL model 142
8.2 IIM UPPAAL model 145
8.3 TPR UPPAAL model 150
8.4 Trackside Train Detection UPPAAL model 151
8.5 Points Management UPPAAL model 153
8.6 Movement SAN model 155
9. Discussion 160
10. Conclusions 163
Bibliography 164
A. Used Formal Languages and Notations 168
A.1 Timed Automata 168
A.2 Networks of Timed Automata Extended with Variables and Broadcast Synchronization 168
A.3 Stochastic Activity Networks 169

Executive Summary

The present document constitutes the Deliverable D2.2 “Moving Block Specification Development”, which is part of Work Package 2 of the “PERformance-based Formal modelling and Optimal tRaffic Management for movING-block RAILway signaling” project (PERFORMINGRAIL). This deliverable provides both semiformal specification and formal modelling of Moving Block (MB) systems; the two activities are the output of Task 2.3 (Specifications for safe and reliable moving-block signalling) and Task 2.4 (Formal Development for moving-block and virtual coupling train operations). The approach is based on the system functional decomposition aimed at taming its complexity. ETCS Use Cases and functional components of both trackside and on-board subsystems are the most important elements in this context.

The semiformal specification activity has been accomplished by creating a SysML model starting from the specification available for the MB systems (i.e., the deliverables of previous S2R projects). The SysML model focuses on functional and behavioural aspects, representing a valid attempt to create an overall model for this signalling system. Starting from this general specification, some preliminary formal models have been defined, exploring the modelling and analysis possibility be means of some formalisms as Stochastic Activity Networks and Timed Automata. The choice of the formalisms has been made to respect the different aspects of the system to model, according to the modelling guidelines already defined in this project. These models try to cope with both quantitative and qualitative properties.

This deliverable makes the following contributions:

- Definition of a semiformal/formal modelling approach for MB systems.
- Definition of a high-level SysML model structure able to embrace the different aspects of a signalling system: from requirements, to interactions between functional components, to the behaviour specifications of the components themselves.
- Specification of a SysML model for a part of the MB systems, able to elevate the knowledge of the ETCS-L3 systems and to detect some problems in system requirements.
- Formal specification of the different functional elements (use cases and components) according to a well-defined approach aimed at detecting functional dependencies, variables, parameters, configuration information, etc.
- Definition of four different formal models, each of one focusing on some internal functions and used for the verification of some simple properties.

Some considerations are needed:

- The coverage of both SysML model and formal models regarding the MB elements is not total. The choice of the part to model has been taken according to the aspect of European Train Control System - Level 3 (ETCS-L3) highlighted in [1].
- Virtual Coupling has not been addressed in this deliverable, due to the lack of proper specification for this advanced signalling mechanism.
- Both SysML model and formal models may be subject to changes and improvements in future documents related to this work.
- The Eulynx Data Preparation (Eulynx DP) approach and has been considered in this deliverable sketching an integration strategy that could be explored in future.

Abbreviations and acronyms

Abbreviation / Acronym	Description
4SECURAIL	FORmal Methods and CSIRT for the RAILway sector
AD	Activity Diagram
ALSP	Axle Load Speed Profile
ASTRAIL	SAtellite-based Signalling and Automation SysTems on Railways along with Formal Method and Moving Block validation
ATO	Automatic Train Operation
ATP	Automatic Train Protection
BDD	Block Definition Diagram
CD	Class Diagram
CRE	Confirmed Rear End
D1.1	D1.1 - Baseline system specification and definition for Moving Block Systems
D1.2	D1.2 - Best practice, recommendations and standardisation to definition of the Railway Minimum Operational Performance Standards
D2.1	D2.1 - Modelling guidelines and Moving Block Use Cases characterization
D2.2	D2.2 - Moving Block Specification Development
D2.3	D2.3 - Moving Block Verification and Validation
DoW	Description of Work
EC	European Commission
EGNSS	European Global Navigation Satellite System (Galileo & EGNOS)
EoA	End of Authority
EoM	End of Mission
ERTMS	European Railway Traffic Management System
ETCS	European Train Control System
ETCS-L2	European Train Control System - Level 2
ETCS-L3	European Train Control System - Level 3
EU	European Union
EUC	ETCS Use Case
Eulynx DP	Eulynx Data Preparation
EVC	European Vital Computer
FM	formal methods
FS	Full Supervision
GNSS	Global Navigation Satellite System
HMI	Human-Machine Interface
IBD	Internal Block Diagram
IP2	Innovation Programme 2
LOS	Line Of Sight
MA	Movement Authority
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MARTE-DAM	Modeling and Analysis of Real-Time and Embedded Systems - Dependability Analysis and Modeling

MaxSFE	Maximum Safe Front End
MB	Moving Block
MDE	Model-Driven Engineering
minSFE	Minimum Safe Front End
MOVINGRAIL	Moving Block and Virtual Coupling Next Generations of Rail Signalling
OBU	On-Board Unit
OMG	Object Management Group
OPS	Operational Scenario
OS	On-Sight
PD	Package Diagram
PERFORMINGRAIL	PERformance-based Formal modelling and Optimal tRaffic Management for movINGblock RAILway signalling
PNs	Petri Nets
PVT	Position Velocity Time
RAT	Requirement Allocation Table
RBC	Radio Block Center
RD	Requirement Diagram
RSM	RailSystemModel
S2R	Shift2Rail
SAN	Stochastic Activity Networks
SD	Sequence Diagram
SLR	Systematic Literature Review
SM	State Machine
SMC	Stochastic Model Checking
SMD	State Machine Diagram
SoM	Start of Mission
SR	Staff Responsible
SSP	Static Speed Profile
STA	Stochastic Timed Automata
STPN	Stochastic Timed Petri Nets
SysML	System Modelling Language
T2.1	Task 2.1 - Modelling approach and guidelines
T2.2	Task 2.2 - Moving block system and scenarios characterization
T2.3	Task 2.3 - Specifications for safe and reliable moving-block signalling
T2.4	Task 2.4 - Formal Development for moving-block and virtual coupling train operations
T2.5	Task 2.5 - Verification and Validation of moving block systems
TA	Timed Automata
TIMS	Train Integrity Monitoring System
TLU	Train Localisation Unit
TMS	Traffic Management System
TPR	Train Position Report
TSA	Track Status Area
TSR	Temporary Speed Restriction
TTD	Trackside Train Detection
UC	Use Case

UCD	Use Case Diagram
UML	Unified Modelling Language
VBD	Virtual Block Detector
VBF	Virtual Block Function
VC	Virtual Coupling
V&V	Verification and Validation
VSSs	Virtual Sub-Sections
VTD	Validated Train Data
X2Rail-1	Start-up activities for Advanced Signalling and Automation Systems
X2Rail-2	Enhancing railway signalling systems based on train satellite positioning, on-board safe train integrity, formal methods approach and standard interfaces, enhancing Traffic Management System functions
X2Rail-3	Advanced Signalling, Automation and Communication System (IP2 and IP5) â€“ Prototyping the future by means of capacity increase, autonomy and flexible communication
X2Rail-5	Completion of activities for Adaptable Communication, Moving Block, Fail safe Train Localisation (including satellite), Zero on site Testing, Formal Methods and Cyber Security
XML	eXstensible Markup Language
XSD	XML Schema Definition
WP	Work Package
WP2	WP2 - Modelling and Analysis of Moving Block Specifications
WP3	WP3 - Fail Safe Train Locationing
WP4	WP4 - Integrated Moving Block architecture for safe and optimised traffic operations

1. Introduction

The Chapter outlines the main obiective of the deliverable and its structure, the adopted modelling approach, and the connections with other PERFORMINGRAIL deliverables.

1.1. Objectives and Scope

The Section outlines the goal of the deliverable and its structure. The main objective of this deliverable is to present some formal and semiformal models of ETCS-L3 moving block systems, reporting the results of the tasks Task 2.3 - Specifications for safe and reliable moving-block signalling (T2.3) and Task 2.4 - Formal Development for moving-block and virtual coupling train operations (T2.4) of the PERformance-based Formal modelling and Optimal tRaffic Management for movINGblock RAILway signalling (PERFORMINGRAIL) project. The models included in this deliverable are of two types:

- a high-level specification of ETCS-L3 in the SysML including both structural, functional and behavioural aspects;
- a set of formal models focusing on “vertical” aspects of ETCS-L3 with the aim of checking specific properties of the system.

The deliverable refines the modelling methodology introduced in [1], also sketching the mapping from high-level models to formal models and enabling fully mechanizable generation processes. This document also considers the languages and the technologies developed by Eulynx, integrating them into the proposed approach.

Since ETCS-L3 specifications are in their early phases of the lifecycle, it is important to stress that modelling in PERFORMINGRAIL has the main objective of **improving the knowledge of ETCS-L3** and of **defining an open framework enabling further refinements of the models**, also coming from different contributors.

In more detail, the deliverable is structured as follows. Chapter 2 recalls the elements already developed/surveyed in PERFORMINGRAIL that are relevant to the presentation; furthermore, essential elements of the Eulynx approach will be also recalled. Chapter 3 reports the ETCS-L3 modelling attempts already available in scientific literature and other deliverables of the project. Chapter 4 describes the methodology underlying the activities of both T2.3 and T2.4. Chapter 5 focuses on the activities of T2.3 describing the approach followed to construct the high-level SysML model. Chapter 6 describes this model in its most meaningful parts. Chapter 7 reports the modelling approach for the formal models, sketching in particular the adopted methods to approach the formal modelling activity. The different formal models are described in Chapter 8. Chapter 9 discusses the devised models, highlighting their scope and limitations. Finally, Chapter 10 concludes the deliverable while addressing the work of Task 2.5 - Verification and Validation of moving block systems (T2.5) which oversees completing PERFORMINGRAIL’s WP2 - Modelling and Analysis of Moving Block Specifications (WP2). Appendix A reports further details on the adopted formalisms used to design the formal models.

1.2. About modelling

The Section introduces the modelling aproach adopded in the deliverable. Modelling is the corpus of the methodologies and the techniques that are devoted to the abstract representations of things of the "real world", according to specific modelling principles and/or languages. It is the cornerstone task to all activities in the process of building or creating an artefact. Models are:

- a means of understanding the issues involved in designing a problem solution;
- an aid to communication between actors involved in the project, especially between the requirement analyst (a development role) and the user, as part of some deliverable;
- a component of the methods used in development activities such as the analysis of the requirements for an artefact and the design of the artefact.

To avoid communication ambiguities, one of the first steps in the modelling task is the definition of a language (i.e., a modelling notation). In the context of system and/or software engineering, both semiformal and formal languages are based on a rigorous and non-ambiguous syntax. But, while semiformal languages (as Unified Modelling Language (UML) and SysML) have not usually a formal semantics, formal languages have. The analysis of formal model can be done both by simulative methods and by analytical methods that provide precise results (i.e., not affected by statistical errors). The formal execution/evaluation of semiformal models is usually not possible.

As detailed in Chapter 4, in this deliverable a two-steps approach in modelling and specifying the ETCS-L3 systems has been adopted. Specification will be done according to the semiformal language of SysML, while the modelling phase will be done using the Timed Automata (TA) and Stochastic Activity Network (SAN) formalisms. All the formalisms have been chosen according to the results of the applicable deliverables [1–3].

1.3. Relationships with other PERFORMINGRAIL deliverables

The Section enlights the connections among the project deliverables. Fig. 1.1 depicts the dependency between D2.2 - Moving Block Specification Development (D2.2) and other PERFORMINGRAIL deliverables, highlighting the information flow for D2.2.

Specifically:

- D1.1 - Baseline system specification and definition for Moving Block Systems (D1.1) contains the description of the ETCS-L3 Use Cases that are modelled in this deliverable[1] [4];
- from D2.1 - Modelling guidelines and Moving Block Use Cases characterization (D2.1), the description of the Operational Scenario (OPS)s are considered to choose the part of the ETCS-L3 to be modelled in the present deliverable[2]. Furthermore, D2.1 contains a discussion on the most proper modelling approaches to adopt [1].

On the other hand, this deliverable will influence *directly* the following activities:

[1] It is worth highlighting the possible ambiguity between the well-known concept of UML's/SysML's Use Cases, and the European Train Control System (ETCS) Use Cases ETCS Use Case (EUC), that are groups of train control functionalities. To this aim, they are represented by different acronyms (Use Case (UC)s and EUCs, respectively).

[2] In this deliverable the Operational Scenarios are abbreviated by OPSs instead of OSs to avoid conflicts with the On-Sight (OS) mode of ETCS.

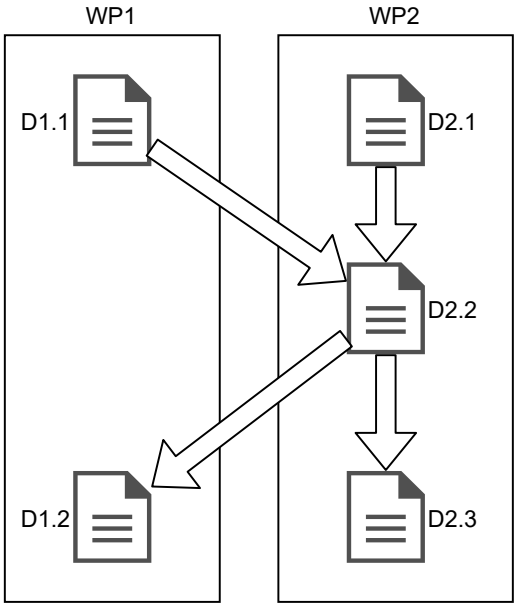


Fig. 1.1. Relationship between D2.2 and other deliverables.

- the experiences and lesson learnt in modelling ETCS-L3 will be collected — with the results collected from WP3 - Fail Safe Train Locationing (WP3) and WP4 - Integrated Moving Block architecture for safe and optimised traffic operations (WP4) — in D1.2 - Best practice, recommendations and standardisation to definition of the Railway Minimum Operational Performance Standards (D1.2);
- the formal models reported in this deliverable will be customized and analysed in D2.3 - Moving Block Verification and Validation (D2.3).

2. Background

The Chapter reports some background notions and terminology useful in the rest of the deliverable. Section 2 recalls the phases of the modelling process. In Section 2.2 the approach of EULYNX is described. In Section 2.3 the desiderable features of a specification languages for the modeling activities are discussed. Eventually, a brief description of the two adopted formalism UPPAL and Möbius are given in Section 2.3.1 and Section 2.3.2, respectively.

2.1. Modelling Phases

The Section recalls the phases adopted in the modelling process. To develop verifiable models for Moving Block (MB) systems, a methodology has been adopted, serving as a guide for the modelling activities in the framework of the PERFORMINGRAIL project. The adopted methodology, whose detailed description can be found in [1], is outlined in the following. The methodology aims at producing generic and adaptable semiformal and formal models for MB systems. In fact, the semiformal modelling activity is an intermediate step towards the achieving of workable formal models that can be used for actual verification and validation activities. Since the MB requirements are mainly written in natural language, a stepwise refinement process is adopted to elaborate the MB formal models. The methodological process includes some iterative steps towards the production of such formal models. This workflow is composed of some interrelated activities that exchange inputs/outputs. Fig. 2.1 shows a high-level view of the workflow, where the main outputs of the workflow are represented by the blue bold boxes.

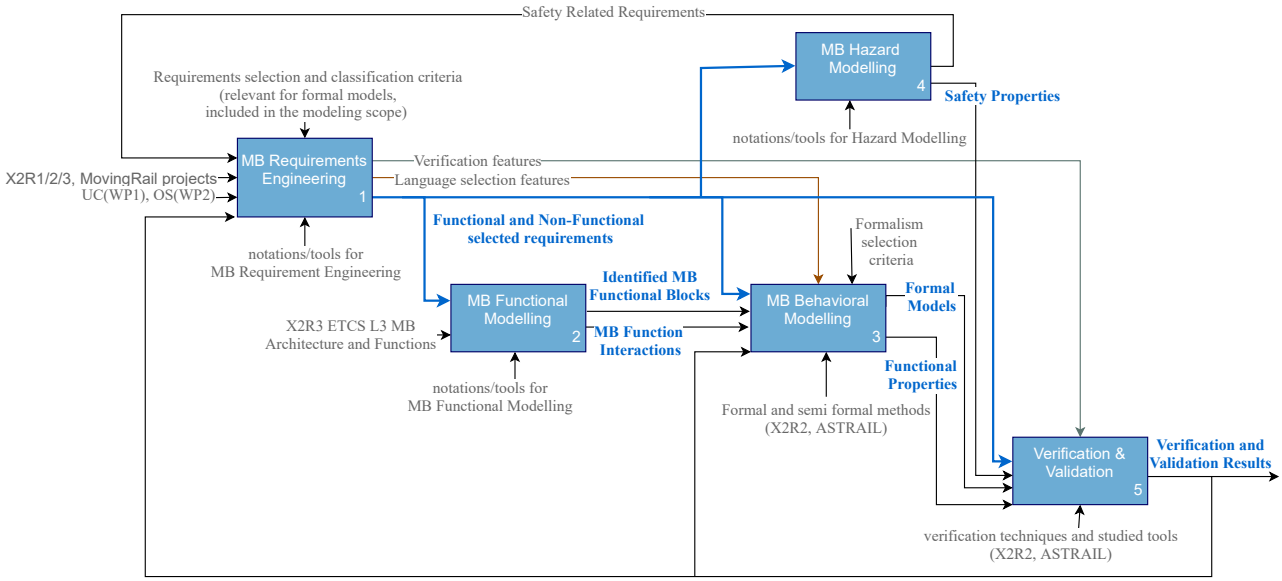


Fig. 2.1. Workflow structure.

The first step, “MB Requirement Engineering”, identifies and classifies the most relevant requirements for the MB system. The selected requirements can be represented using SysML requirement diagrams. The second step, “MB Functional Modelling”, aims at identifying, from the selected requirements, the various functions within the MB scope, as well as the

interaction among them. The identified functions can be modelled using SysML state machines and the interactions they induce are modelled by using SysML sequence diagrams. The “MB Behavioural Modelling” step develops parametrizable formal models for the various MB functions and identifies functional properties. The “Hazard Modelling” step aims at modeling the identified hazards that are related to the MB system and at specifing the safety properties to be verified. Finally, the “Verification & Validation” step is introduced to verify and validate the formal models developed in the previous step.

2.2. The EULYNX Approach

The Section contains a brief description of the EULYNX project. EULYNX is an initiative by European infrastructure managers to standardize signalling systems, and in particular the interfaces between interlocking and controlled field elements.

EULYNX also defines Eulynx Data Preparation (Eulynx DP), a standard for the exchange of signalling information. A dataset that is complete and respects this standard should allow the signalling supply industry to design, test and build a complete signalling system. As a consequence, the resulting model is highly detailed.

The Eulynx DP model is a UML class model that can be accessed at https://dataprep.eulynx.eu. It is organized as a stack of loosely coupled packages or layers. The lower layers are defined by International Union of Railways (UIC)’s RailSystemModel (RSM) (https://rsm.uic.org) and model topology, location, position and rudimentary (signalling) equipment. The network topology is a set of topological elements, typically linear elements, that represent tracks from one place (point/crossing/bufferstop) to the next. Linear elements are connected by positioned relations that inform that a train can travel from one track to the next.

A location informs where an equipment is situated on the topology. For instance, one can state that a signal has a spot location that is attached to linear element *a* at relative position 0.8729 and applies in the up-direction. A position informs where on the surface of earth, screen or paper plant, an equipment is situated.

EULYNX adds more layers of detailed information to the RSM.

Eulynx DP defines a wide range of classes representing signalling equipment, their semantics and relations. Thus, a data designer can capture detailed information stating about a signal, namely:

1. where the signal is, in the railway network and on the surface of the earth, and in which direction does it apply;
2. what message can the signal send to the train. This message is a composition of one or more signal aspects;
3. what the physical components are, ranging from the physical support to individual signal frames;
4. which controllers, including interlocking, switch the signal.

Route information informs about:

1. the route path, from entry to exit signal;
2. properties of the route such as speed and type of route, e.g., shunting or normal operation.

An EULYNX dataset is an eXstensible Markup Language (XML) file that validates against a

set of Eulynx DP XML Schema Definitions (XSDs). These XSDs were automatically generated from the UML model.

Whereas the Eulynx DP use case describes the exchange of information for building a signalling system, the level of detail and scope will more than likely satisfy other use cases.

In fact, use cases such as Automatic Train Operation (ATO) and MB need input for algorithms that compute distance to run and braking curves. RSM and Eulynx DP define alignment, gradient profiles, cant and speed profiles. This information, combined with stopping points defined by signals platforms and routes, provides the needed input for accurately simulating train dynamics.

2.3. Adopted Languages

The Section describes the criteria used for choosing the two specification frameworks UPPAAL and Möbius. In the framework of PERFORMINGRAIL modelling activities, several languages and notations have been adopted to specify the formal behavioural models of the various targeted functions involved in the MB operation. In fact, the choice of the languages was guided by a number of aspects as detailed below:

- the features to be considered in the modelling activity: for a given function, depending on the relevant corresponding specifications and depending on the properties to be checked, a number of features to be inferred. Namely, relevant features may include the following:
  - (quantitative) temporal aspects to be considered, e.g. timeouts for an event to occur, or a duration for an action to be completed;
  - probabilistic aspects, such as, for instance when several possible evolutions may be taken from a given state which can be characterized by some likelihood value;
  - stochastic aspects, such as, for instance when the occurrence time of some given event can be characterized using some stochastic distributions.
- the availability of supporting tools: although some notations and languages may show interesting features, the lack of dedicated and sufficiently mature tools necessarily hampers the adoption of such notations. In this respect, since implementing supporting tools in the framework of the project is a risky task, it is important that the chosen languages are supported by available efficient and mature tools. The issue of tool licence is also important; namely, whether these tools are free or they require a licence is also relevant.
- experience and mastering of the modelling tools: the various teams that are involved in the formal modelling activities have previous experience with different formal methods (FM) tools. The choice of the tool to be adopted within the PERFORMINGRAIL project is necessarily impacted by these previous experiences and the level of expertise gained on the facilities offered by the tools. Another aspect that may also impact the choice of a new tool is the effort needed to master the tool facilities in comparison with the additional benefits the new tool could provide with respect to tools on which the partner has a previous relevant expertise.

The tools used in this deliverable for formal modelling activities are UPPAAL[1] and Möbius[2].

[1] https://uppaal.org/
[2] https://www.mobius.illinois.edu/

As for UPPAAL, the extended timed automata provided by this tool offer rich semantics that allow coping with various modelling features that are relevant for the modelling activities of the project. Moreover, UPPAAL offers interesting facilities in terms of model edition, generation, formal verification and simulation. As for Möbius, it is adopted in this work for performability analysis, in particular for the facilities it offers in terms of depicting stochastic behaviour with SANs. The previous experience of the teams in mastering these modelling tools was also a criterion supporting the choice.

Even if the modelling language provided by UPPAAL and Möbius have already been discussed in D2.1, to ensure self-containment of the deliverable, their main features are briefly summarized in the following. Full descriptions of these frameworks are provided in the Appendix A.

2.3.1. UPPAAL timed automata

The Section reports a brief description of the specification framework UPPALL. TA are a variant of state-transition machines that are extended with real-time features using clocks. TA use a dense-time model where a clock variable evaluates to a real number. All the clocks progress synchronously. The model-checker UPPAAL is based on the theory of timed automata, and extends the TA formalism with various features such as bounded integer variables, urgency and communication channels. In UPPAAL, the system to be investigated is modelled as a network of timed automata that operate in parallel. The bounded discrete variables are part of the state and can be handled as in programming languages. Namely, they can be read, written, and are subject to common arithmetic operations. A system's state is defined by the locations of all automata, the clock values, and the values of the discrete variables. Every automaton may fire a transition independently with respect to the other automata, or by synchronizing with another (resp. several) automaton (resp. automata) using binary (resp. broadcast) communication channels. The firing of a transition or the progress of time leads to a new state.

In fact, UPPAAL is an integrated environment for modelling, validation and verification of real-time systems. Verification can be performed by exploiting model-checking techniques with properties expressed in a suitable fragment of a branching time temporal logic. UPPAAL also allows simulation, which can be used to provide useful insights into the working of the considered real-time system. Simulation can also be advantageously used during the model debug phases. Moreover, some extensions make it possible to consider stochastic aspects, while in this case, the verification can be performed exploiting a Stochastic Model Checking (SMC). A more detailed description of timed automata and networks of timed automata is reported in Appendix A.

2.3.2. Stochastic Activity Networks

The Section reports a brief description of the specification framework Möbius. SANs are a variant of stochastic Petri Nets (PNs) that have been introduced to facilitate performability evaluation of large systems. A SAN model comprises four primitives: places, activities, input gates and output gates.

Places are used to represent local system states, e.g., conditions or situations as in PNs, but differently from PNs, they can be of two types: ordinary and extended. Ordinary places are as in PNs; they can contain tokens, and the marking of an ordinary place is defined as the number of tokens it contains. Extended places are associated to a variable that can be of any type: atomic, array, matrix or a data structure. The value of the associated variable

is the marking of the extended place. These variables cannot be removed or added to the extended places as the tokens in the ordinary places, but their value can only be read or changed.

Activities are as transitions in PNs: they can be either timed or instantaneous. Timed activities represent time-consuming activities whose duration has an impact on the system performance, they can have either a deterministic or a stochastic duration. Instantaneous activities are used to model the verification of logical conditions or the completion of activities that require negligible amounts of time. Each activity has a non-zero number of cases, where a *Case* denotes a possible action that may be taken upon the completion of the activity.

Gates are introduced to allow for greater flexibility in defining enabling and completion rules.

Input gates are used to control the activation of the activities, and the effect of their completion on their input places. The input gate connected to an activity defines the enabling condition of that activity (by specifying an *enabling predicate*) and how the new marking of its input places has to be evaluated after the activity completion (by specifying an *input function*). Therefore, when connected to an extended place, an input gate allows the reading of its associated variable.

Output gates define the marking change on the output places of an activity when it completes through the specification of an *output function*. Therefore, when connected to an extended place, an output gate allows the writing of its associated variable.

SANs are formally defined in [5]. In the Appendix, an example is presented to provide more details on SAN modelling and introduce some features of the Möbius tool [6], that is being used in PERFORMINGRAIL to develop a performability model of the MB system. In particular, Möbius allows for compositional modelling, so enabling the development of a set of sub-models and the adoption of a modular approach in system modelling. More details regarding SANs are provided in Appendix A.

3. ETCS-L3 Modelling

The adoption of semiformal and formal methods has always been pursued in railway signalling to take advantage of several benefits they can bring by leveraging on formal specification, modelling, development, and verification of systems. Such benefits include, among others, enhanced safety and security, improvement of specifications, designs and architectures, formal evidence of properties in certification processes, standardization of protocols, and reduced costs. Despite the potential advantages, the take-up of formal methods in industrial settings is still limited.

A recent report [7] presents the current status of adoption of formal methods in signalling, including the work conducted within the S2R programme which opened specific streams of research on the application of formal and semiformal methods. The modelling activities carried out in PERFORMINGRAIL WP2 and reported in the present deliverable consider both the previous work conducted in S2R IP2 projects and the results available in the scientific literature. This chapter provides a review and analysis of the current state-of-the-art in ETCS-L3 modelling.

3.1. Semi-Formal and Formal Methods in S2R IP2 projects

This section gives a bird's-eye view of the work conducted in IP2 on semiformal and formal modelling, based on the *available public documents*. The main source for the information contained in this section are the S2R IP2 project pages as well as the public websites of the projects. The following IP2 projects address formal methods for signalling systems within the TD2.7 (Formal methods and standardisation for smart signalling systems): Enhancing railway signalling systems based on train satellite positioning, on-board safe train integrity, formal methods approach and standard interfaces, enhancing Traffic Management System functions (X2Rail-2) [8], SAtellite-based Signalling and Automation SysTems on Railways along with Formal Method and Moving Block validation (ASTRAIL) [9], FORmal Methods and CSIRT for the RAILway sector (4SECURAIL) [10], Completion of activities for Adaptable Communication, Moving Block, Fail safe Train Localisation (including satellite), Zero on site Testing, Formal Methods and Cyber Security (X2Rail-5) [11] and PERFORMINGRAIL [12].

X2Rail-2 proposed a classification of formal methods for the development and Verification and Validation (V&V) of railway signalling systems, providing a framework for the following activities in S2R [13].

ASTRAIL was a complementary project of X2Rail-2. The objective of the FM related activities was twofold: i) benchmarking and ranking formal and semi/formal methods to identify the most suitable languages and tools to be applied for the development of railway systems, and in particular of signalling systems; ii) experimenting with the usage of a set of selected formal methods through the modelling of the moving-block system and validating the usage of the selected formal methods by integrating the moving-block model with automated driving technologies. In particular, ASTRAIL also considered the Global Navigation Satellite System (GNSS) in combination with moving-block.

As for the first objective, an in-depth analysis was conducted by ASTRAIL. One of the results of the Systematic Literature Review (SLR) carried out during the project is that UML is the most common semiformal language used for the high-level specification of railway

systems; whereas from the survey conducted among practitioners, the formal tools and methods most mentioned by industrial or academic users are: Simulink[1] and Stateflow[2], SCADE[3], CPN tools[4], Monte Carlo Simulation, NuSMV[5] and Spin[6]. The main quality aspects of a (semi-)formal method/tool should have to be concretely and effectively adopted in the railway industry, according to relevant stakeholders, are maturity and ease of learning. Some of the findings from the analysis phase performed by ASTRAIL are summarized in [14].

As for the second objective, several semi/formal and formal models have been developed with different purposes, along two research streams.

a) The logical functionality of the MB signalling system without trackside train detection has been modelled as an input for the following hazard analysis. The functional model includes eight UML State Machine Diagrams (SMDs) in eight regions, each of them representing a function or process performed in the system [2]. This model aims at visualising the workflow, structure, and behaviour of the system, relationships and interaction of its elements, and it is used as a basis for the modelling activities in the second research stream, briefly summarized at point b). In the following, this model is called *MB model*.

A set of system scenarios based on the MB architecture at general functional level have also been defined and modelled with UML SMD. This modelling activity mostly refers to the Start of Mission (SoM) EUC, and on a transition scenario from Full Supervision (FS) to TRIP mode. Specifically:

1. **SoM when the train position is valid** and GNSS has the required availability (i.e., Line Of Sight (LOS)). This scenario corresponds to the normal operation.
2. **SoM when the train position is invalid/unknown** under the following conditions: the position cannot be acquired from the Train Localisation Unit (TLU) (erroneous or unavailable position), the train integrity is confirmed, the communication session with the Radio Block Center (RBC) is correctly set, and the train information is correct. This scenario corresponds to the degraded operation where the train position is invalid or unknown and GNSS positioning information is out of range.
3. **SoM when the train integrity is not confirmed** under the following conditions: the position can be correctly acquired from the location unit, the train integrity is not confirmed, the communication session with the RBC is correctly set, and the train information is correct. This scenario corresponds to the degraded operation where the train integrity is not confirmed during the SoM procedure, nevertheless the position of the train can be acquired.
4. **Transition from FS to TRIP if the train position is invalid/unknown**. This scenario corresponds to the degraded operation where during the mission (train in FS) either train integrity is not confirmed, or train position becomes invalid/unknown. The provided sequence diagram covers the case in which the train position is invalid/unknown.

The ASTRAIL *MB model* and the sequence charts listed above are available and illustrated in [2]. All the above scenarios are said to be valid for each railway profile.

[1] https://it.mathworks.com/products/simulink.html
[2] https://it.mathworks.com/products/stateflow.html
[3] https://www.ansys.com/products/embedded-software/ansys-scade-suite
[4] https://cpntools.org/
[5] https://nusmv.fbk.eu/
[6] https://spinroot.com/spin/whatispin.html

b) Selected languages and tools have been applied to the *MB model* enabling experimental parallel application of multiple techniques/tools to evaluate and compare their usability and applicability in the domain. The models produced with eight different formal techniques were used to provide a usability assessment. The industrial users indicated Simulink/Stateflow and SCADE as the most usable tools. The different formalizations based on the UML state machine diagrams of the *MB model* include UML State Machines, Event B State Machines, and Stochastic Timed Automata (STA) modelled in UPPAAL. Some of the results of this research have been described in [15,16]. In particular, the UPPAAL models are also presented in [17–19].

One of the objectives of 4SECURail was to perform a cost-benefit analysis for the adoption of formal methods by prototyping a formal method Demonstrator to be exercised with a selected case study. Specifically, the goal was to study the possible impact of the introduction of formal methods **in the system requirements definition process**. The current release of the formal method demonstrator is described in [3]. It consists of a *process* for the demonstration and evaluation of techniques based on formal methods. Starting from the initial natural language requirements, several steps are cyclically performed, until the output of the analysis process becomes usable. The main steps are here sketchily reported:

1. development of a standardised description of the systems specification based on UML/SysML diagrams, in particular state machines and sequence diagrams (behavioural diagrams), according to the indications from EULYNX and the X2Rail projects (from natural language to semiformal language);
2. (automatic) transformation of the standard UML/SysML description into verifiable formal models (from semiformal to formal language(s)) and verification of the properties of interest;
3. rigorous natural language rewriting of the requirements (from formal, to semiformal, to natural language).

The process ends when several conditions are fulfilled, i.e., the complete system is entirely designed, all the properties of interest are satisfied, etc. The formal method demonstrator is introduced in [20, 21] and its final release is presented in [3]. The signalling case study selected for experimenting the demonstrator in 4SECURail is the RBC/RBC Handover.

3.2. Review of the Scientific Literature

In the scientific literature, few works specifically address the application of formal and semiformal methods to ETCS-L3, Table 3.1 and Table 3.2 summarize the current state-of-the art for the works whose subjects are RBC, TLU, On-Board Unit (OBU), Virtual Sub-Sections (VSSs) or other relevant subsystems.

A number of the papers addressing full moving block describe results from ASTRAIL and 4SECURAIL projects. Among them, [17], [18], [22] and [19] address a ETCS-L3 moving block scenario related to Movement Authority (MA) communication. In [17], [18] the system behaviour is modelled by Simulink and UPPAAL, and then analysed by using statistical model checking. The first work focuses on the evaluation of the probability of reaching a safe state (i.e., a quantitative property), the second work focuses on fine-tuning communication parameters which has impact on the system reliability. In [22] the authors combine statistical model checking with Reinforcement Learning to synthesise a safe strategy guaranteeing that the train does not exceed the MA. In [19] a refinement of the UPPAAL model from [18]

Ref.	L3	Goal	System	Method(s)	Tool(s)	Funding
[17]	MB	Safety	TLU,OBU,RBC	Stateflow TA	Simulink UPPAAL SMC	S2R, Regional
[18]	MB	Parameters tuning	TLU,OBU,RBC	Stateflow TA	Simulink UPPAAL SMC	S2R, Regional
[22]	MB	Safety	TLU,OBU,RBC	TA	UPPAAL STRATEGO	S2R, National
[19]	MB	Multiple trains Parameters tuning	TLU,OBU,RBC	TA	UPPAAL SMC	S2R
[23]	MB	Availability Evaluation	Communication	Stochastic Timed Petri Nets (STPNs)	ORIS	-
[24]	MB	Performability Evaluation	Communication	STPNs	ORIS	-
[25]	MB	Safety Evaluation	GNSS Localization	TA	UPPAAL	-
[26]	MB	Safety and Performance Evaluation	GNSS Localization	TA	UPPAAL SMC	-

Table 3.1: Scientific papers addressing ETCS-L3 formal modelling (part 1)

is presented, which models a scenario with more trains on a line.

STPN models are proposed in [23] and [24] to perform quantitative analyses of communication failures in ETCS-L3 (and consequent emergency stops). A model-driven, modular approach is described in [25] and [26] for safety and performance evaluation of GNSS-based railway localisation function. This approach also exploits TA, Model checking and SMC with UPPAAL. Modularity allows considering a variety of system architectures in different operational context.

Few works have also been published in the literature that specifically cope with formal and semiformal modelling of Virtual Coupling. In [22] the authors mention the possibility of further reducing the headways between trains. Virtual Coupling (VC) is explicitly investigated in [39] and [40]. The work described in [39] presents a SAN model for the quantitative evaluation of capacity increase in VC scenarios. The study is based on a fine-grained discretization of the railway track that allows to represent the moving block concept by a step-by-step movement of the trains.

Most of the above publications on full MB can be easily clustered by authors and their underlying research project. On the contrary, more works from diverse groups of authors focus on Hybrid ETCS-L3. This different attention paid by the scientific community to the two signalling systems seems due to two concomitant circumstances: 1) the availability of a public document describing in detail the Hybrid L3 principles from the European economic interest grouping (EEIG) European Railway Traffic Management System (ERTMS) Users Group, and 2) the case study track of the $6th$ ABZ conference on state-based and machine-based formal methods, specifically addressing Hybrid ERTMS/ETCS-L3. Such circumstances put a special attention on the Hybrid system, but on the other hand, all these works focus on the management of the fixed VSS that was the scope of the proposed case study. These studies

Ref.	L3	Goal	System	Method(s)	Tool(s)	Funding
[27]	Hybrid	V&V	VSS	Electrum	Analyzer	ERDF, National
[28]	Hybrid	V&V	VSS	Promela	Spin	-
[29]	Hybrid	V&V, Animation	VSS	B-Method (Model Checking (MC))	ProB	-
[30]	Hybrid	V&V	VSS	Event-B	Rodin	Res. Council of Canada, National
[31]	Hybrid	V&V	VSS	Event-B	Rodin	National
[32]	Hybrid	V&V	VSS	Event-B	Rodin	-
[33, 34]	Hybrid	Functional Safety	VSS	Event-B, iUML-B	ProB	ECSEL JU
[35]	Hybrid	Verification and Testing	RBC (MA), TTD, VSS	Event-B, iUML-B	Rodin, MoMuT	ECSEL JU, BMVIT
[36]	Hybrid	Requirements Specification, and scenarios validation	OBU TPR), RBC (TTD), VSS	Abstract State Machines	ASMETA toolset	-
[37]	Hybrid	Safety	Train, TTD, RBC (MA), VSS	UML Event-B	ProB, B4MSecure	IRT Railenium
[38]	Hybrid	Verification	TTD, Trackside, VSS	mCRL2	mCRL2 Toolset	-
[39]	VC	Capacity Evaluation	Trackside OBU	SAN	Mobius	S2R
[40]	VC	Safety, Performance, Functional Evaluation	V2V Interaction and Communication	SysML/CPN	CPN-Tool	National

Table 3.2: Scientific papers addressing ETCS-L3 formal modelling (part 2)

follow two main approaches: The works described in [27], [28] and [29] model the specification as faithfully as possible, checking and animating the specification for verification and validation purposes in some operational scenarios. Electrum (an extension of Alloy) and the Analyzer model checker are used in [27], Promela and the Spin model checker are used in [28] while in [29] the introduction of a Virtual Block Function (VBF) computing the occupation states of the virtual subsections is described. The VBF is implemented as a formal B model executed at runtime using ProB, and it has been used in a field demonstration.

The studies from [30] to [34] make use of abstraction and refinement, regardless of scenarios. [30–32] use Event-B and theorem proving to verify the main principles, considering a subset of the requirements. In [33, 34] a systematic modelling method based on the state and class diagrams of iUML-B and Event-B is proposed, which uses abstraction to verify the principle of movement authority and develop the Virtual Block Detector (VBD) component according to a stepwise refinement approach, verifying that it preserves the safety proper-

ties. Two interesting works also consider the presence of the Trackside Train Detection (TTD) system [37, 38]. Finally, a few pioneer papers describe research on VC, e.g., in [39, 40].

3.3. Comparison with PERFORMINGRAIL contribution

The activities conducted in WP2 aim at modelling the behaviour of the MB system as defined in the Deliverables D4.2 part 2 and part 3 (Moving Block Specifications – System Definition and System Specification [41, 42]). These documents were not available before the end of 2020; therefore the work done in previous projects, and the modelling activity described in the scientific literature so far, had necessarily to refer to partial knowledge of the system and to formulate hypotheses about its behaviour. In addition, the works presented in the literature are intended to model specific functionalities or scenarios of the system. In contrast, the work described in this deliverable has taken up the challenge of modelling a large part of the system's behaviour, addressing a number of components and features that have never been considered at the same time in previous modelling attempts. Hence, regarding the works currently reported in the literature, the modelling effort being performed in PERFORMINGRAIL WP2 is based on a preliminary definition of the system specifications; on the other hand, it had to deal with the difficulties arising from the complexity of the system, and the need to make up for the lack of some information necessary for such extensive modelling. This required the definition of a suitable model development methodology, based on software engineering principles, more complex and articulated from that normally required for the development of models related to specific features or scenarios.

In doing that, the continuity with the previous S2R projects has been preserved, in particular, some modelling choices from ASTRAIL and the adherence (as possible) to the formal methods demonstrator developed by 4SECURAIL. Among the aspects pointed out by 4SECURAIL, the following are especially relevant for the work conducted in PERFORMINGRAIL [3, 20, 21]:

- The importance of UML/SysML artefacts to effectively complement the specification of system requirements.
- The importance of formal methods diversity, i.e., the development of formal models of different types.
- The importance of automated (mechanical) generation of the formal models.

The modelling approach taken in WP2 is based on these key points, in particular the extensive usage of sequence and state machine diagrams allowed to develop formal models for which automatable transformations from the SysML artefacts can be defined.

4. The Overall Modelling Process

This chapter is devoted to the description of the modelling process adopted in T2.3 and T2.4. The described modelling activities are distinct, as they focus on two different aspects: *high-level specifications* and *developing formal models*.

The process adopted in these activities targets accomplishing the following objectives:

- interoperability: the languages and tools used in the process require a high level of interoperability, and must adhere to a well-known and adopted standard. The objective is to foster the reuse of the produced models by the scientific community;
- integration with EULYNX: the results of the modelling activities are integrable with the Eulynx DP;
- automatable generation process: it is possible to mechanize the generation of formal models from the high-level specification of ETCS-L3;
- incremental design: the modelling approach reflects a modular design paradigm that facilitates extending existing models with more refined versions;
- early-stage level: the results of the modelling approach reflect the status of maturity of specification and standardization of ETCS-L3 systems;
- demonstration-oriented: even if the final goal of any modelling activity is to define an abstraction of a system, in order to be able to infer several system properties from model analyses, modelling (and analysing) the entire ETCS-L3 system is a hard task. To this end, the conducted activities have been oriented to demonstrate the feasibility of the approach by eliciting selected and relevant validation scenarios (see Chapter 5).

To pursue the above objectives, an “agile modelling” approach has been adopted in this project. A very high-level overview of this approach is depicted by the UML’s UC in Fig. 4.1.

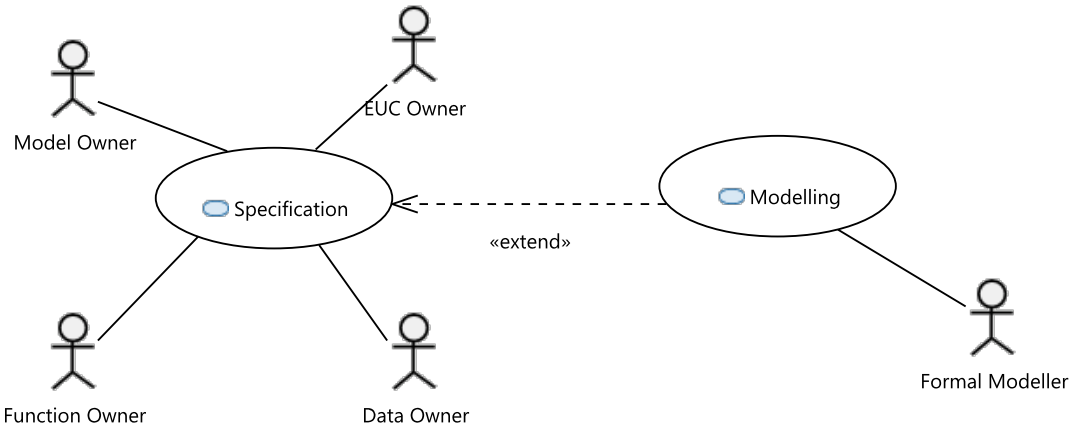


Fig. 4.1. High-level view of the modelling approach.

Two distinct sub-activities are reported: *Specification*, that is responsible for defining the high-level model (T2.3), and *Modelling*, for definition of formal models (T2.4), which depends on the first sub-activity, as represented by the $\ll extend \gg$ relationship. This diagram is also important, as it details the different roles participating in the process and contributing to the

Work Package (WP). On the one hand, the Model Owner, the Data Owner, EUC Owners, Function Owners specify and update a specific part of the high-level model, respectively (see Table 4.1 for details). On the other hand, a Formal Modeller defines and updates one of the formal models developed in T2.4.

Table 4.1: Roles and responsibilities

Role	**Responsibility**
Model Owner	To maintain and update the entire model, and integrate the different contributions. To specify the parts of the model that are common to all of the sub-models (i.e., the architecture, the requirement structure, etc.).
Data Owner	To define and maintain the representation of the data for train and trackside.
EUC Owners	To specify the EUC with the specific purpose of highlighting the interactions among the functional components.
Function Owners	To specify the inner behaviours of the functionalities of both train and trackside.
Formal Modellers	To model specific internal functions according to a concrete formalism.

Fig. 4.2 details the interactions between the mentioned two sub-activities, by illustrating the iterative nature of the “agile approach” followed in WP2.

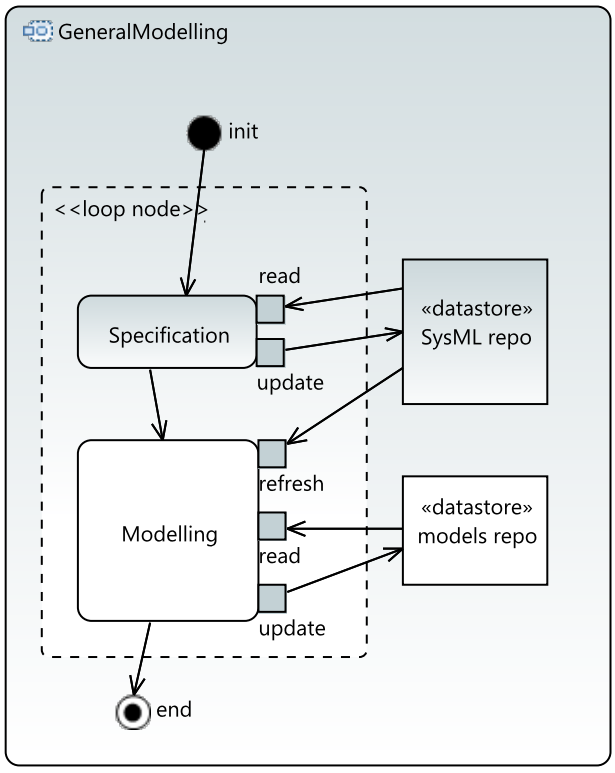


Fig. 4.2. The specification and modelling approach.

Through this UML's Activity Diagram (AD), the *Specification* and the *Modelling* activities are carried out in a loop. Each iteration of this loop represents a different sprint of such kinds of approaches. The *Specification* activity updates (using two ports) the *SysML repo* datastore (e.g., concretely, a repository of high-level models), while the *Modelling* one updates another repository, the *models repo*. The $\ll extend \gg$ relationship is represented by the input from the *SysML repo* to the *refresh* port of *Modelling*.

Here, a brief summary on how the deliverables of other S2R projects affected the PERFORMINGRAIL baseline: (1) the results of [43] of MOVINGRAIL affected the SysML's functional architecture; (2) the requirements of [42] of have been considered as a baseline for all the modelling activities; the results of ASTRAIL reported in [20] strongly influenced the choice of languages and of modelling approaches here followed.

In the rest of this document, the following naming convention is adopted:

- Automata: between “ ”, first letter in capital, e.g. “Name_of_automaton”;
- Clocks: small capital letters with _ separating words, e.g. CLOCK_1;
- Variables: between dollars, e.g. x;
- Signals: between dollars and ’ ’ with _ separating words, e.g. $'MA\_request'$;
  Operations: use MATHIT command with _ separating words, e.g. send_MA();
- Functions: use MATHIT between “ ”, first letter in capital, e.g. “Trains_management”;
- Timers: use MATHIT command and small capital letters with _ separating words, e.g. SYNCHRONIZATION_TIMER;
- Constants: small capital letters with _ separating words, e.g. NB_TRAINS;
  UML diagrams: between “ ”, use MATHIT, e.g. “TTD_Management_SD”;
- Requirements: between “ ”, respecting the way the requirement is spelled in the specs, e.g. “REQ-Trainloc-5”.
- States: use italic between ’ ’ , first letter in capital, e.g. *’Idle’*;
- External Actors: use italic e.g.*Driver*;

5. The Specification Approach and the SysML Structure

This chapter shows the approach followed for the Specification phase, as described in Chapter 4 — i.e., the first phase of the entire work and the purpose of T2.3. The description is articulated in:

- a description of the scope of the specification activity and the consequent SysML model structure (Section 5.1);
- a formalization of the entire specification process (Section 5.2);
- the description of the proposed SysML structure, also through a running example (Section 5.3);
- some considerations on the integration of the proposed modelling approach with Eulynx DP (Section 5.4);
- a description of the ETCS-L3 functional elements actually chosen (Section 5.5);
- some considerations about tools and concrete artefacts containing the model (Section 5.6).

5.1. The Modelling Scope

This sub-section presents the scope of the specification, and the SysML model structure. Fig. 5.1 reports a Class Diagram (CD), describing the elements of the ETCS-L3 as well as the parts of SysML, involved in the *Specification* activity. The blue blocks are the elements of ETCS-L3, while the red blocks represent the SysML diagrams used to specify one or more blue blocks.

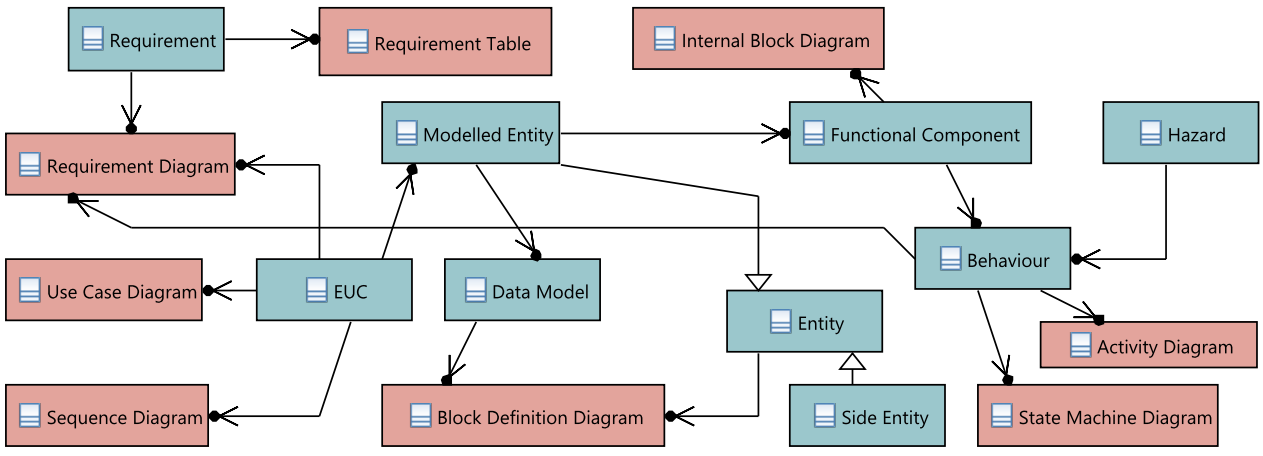


Fig. 5.1. Overview of the ETCS-L3 modelled elements

Entity represents the different (physical) subsystems and components, and it can be the subject of modelling (i.e., *Modelled Entity*) or not (i.e., *Side Entity*). Modelled Entities are referred by *EUCs* that are characterized by one or more *Requirements*. Modelled entities

(i.e., Trackside and On-board) are characterized by a set of internal functions (named *Functional Components*). A Functional Component is responsible for implementing one or more requirements that are related to the same aspect of the ETCS-L3: the way a Functional Component implements such requirements is through a *Behaviour*. A Behaviour can be affected by one or more *Hazards*, and characterized by one or more *Requirements*, too. Modelled entities are also described by a static description of their data (using a *Data Model*).

The description of these “blue blocks” and their relationships is captured by SysML diagrams (represented in the figure by “red blocks”). Table 5.1 reports such kinds of diagrams, explaining their role in the general ETCS-L3 SysML model.

Table 5.1: Mapping between ETCS-L3 concepts and SysML.

Domain Element	Diagram	Description
Data Model	Block Definition Diagram (BDD)	The Data Model describes the software and data part of the ETCS-L3, focusing on elements such as protocol messages, configuration parameters, living variables, etc.
Architecture Model	BDD	This diagram is used to describe physical parts of the model and their static relationships. They are used to describe the ETCS entities involved in the model.
Functional Architecture	Internal Block Diagram (IBD)	This diagram describes the functional components of both trackside and on-board in a more in-depth way, highlighting the interactions among them, in terms of exchanged signals and conveyed data from a component to another.
EUC / Behaviour	Requirement Diagram (RD)	These sub-models depict the requirements related to the EUC or the internal function behaviour, as well as the allocation of requirements on model elements, to tackle the traceability concern.
EUC	Use Case Diagram (UCD)	It structures EUC, by defining relationships with actors and sub-functionalities.
EUC	Sequence Diagram (SD)	This diagram explains the interactions between involved actors and functionalities.
Behaviour	SMD	This diagram depicts, for each internal function of both trackside and on-board, the evolution of the functional component.
Behaviour	AD	These diagrams may be used to describe some actions triggered by the evolution of a functional component's behaviour (i.e., by the State Machine (SM)).
Requirements	Requirement Allocation Table (RAT)	This table summarizes the considered requirements and their respective allocation to one or more SysML model elements.

5.2. Specification Process Description

This sub-section presents the details of the specification approach, as follows. Fig. 5.2 reports the description of the sub-activities involved in the Specification process. First, there are two one-time actions that are carried out in the first iteration of the process:

- ETCS element selection — that is devoted to the definition of the actual (functional) elements of the system that are specified and modelled in this document;

- role assignment — where the roles discussed in Chapter 4 are apportioned to PERFORMINGRAIL's partners.

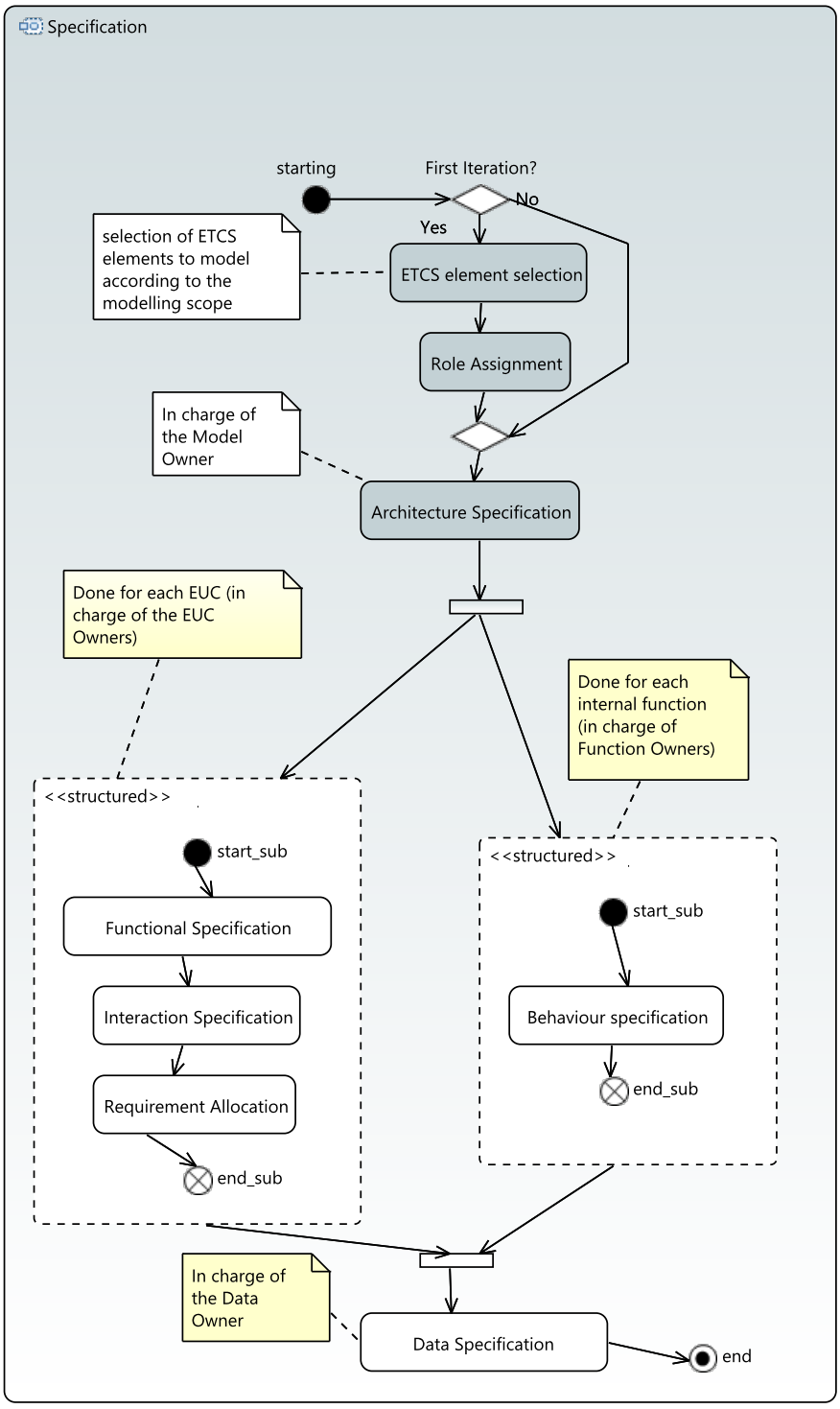


Fig. 5.2. Details on the specification approach.

Then, other activities are carried out every time the Specification process starts, to refine the work previously done. The first of these activities is the Architecture Specification, whose objective is to define the architectural sub-model of all the involved physical entities,through a

BDD. Such Architecture is then improved, in a second instance, by a Functional Architecture, highlighting the interactions between components. Next, different activities start in parallel:

- the specification of each chosen EUC, a sequence of (1) *Functional Specification* (with a UCD), (2) an *Interaction Specification* (with a SD), and (3) a *Requirement Specification* (with a RD);
- the specification of each chosen Functional Component using a *Behavioural Specification* (with one or more SMDs/ADs).

When all the previous activities are completed, *Data Specification* defines the structural view of the data of all modelled entities, ending the Specification process.

5.3. Building the SysML model

This sub-section describes the proposed SysML model structure, and exemplifies it on a selected example. Since the SysML model is complex and structured in different parts and diagrams, some considerations are due in discussing the detailed ways of creating such sub-models. These considerations are displayed by employing a running toy example, where a train is subject to the emergency commands of a trackside controller and has to activate/deactivate its emergency brake.

Let us start from the two considered requirements:

- "REQ-1": Given a running train, when the Braking Supervision receives an emergency stop message, then it activates the emergency brake.
- "REQ-2": Given a braking train, when its speed is at zero, then the Braking Supervision function deactivates the emergency brake.
- "REQ-3": Given that the Controller has sent an emergency stop message, when the Braking Supervision function receives such a message, then it sends back to the Controller an acknowledgment message.

Architecture Specification: Let us consider the functional architecture presented in the form of SysML's BDD/IBD (see Fig. 5.3).

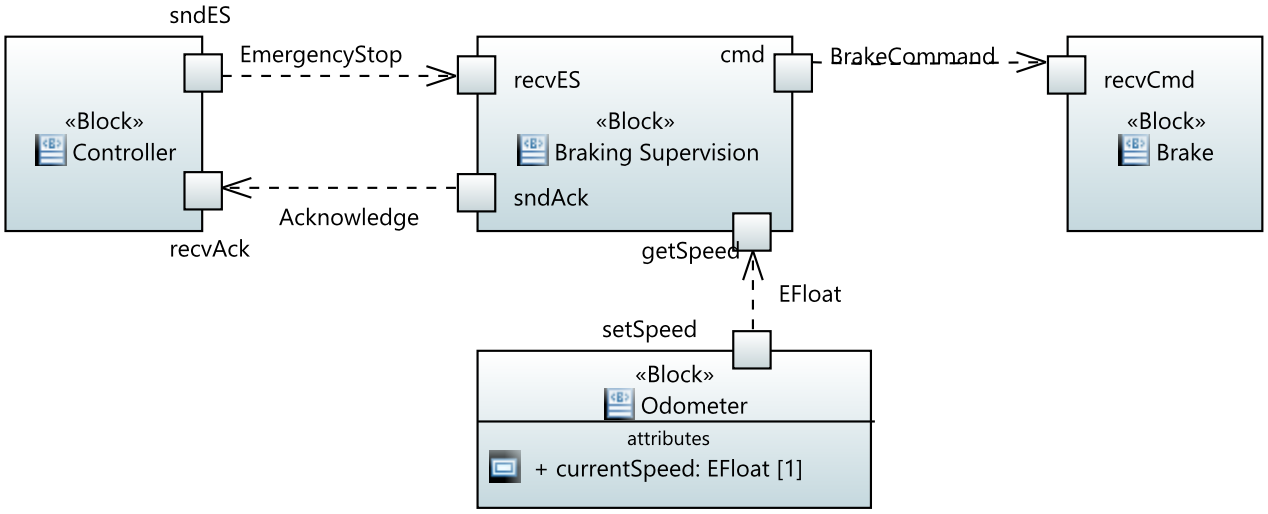


Fig. 5.3. Functional Architecture (running example).

The key elements of this functional architecture are the blocks, representing physical (i.e., “Controller”, “Brake”, and “Odometer”) and “software” components (i.e., the “Braking Supervision”) that are internal functions of other physical components (the “Train” in this case, not reported in the diagram for simplicity). The SysML model presents the internal functions as encapsulated in homonymous SysML’s blocks, to allow adding ports and signals. Ports and signals are the means by which blocks communicate with each other. The ports can be Output Ports and Input Ports according to the direction of the item flows that connect them. Input Ports — e.g, source ports for the item flows — are characterized by a SysML’s signal, whose role is crucial in the description of components’ behaviour. Item flows are characterized by the type conveyed onto it. This data type represents the type that the concrete data transported by the flow are conforming to. In this specific model, the following signals are present (even if not shown in the diagram), all involving the “Braking Supervision” component: $'emergencyStop'$ (communication from “Controller”), $'brakeCommand'$ (communication to the “Brake” block), $'acknowledged'$ (communication to “Controller”), and the $'updatedSpeed'$ signal (communication from “Odometer”). The conveyed types are described in the following Data Model.

Data Model: Fig. 5.4 reports the data model of the running example. In such a model, each class represents a unit of data/software according to the traditional software engineering processes. Classes/Blocks can be characterized by attributes (representing the current state of the subsystem), parameters (whose values are set at configuration-time and are, in general, read-only), and operations (which can be used to hide complex computations from the algorithmic perspective). As in this example diagram, the data model can contain the definition of specific data types as well, e.g., enumeration types.

Some considerations are due, in order to clarify the role of the Data Model in the whole Specification approach. Since the ETCS-L3 signalling system is still subject of research, there are few attempts to create concrete experimental implemented solutions. In PERFORMINGRAIL, there is no will to propose such a solution, considering that the requirements need further assessment. Hence, the underlying hypothesis is the presence of a global database, containing all the variables, parameters and operations, able to read, write and manipulate data, by functional components and their behaviours. Such a database is shared among the components, without further hypotheses on the concrete mechanisms, which are left to future research projects.

Use Cases: Use cases represent general scenarios involving different internal functions. Each of these scenarios is described by utilizing three SysML diagrams, according to Fig. 5.2:

- Functional Specification, used to define the context of the use case, according to the involved external blocks and internal functions. It is represented by a UCD. In the running example, Fig. 5.5 depicts this view.
- Interaction Specification, used to define the sequence of the messages exchanged among the different actors identified in the previous diagram. In this approach, it is represented by a SD. For the messages that are related to a communication that is also represented in the functional architecture, some links with the functional architecture must be set. This is concretely done by setting the signature of the message to the

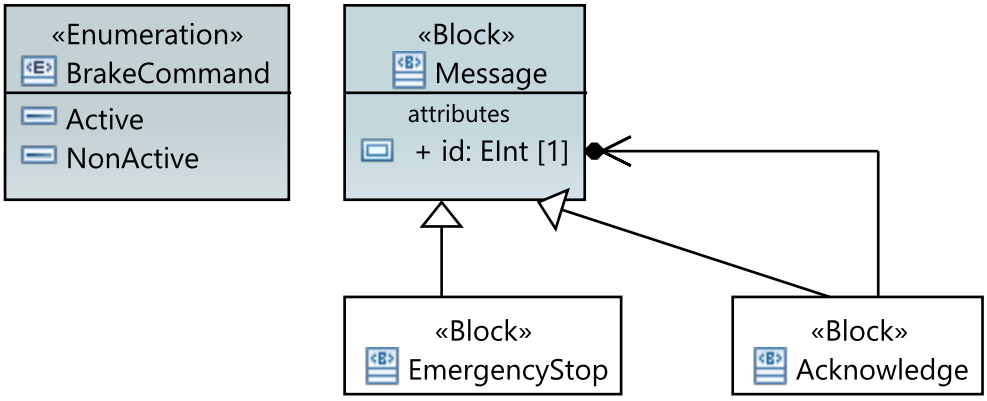


Fig. 5.4. Data Model (running example).

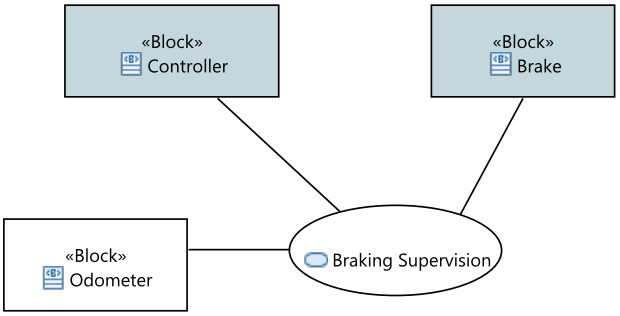


Fig. 5.5. Use Case Functional Specification (running example).

signal of the communication, according to the functional architecture. In the running example, Fig. 5.6 depicts this view.

- Requirement Allocation, where the system requirements considered for the use case are mapped to model elements. A more comprehensive discussion about this mapping is presented at the end of this section.

Functional Components: for each considered internal function, the description of its behaviour is captured by a SMD, respectively. States and transitions are determined by analysing the requirements that are apportioned to the functional component, as well as in analysing the messages that the functional component receives in the different SDs that it is involved in. The Data Model is also crucial in determining the correct behaviours, by-setting the proper guards and actions. Local variables in the state machines are possible, simplifying their structures.

Two points worth discussing follow. The first point regards the transitions of the SMD, linking the Behaviour Specification to the Functional Architecture. When the SMD refers to a functional component that receives a message — i.e., is the target of an incoming message in the SD, and it has an incoming item flow in the functional architecture — one or more transitions in the SMD can be triggered by the receiving of this message. Consequently, the "accepting port" related to the signal is set as in the port field of the transition's trigger.

The second discussion point is related to the actions. As some SMDs need to send mes-

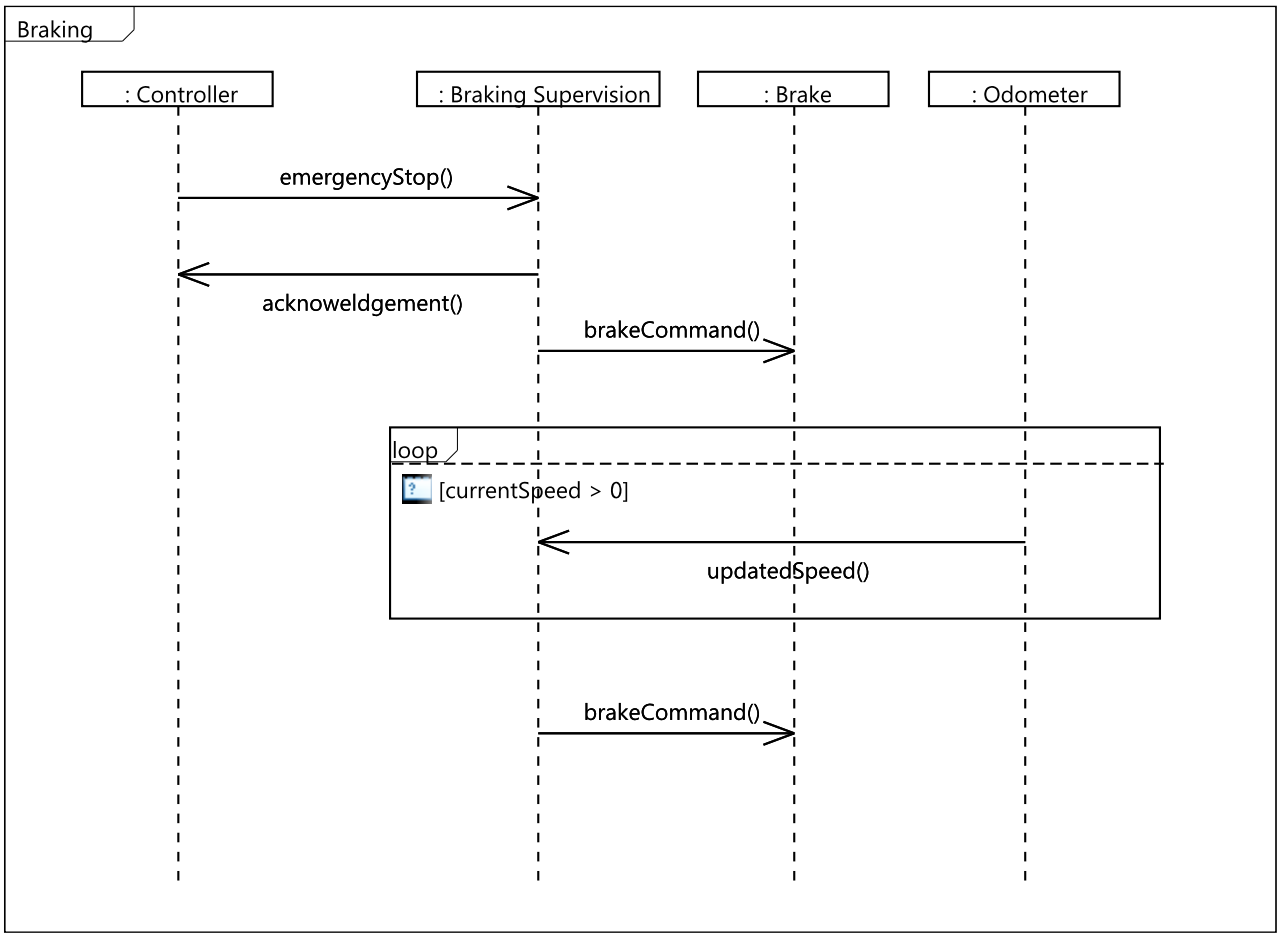


Fig. 5.6. Use Case Interaction Specification (running example).

sages, activities can be set in the effect field of such transitions. These activities can be then developed into ADs, possibly considering SysML's *Send Signal Action*.

Fig. 5.7 reports the SMD describing the behaviour of the "Braking Supervision" functional component, while Fig. 5.8, Fig. 5.9 and Fig. 5.10 report the ADs of the actions related to the transition from *'running'* to *'braking'*, from *'braking'* to *'running'*, and on the self-loop on *'theupdateWaiting'* state, respectively.

Requirement Allocation: to illustrate the number of requirements that are considered during the SysML modelling activities, the considered requirements are mapped onto model elements. A set of model elements are used to satisfy the considered requirements through the `satisfied by` relationship into a RD: transitions, actions, use cases and combined fragments can be used as well to this aim. Next, a RAT is automatically generated, summarizing this mapping into a table-form view.

Fig. 5.11 shows the allocation of the requirements in the considered example, while Table 5.2 reports the example's RAT.

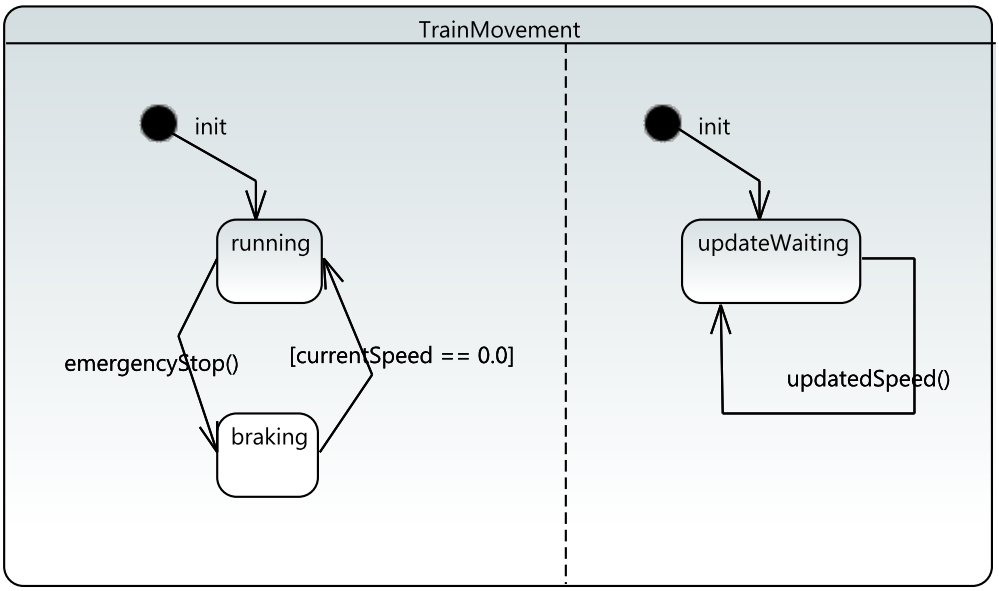


Fig. 5.7. Behaviour Specification — SMD (running example).

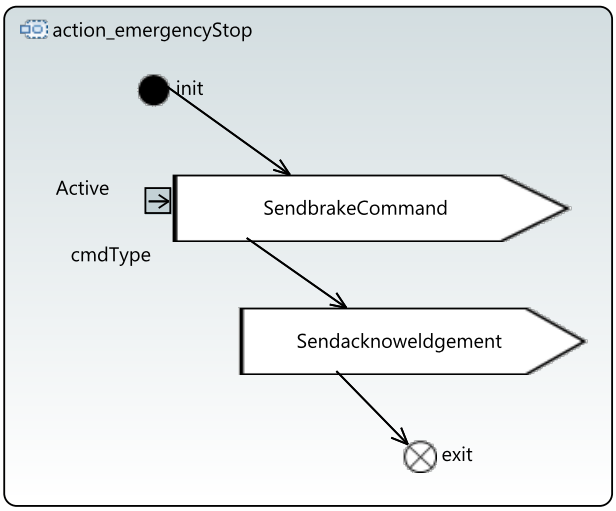


Fig. 5.8. Behaviour Specification — AD/1 (running example).

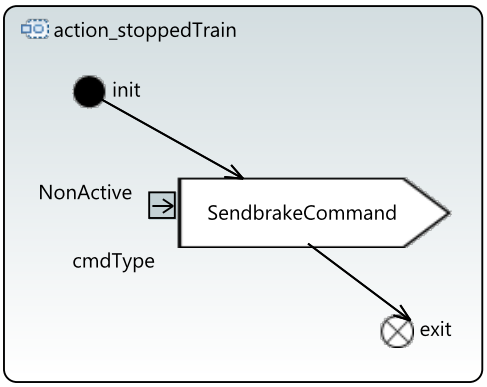


Fig. 5.9. Behaviour Specification — AD/2 (running example).

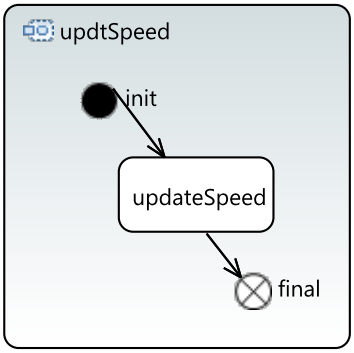


Fig. 5.10. Behaviour Specification — AD/3 (running example).

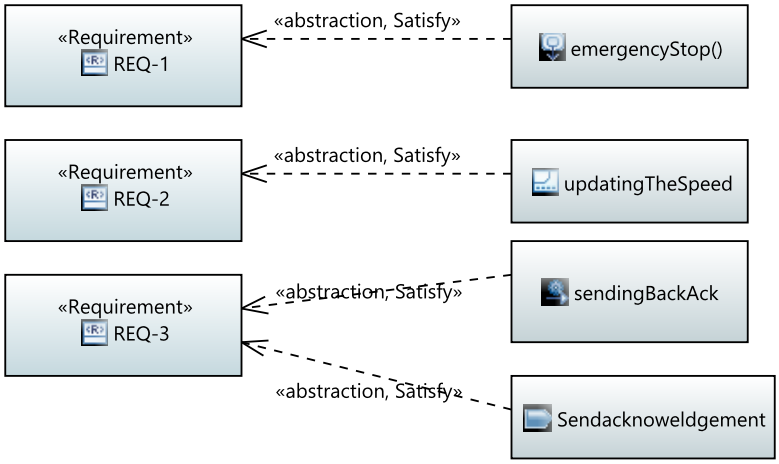


Fig. 5.11. Requirement Allocation Diagram (running example).

Table 5.2: Example RAT.

Requirement	Satisfying Elements
REQ-1	emergencyStop()
REQ-2	updatingTheSpeed
REQ-3	sendingBackAck, Sendacknoweldgement

5.4. Integrating EULYNX DP

This sub-section describes aspects regarding the integration of the proposed modelling approach with Eulynx DP. The Eulynx DP model provides a highly detailed static class model of trackside systems. The information needs of the present project overlap with the information provided by Eulynx DP, yet the level of detail differs. For instance, Eulynx DP models track vacancy detection sections, the associated technical systems and components, e.g., axle counters, which implement train detection. The PERFORMINGRAIL system architecture hides this complexity inside the TTD manager, which reports track vacancy. This indicates that, at this stage, classes such as the TTD manager can be regarded as facades of the detailed Eulynx DP model.

However, state machines that use train positions and distance calculations intensively are being implemented. These state machines use the same model for representing topology and topography as Eulynx DP, which is UIC's RSM. This matters because the signalling equipment, routes, speed profiles and other relevant input are located in the railway network through the RSM model.

The software that can process routes, profiles and signalling equipment *located* in the railway network using the Eulynx DP/RSM model, is hence reusable, since reading in configuration data will be substantially easier.

5.5. Choice of the Functional Elements

This sub-section details the steps followed in defining the boundary of the ETCS-L3 subset of elements that are the subject of the specification and modelling activities. These actions refer to the *ETCS element selection* subprocess, defined in Section 5.2.

The starting point is determined by the OPSs defined in [1]. In this document, ten OPSs have been defined by summarizing some specific situations of ETCS-L3, rather than focusing on entire train trips. In this document, four OPSs have been selected as candidates for guiding the verification and the demonstration of the results conducted in WP2. These four scenarios are:

- Trackside Initialisation
- Points Control
- Loss/Restore of Communications
- Loss of Train Integrity.

In [4], a coverage table from OPSs to EUCs is reported, describing for each OPS what EUCs are involved by the OPS, respectively. By this EUC-OPS mapping, it is possible to determine a subset of the EUCs that is obtained by considering only the selected OPSs. Such a subset contains the EUCs that are necessary to accomplish the demonstration of the modelling approach, and are hence considered in the rest of this document. Such EUCs are as follows:

- Trackside Initialization
- Normal Train Movement
- On-Sight (OS) Movement
- Loss/Restore of Communication
- Loss of Train Integrity

- Staff Responsible (SR)
- Points Control
- Sweeping.

In addition, Functional Components are detected from Figure 4 in [4], where main functions of the ETCS-L3 are reported. The list of these trackside functions are:

- Track Status Management
- Reserved Status Management
- Trains Management
- MA Management
- Route Management
- TTD Management
- Manage Low Adhesion Areas
- Manage Temporary Speed Restrictions
- Points Management
- Communication Management

whereas On-board functions are:

- Train Position Reporting
- Integration Information Management
- Manage Dynamic Speed Profile
- Speed and Distance Supervision

It is important to underline that these functions rely on other ETCS-L2 functions that are not the main objective of this modelling activity and, hence, they will be considered only if necessary. Furthermore, also between these ETCS-L3 functions — as they are mentioned in [41] — Manage Low Adhesion Areas and Manage Dynamic Speed Profile are partially modelled. Both of them are quite hard to model by means of discrete SM-based formalisms.

5.6. Tooling

This sub-section overviews the final considerations regarding the tools adopted in T2.3. The Eclipse Papyrus[1] tool has been chosen to match the requirements of interoperability and adopting an open-source tool able to foster the exploitation of the produced SysML model. Another important feature of Papyrus that justifies its choice is the possibility to “branch” a part of a model into a separate Papyrus file. This possibility enables the apportionment of the different sub-models (each devoted to a EUC or functional component, respectively) to different roles, simplifying the versioning of the whole model with the usage of a git repository (i.e., the *SysML Repo*). This practice enables a distributed and concurrent modelling practice with Eclipse Papyrus, without building/adopting a complex/closed solution. The result of this

[1] https://www.eclipse.org/papyrus/

approach is available at the GitHub repository of the model https://github.com/stefanomarrone/performingrail.

6. The Detailed SysML Model

This chapter presents the results of applying the Specification Approach on ETCS-L3 and provides details on the different sub-models that compose the SysML model. The sub-models are described in the following order:

- *Architectural Specification*. This sub-model defines the structure and behaviour of the ETCS-L3 system, including its components, interfaces, ports and connectors.
- *Data Model*. This sub-model specifies the data types and enumerations used by the ETCS-L3 system, as well as their relationships and constraints.
- *ETCS Use Cases*. This sub-model captures the functional requirements of the ETCS-L3 system, using use case diagrams and sequence diagrams to describe the interactions between the system components and its actors.
- *Functional Component*. This sub-model describes the internal behaviour of the ETCS-L3 system (divided into trackside and onboard) to model its logic by means of state machines.
- *Requirement Allocation*. This part traces the requirements of the ETCS-L3 system to the elements of the other sub-models to show how each requirement is satisfied by the system design.

The complete SysML model is available at `https://github.com/stefanomarrone/performingrail` and has been developed according to the toolset described in Section 5.6.

6.1. The Architectural Specification

This subsection presents the Architectural Specification of the ETCS-L3 system, using a BDD to show its components and their relationships. The BDD is depicted in Fig. 6.1. This structural diagram shows the concepts as in Fig. 5 of [41] but in a SysML style. Some associations among these blocks are annotated with a comment reporting the ERTMS-related documents that support the association.

The core of the model consists of the *Trackside* and *On board* blocks. They interact with the *ETCS-L3 External Users*, who are the actors that use the system — namely, the *Infrastructure Manager*, the *Driver* and the *Dispatcher* — and with *External Interfaces*, which are other systems related to ETCS-L3-related systems — namely, the *Object Controller*, the *TTD*, the *Train/Engine*, other *Adjacent Signalling Systems*, and the *Traffic Management System (TMS)*. The TLU[1] and the Train Integrity Monitoring System (TIMS) are also explicitly mentioned because they play a crucial role in the ETCS-L3 signalling system.[2]

[1] In this deliverable, the TLU is considered as the summarization of all the HW/SW technologies able to determine the position of a train including Eurobalises, GNSS's odometry and other related mechanisms.
[2] The attribute *External* is consistent with the nomenclature of [41].

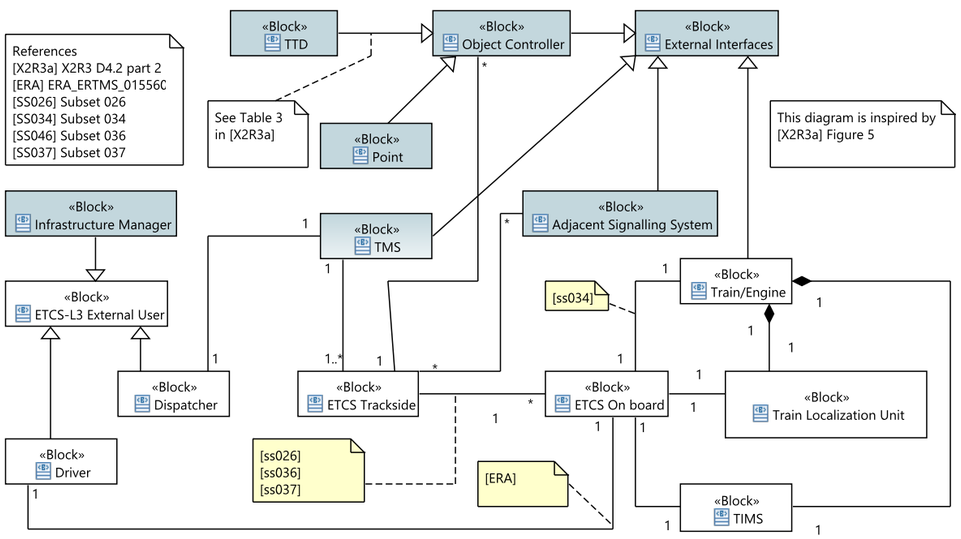


Fig. 6.1. Architecture Specification.

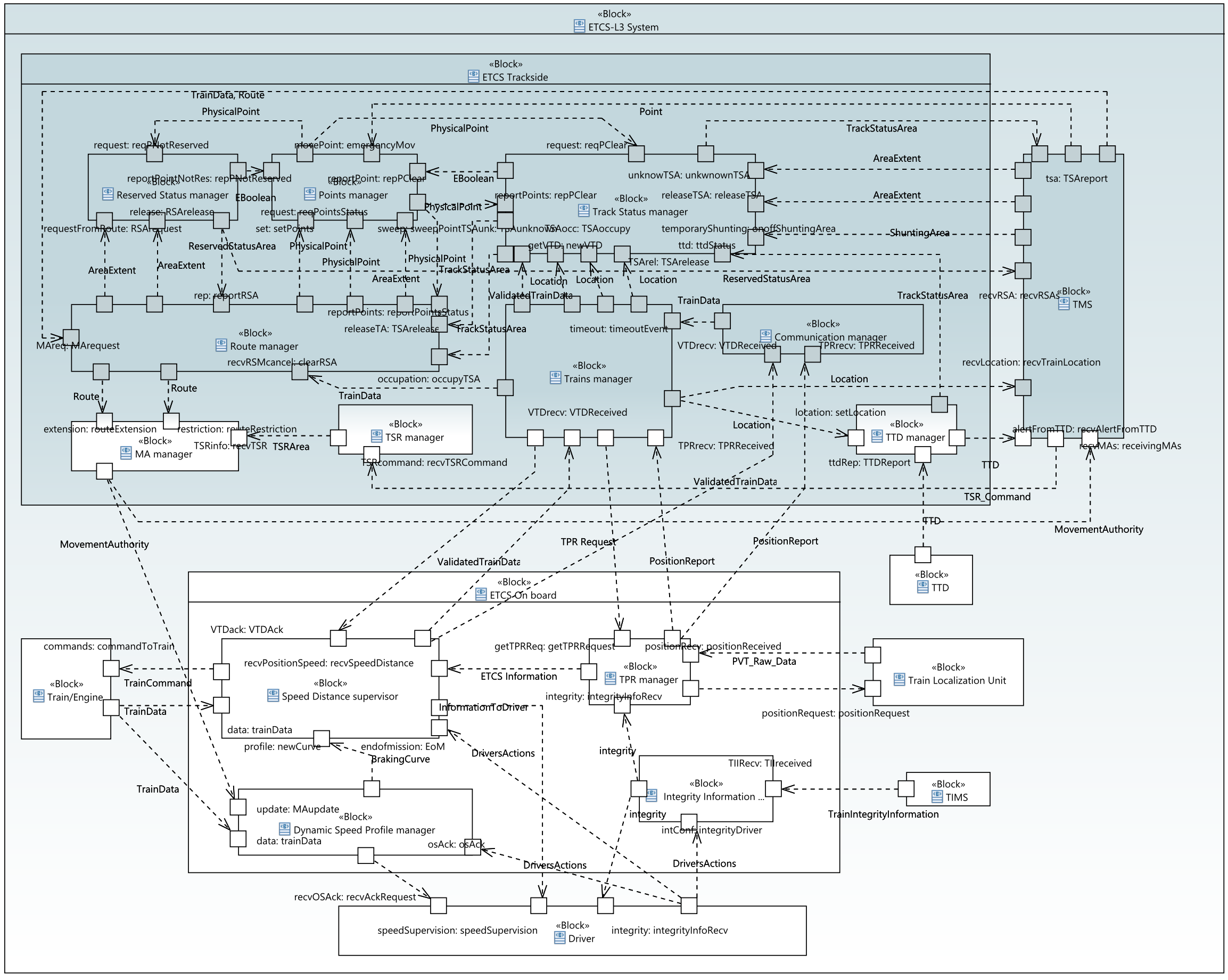


Fig. 6.2. Functional Architecture Specification.

This diagram serves as the basis for the Functional Architecture, which is a key representation of the approach. The Functional Architecture has been derived from similar diagrams presented in [41, 44]. The relation between this model and these references is explained in [4], while this deliverable shows its SysML representation as an IBD (Fig. 6.2). In this Functional Architecture, the Trackside and On-board subsystems are decomposed, showing their internal functions and the relations among them and with the external components. The basic mechanism for the interaction is described in Section 5.3, while the details of the specific interactions are given in Tables 6.1, 6.2, 6.3, and 6.4. In these tables, the signals are grouped by the block (e.g., the functional or component) that receives the signal (the Owner). These interactions are only a subset of the real interactions: some components (e.g., *TLU*) are only considered for their interactions with ETCS-L3 components; other interactions are not related to ETCS-L3 or depend on the technology used.

Table 6.1: Interactions in the Functional Architecture (part 1).

Signal	Source	Conveyed Type	Description
Driver			
$'integrityInfoRecv'$	Integrity Information Manager	integrity	It receives train integrity information as detected by TIMS.
$'speedSupervision'$	Speed Distance Supervisor	Information To Driver	Light/Sound alarm to the Human-Machine Interface (HMI).
$'recvAckRequest'$	Dynamic Speed Profile Manager	-	Request for acknowledge in case of OS mode.
Train/Engine			
$'commandToTrain'$	Speed Distance Supervisor	TrainCommand	Traction and brakes commands, given to the train by the Automatic Train Protection (ATP) functions[3].
TMS			
$'TSAreport'$	Track Status Manager	Track Status Area	Reporting the status of the areas to the TMS.
$'recvLocation'$	Trains Manager	Location	Reporting the position of the trains to the TMS.
$'receivingMAs'$	MA manager	Movement Authority	Reporting th e MAs sent to the trains to the TMS.
$'recvRSAs'$	Reserved Status Manager	Reserved Status Area	Reporting the status of the track reservation to the TMS.
$'recvAlertFromTTD'$	TTD manager	TTD	Alerting the TMS in case of suspect track occupancy.
TLU			
$'positionRequest'$	Integrity Information manager	-	Request for an estimated position.
Trackside::Trains Manager			
$'VTDReceived'$	Speed Distance Supervisor	Validated Train Data	Validated Train Data (VTD) message to the trackside. Used to know the length and other parameters of the train.
$'TPRReceived'$	Train Position Report (TPR) manager	PositionReport	TPR message received by the train to know position and integrity information.
$'timeoutEvent'$	Communication manager	Train Data	Used to know if one or more of the communication timer have expired.
Trackside::MA Manager			
$'routeExtension'$	Route Manager	Route	Used to updating a MA
routeRestriction	Route Manager	Route	Used to restrict an existing MA
$'recvTSR'$	Temporary Speed Restriction (TSR) manager	TSR Area	Sending activated TSR needed to compute MA.

Table 6.2: Interactions in the Functional Architecture (part 2).

Signal	Source	Conveyed Type	Description
Trackside::Route Manager			
$'reportRSA'$	Reserved Status Manager	Reserved Status Area	It gives information about the status of a reservation.
$'MArequest'$	TMS	Train Data, Route	Triggered when the TMS asks for and MA for a given train on a specific route.
$'reportPointStatus'$	Points manager	Physical Point	It gives information about a point.
$'TSArelease'$	Track Status manager	$'TrackStatusArea'$	Triggered when an area of the track is to clear according to the passage of a train.
$'clearRSA'$	Trains Management	TrainData	Triggered by the Trains manager when a train disconnects and its related reserved area is to clear.
Trackside::TTD Manager			
$'setLocation'$	Trains Manager	Location	Position of a train (used to avoid false positives).
$'TTDreport'$	TTD	TTD	HW signals from the track circuits.
Trackside::TSR Manager			
$'recvTSRCommand'$	TMS	TSR_Command	Used when the TMS commands a TSR.
Trackside::Points Manager			
$'setPoints'$	Route manager	PhysicalPoint	Used to command a point to move.
$'reqPointStatus'$	Route manager	PhysicalPoint	Used to query the status of a point.
$'repPClear'$	Track Status manager	Boolean	Used to know if a point does not belong to an occupied/unknown area.
$'repPNotReserved'$	Reserved Status manager	Boolean	Used to know if a point does not belong to a reserved area.
$'emergencyMov'$	TMS	PhysicalPoint	Used to manually override point movement protection.
$'sweepPoint'$	Route manager	PhysicalPoint	Used to sweep points by a sweeping train.
Trackside::Reserved Status Manager			
$'reqPNotReserved'$	Points manager	PhysicalPoint	Used by Points management to query whether a point belongs or not to a reserved status area.
$'RSArelease'$	Route manager	AreaExtent	Triggered when an area is requested to be released.
$'RSArequest'$	Route manager	AreaExtent	Triggered when an area is requested to be reserved.
Trackside::Communication Manager			
$'TPRReceived'$	TPR manager	PositionReport	Used to compute communication timers.
$'VTDReceived'$	Speed Distance Supervisor	Validated Train Data	Used to compute communication timers.

Table 6.3: Interactions in the Functional Architecture (part 3).

Signal	Source	Conveyed Type	Description
Trackside::Track Status Manager			
$'repPClear'$	Points manager	PhysicalPoint	Used by Points management to inform about the status of a point.
$'reqPClear'$	Points manager	PhysicalPoint	Used to understand whether a point belongs or not to an occupied/unknown area.
$'TSAunknown'$	Trains manager	Location	Used by Trains manager to signal to update to unknown the status of an area according to the train location.
$'TSAoccupy'$	Trains manager	Location	Used by Trains manager to signal to update to occupy the status of an area according to the train location.
$'TSArelease'$	Trains manager	Location	Used by Trains manager to signal to update to clear the status of an area according to the train location.
$'ttdStatus'$	TTD manager	TrackStatusArea	To get occupancy information about a TTD.
$'onoffShuntingArea'$	TMS	ShuntingArea	Used to enable/disable a Shunting Area.
$'unknownTSA'$	TMS	AreaExtent	Manual overriding operation from TMS.
$'releaseTSA'$	TMS	AreaExtent	Manual overriding operation from TMS.
$'newVTD'$	Trains manager	ValidatedTrainData	Used to react to a change in VTD under the aspect of track occupancy.

Table 6.4: Interactions in the Functional Architecture (part 4).

Signal	Source	Conveyed Type	Description
On-board::TPR manager			
$'getTPRRequest'$	Trains manager	TPR Request	Used by the trackside to request a TPR to a train.
$'integrityInfoRecv'$	Integrity Information manager	integrity	Integrity related information to include in the TPR.
$'positionReceived'$	TLU	PVT_Raw$_{D}ata$	Raw data about the Position Velocity Time (PVT).
On-board::Speed Distance Supervision			
$'trainData'$	Train/Engine	TrainData	Data of the train used to compute braking curves.
$'VTDAck'$	Trains manager		Acknowledge of the Validated Train Data message.
$'recvSpeedDistance'$	TPR manager	ETCS Information	Update of the position/speed of the train for the supervision.
$'newCurve'$	Dynamic Speed Profile manager	BrakingCurve	Update of the braking curves.
$'EoM'$	Driver	DriversAction	End of Mission (EoM) command to close the supervision mechanism.
On-board::Integrity Information manager			
$'TIIreceived'$	TIMS	Train Integrity Information	Integrity confirmation by the train.
$'integrityDriver'$	Driver	DriversAction	Integrity confirmation by the driver.
On-board::Dynamic Speed Profile			
$'MAupdate'$	MA manager	Movement Authority	Update of the MA and related information (e.g., Static Speed Profile (SSP)).
$'trainData'$	Train/Engine	TrainData	Data of the train used to compute braking curves.
$'osAck'$	Driver	DriversAction	Driver confirming for entering in OS mode.

6.2. The Data Model

This subsection describes the Data Model, which is the structural part of the SysML model that represents the software aspects of the system. Tthe Data Model acts as a global database, storing all the configuration parameters and the current variables of both track-side and on-board subsystems. The Data Model also serves as a blackboard that allows each internal function to write and read data to communicate with the other functions. The Data Model is composed of four main packages, as shown in the Package Diagram (PD) in Fig. 6.3.

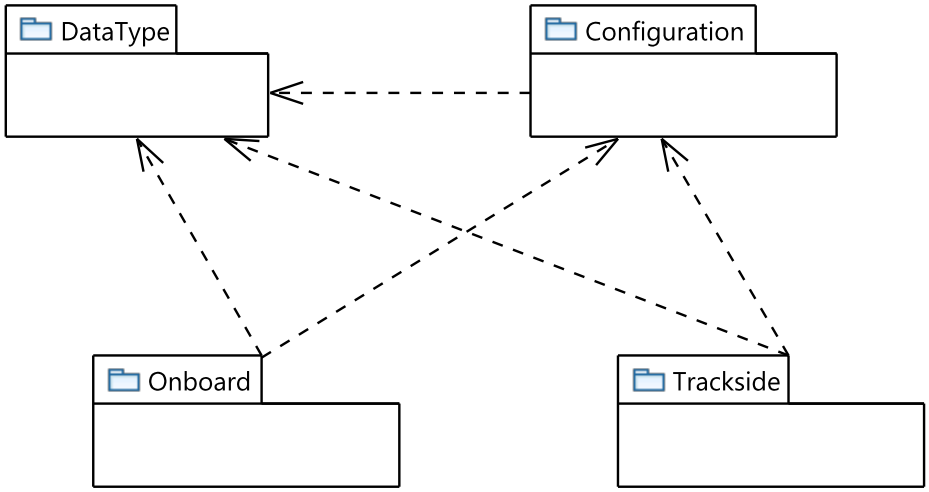


Fig. 6.3. Data Model Package Diagram

The **DataType** package defines the `Enumerations` and `Types` that are used by other packages. Many of these types are derived from the ETCS-L2 and ETCS-L3 documents. One notable type is the `AreaExtent`, which represents a contiguous area of railroad, extending from `start` to `end`. The **DataType** package is shown in Fig. 6.4.

The packages **DriverInterface** (Fig. 6.5) and **TrainInterface** (Fig. 6.6) defines the data and the types of the information exchanged respectively between the system and the driver and the train, respectively.

The **Configuration** package, shown in Fig. 6.7, contains the SysML's blocks that specify the configuration parameters of a generic ETCS-L3 system, such as communication and lineside aspects.

The **Trackside** package, shown in Fig. 6.8, describes the core software data structures that form the basis of a ETCS-L3 trackside system.

The **Onboard** package, shown in Fig. 6.9, describes the core software data structures that form the basis of a ETCS-L3 on-board system.

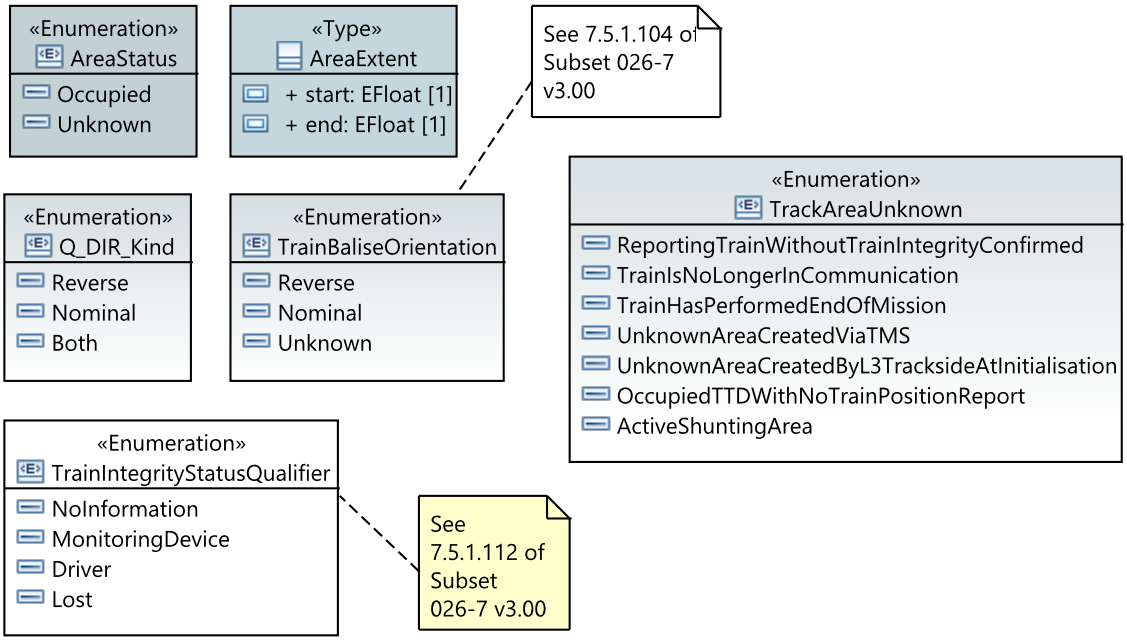


Fig. 6.4. The DataType package BDD

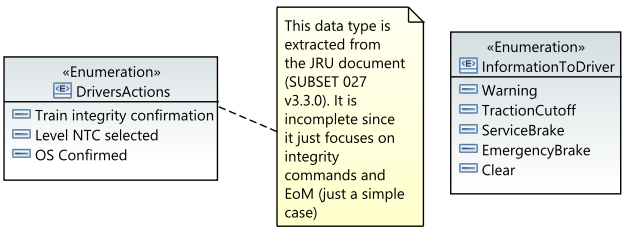


Fig. 6.5. The DriverInterface package BDD

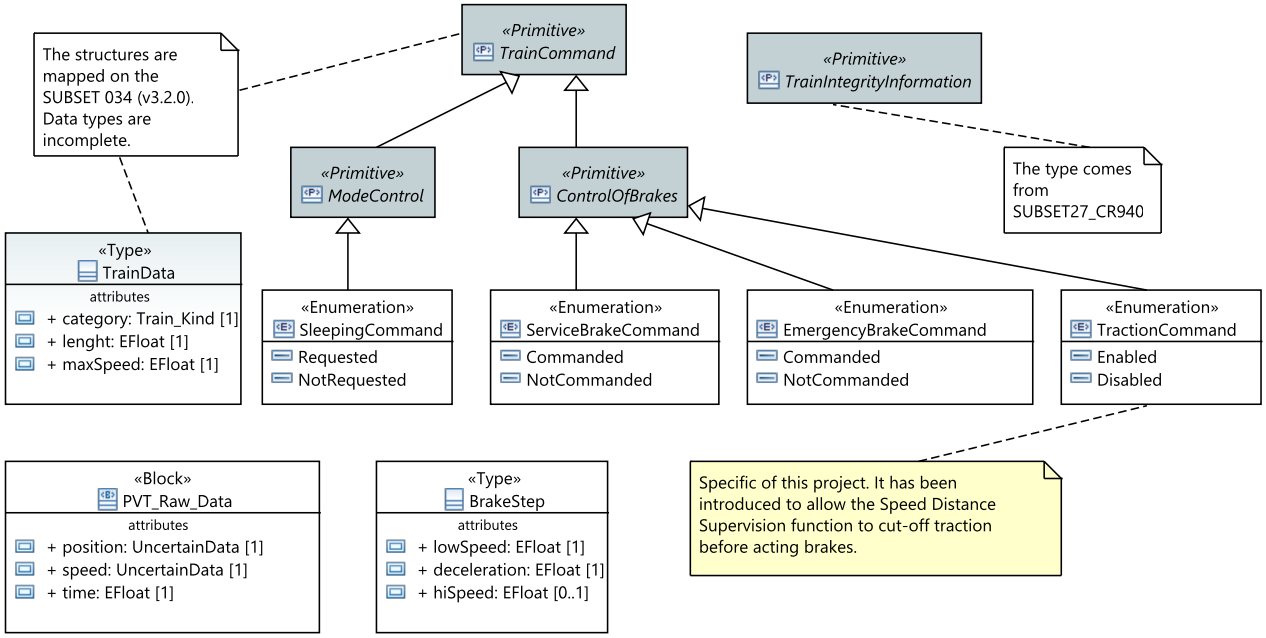


Fig. 6.6. The TrainInterface package BDD

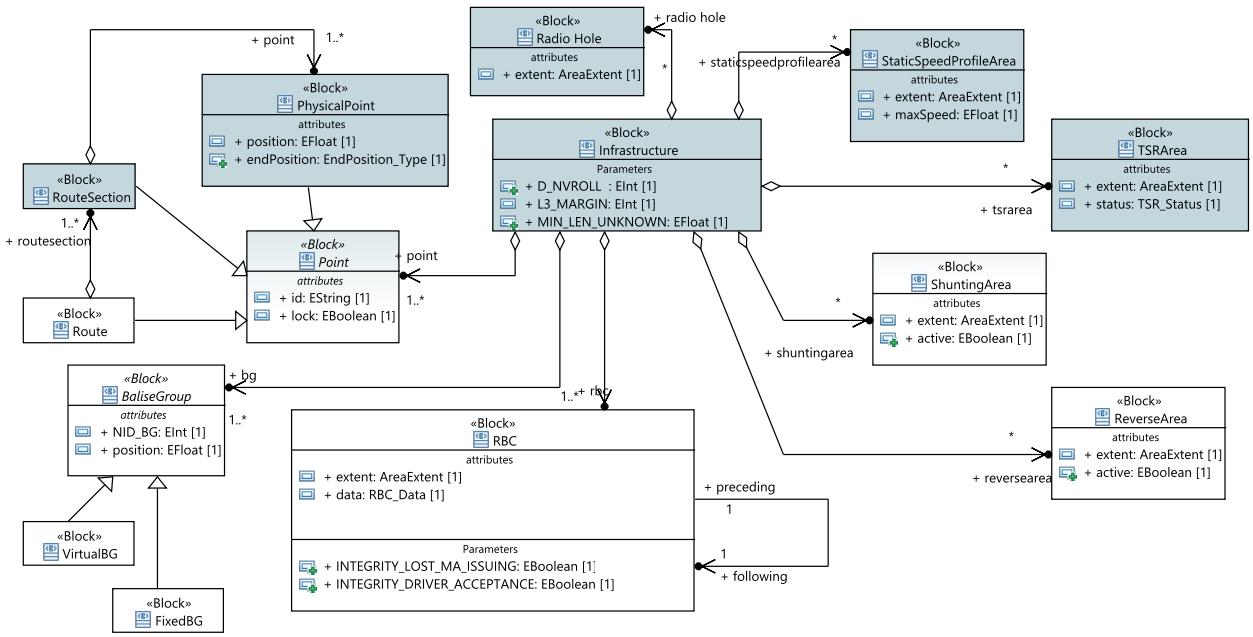


Fig. 6.7. The Configuration package BDD

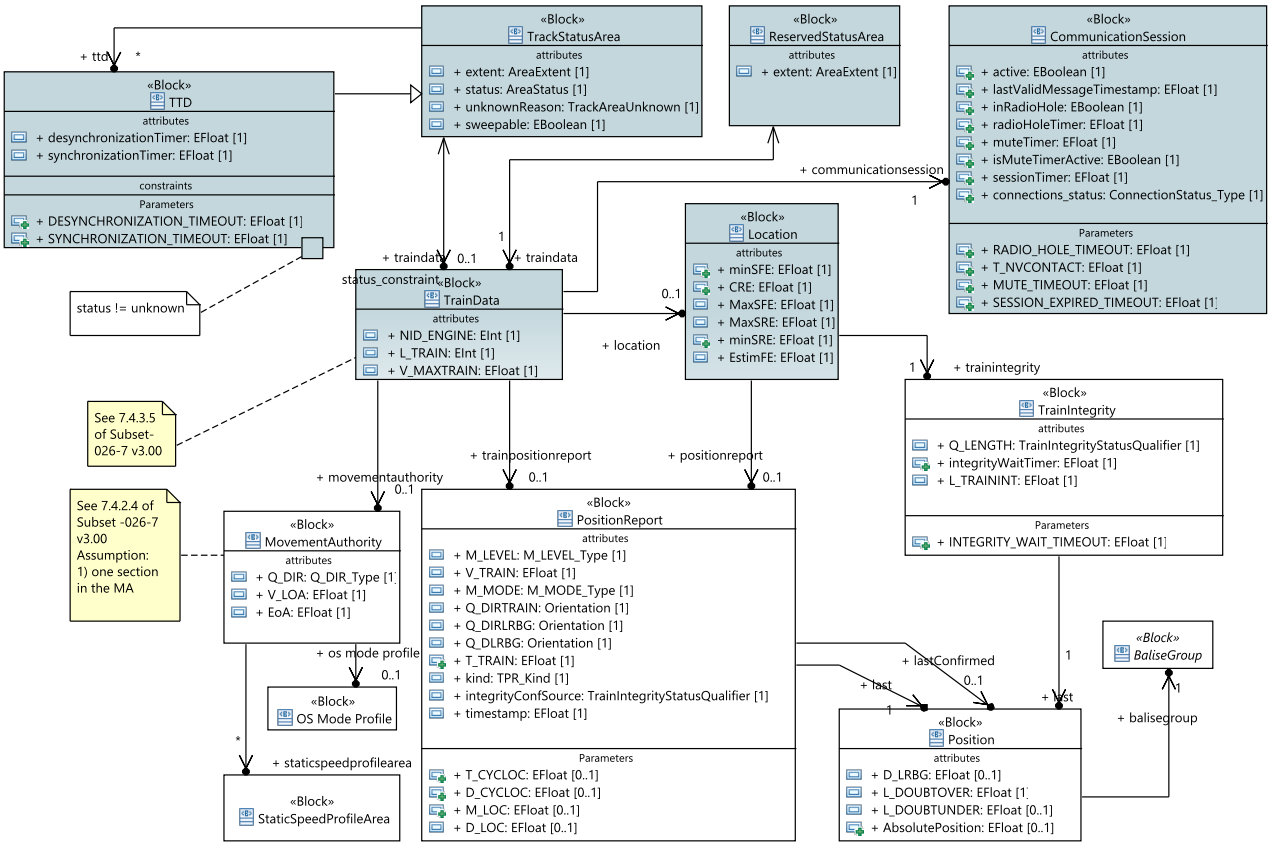


Fig. 6.8. The Trackside package BDD

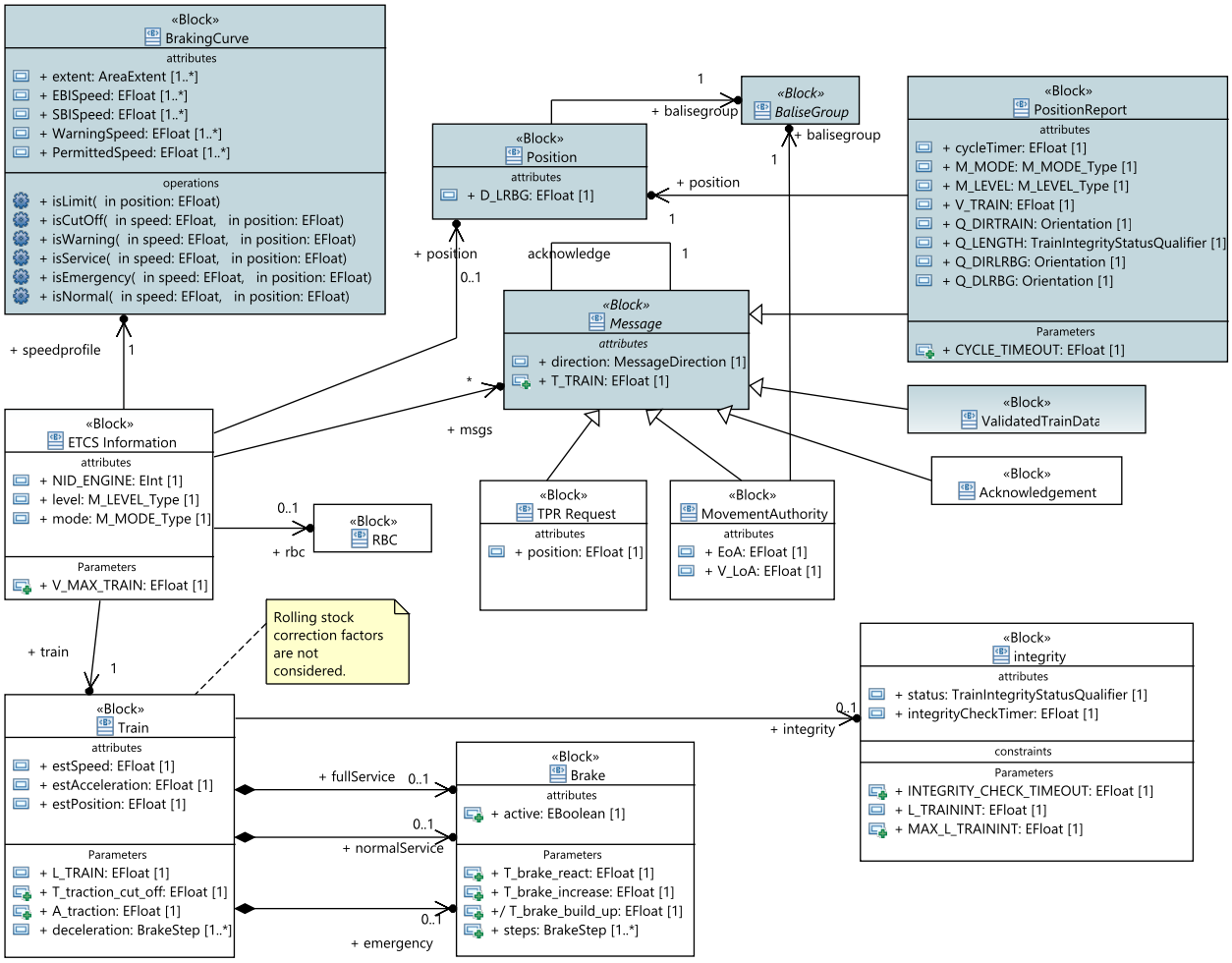


Fig. 6.9. The Onboard package BDD

6.3. The ERTMS Use Cases

This subsection presents the SysML sub-models of the ETCS-L3 use cases, using RDs, UCDs, and SDs.

6.3.1. Trackside Initialisation

The *Trackside Initialisation* use case describes the process of bringing the trackside equipment to a state that enables safe operation to begin.

The trackside system is a vital signalling system. Such a system defines a fail-safe state which is typically attained due to loss of power; after power-on, all detectors report the state “unknown”. However, it is assumed that configuration data and some vital settings such as temporary speed restrictions (TSR, slippery track) are stored in non-volatile memory. In a similar way, it can be safely assumed that trains store vital status information such as train length and integrity information. The latter kind of information has a long time-to-live after which it becomes stale. Once the track-train communication is re-established, trains communicate this information to the RBC.

After power-on, the trackside system acquires point detection from the point control subsystems and other systems such as movable bridges and hot-box detectors (out of the scope of this project).

Trackside information includes point detection and TTD. Train detection using track circuits is relatively straightforward because it detects the presence of rolling stock. Axle counting systems behave differently during power-on; the nature of axle counting systems implies that train movement during a “dark phase” is not detected. Therefore, any TTD sections using axle counters enter the state occupied after initialisation. As a consequence, the RBC cannot issue FS MA but only SR MA’s.

Crucial to trackside initialisation is the acquisition of the safe position of trains. This subsumes that procedures must ascertain that trains haven’t coupled or split and that no non-reporting rolling stock has entered the network, e.g., to tow stranded trains. This information cannot be acquired technically during a dark phase, the management of dangerous situations must be described in proper procedures involving drivers and signallers. Such procedures are beyond the scope of this project but should be drawn up and respected by the Infrastructure Managers (IMs). The Trackside Initialization diagrams are reported in Fig. 6.10 (UCD), in Fig. 6.11 (SD) and in Fig. 6.12 (RD).

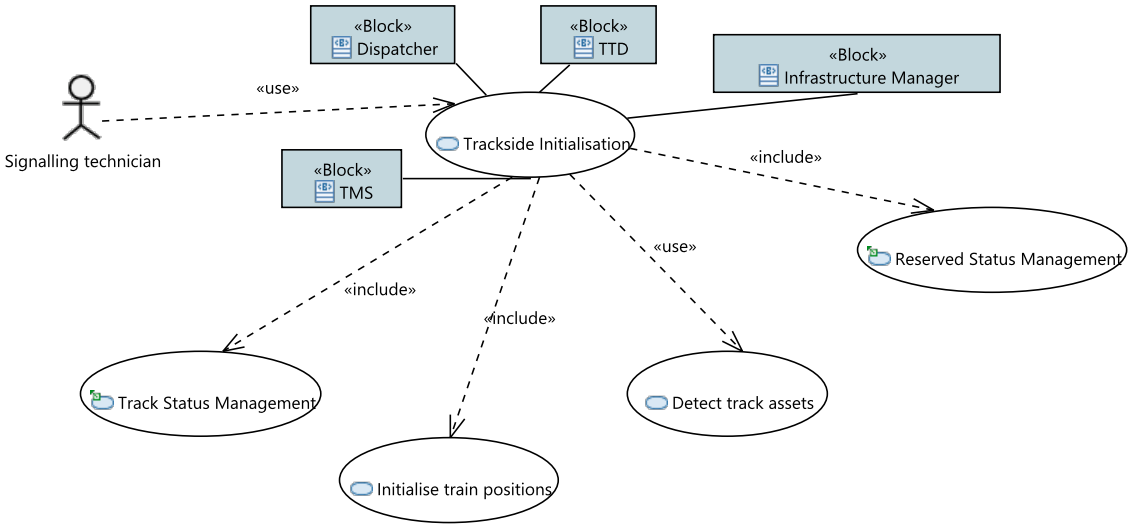


Fig. 6.10. The Trackside Initialization EUC UCD

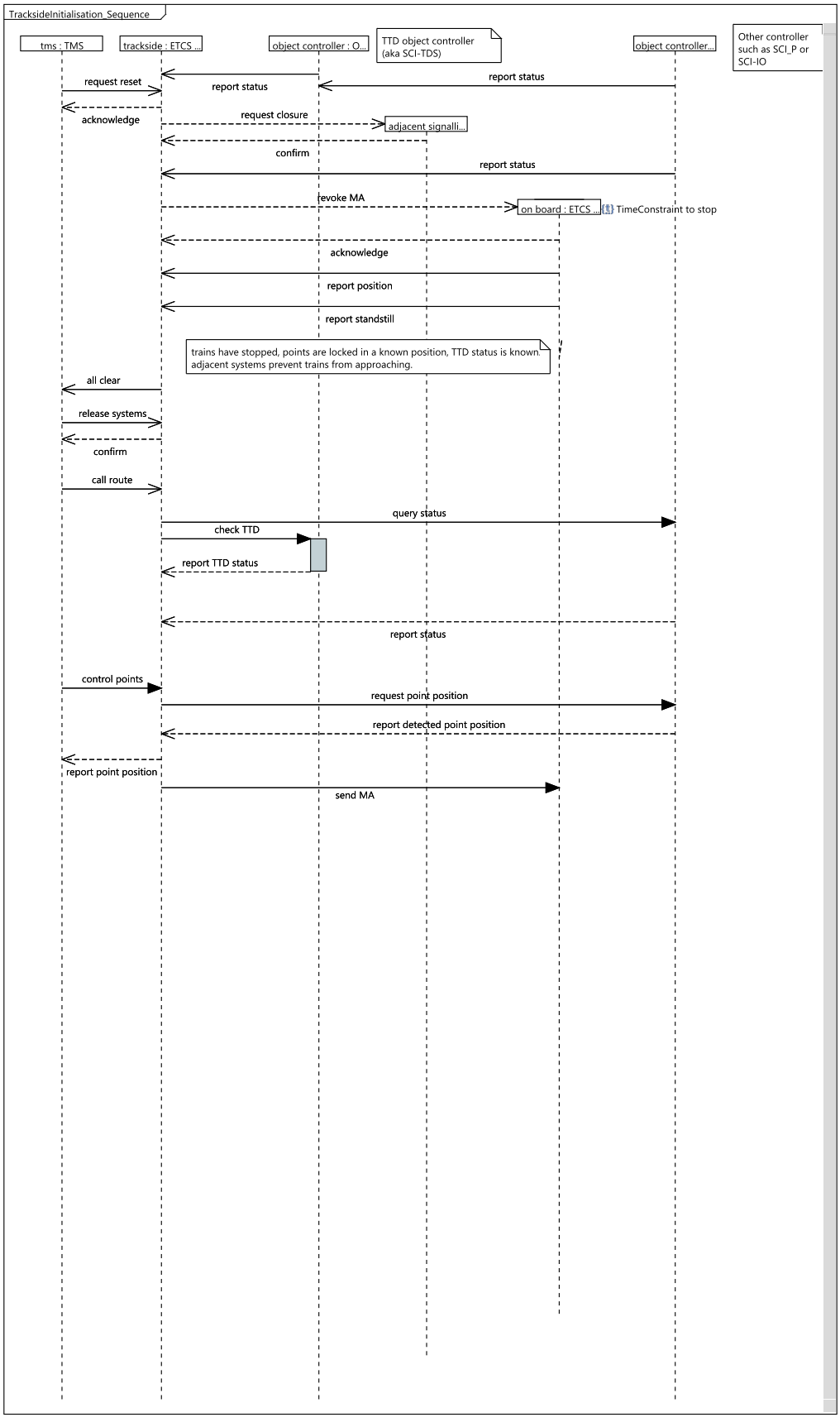


Fig. 6.11. The Trackside Initialization EUC SD

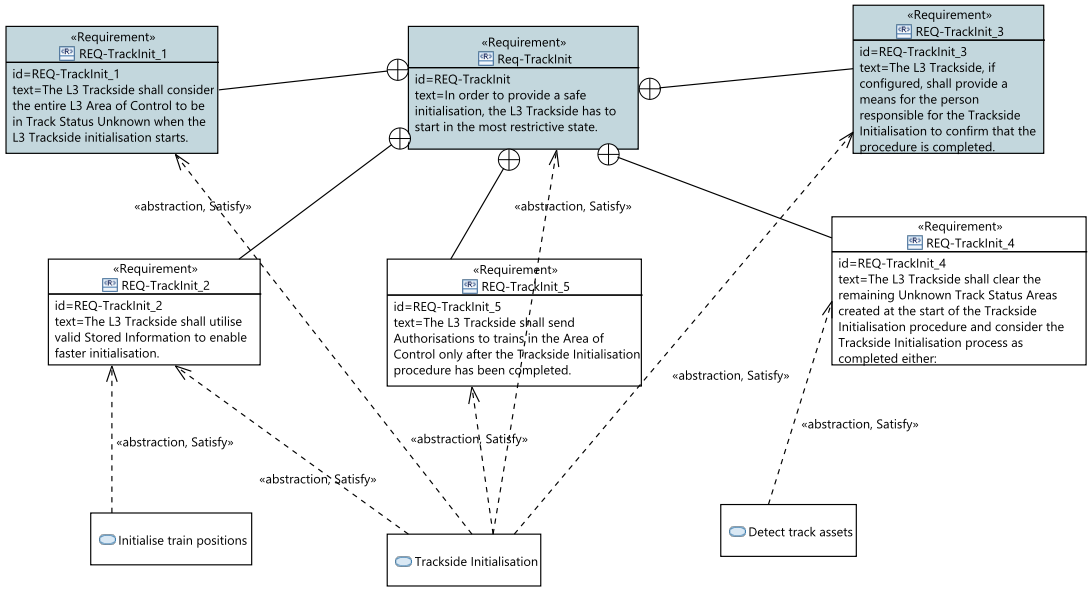


Fig. 6.12. The Trackside Initialization EUC RD

6.3.2. Normal Train Movement

The *Normal Train Movement* use case describes the nominal actions performed by the trackside and a train during its running. In particular, this use case considers the normal running of a single train with integrity confirmed either by an external device or by the driver. Also, the presence of the TTD is considered as optional.

Given its scope and regarding high-level topics, requirements of Normal Train Movement are spread among the different chapters of [42]. Specifically, after a careful analysis of the cited document, a set of applicable requirements has been identified and reported in the SysML model. The resulting RDs are depicted in Fig. 6.13, Fig. 6.14 and Fig. 6.15.

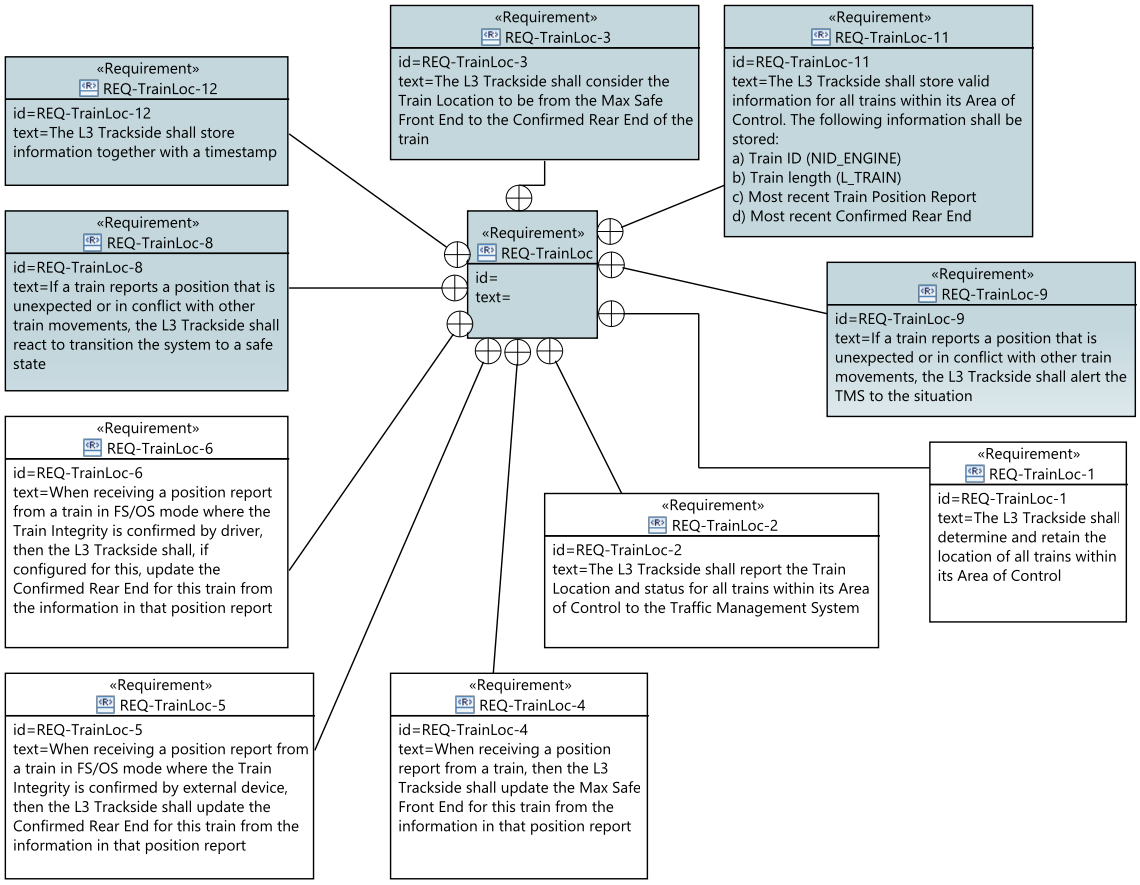


Fig. 6.13. Normal Train Movement Requirement Diagram (1/3).

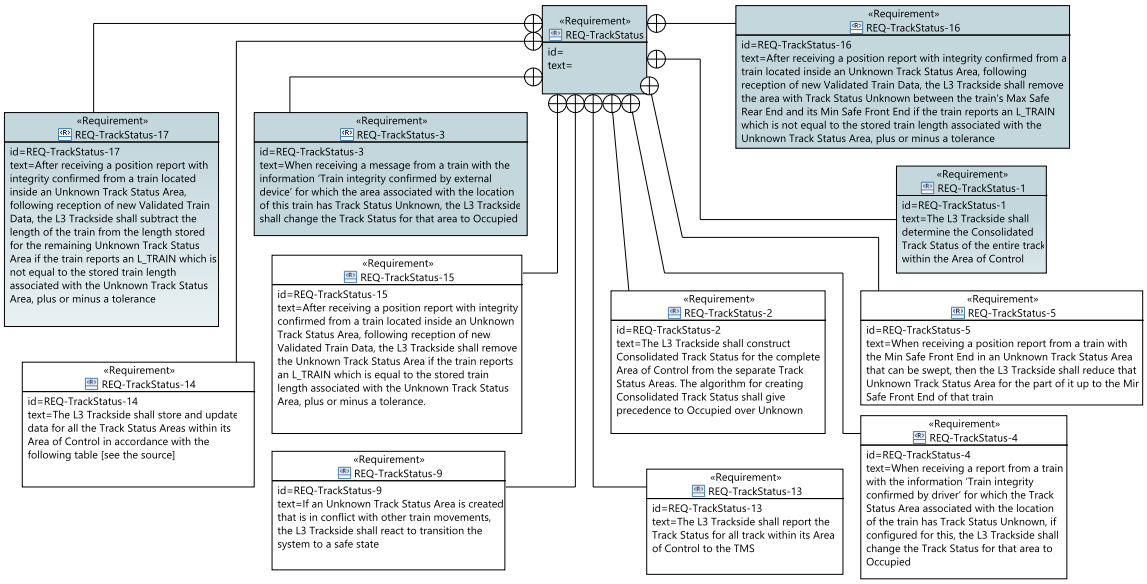


Fig. 6.14. Normal Train Movement Requirement Diagram (2/3).

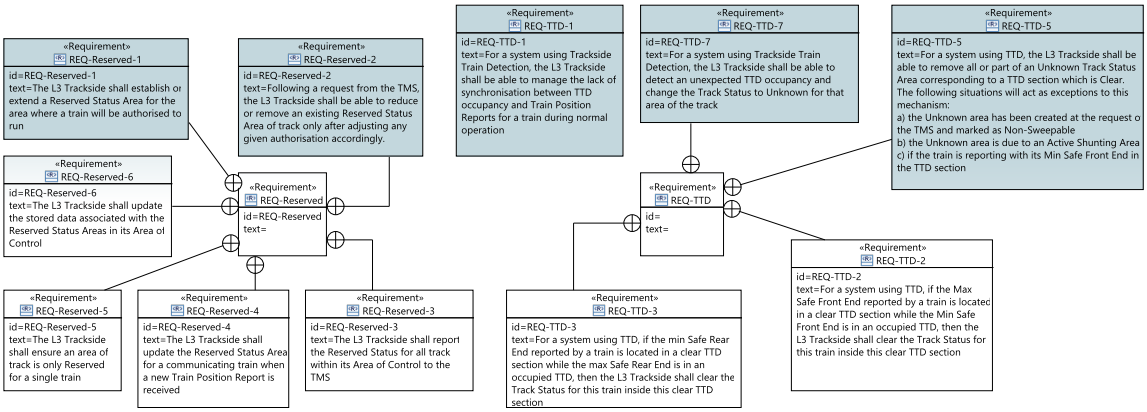


Fig. 6.15. Normal Train Movement Requirement Diagram (3/3).

Based on the requirements reported above, the UCD has been designed, and it is reported in Fig. 6.16. The external actors involved in the use case, represented as blocks in the diagram, are *Driver*, *Trackside*, *TMS* and *TTD*. The use case has also some *include* relationships with the impacted use cases, which are *Train_Position_reporting*,*Integrity_Information_Management*, *Communication Management*, *Trains_Management*, and *Track_Status_Management*.

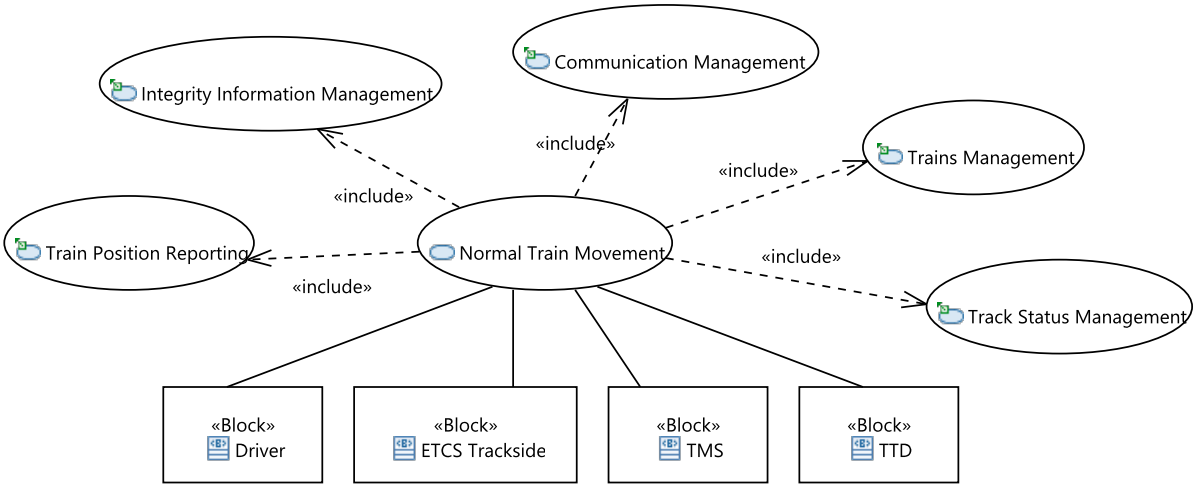


Fig. 6.16. Normal Train Movement Use Case Diagram.

The detailed behaviour of the use case is modelled by the SD in Figure 6.17. The diagram depicts the scenario in which a train, specifically the *"Train_Position_Reporting"* subcomponent, sends the TPR to the *Trains Manager* of the trackside. When this component receives this information, it computes the updated Train Location and, consequently, updates the Max Safe Front End and the Confirmed Rear End based on the train integrity confirmation. At last, it updates the TMS with the information regarding the train location. The underlying hypothesis of this use case is that the *TIMS* confirms the train integrity, hence the *Trains Manager* is also in charge of sending the two signals, $'TSAoccupy'$ and $'TSArelease'$, to *Track Status Manager*. This latter component updates the extent of the corresponding $Track\_Status\_Areas$, also deleting it and clearing the track when the train reaches the end of the track area controlled by the trackside. In the specific case in which the train is moving over an Unknown Track Status Area, in the OS-SR mode, the *Trains Manager* sends the $'TSAunknown'$ to the *Track Status Manager*. At last, in the case in which the train integrity is reported as lost, the Trains Manager also sends the $'TSAunknown'$ to the *Track Status Manager*. In all the cases, the *Track Status Manager* reports the status of the $Track\_Status\_Areas$ to the *TMS*.

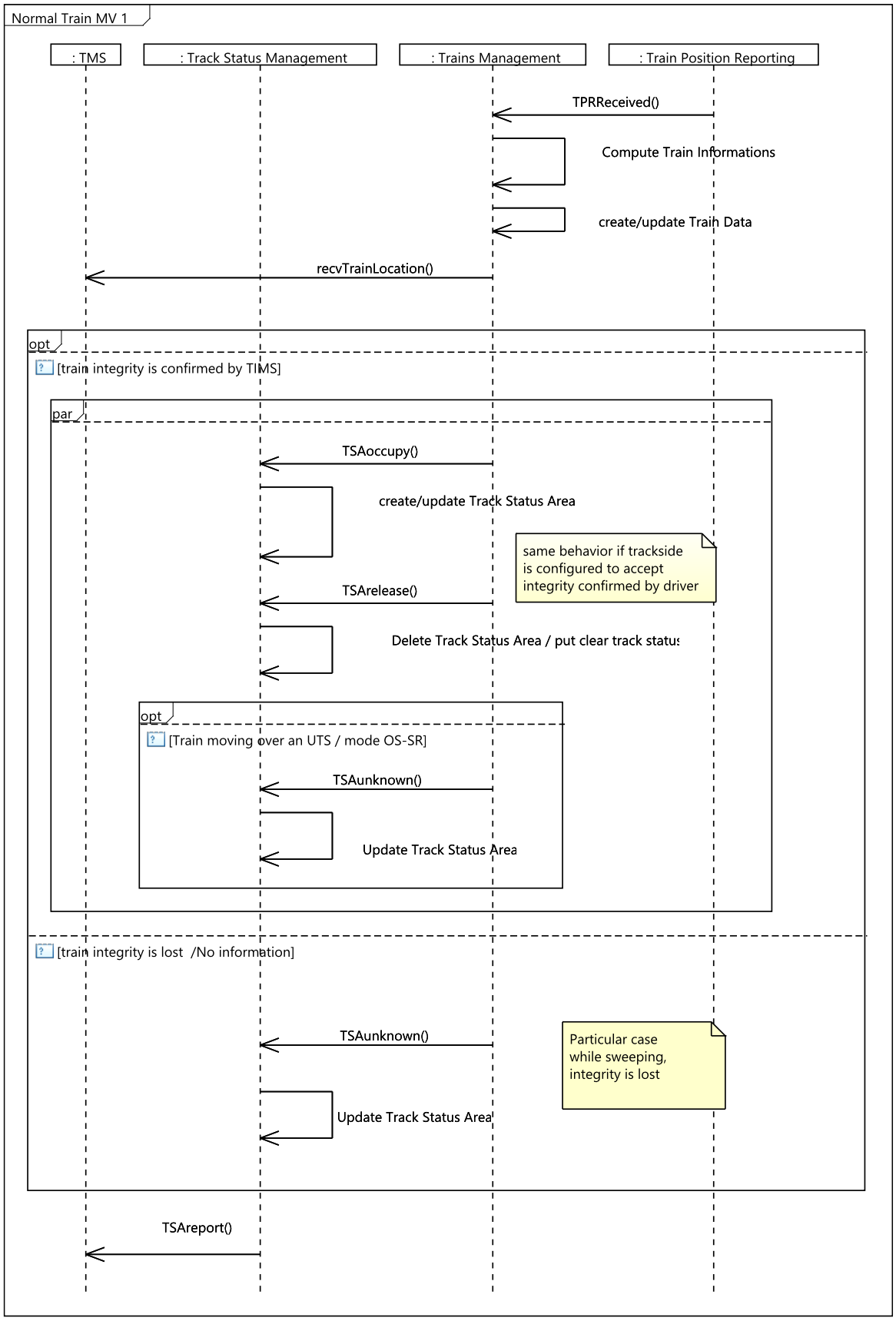


Fig. 6.17. Normal Train Movement Sequence Diagram - receive TPR from a train.

In the same use case, the alternative scenario of an $'MA\_request'$ raised by the *TMS* is considered; the corresponding sequence diagram is reported in Fig. 6.18. In this scenario, the message is received by the *Route Manager*, which asks for setting the points to the *Points Manager*. This latter component locks the points (after ensuring that they are in a clear Track Status Area, and they are not included in a Reserved Status Area) and reports the status to the *Route_Management*. Under the hypothesis that the points are effectively locked, The *Route Manager* locks the route and asks for the creation/update of a Reserved Status Area, which is at last reported to the *TMS*.

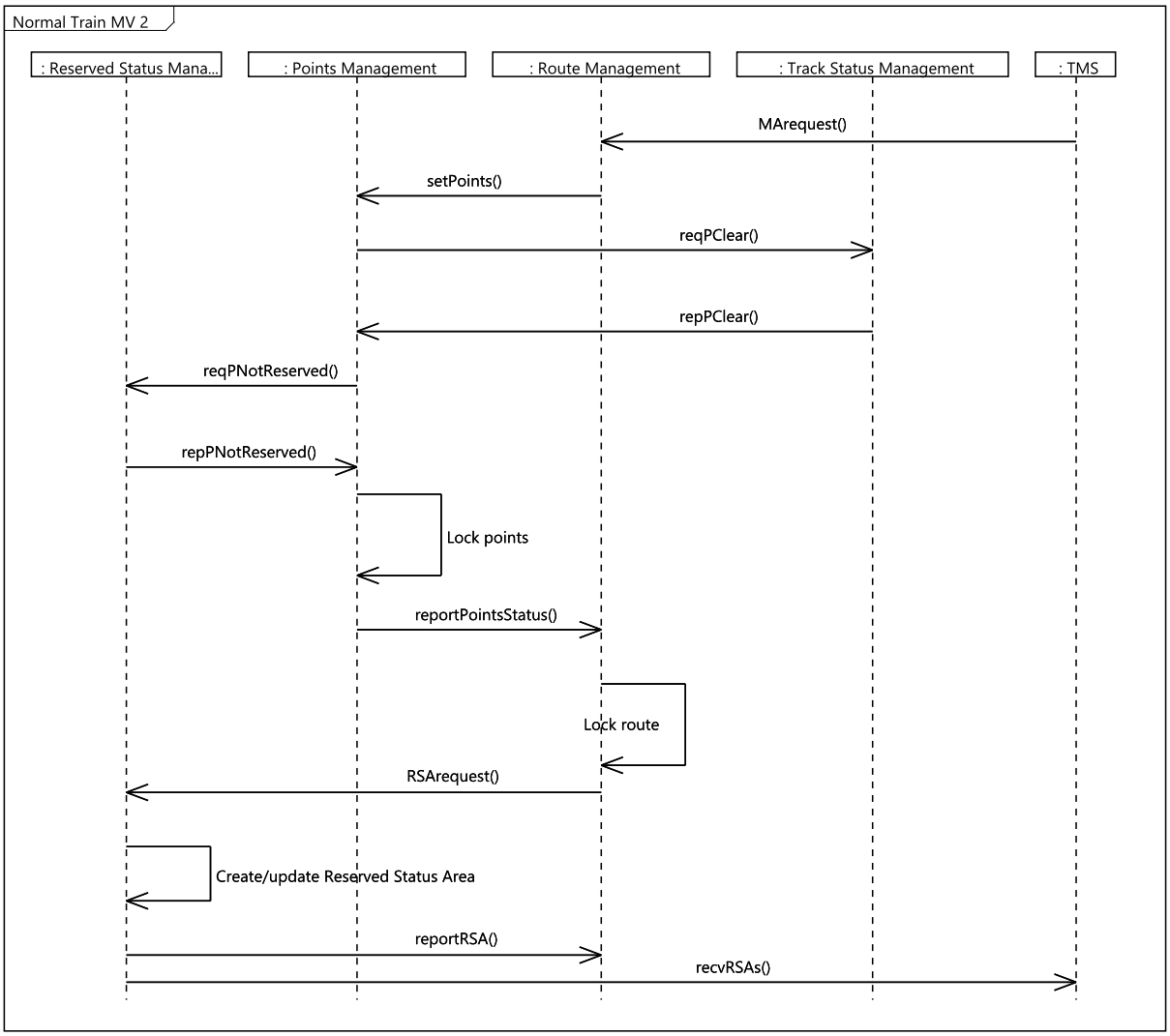


Fig. 6.18. Normal Train Movement Sequence Diagram - update Train.

A third possible scenario in this use case considers a route extension request from the *Route Manager*. The corresponding sequence diagram is reported in Fig. 6.19, and it depicts the actions of creating/updating the MA and, after the recalculation, its delivery to the *Dynamic Speed Profile Manager* component of the train and the update of the *TMS*.

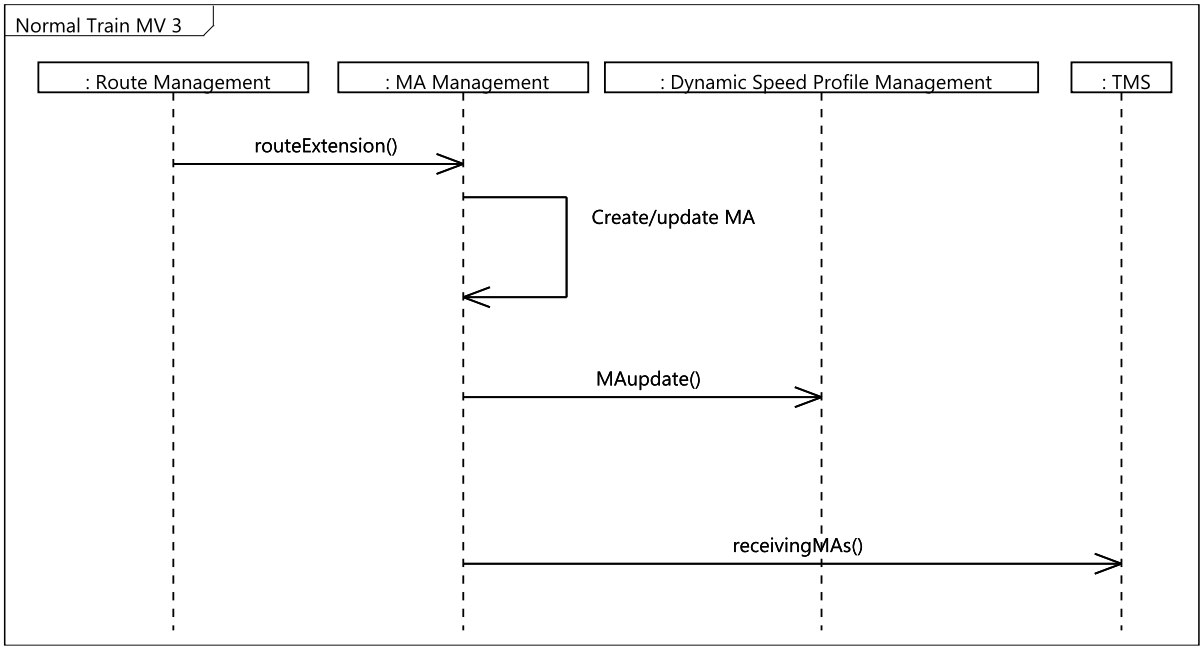


Fig. 6.19. Normal Train Movement Sequence Diagram - unexpected situations.

6.3.3. On Sight Movement

The *On-Sight Movement* use case allows the trains to enter an occupied line for the purpose of joining or checking for infrastructure defects. The movement in On-Sight mode cannot be selected by the driver, but shall be entered automatically when commanded by trackside and all necessary conditions are met. The On-Sight Movement EUC is the UC where the train moves in On-Sight mode. This EUC is described using a RD, a SD and a UCD.

There is no dedicated section for On-Sight requirements in [42]. To deal with this issue, a search for the keywords On-Sight/OS/On Sight is performed. As a result, six requirements are identified and are represented by a SysML RD, as shown in Fig. 6.20.

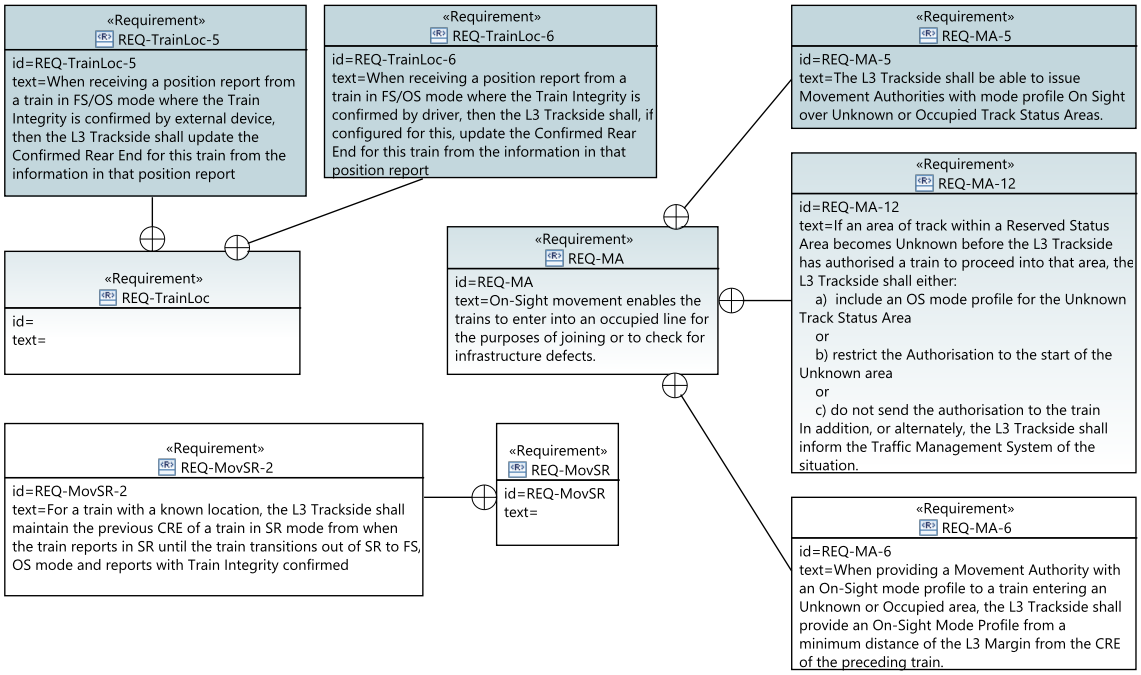


Fig. 6.20. On-Sight Requirement Diagram.

The UCD is designed by analysing the requirements of the RD. It is then refined using the OS procedure described in subset-26 part 5 page 35. It is important to note that the OS UC was not described in [4]. It was reported that the On Sight Movement use case was merged with the Sweeping use case since the On-sight L3 features are related to the sweeping functionality. To this aim, the description of Sweeping UC reported in [4] has been analysed. The external actors detected are *TTD*, *TMS*, *Driver* and *Dispatcher*. The related internal functions detected are *Route_Management*, *TTD_Management*, *Track_Status_Management*. The UCD is represented in the Figure 6.21. In this figure, the elements coloured in magenta are not reported in the description of Sweeping UC in [4]. Indeed, from the requirement “REQ-TrainLoc-5”, the TIMS (external device) and the On-board functions *Train_Position_Reporting* and Integrity_Information_Management are identified. From the requirement “REQ-MovSR-2”, the *Train Localization Unit* is detected. From the requirement “REQ-MA-12”, the Trackside functions *MA_Management* and *Reserved_Status_Management* are detected. Finally, from the procedure OS described in subset-26, the On-board functions *Speed_and_Distance_Supervision* and *Manage_Dynamic_Speed_Profile* are identified.

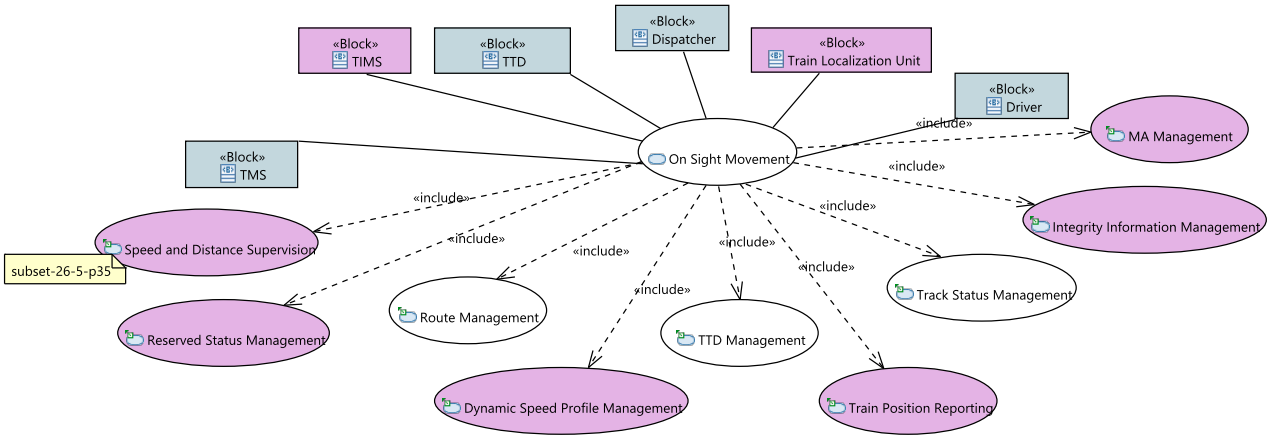


Fig. 6.21. On-Sight UC Diagram.

Two SDs related to the UC are designed by analysing the RD: the first describes the transition from the SR mode to the OS mode and the second reports an example of interactions within the OS mode.

The first SD is depicted in Fig. 6.22. The TIMS sends a message $'TIIreceived()'$ to the *"Integrity_Information_Management"* function which computes integrity information and sends a message $'integrityInfoRecv()'$ to the *Train_Position_Reporting* function. The latter computes the train position report and sends it in a message to the *Trains_Management* which updates the Confirmed Rear End (CRE) and the Maximum Safe Front End (MaxSFE) of the train. In the case that the train, moving in FS mode, is located in the rear of an adjacent unknown/occupied track status area, the *MA_Management* function computes an OS mode profile for this train and issues an MA with mode profile OS over the Unknown/Occupied Track Status Area (TSA) ($'MAupdate()'$ in Fig. 6.22). Then, the *Manage_Dynamic_Speed_Profile* function sends to the *Driver* a request for acknowledgement for OS mode ($'recvAckRequest()'$). This is represented by a synchronous message in the SD. The *Driver* sends a reply message Acknowledgement (ACK) for OS mode to the function *Manage_Dynamic_Speed_Profile*. Finally, the on-board system moves to the mode OS ($'Transition\ to\ OS\ mode'$ in the figure).

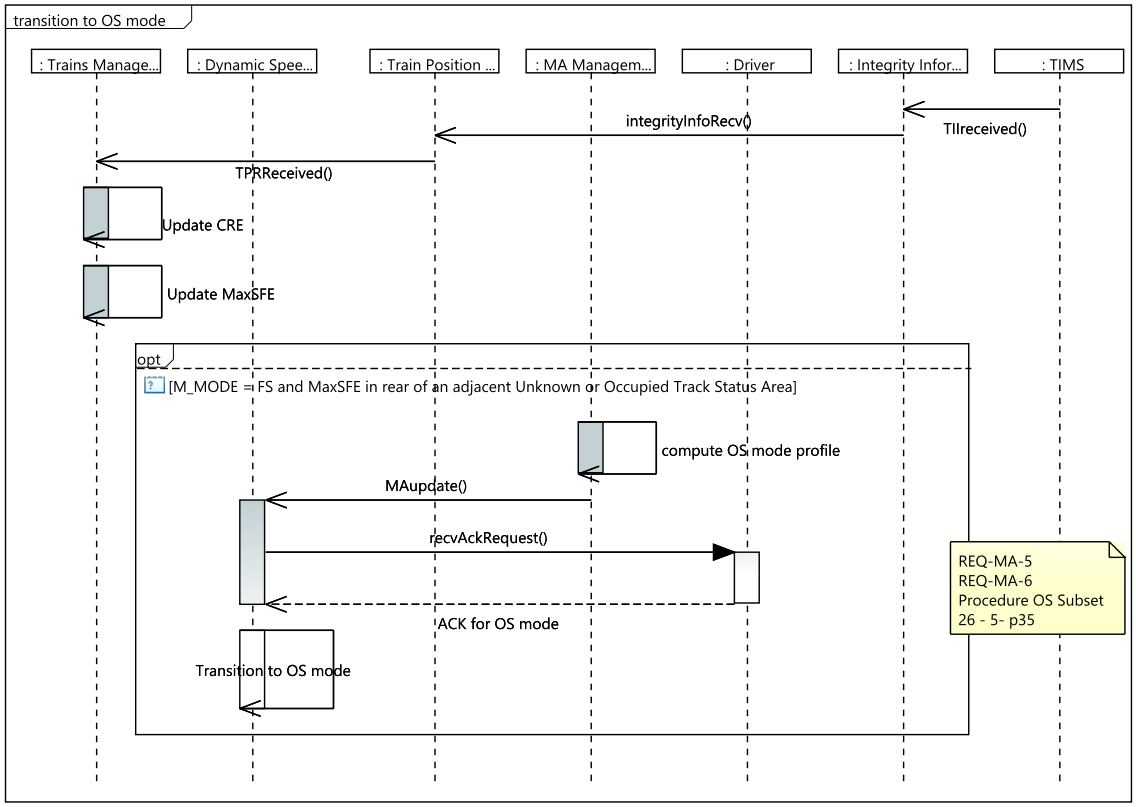


Fig. 6.22. On-Sight Sequence Diagram: Transition to OS mode.

The second SD is depicted in the Fig. 6.23, and it describes the behaviour of trackside and on-board systems to represent the requirements “REQ-TrainLoc-5” and “REQ-TrainLoc-6”. For the requirement “REQ-TrainLoc-5”, the *TIMS* sends a message $'TIIreceived()'$ to the *Integrity_Information_Management* function. The last computes integrity information and sends a message $'integrityInfoRecv()'$. The *Train_Position_Reporting* function sends a request for position $'positionRequest()'$ to the *TLU* which replies with the position $'positionReceived()'$. The *Train_Position_Reporting* function computes the train position report and sends it in a message to the *Trains_Management*. In the case that the mode is On-Sight and the integrity is confirmed by TIMS, the *Trains_Management* updates the CRE. For the requirement “REQ-TrainLoc-6”, the behaviour is similar to that of “REQ-TrainLoc-5”. The *Driver* sends a message $'integrityConfirmedByDriver()'$ to the *Integrity_Information_Management* function. In the case that the mode is OS, the integrity is confirmed by the Driver and the trackside is configured to accept driver confirmation, the *Trains_Management* updates the CRE.

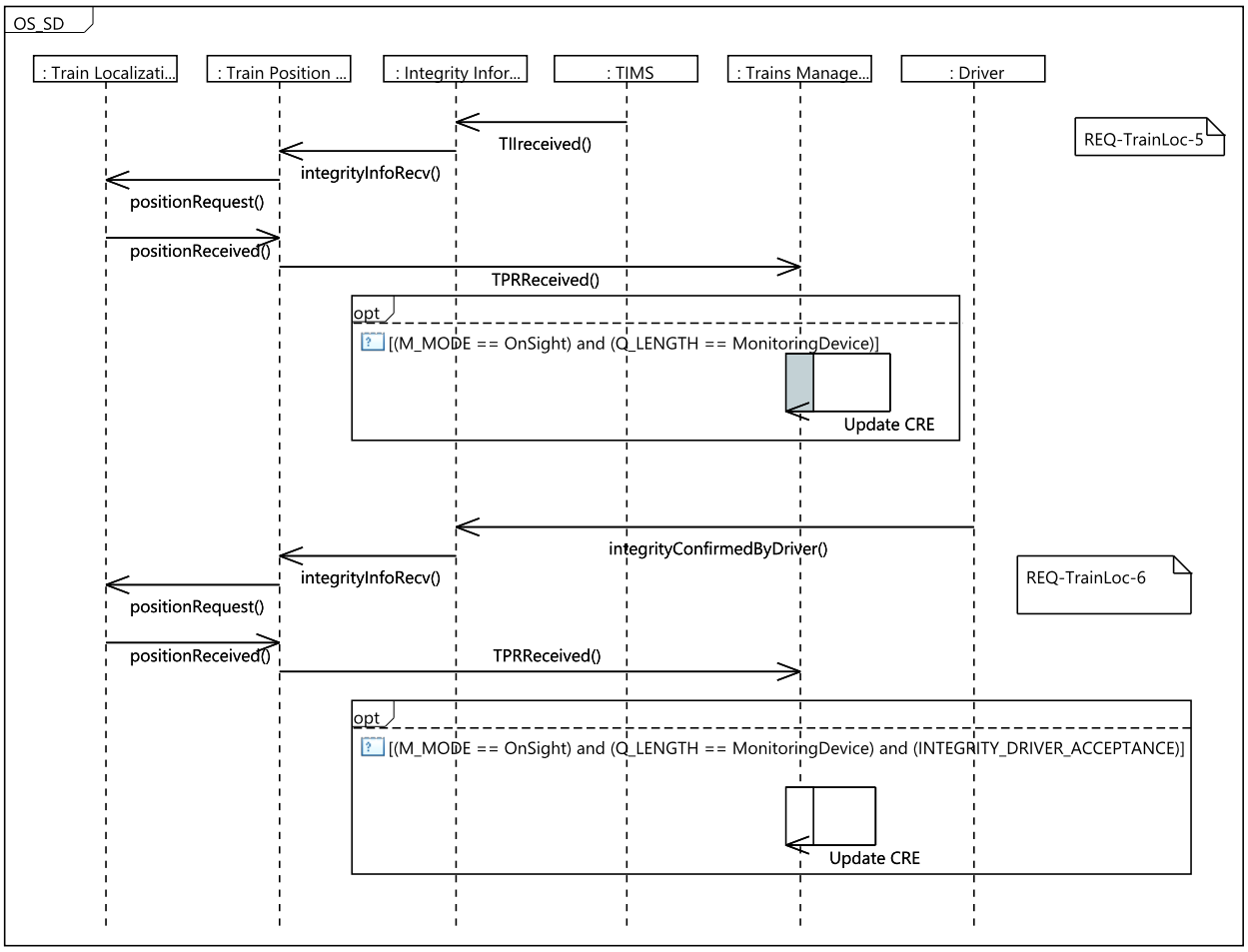


Fig. 6.23. On-Sight SD: Train in OS mode.

6.3.4. Loss of Train Integrity

The Loss of Train Integrity (LTI) EUC is a hazardous scenario. It corresponds to the situation where some wagons are unintentionally unleashed from the train, which may induce severe accidents if not timely detected. The train integrity is considered to be lost in either of the following cases: a loss of integrity is explicitly reported, or integrity is assumed to be lost, namely due to the expiry of the Integrity Wait Timer or in case new Validated Train Data are received. In fact, if a train splits unintentionally, the dispatcher needs to take appropriate measures, in particular, to prevent the collision of the dislocated part of the train with some following trains. It is also worth mentioning that under ETCS-L3 operation, the train integrity information has a significant impact on the performance of the line.

The LTI EUC is described through three SysML diagrams: RD (*"LTI_RD"*), SD (*"LTI_SD"*) and UCD (*"LTI_UCD"*).

To design the *"LTI_RD"*, ten requirements in the section Loss of Train Integrity (section 3.17) in [42] are identified. Detected requirements are represented by a SysML RD, as shown in Fig. 6.24.

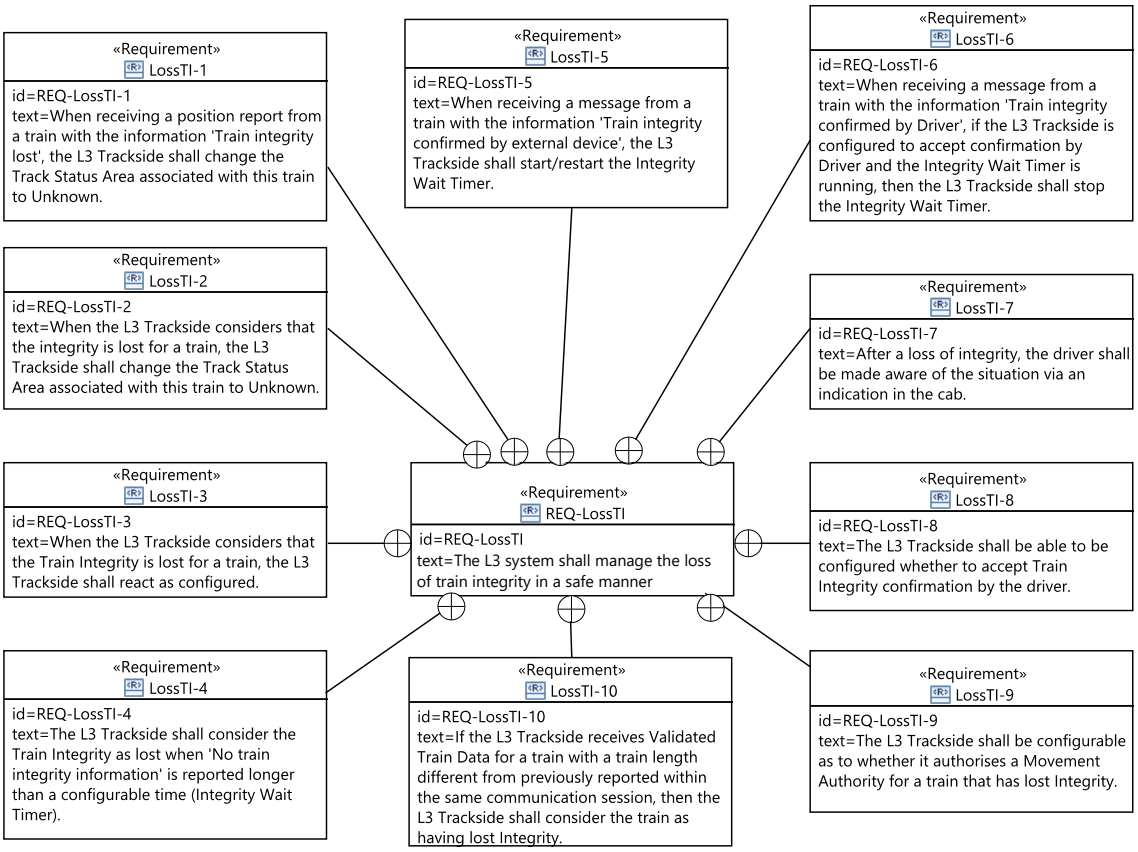


Fig. 6.24. Loss of Train Integrity Requirement Diagram.

The UCD related to LTI EUC is designed by analysing the related description reported in [4]. Then, it is refined using the *"LTI_RD"* (cf. Fig. 6.25). According to the requirement "REQ-LTI-5", the external device (*TIMS*) is missing in [4]. This external actor is represented with a magenta colour in Fig. 6.25 representing the *"LTI_UCD"*. The *TLU* and the trackside function "$Trains\_Management$" are also added to the *"LTI_UCD"*, and they are represented

with a magenta colour. Indeed, from the functional architecture, the *"Trains_Management"* function is in charge of receiving the train position report from the On-board system and sending messages to *"Track_Status_Management"* function in order to update the status of track areas. The *TLU* is required for computing the train localization included in the train position report.

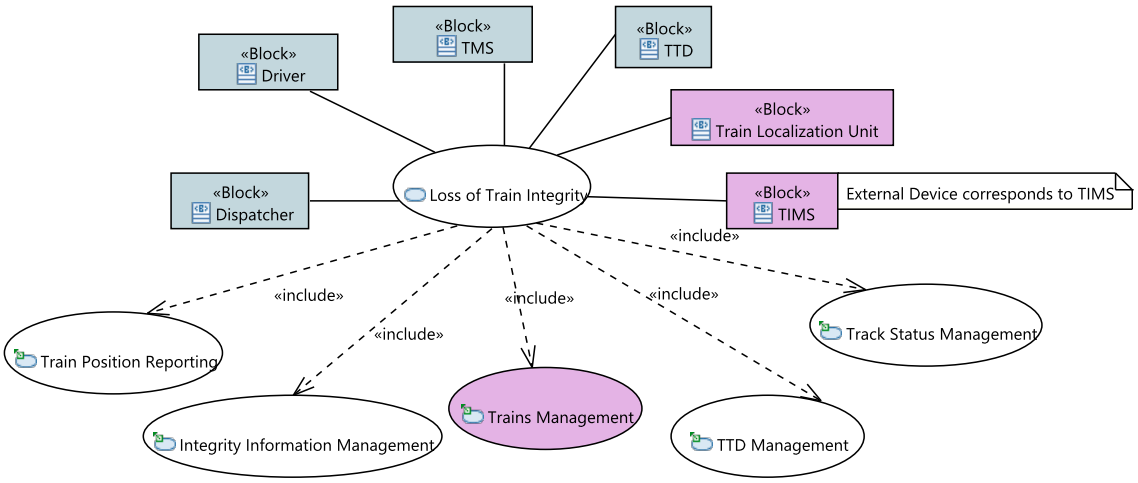


Fig. 6.25. Loss of Train Integrity UCD.

From the identified LTI related requirements, the interaction between LTI EUC and ETCS-L3 actors is depicted using two SysML SDs.

The first "LTI_SD" (see Fig. 6.26) represents an example of interactions of the LTI EUC with ETCS-L3 actors. The onboard function "$Integrity\_Information\_Management$" receives Train Information Management (TII) either from the external device *TIMS* ('$TIIreceived()$') or from the *Driver* ('$integrityConfirmedByDriver()$'). Then, the "$Integrity\_Information\_Management$" function computes the integrity status and sends it to the "$Train\_Position\_Reporting$" function ('$integrityInfoRecv()$'). Then, the "$Train\_Position\_Reporting$" requests position from the *Train Localization Unit* which computes train position and replies to the request ('$positionRequest()$' and '$positionReceived()$'). The "$Train\_Position\_Reporting$" sends the train position report to the "$Trains\_Management$" ('$TPRReceived()$').

The interactions between the LTI EUC and ETCS-L3 actors and ETCS-L3 internal functions depend on the status of integrity received in the TPR (Q_LENGTH). Indeed, the variable Q_LENGTH contains the status of integrity computed by the function "$Integrity\_Information\_Management$".

- $Q\_LENGTH = MonitoringDevice$ means that the integrity in confirmed by the TIMS. In this case, the "$Trains\_Management$" function restarts the wait integrity timer ("REQ-LossTI-5"), and computes the areas of track which are released/occupied by the train. Then it sends the information to the "$Track\_Status\_Management$" to update the track status areas ('$TSAoccupy()$', '$TSArelease()$').
- $Q\_LENGTH = Lost$ means that the integrity is lost, and it is reported by the TIMS. In this case, the $mathit"Trains\_Management"$ function computes the area of track which is unknown. Then, it sends the information to the "$Track\_Status\_Management$"

to update the track status area ($'TSAunknown()'$, “REQ-LossTI-1”). The *Driver* is aware of the situation by the “$Integrity\_Information\_Management$” Function ($'integrityInfoRecv()'$, “REQ-LossTI-7“)

- $Q\_LENGTH = NoInformation$ means that the integrity is unknown, and it is reported by the TIMS. In the case that the wait integrity timer expires, Trackside considers that the integrity is lost (“REQ-LossTI-4”). The “$Trains\_Management$” function computes the area of the track which is unknown. Then, it sends the information to the “$Track\_Status\_Management$” to update the track status area ($'TSAunknown()'$, “REQ-LossTI-2”) and it reacts as configured (“REQ-LossTI-3”).

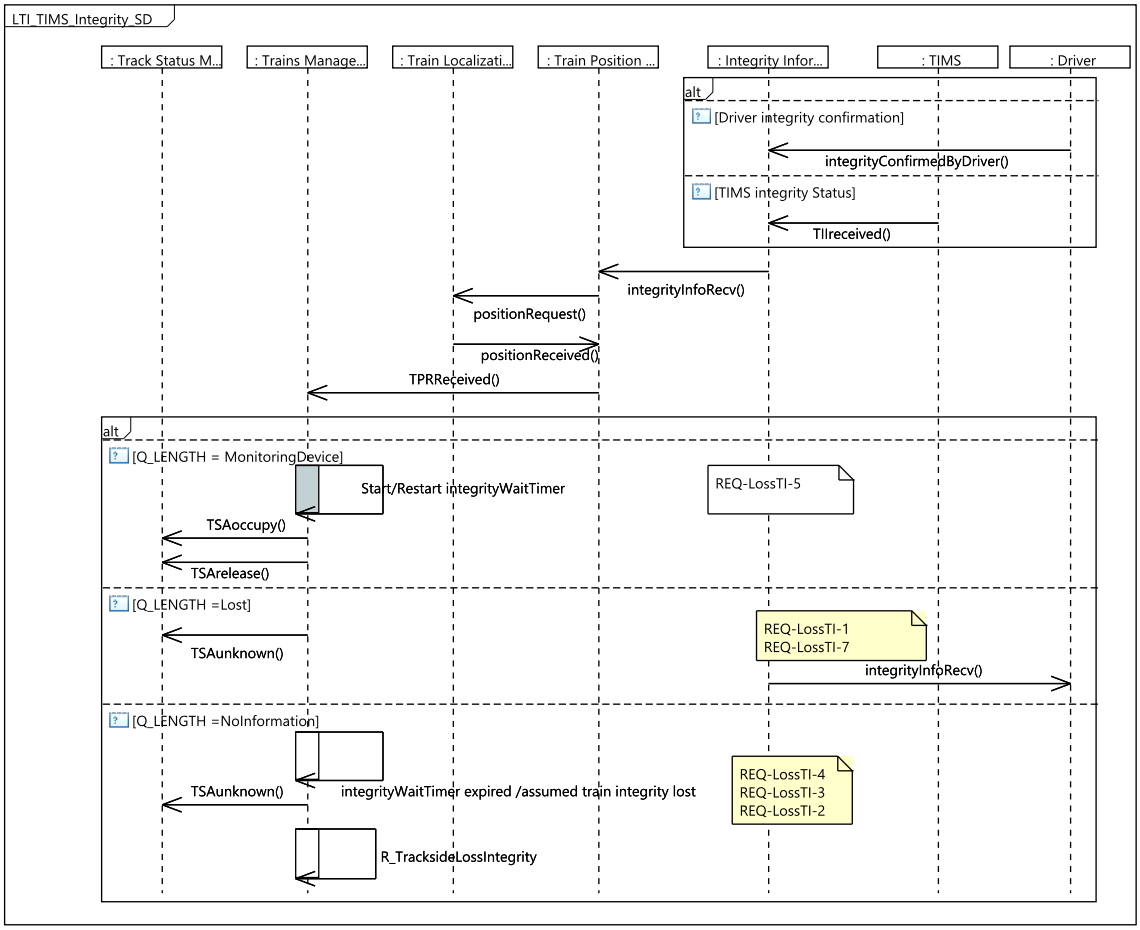


Fig. 6.26. Loss of Train Integrity Sequence Diagram 1.

The second “LTI_SD” (see Fig. 6.27) represents an example of interactions of the LTI EUC with ETCS-L3 actors in the case that the integrity is confirmed by the driver.

While modelling this sequence diagram, some missing requirements have been identified. For instance, if the trackside is not configured to accept train integrity from the driver, then its expected reaction is not specified. To address this issue, the following additional requirement is proposed: *if the trackside is not configured to accept a confirmation of the train integrity by the driver, then it shall react as if the received TPR does not hold any train integrity information*. This solution is, at the same time, consistent with guidance of “REQ-LossTI-4” (if the L3 Trackside is configured not to accept Train Integrity confirmed by Driver, and Train

Integrity confirmed by Driver is reported, then the L3 Trackside will treat this as “No train integrity information”) and inconsistent with guidance of “REQ-LossTI-8” (if confirmation of Integrity by the driver is not accepted, then the L3 Trackside can ignore any reports with Train Integrity Confirmed by Driver). The proposal is to delete the guidance of “REQ-LossTI-8”.

The description of interactions is as follows. If the Trackside is configured to accept integrity confirmation by the driver (“REQ-LossTI-8”), the “Trains_Management” function stops the wait integrity timer (“REQ-LossTI-6”). It computes the areas of track which are released/occupied by the train. Then it sends the information to the “Track_Status_Management” to update the track status areas ($'TSAoccupy()'$, $'TSArelease()'$). In the case that the Trackside is not configured to accept integrity confirmation by the driver, it reacts as receiving a position report with ”no train integrity information”.

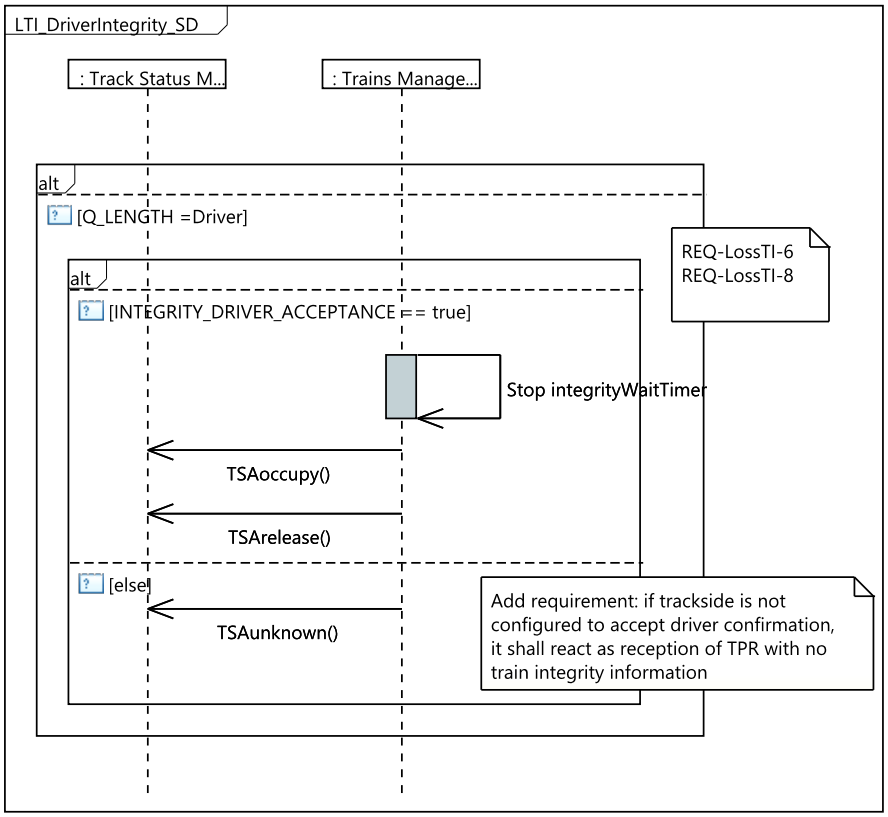


Fig. 6.27. Loss of Train Integrity Sequence Diagram 2.

It is important to underline that identifying missing requirements as early as possible is crucial since that has a positive impact during the subsequent engineering activities [45].

6.3.5. Staff Responsible

SR mode is the primary way to move a non-communicating train or a communicating train without a known location. The procedure to authorize the movement is out of the scope of this use case. The detailed requirements are given in a specific chapter of [42], leading to a Requirement Diagram with five requirements as depicted in Fig. 6.28.

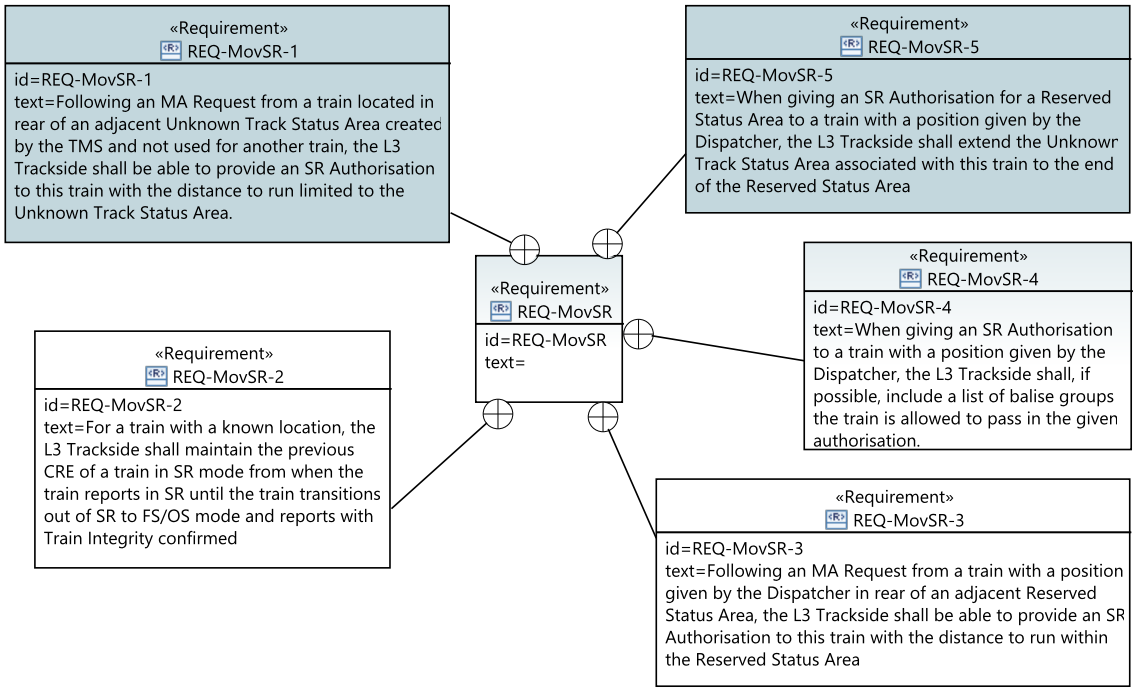


Fig. 6.28. Staff Responsible Requirement Diagram.

The UCD is reported in Fig. 6.29. The main external actors are the *Driver* and the *Dispatcher*. The use case has direct relationships with the other three use cases, as depicted in the figure.

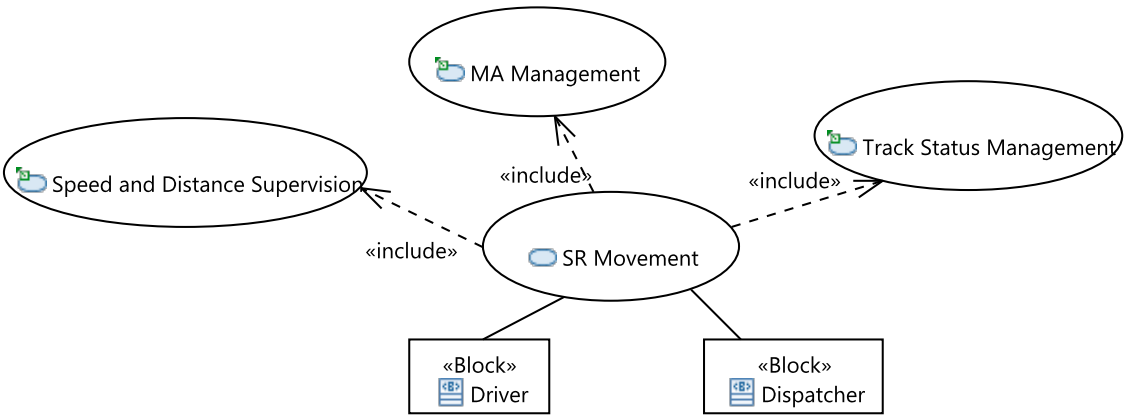


Fig. 6.29. Staff Responsible Use Case Diagram.

The behaviour of this use case has been modelled with the SD in Fig. 6.30. Following an

MA Request from a train in SR mode, the trackside has to provide an authorization modelled through the $'MAupdate'$ message. Three different scenarios are possible: the train is located in rear of an adjacent Unknown TSA created by the *TMS* and not used for another train; the train has a position given by the *Dispatcher*, and it is in the rear of an adjacent ; the train has a known location. In the first case, the distance to run is limited to the adjacent Unknown Track Status Area. In the second case, the distance to run is within the and, if possible, the trackside includes the list of the balise groups the train is allowed to pass in the given authorization. In the last case, the trackside maintains the previous CRE from when the train reports in SR until the train transitions out of SR to FS/OS mode and reports with Train Integrity confirmed; as also described in the Normal Train Movement, the corresponding TSA is set to UNKNOWN.

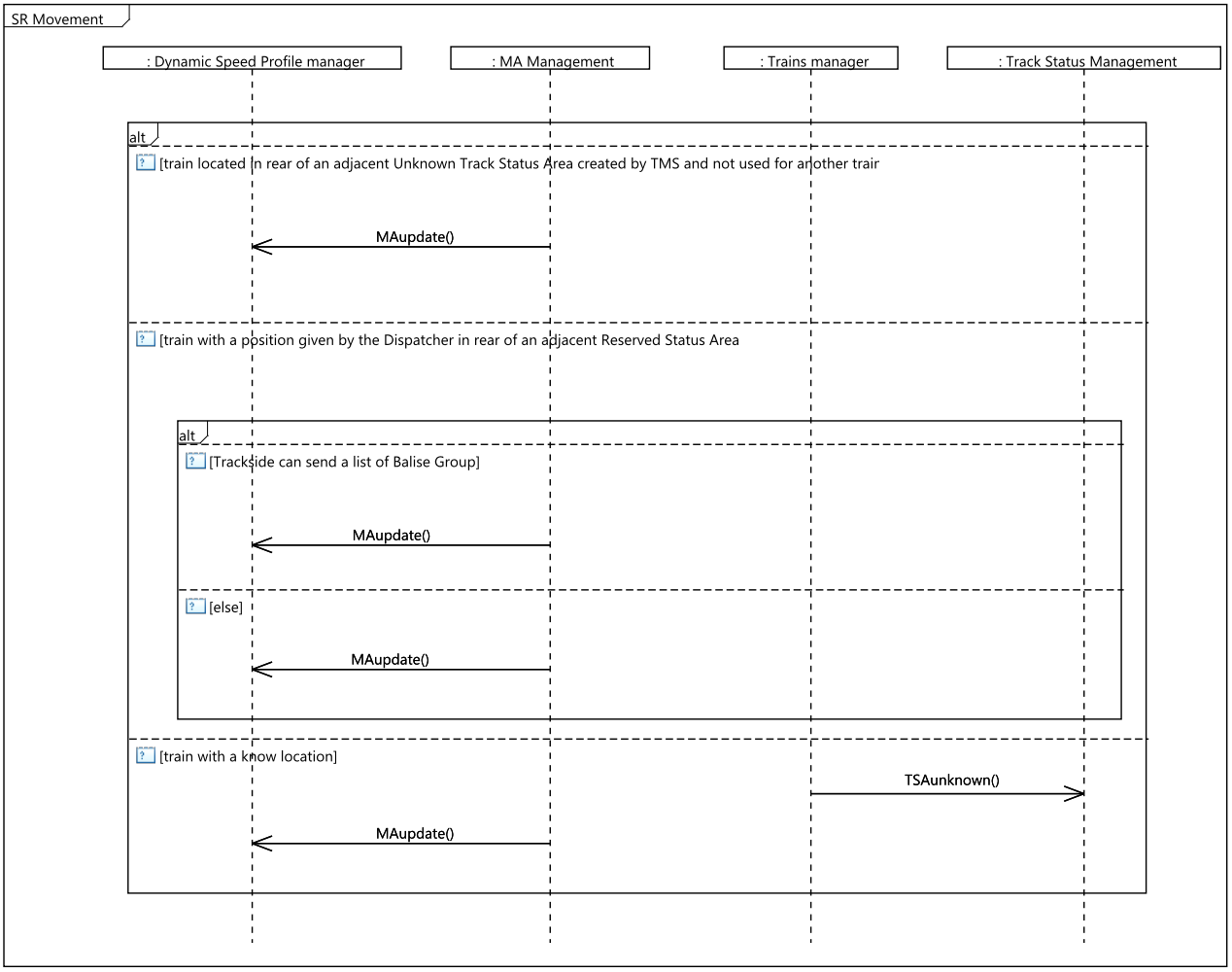


Fig. 6.30. Staff Responsible Sequence Diagram.

6.3.6. Points Control

An UCD is built and depicted in Fig 6.31: Points Control's EUC interacts with four trackside internal functions (i.e., Points Management, "*Reserved_Status_Management*", "*Track_Status_Management*" and "*Route_Management*") and TMS as an external actor.

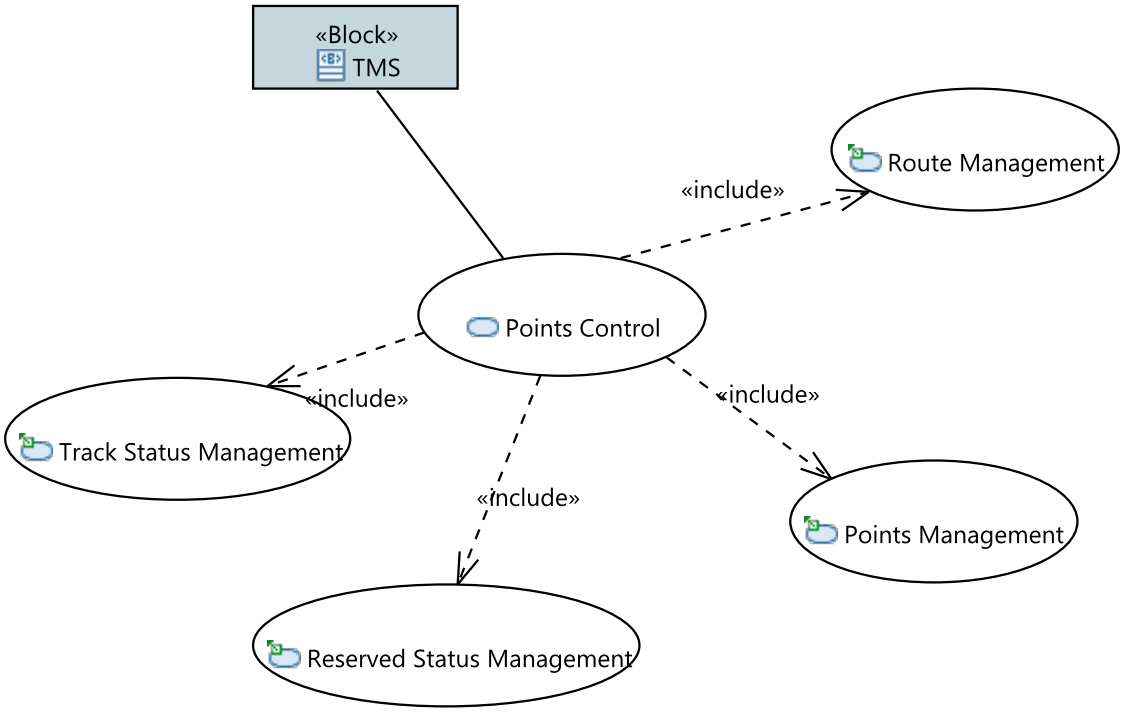


Fig. 6.31. Points Control UCD.

On the base of the UCD and by analysing the reported requirements, three scenarios are detected.

Nominal scenario: referring to "REQ-PTS-1" and "REQ-PTS-2", it describes the interactions between the different actors when the TMS just requests the creation of a route (see Fig. 6.32).

Degraded scenario: referring to "REQ-PTS-3", it describes the exchange of messages between TMS and the "Points_Management" function when *TMS* forces the release of some points (see Fig. 6.33).

Sweeping scenario: referring to "REQ-PTS-4", it describes the case of a sweeping train that frees some points: in this case, the Point Management internal function communicates with the "Track_Status_Management" to restore the correct status of an area (see Fig. 6.34).

According to the description of the scenarios already provided, Fig. 6.35 depicts a RD where the four considered requirements for this EUC are reported as well as their satisfying model elements.

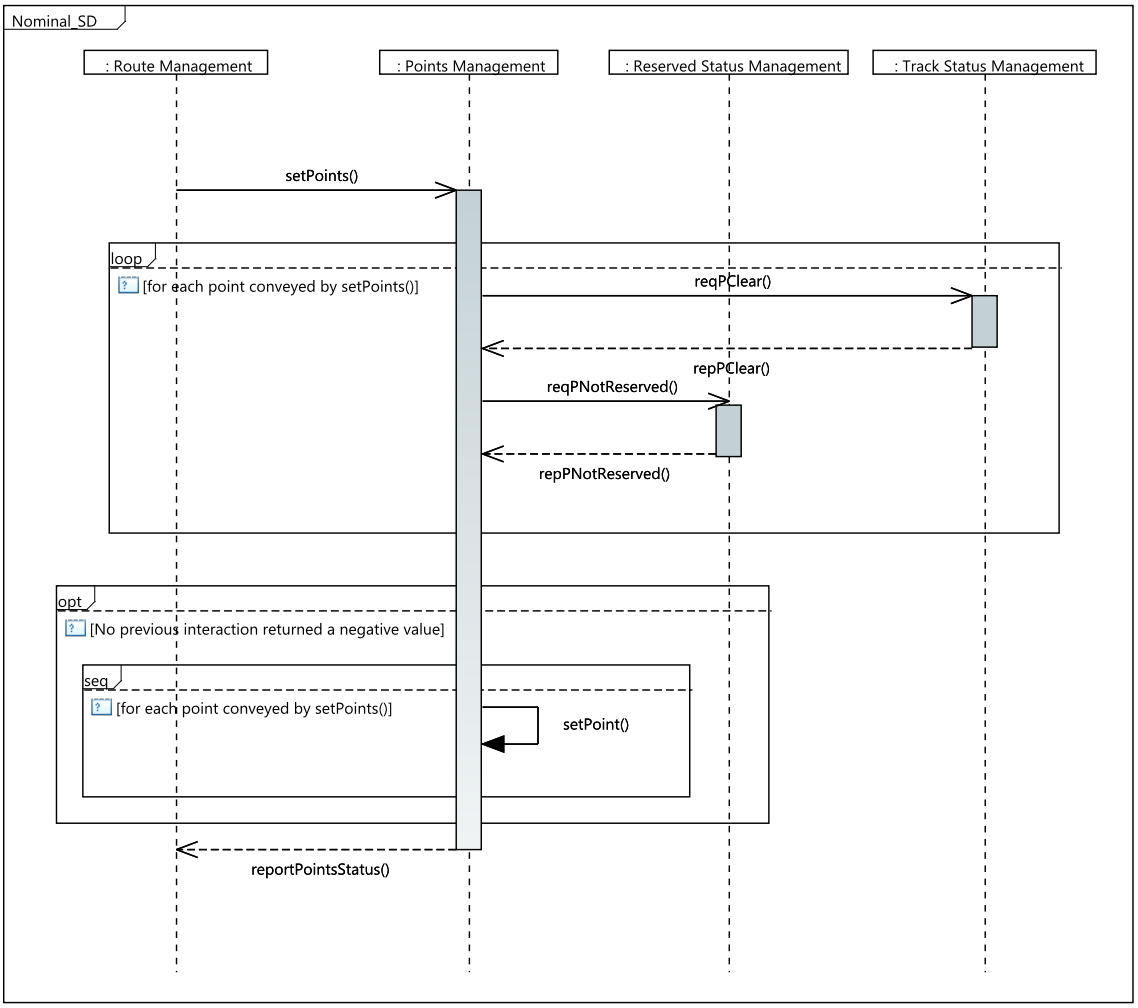


Fig. 6.32. Points Control SD (nominal scenario).

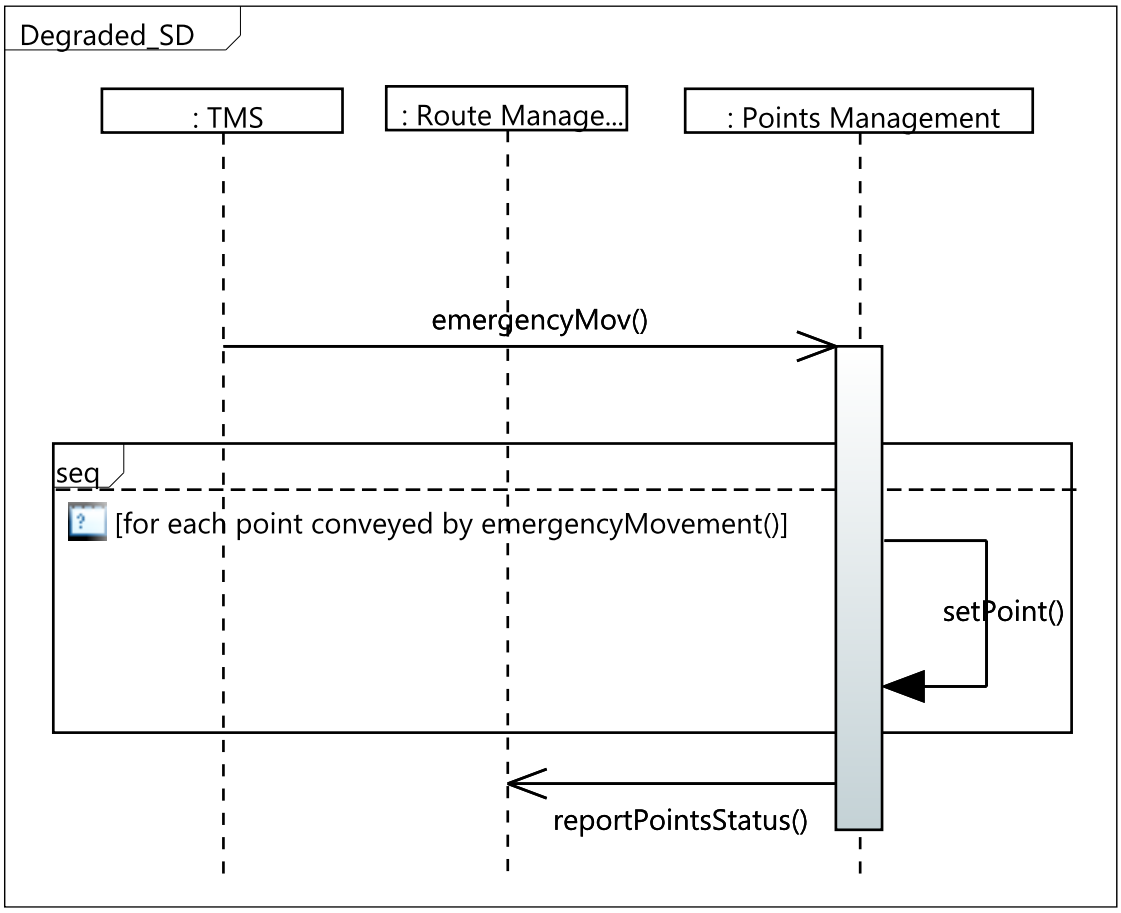


Fig. 6.33. Points Control SD (degraded scenario).

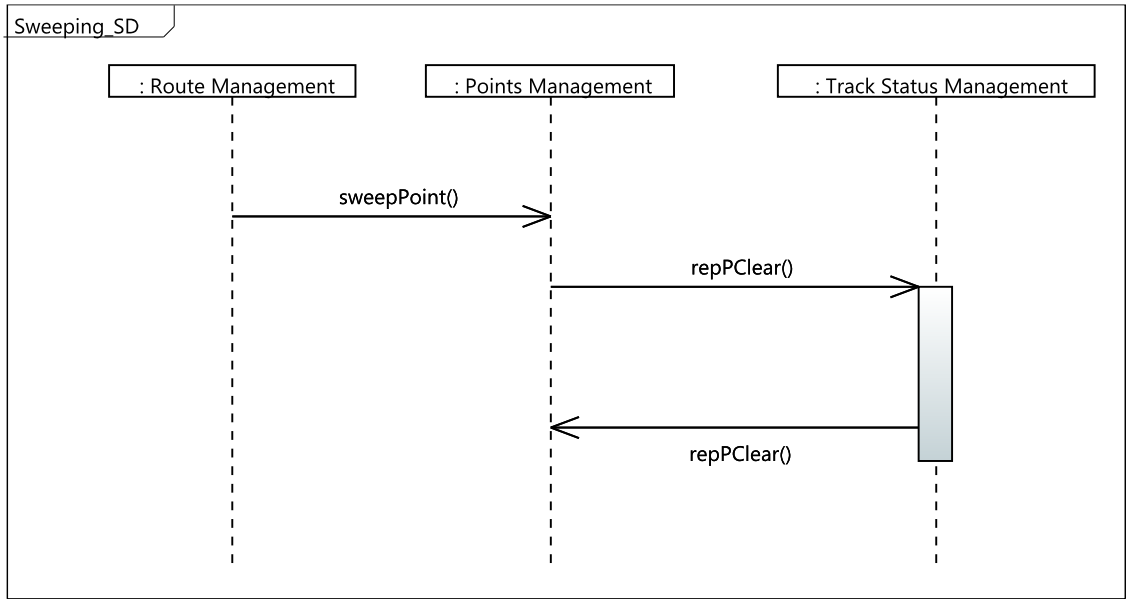


Fig. 6.34. Points Control SD (sweeping scenario).

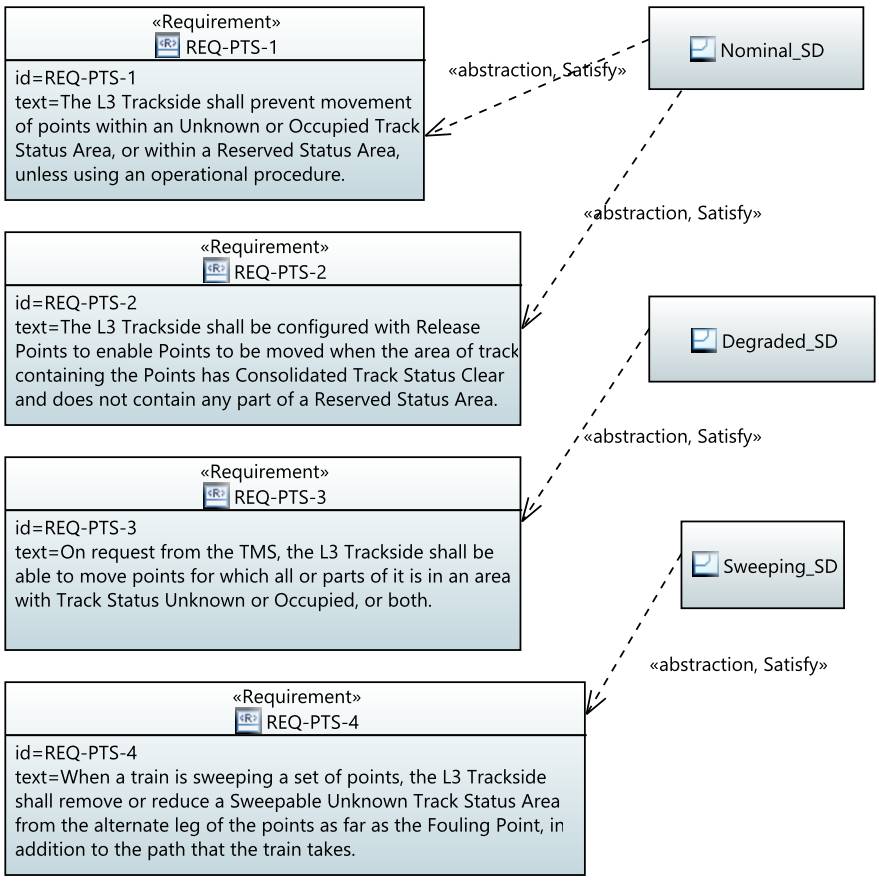


Fig. 6.35. Points Control allocation RD.

6.3.7. Sweeping

Sweeping is that state of the system where a train, that is authorized by the L3 Trackside in OS or SR mode, moves into a track area that is in the unknown state. The objective of this functionality is to have a train that "cleans" the area safely. The Sweeping EUC is the UC implementing this functionality. It is described using three SysML diagrams: RD, SD and UCD.

Since no dedicated section for sweeping requirements is found in [42], a search for keywords swept/sweeptable/sweeping is performed. As a result, eight requirements are identified and are represented by a SysML RD, as shown in Figure 6.36.

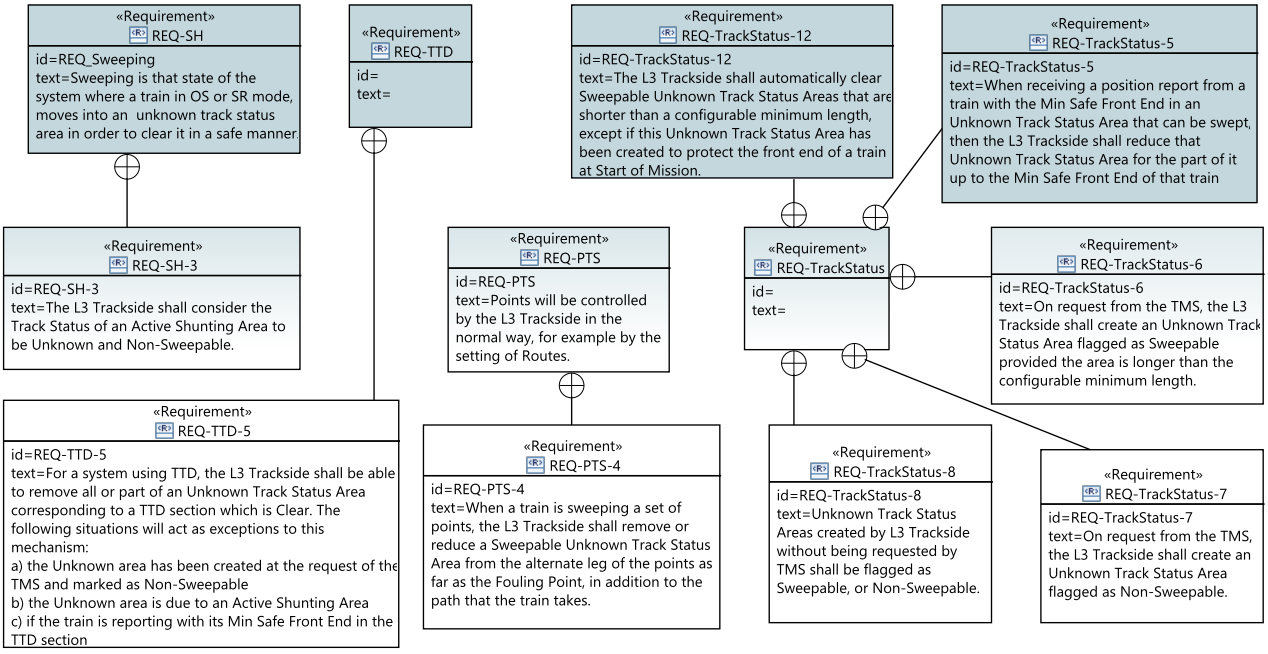


Fig. 6.36. Sweeping Requirement Diagram.

The UCD related to Sweeping EUC is designed by analysing the description of Sweeping EUC reported in [4]. Then, it is refined using the Sweeping RD (cf. Figure 6.36). According to the requirement "REQ-TrackStatus-5", the On-board function "$Train\_Position\_Reporting$" is missing in [4]. This function is represented in magenta colour. The train position report is received by the function "$Trains\_Management$" which is also added in magenta colour in Fig. 6.37 representing the Sweeping UCD.

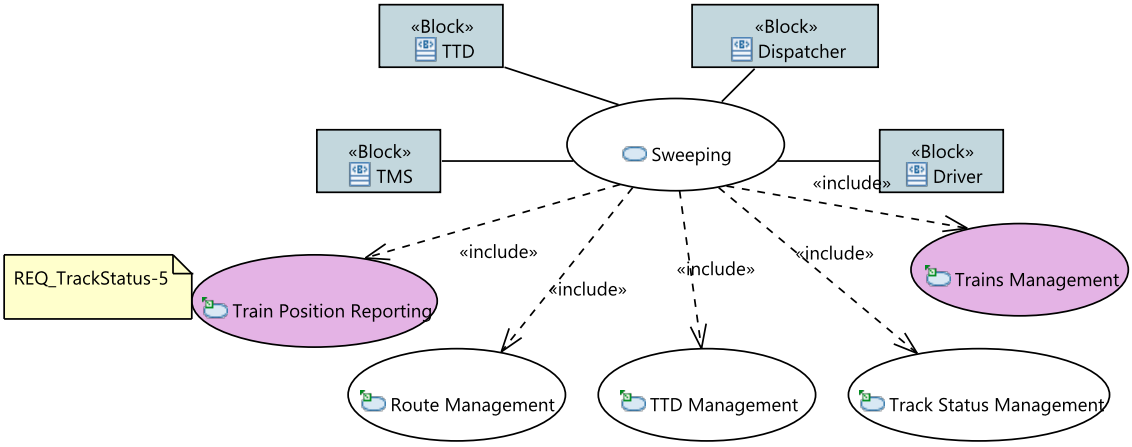


Fig. 6.37. Sweeping UC Diagram.

From the identified requirements, the interaction between Sweeping EUC and ETCS-L3 actors and ETCS-L3 functions is depicted using two SysML SDs. The first "$Sweeping\_SD$" is depicted in the Figure 6.38. The interaction is related to the requirements "REQ-TrackStatus-6" and "REQ-TrackStatus-7": the *TMS* sends a request to "$Track\_Status\_Management$" to create an Unknown TSA ('$unknownTSA()$'). If the area is sweepable, the "Track_Status_Management" computes the length of TSA. If the length is longer than a configurable minimum length (MIN_LEN_UNKNOWN), the "Track_Status_Management" creates the TSA. Else, nothing happens. If the area is non-sweepable, the "Track_Status_Management" creates the UTSA without any length verification.

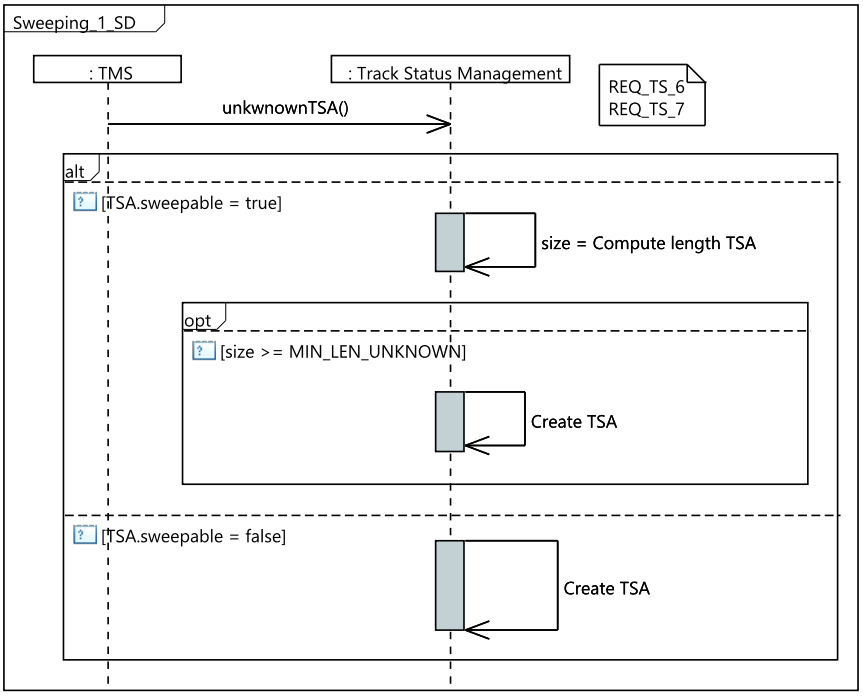


Fig. 6.38. Sweeping Sequence Diagram: TMS request.

The second “Sweeping_SD” is depicted in Fig. 6.39. The interaction is related to the requirements “REQ-Shunting-3” and “REQ-TrackStatus-5”.
For the requirement “REQ-Shunting-3”, The *TMS* sends a request to the “Track_Status_Management” to enable a shunting Area ($'enableDisableShuntingArea()'$). The “Track_Status_Management” sets this area as unknown and non-sweepable.
For the requirement “REQ-TrackStatus-5”, the “Train_Position_Reporting” function sends a TPR to “Trains_Management” function ($'TPRReceived()'$). The “Trains_Management” computes the Minimum Safe Front End (minSFE). In the case that the minSFE is located in an unknown track status area (UTSA), the “Trains_Management” computes the new size of the UTSA and sends a request ($'unknownTSA()'$) to “Track_Status_Management” function in order to reduce it. The “Track_Status_Management” updates the TSA with the received information.

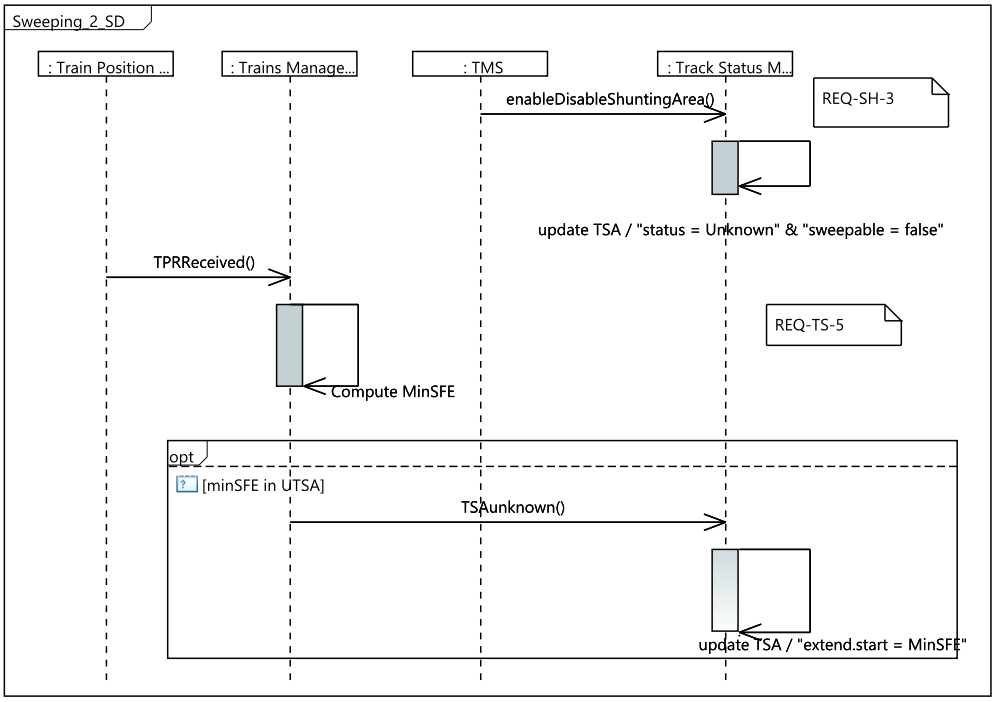


Fig. 6.39. Sweeping Sequence Diagram.

6.3.8. Loss of Communication

The Loss/Restore of Communication use case is related to the degraded mode occurring when the L3 Trackside loses the supervisions of one or more trains under its control. L3 Trackside manages a train losing its connection and involves the train status management and the track status. The use case also considers the actions that need to be taken to restore the communication between on-board and trackside subsystems. This use case assumes that the Train is not currently, and does not enter subsequently, in a RadioHole and has not been sent Reversing Area Information. Loss of communication in these last two situations are dealt with by dedicated Use Cases.

The specification requirements relevant to this use case have been extracted from [42] and depicted in the requirement diagram reported in Figure 6.40.

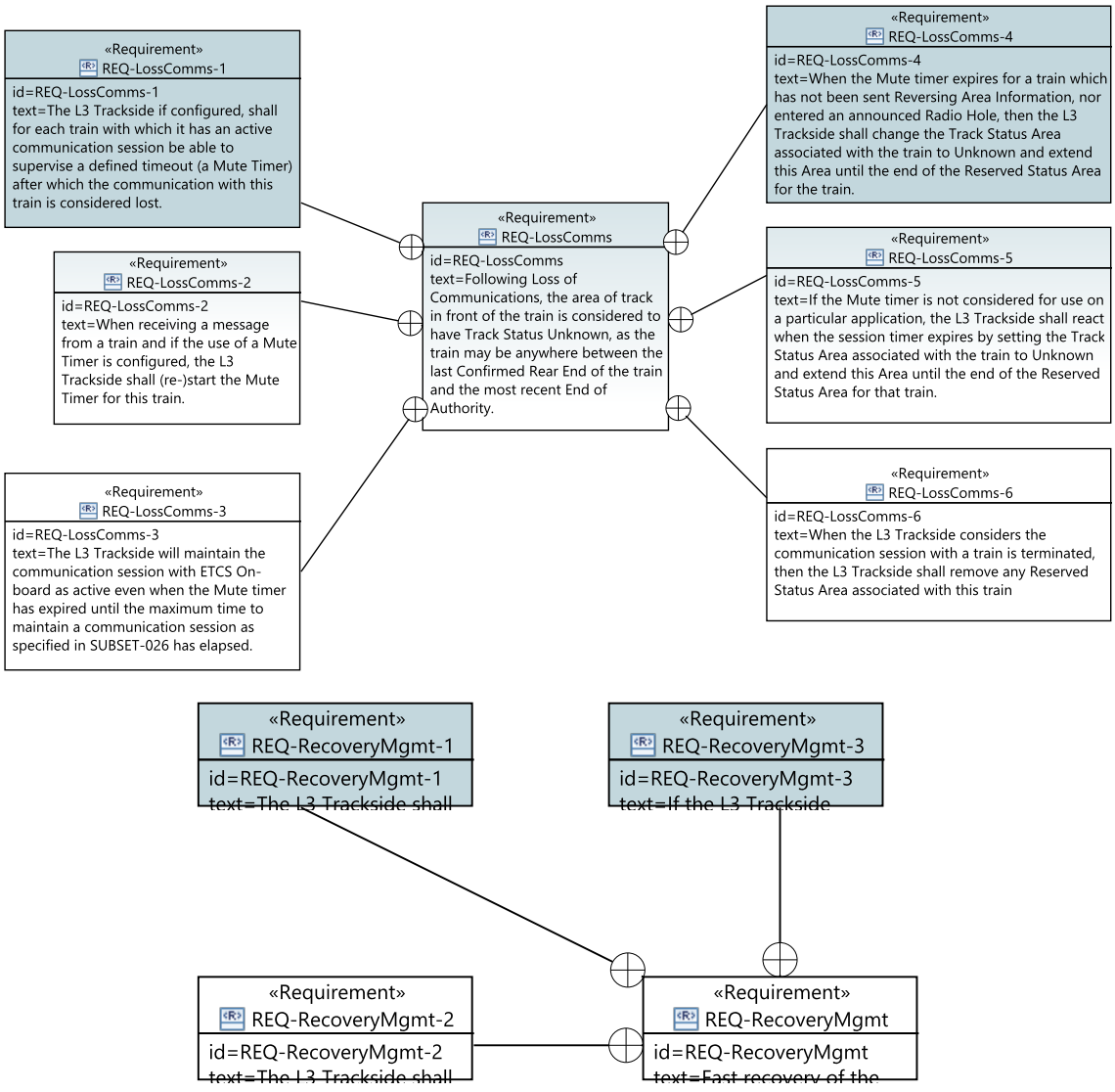


Fig. 6.40. Loss of Communication Requirement Diagram.

The Use Case Diagram for Loss of Communication is designed on the basis of the require-

ments reported above and is illustrated in Figure 6.41. The external actors that partake in the use case are the Traffic Management System (<<Block>> TMS) and the Trackside Train Detection (<<Block>> TTD). In addition, the Infrastructure Manager actor has been included according to the Engineering Rule ENG-LossComm-1 in Section 2.14.2 of [46]. The functions involved, instead, are Communication Management, Track Status Management, Train position reporting, TrainsManagement and Reserved Status Management (see the <<include>> relation).

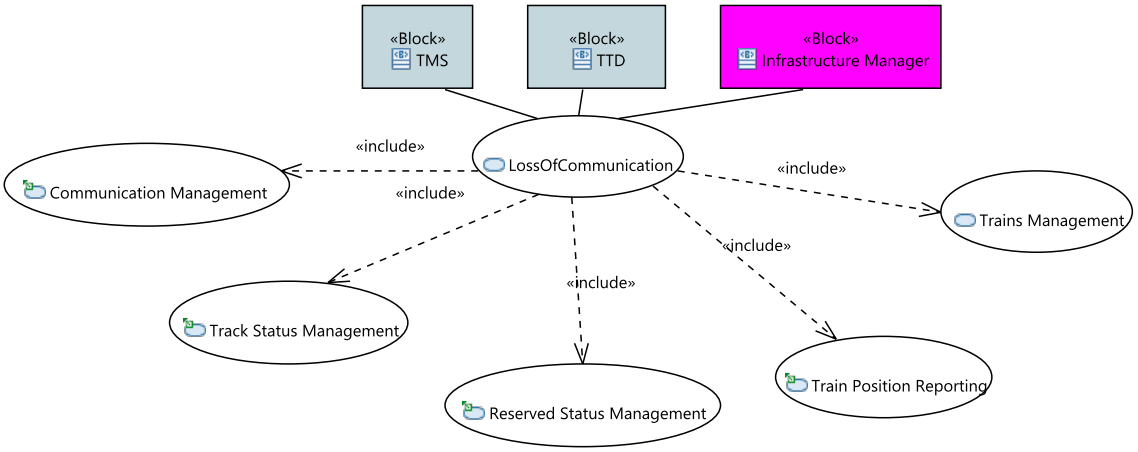
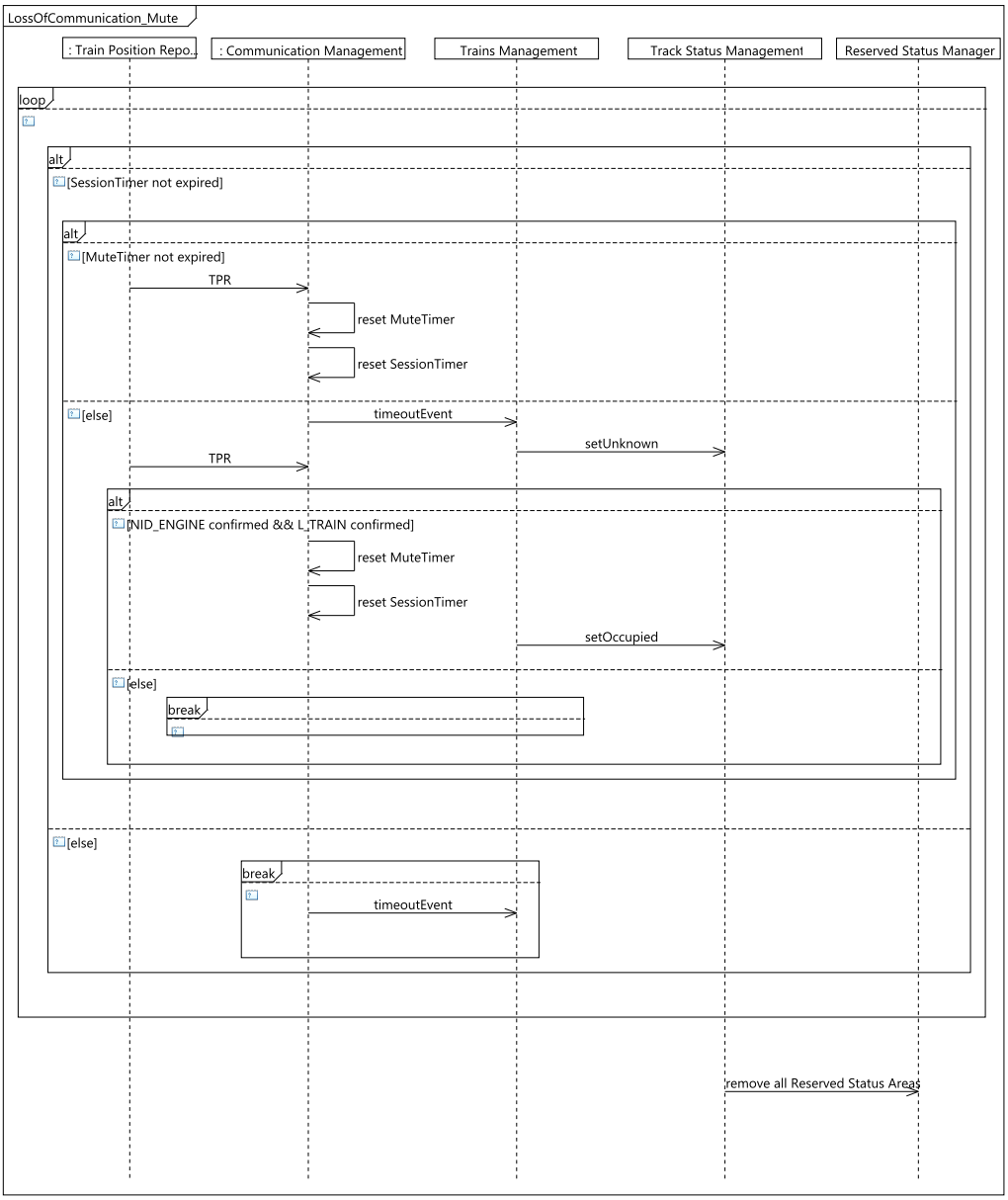
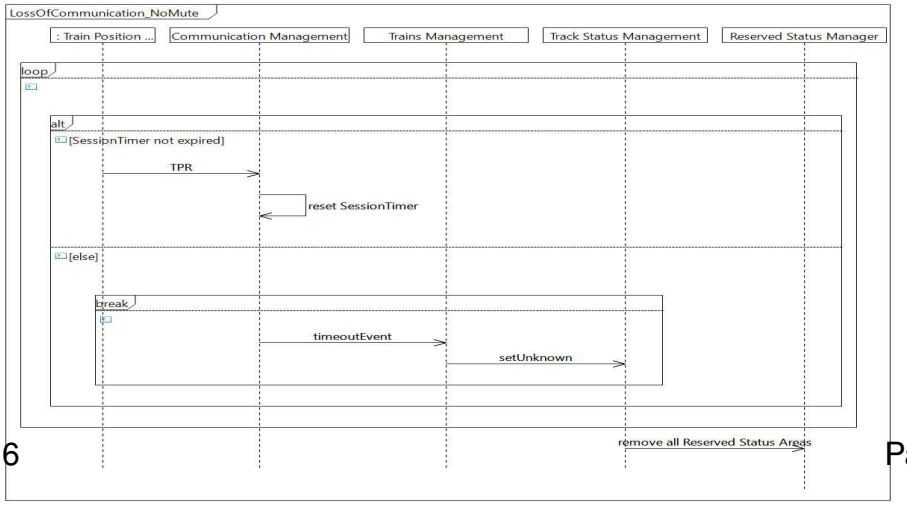


Fig. 6.41. Loss of Communication Use Case Diagram.

The detailed behaviour of the Use Case specified in the Sequence Diagrams reported in Figure 6.42. According to requirements in Figure 6.41, for each connected train, the L3 Trackside has a Session Timer and can be configured to also depend on a Mute Timer. It is assumed that the timeout for the Mute Timer, if configured, is smaller than the one for the Session Timer. Figure 6.42 illustrates the behaviour of the Use Case both when the Mute Timer is configured (top diagram) and when it is not (bottom diagram). When the Mute timer is configured, every time a valid message is received from the train, the L3 Trackside resets both timers. When the timeout for the Mute Timer triggers, the communication with this train is considered lost; in this case, the area of the track in front of the train is considered to have Track Status Unknown. If the communication session is restored before the Session Timer expires, the Track Status will be recovered, after L3 Trackside has checked that TRAIN_ID and TRAIN_LENGTH, sent in the Train Position Report, have not changed. In this case, Track Status is cleared according to the presence or not of TTD and MA assignment can proceed according to the other functional requirements. If the communication session is not restored in time, or communication is restored, but the train is not recognised as the same train, then the Track Status will remain Unknown and any reserved status area for the train is released. Since the Mute Timer is not mandatory, if not used, the SESSION_EXPIRED_TIMEOUT plays its role. When this timeout triggers, the effects of the track status and reserved status are the same as the ones described when the Mute Timer is configured.


LossOfCommunication_Mute
: Train Position Repo...
: Communication Management
Trains Management
Track Status Management
Reserved Status Manager
loop
alt
[SessionTimer not expired]
alt
[MuteTimer not expired]
TPR
reset MuteTimer
reset SessionTimer
[else]
timeoutEvent
setUnknown
TPR
alt
[NID_ENGINE confirmed && L_TRAIN confirmed]
reset MuteTimer
reset SessionTimer
setOccupied
[else]
break
[else]
break
timeoutEvent
remove all Reserved Status Areas


LossOfCommunication_NoMute
: Train Position ...
Communication Management
Trains Management
Track Status Management
Reserved Status Manager
loop
alt
[SessionTimer not expired]
TPR
reset SessionTimer
[else]
break
timeoutEvent
setUnknown
remove all Reserved Status Areas

6.4. The Trackside Behaviour

This subsection presents the SysML sub-models of the ETCS-L3 trackside internal functions, using SMDs.

6.4.1. Track Status Management

The "Track_Status_Management" function is the trackside function in charge of determining the Track Status within its Area of Control. The $Track\_Status$ represents the information held within the trackside about whether the track is occupied or not; any area of track within the Area of Control can be OCCUPIED, UNKNOWN or CLEAR. The Track Status is OCCUPIED if the trackside considers that there is a communicating train on the track with $Train\_Integrity$ confirmed. Instead, it is CLEAR (also known as FREE) when the trackside considers that there is no obstacle on the track. At last, the $Track\_Status$ is UNKNOWN when the trackside is unsure whether there is a train or an obstacle on the track, or it is certain that there is a train, but it does not know the location of the train within the area. Hence, the *'Unknown'* state is used both for a specific train and also for any other reason, not associated with a train (e.g., created by the Dispatcher, or corresponding to an Active Temporary Shunting Area). An Unknown Track Status Area created by the Dispatcher can be Sweepable or Non-Sweepable. Among the states, in case of overlap, the function gives precedence to Occupied over Unknown as it is more restrictive.

Track Status can be considered as a collection of Track Status Areas, each of which can be Occupied or Unknown. Any area of track that is not covered by a Track Status Area (Occupied or Unknown) is considered as Clear. An Occupied Track Status Area is possible only when the trackside receives at least one TPR, with Train Integrity confirmed. An Unknown Track Status Area is consequent to several reasons:

- the trackside has received at least One Position Report, but the Train Integrity is lost or never confirmed;
- the trackside is no longer receiving the Train Position Report (i.e., the communication is lost);
- the trackside has received an End of Mission;
- the Dispatcher has created an Unknown Track Status Area (Sweepable or Non-Sweepable) via the TMS;
- during the initialization procedure of the trackside, where all the Area of Control is Unknown;
- if present, the TTD input is Occupied, but there is no corresponding Train Position Report;
- the track within an Active Shunting Area is Unknown.

The objective of the function is to calculate the Consolidated Track Status of the entire track within the Area of Control according to the current state of all Track Status Areas. In addition, it sends information to the Points manager. Therefore, "Track_Status_Management" is modelled by two state machines. The machine in Figure 6.43 models the global behaviour of the Track Status manager. The safe handling of overlapping Unknown track sections is in charge of the algorithm (do activity) calculating the Consolidated Status Area.

The machine in Fig. 6.44 describes the internal behaviour of a generic Track Status Area and must be instantiated for each Track Status Area. An assumption is that information is

available, allowing the association of the incoming signals to the specific track to which they are sent. The three possible entry points of the machine allow the trackside to immediately create an Occupied or Unknown (either Sweepable or Non-Sweepable) Track Status Area. Different transitions exist, which can vary the state of a Track Status Area or delete the information, enabling the trackside to consider the Area as Clear. Mainly, when the train integrity is confirmed, either by an external device or by the driver and the system is configured for this, the status is updated to Occupied. Similarly, after TMS requests, the state can be updated.

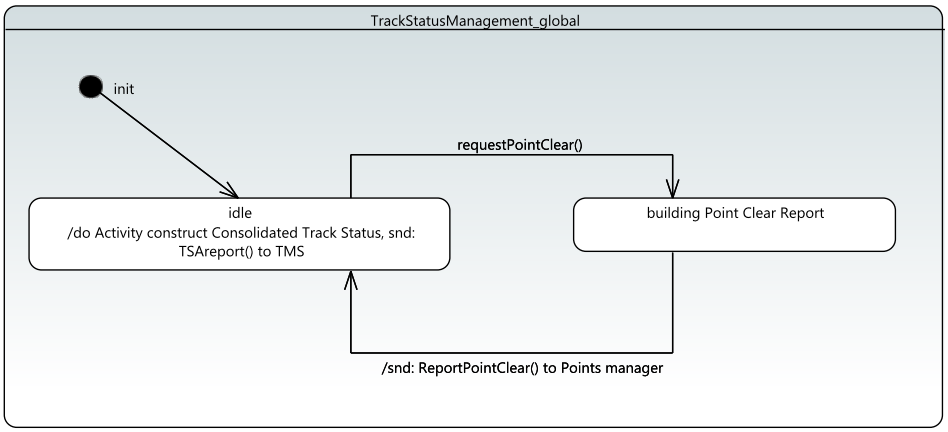


Fig. 6.43. State Machine Diagram for the “Track_Status_Management” (global).

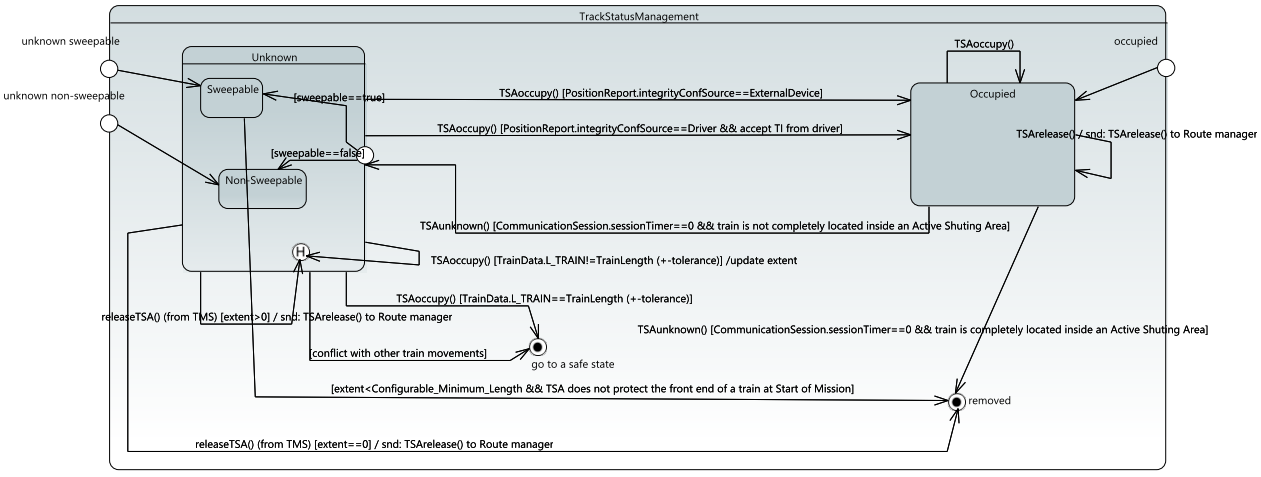


Fig. 6.44. State Machine Diagram for a single Track Status Area.

6.4.2. Reserved Status Management

The “Reserved_Status_Management” is a trackside function devoted to the reserving of an area into which the train is authorised to move. Specifically, it manages the information held within the trackside about whether the track is reserved for a train or not; if any part of the track within the Area of Control is reserved for a train, the Reserved Status is Reserved. An area of the track must be Reserved before the trackside authorizes a train to move through that area (i.e., before sending a $'Movement\_Authority'$ to a train for that area of the track, authorizing an SR movement or sending Route Related Information as part of handover to an adjacent trackside). The Reserved State is separate from the states defined for Track Status. Similarly to the Track Status, Reserved Status can be considered as a collection of Reserved Status Areas. A Reserved Status Area is always for a specific train, even if the Train ID is not known yet.

The management of a Reserved Status Area follows three possible causes:

- the trackside will issue an SR Authorization to a train;
- the trackside will issue a new or extended Movement Authority to a train;
- the trackside may retain a Reserved Status Area behind a train in a Reversing Area.

The individual Reserved Status Areas can overlap Track Status Areas which are Unknown or Occupied. Reserved Status Areas do not overlap with each other, and any area of track that is not covered by a Reserved Status Area is not reserved [42]. Reserved Status Areas can be created or extended, but also reduced following a request to shorten an authorisation, for example, if there is a cooperative shortening of a Movement Authority. After its removal, the corresponding area is not reserved.

The behaviour of the “Reserved_Status_Management” is represented by the SMD in Figure 6.45. When receiving an $'RSArequest'$ message, the state machine creates/updates each Reserved Status Area requested in the received message. For each create/update request, the internal, machine modelling a single Track Status Area is updated, and, if necessary, created or removed. After the complete update of the Reserved Status Areas, the state of the general state machine comes back to *’Idle’*, releasing the $'reportRSA'$ message. Similarly, when the machine receives the signal $'RSArelease'$, it behaves similarly. At last, when the machine receives the requestPointNotReserved message, it enters the state points requested, where it computes the locations of the points. After finishing the computation, the machines come back to *’idle’*, sending the reportPointNotReserved message.

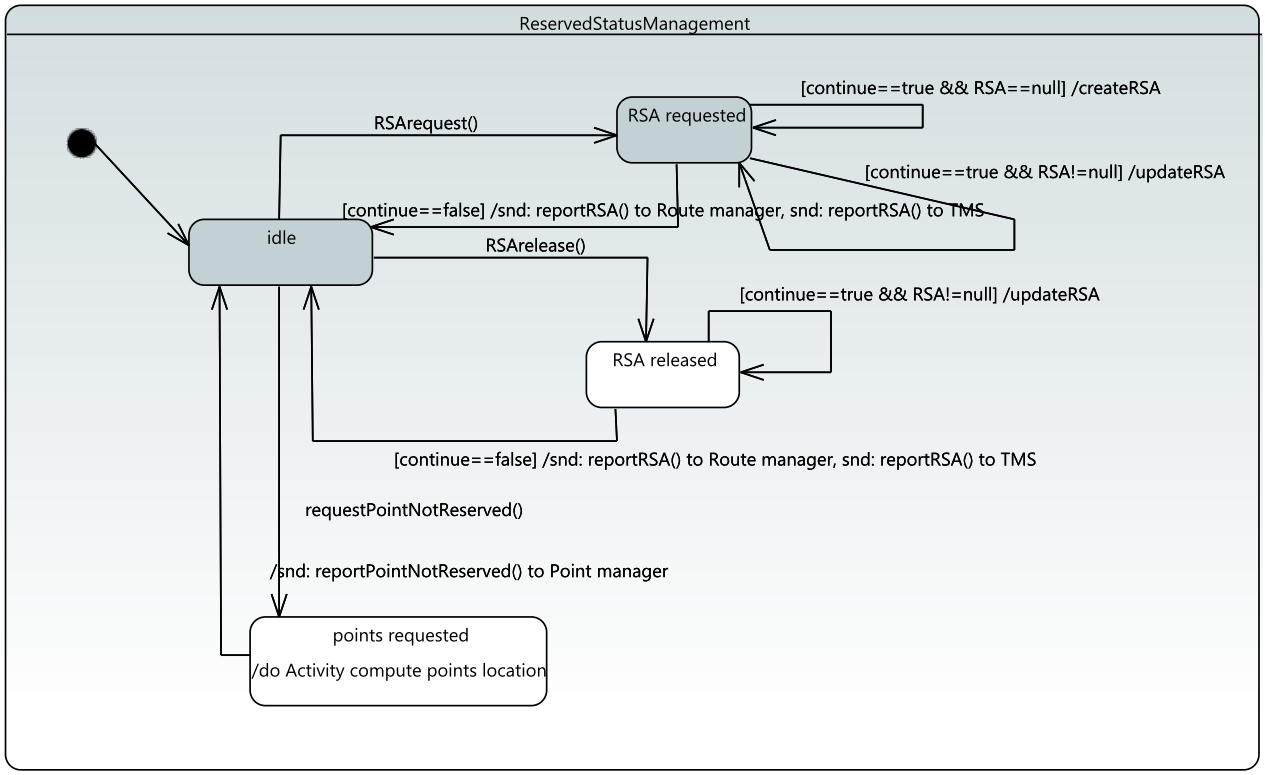


Fig. 6.45. State Machine Diagram for the Reserved Status Management.

6.4.3. Trains Management

The “Trains_Management” function is similar to the corresponding function in ETCS L2, but it takes different inputs [41]. The following main inputs are considered:

- The Position Report (TPR, from the ETCS On Board);
- The Validated Train Data (VTDR, from the ETCS On Board);
- The timeoutEvent (from the Communication manager).

The state machine modelling the internal behaviour of this function is reported in Figure 6.46. This state machine is instantiated for each train connected to the trackside. Each machine is supposed to communicate with the train whose NID_ENGINE is specified by the Train Data and locally stored by the “Trains_Management”. The machine is activated after the SoM (not modelled); similarly, termination after EoM is not modelled. The objective of the modelled function is to determine the train location and send the Track Status Manager the proper signals to update the Track Status Area (i.e., the Occupied, Unknown and Release signals). Sleeping and non-leading engines are not considered in this model. The signals to the Track Status manager are determined by checking the train length, updating the $Front\_End$ and the $Rear\_End$ of the train. State transitions are specified according to the requirements in [42], the transitions are enabled by the evaluation of conditions, including the availability of information about the train integrity. If an unexpected or conflicting position of the train is detected, an alert is sent to the TMS and a safe procedure must be performed. In case the communication with the train is lost (the mute timer is expired), this machine receives the $'timeoutEvent'$ from the “Communication_Management and waits in the state mute timer expired for a possible valid reconnection of the train. Waiting in this state, if the machine is alerted for a session timer expiration, then it asks the “Track_Status_Management” to release the Track Status Area. Similarly, if the TRAIN_INTEGRITY_WAIT timer expires, the machine asks the “Track_Status_Management” to set the specific Track Status Area as unknown.

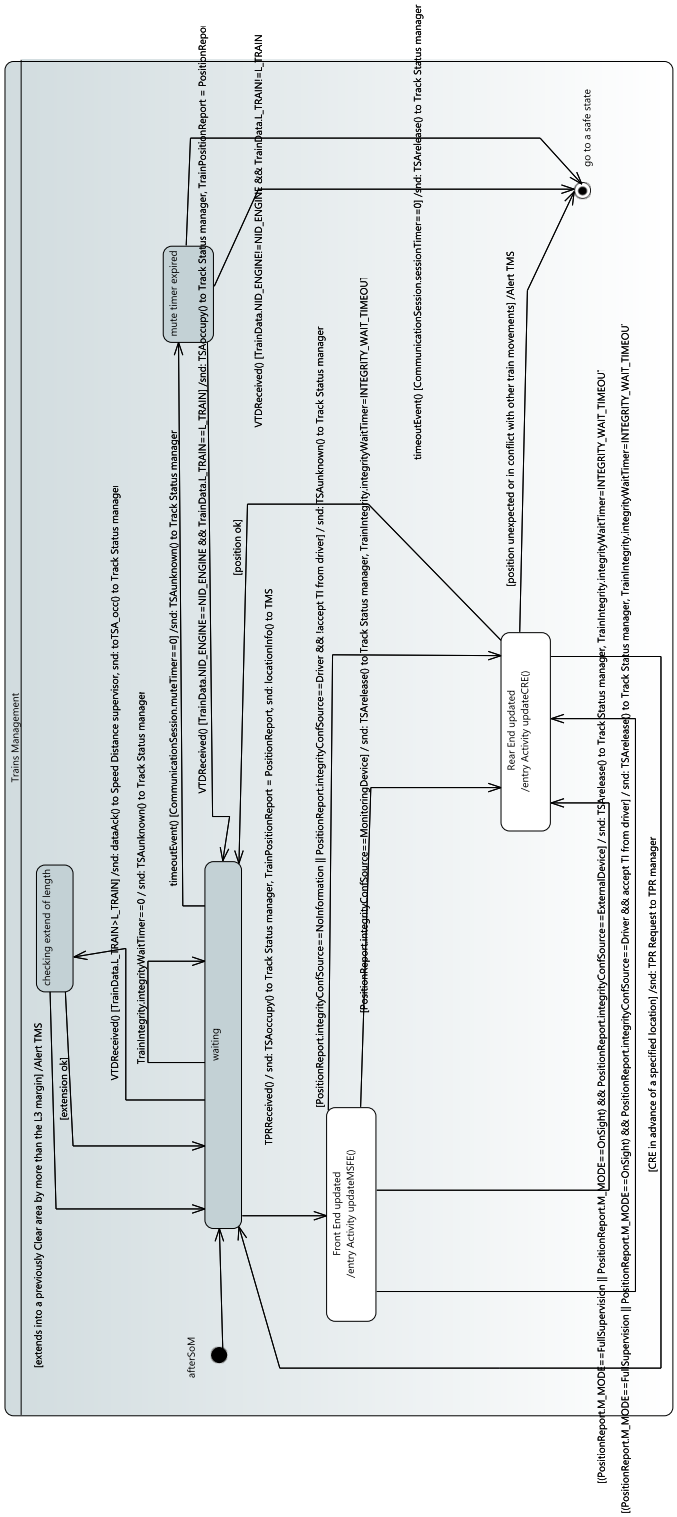


Fig. 6.46. State Machine Diagram for the Trains Management trackside function

6.4.4. Movement Authority Management

The "Movement_Authority_Management is the function in charge of delivering the MAs to the train. The state machine representing the internal behaviour of this function is depicted in Fig. 6.47. This state machine is triggered by an $'MA\_request'$ and leads the machine to the state *'Idle*. Then, to manage a single MA request, the internal machine is entered, and the specific validation of the MA is performed. Specifically, the validity of the request is validated against the conditions that a Danger Point should be at or in the rear of the CRE of the preceding train, and there should be at least the distance of the L3 Margin between the End of Authority (EoA) and the CRE. If the conditions are satisfied, the MA boundaries are computed and the MA is issued. Otherwise, the procedure fails, and the MA is not delivered.

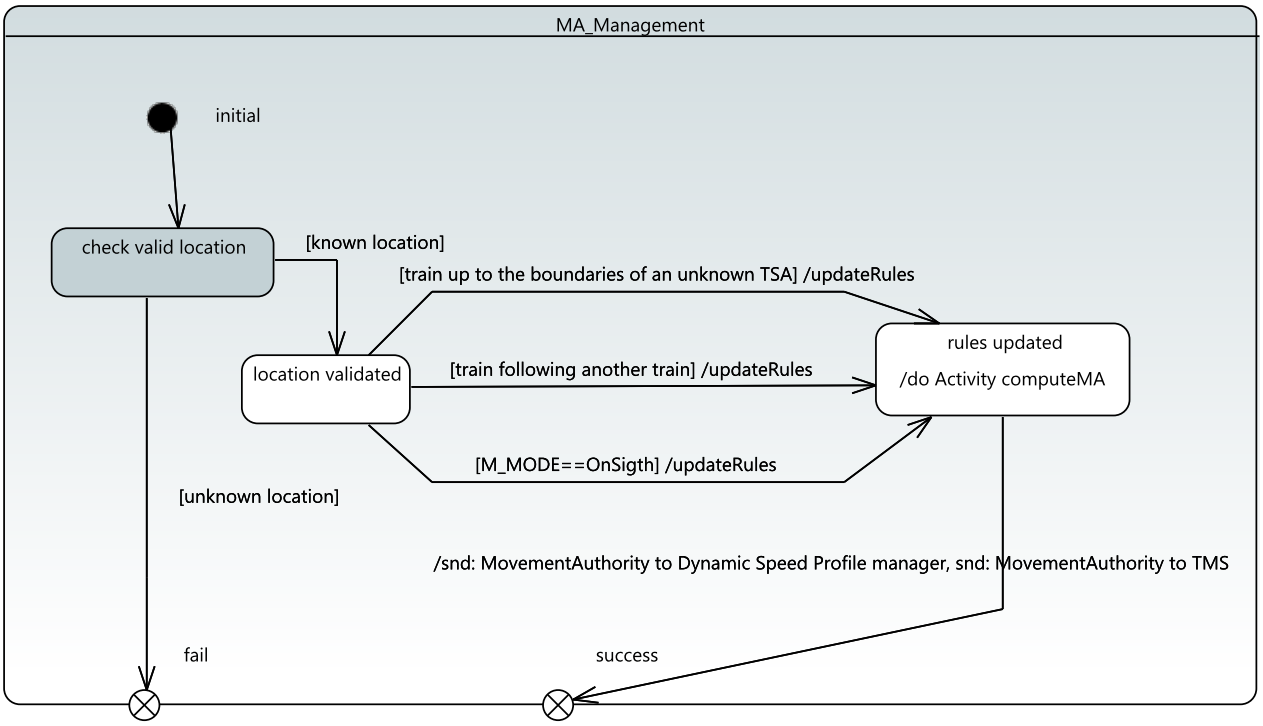


Fig. 6.47. State Machine Diagram for the Movement Authority Management trackside function

6.4.5. Route Management

The “Route_Management” function is in charge of managing the setting, releasing and locking of Routes. As stated in [41], this function is the same as in ETCS-L2. Specifically, ETCS-L2 assumes the availability of an external interlocking system, whereas in ETCS-L3 the interlocking functions are assumed to be contained within the ETCS-L3 functions. However, no new requirements for “Route_Management” are introduced in [42]. Interlocking for moving block was discussed in MOVINGRAIL (Deliverable D1.1 [43]) that proposed to include the flank protection function into the ETCS-L3 “Route_Management”. This point is undecided yet, and it could bring to a possible future extension.

Anyway, an abstract high-level model of this function is provided as it has several interactions with other trackside components and functions, including “Reserved_Status_Management” (that is L3 specific).

The state machine in Figure 6.48 models the general behaviour, and it must be instantiated for each route. The TMS sends the route request and the related information to the Route manager. The request may have been raised by the first MA or by subsequent MAs. The function interacts with the Points management function to set the points for the route, i.e., all the points in the running route. The set of points could also include any adjacent points for flank protection, according to what has been proposed by the MOVINGRAIL project.

The Points manager provides the “Route_Management” with a report about the points status, if the required points cannot be set the route cannot be created/extended/reduced, otherwise the “Route_Management” interacts with the Reserved Status manager to check if the route can be reserved. Again, if the check fails, the route cannot be reserved and the procedure represented in the state machine is aborted, otherwise information about the route extension/reduction is sent to the MA manager and the route is reserved. It is locked as soon as the train occupies the first track. The route is released when the train fully clears all the tracks in the route.

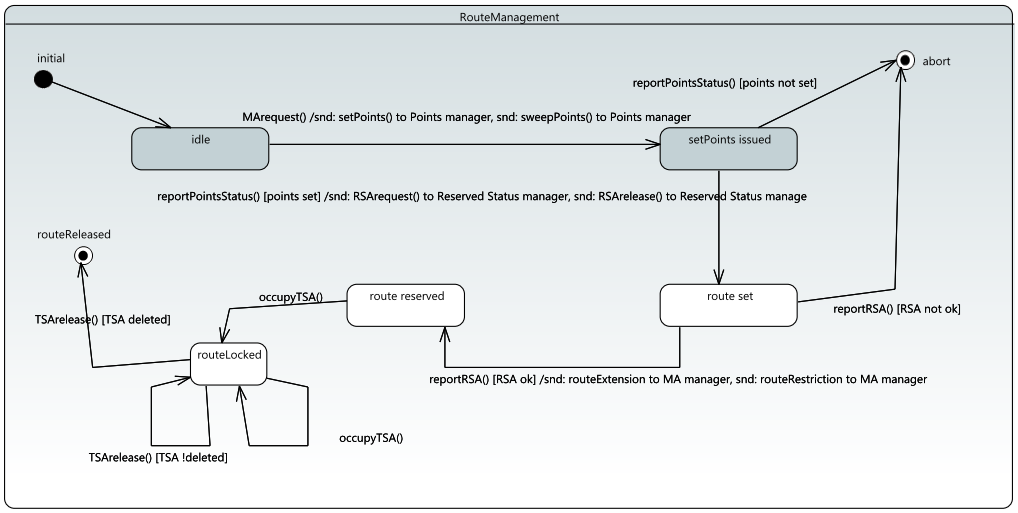


Fig. 6.48. State Machine Diagram for the Route Management trackside function

6.4.6. TTD Management

The Trackside Train Detection(TTD) Management function is a trackside function devoted to managing the status of TTDs. It is used only for an MB system using Trackside Train Detection. The main objectives of the "TTD_management" function are:

- receive and compute a report from the Trackside Train Detection.
- receive and compute train location information (minSFE, MaxSFE, Minimum Safe Rear End (minSRE), Maximum Safe Rear End (MaxSRE)) from "Trains_Management" function.
- manage the lack of synchronisation between TTD occupancy and trains information during normal operation.

The state machine for "TTD_management", represented in Fig. 6.49, is designed using TTD requirements reported in [42] and following the architecture defined in section 6.1.

To describe the behaviour of this state machine, some variables and functions are defined:

- $'ReceiveTrainInformation()'$ updates the information of trains using the received information from "Trains_Management" function.
- $'ReceiveTTDReport()'$ updates the information on TTD status using the received TTDR. TTDRs are stored with a timestamp.
- $'TTD\_STATUS(n)'$ returns the status of TTD number n. Two values are possible: CLEAR and OCCUPIED.
- $'Start(synchronizationTimer)'$ starts the synchronization timer.
- $'Reset(synchronizationTimer)'$ restarts the synchronization timer.
- $'TTD(position)'$ returns the TTD number where the "position" is located.
- $'MaxSFE(Train)'$ returns the position of the Max Safe Front End of the Train passed as a parameter.
- $'MinSFE(Train)'$ returns the position of the Min Safe Front End of the Train passed as parameter.
- $'MaxSRE(Train)'$ returns the position of the Max Safe Rear End of the Train passed as parameter.
- $'MinSRE(Train)'$ returns the position of the Min Safe Rear End of the Train passed as a parameter.
- $'send(Track\_Status\_Management, Clear(Track\_Status(TTD(MaxSFE(Train)), train)))'$ sends to the function "Track_Status_Management" a message to clear the track status identified by the ID of the train and the TTD occupied by the MaxSFE of the train. The same function is defined with the minSRE.
- The DESYNCHRONISATIONTIMER is configured at L3 Trackside level. It is assigned to a TTD if the train position refers to an occupied track section and the corresponding TTD is free, or if a TTD is occupied and there is no corresponding train position for the same track section. Indeed, for a train during normal movement, it may occur that the train physically occupies a TTD before it has reported its position within the TTD boundary (or vice versa).

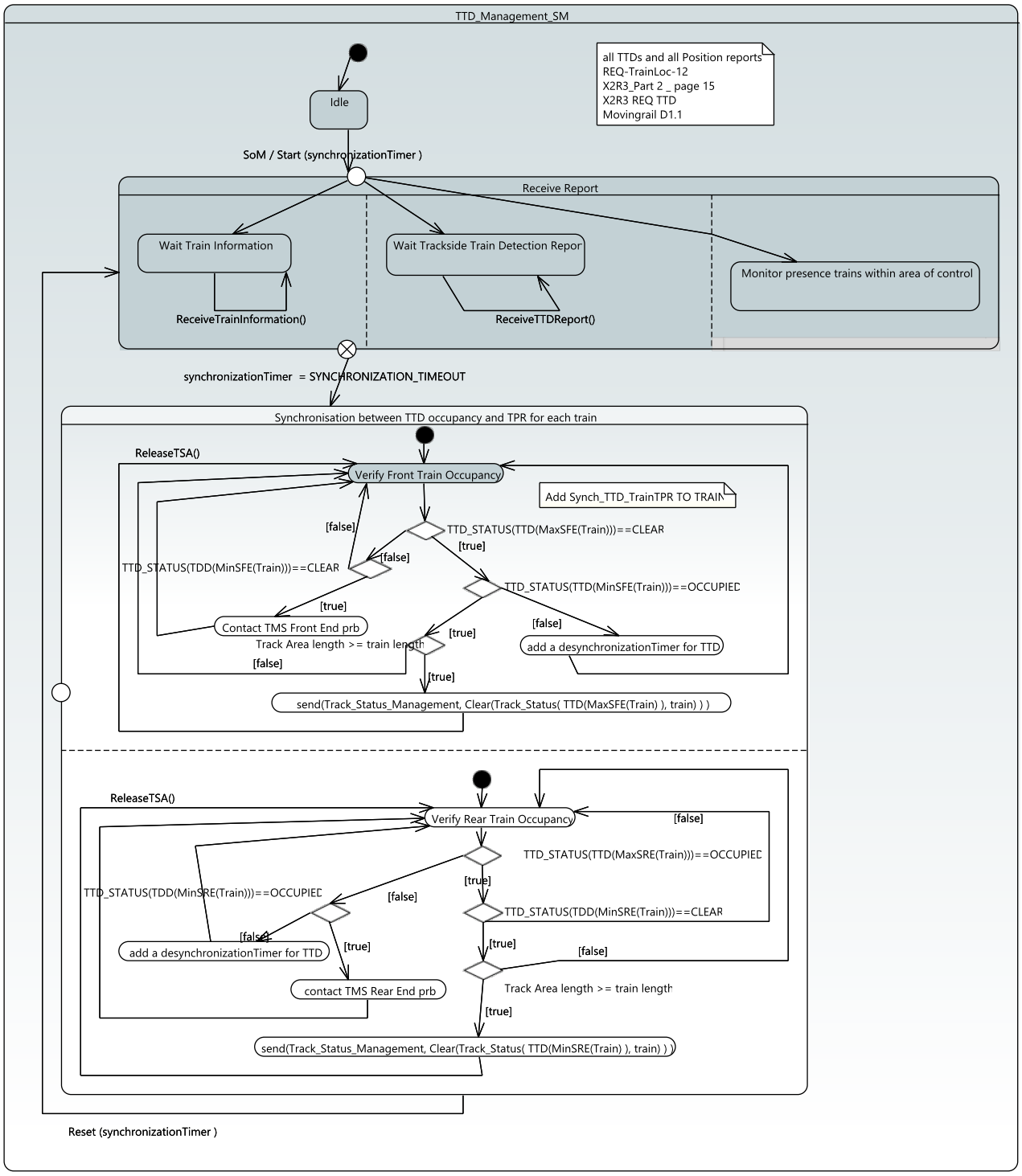


Fig. 6.49. State Machine Diagram for the Trackside Train Detection Management function

Initially, the TDD Management State Machine ("TTD_SM") is in an *'Idle'* state. After a SoM, the DESYNCHRONISATIONTIMER is started ('$Start(synchronizationTimer)$') and the "TTD_SM" passes to the macro state *'Receive Report'*. In this state, the "TTD_SM" performs at the same time three parallel activities represented by states.

- The first is the state *'Wait Train Information'*: when the trackside function

"Trains_Management" sends the train location information, the "TTD_SM" can compute the ID of the train and update the information about this train.

- The second state is *'Wait Trackside Train Detection Report'*: when the external actor TTD sends a Trackside Train Detection Report, the "TTD_SM" can update the information on TTD status.
- The third state is *'Monitor presence trains within area of control'*. In this state, the "TTD_SM" updates the list of trains within the area of control using the received train location information: if the MaxSFE of the train is located in the first TTD of the area of control, the ID of the train is added to the list. If the MinSRE comes out the last TTD, the ID of the train is removed from the list.

If the DESYNCHRONISATIONTIMER is elapsed, the "TTD_SM" moves to the state *'Synchronisation between TTD occupancy and TPR for each train'*. For each train within the area of control, the "TTD_SM" verifies the front train Occupancy ("REQ-TTD-2") and the Rear Train Occupancy ("REQ-TTD-3").

- In the state *'Verify Front Train Occupancy'*, the "TTD_SM" checks if the Max Safe Front End of the train is located in a clear TTD ($'TTD\_STATUS(TTD(MaxSFE(Train))) == CLEAR'$) and the Min Safe Front End of the train is located in an occupied TTD ($'TTD\_STATUS(TTD(MinSFE(Train))) == OCCUPIED'$) and the length of track area occupied by the train is not shorter than the train length ($'TrackArea >= trainlength'$), then the "TTD_SM" sends to the function "Track_Status_Management" a message to clear the track status identified by the ID of train and the TTD occupied by the MaxFSE of the train ($'send(Track\_Status\_Management, Clear(Track\_Status(TTD(MaxSFE(Train)), train)))'$), see Fig. 6.50.

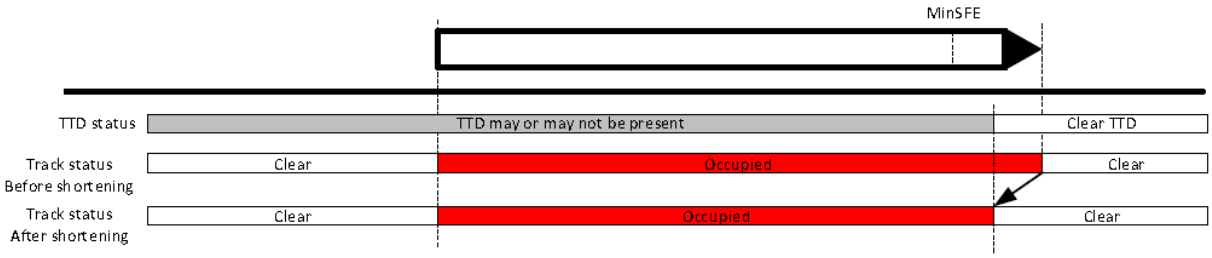


Fig. 6.50. Shortening of front of Track Occupancy due to clear TTD (FMB)

- In the state *'Verify Front Train Occupancy'*, the "TTD_SM" checks if the MaxSFE and the minSFE of the train are located in a clear TTD then the "TTD_SM" shall correlate the TTD occupancy to the train location information by adding a delay timer which is the DESYNCHRONISATIONTIMER (see Fig. 6.51).

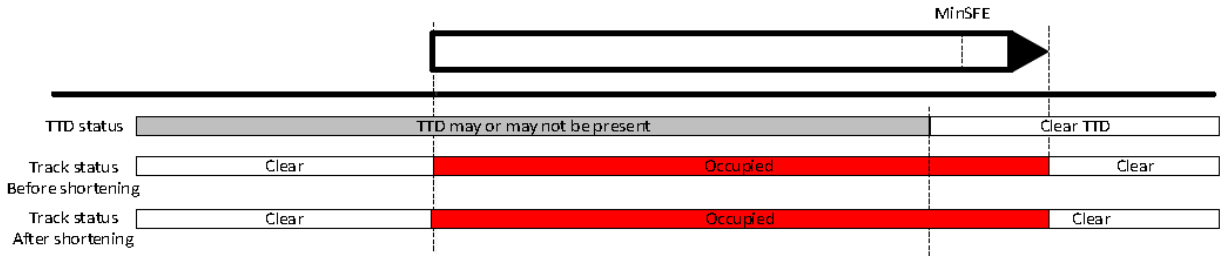


Fig. 6.51. The Max Safe Front End and the Min Safe Front End of the train are located in a clear TTD (FMB)

- In the state *'Verify Front Train Occupancy'*, the "TTD_SM" checks if the Max Safe Front End and the Min Safe Front End of the train are located in an occupied TTD then this situation is normal (see Fig. 6.52).

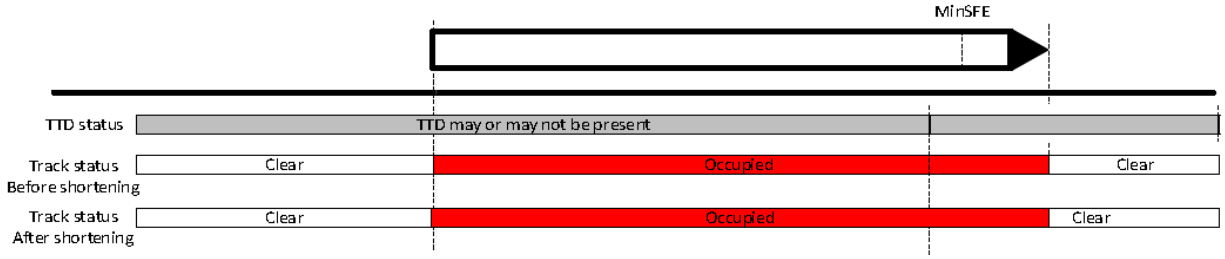


Fig. 6.52. The Max Safe Front End and the Min Safe Front End of the train are located in an occupied TTD (FMB)

- In the state *'Verify Front Train Occupancy'*, the "TTD_SM" checks if the Max Safe Front End of the train is located in an occupied TTD and the Min Safe Front End of the train is located in a clear TTD then this situation is not possible and the "TTD_SM" shall contact the *TMS* (state *Contact TMS Front End prb*).
- In the state *'Verify Rear Train Occupancy'*, the "TTD_SM" checks if the Max Safe Rear End of the train is located in an occupied TTD ($'TTD\_STATUS(TTD(MaxSRE(Train))) == OCCUPIED'$) and the Min Safe Rear End of the train is located in a clear TTD ($'TTD\_STATUS(TTD(MinSRE(Train))) == CLEAR'$) and the length of track area occupied by the train is not shorter than the train length ($'TrackArealength >= trainlength'$), then the "TTD_SM" sends to the function "Track_Status_Management" a message to clear the track status identified by the id of train and the TTD occupied by the minSRE of the train ($'send(Track\_Status\_Management, Clear(Track\_Status(TTD(MinSFE(Train)), train)))'$), see Fig. 6.53.

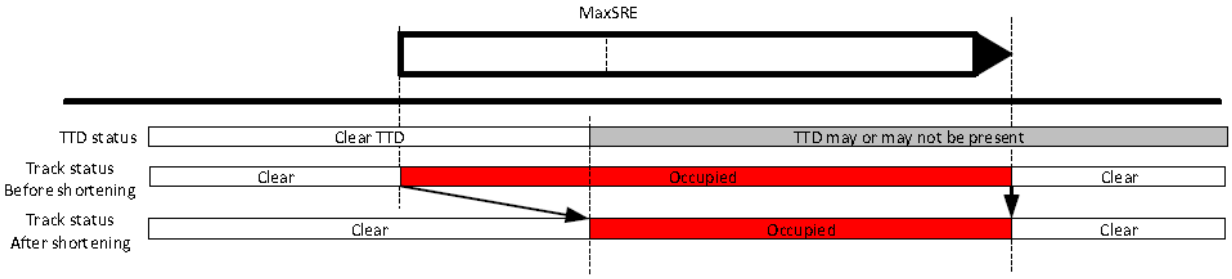


Fig. 6.53. Shortening of rear of Track Occupancy due to clear TTD (FMB)

- In the state *'Verify Rear Train Occupancy'*, the "TTD_SM" checks if the Max Safe Rear End of the train and the Min Safe Rear End of the train are located in a clear TTD then the "TTD_SM" shall correlate the TTD occupancy to train location information by adding a delay timer which is the DESYNCHRONISATIONTIMER(see Fig. 6.54).

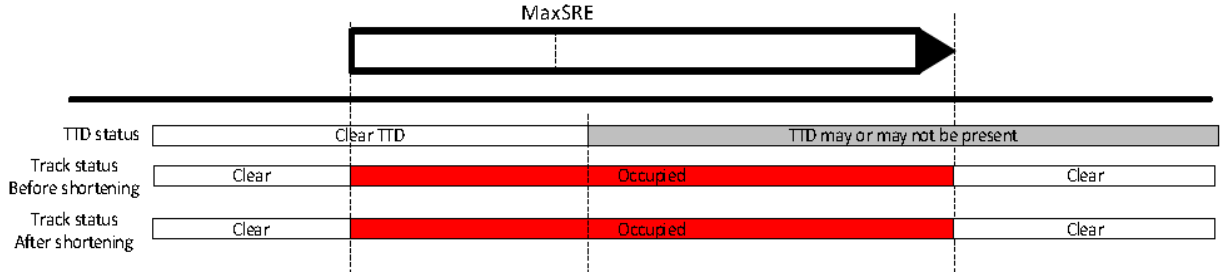


Fig. 6.54. The Max Safe Rear End and the Min Safe Rear End of the train are located in a clear TTD (FMB)

- In the state *'Verify Rear Train Occupancy'*, the "TTD_SM" checks that the Max Safe Rear End and the Min Safe Rear End of the train are located in an occupied TTD then this situation is normal (see Fig. 6.55).

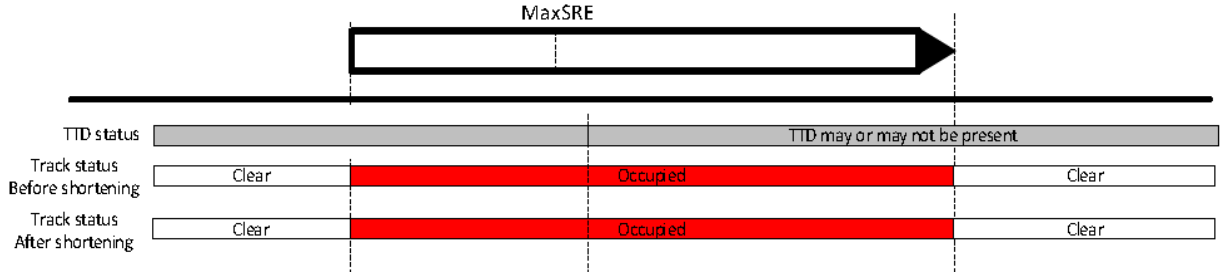


Fig. 6.55. The Max Safe Rear End and the Min Safe Rear End of the train are located in an occupied TTD (FMB)

- In the state *'Verify Rear Train Occupancy'*, the "TTD_SM" checks that the Max Safe Rear End of the train is located in a clear TTD and the Min Safe Rear End of the train is located in an occupied TTD then this situation is not possible and the "TTD_SM" shall contact the TMS (state *'Contact TMS Rear End prb'*).

After verifying the rear and the front occupancies for all trains within the area of control, the "TTD_SM" restarts the DESYNCHRONISATIONTIMER.

6.4.7. Manage Temporary Speed Restrictions

The “Manage_Temporary_Speed_Restrictions is the trackside function in charge of managing Temporary Speed Restrictions (TSRs). This procedure is described in [47], together with the possible requests and the detailed list of their attributes. Briefly, this function is solicited by the following requests:

- Establish a TSR;
- Activate a TSR;
- Deactivate a TSR;
- Remove a TSR;
- Purge a TSR;
- Cancel the purge of a TSR.

The State Machine Diagram for each TSR is depicted in 6.56. In brief, after establishing a TSR, it is in its *'Established'* state. After the activation with the proper message, the state is updated to *'Activated'*. When the conditions leading to the TSR are restored, it can be (in sequence) deactivated, removed and purged. After the removal of a TSR, the trackside could cancel the purge of a TSR, so returning to the state *'Activated'*.

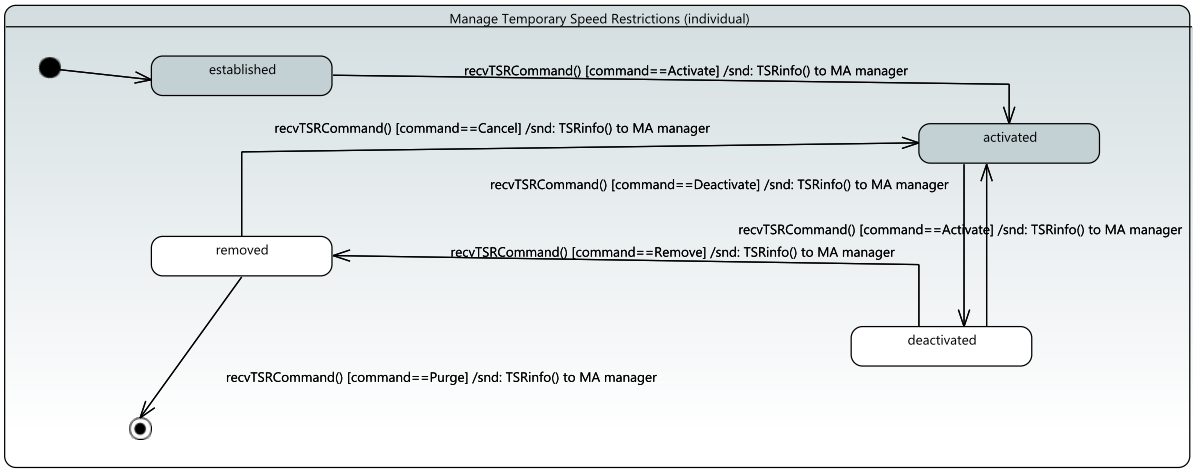


Fig. 6.56. State Machine Diagram for the Manage TSRs trackside function

6.4.8. Points Management

Points Management is the function of the trackside devoted to the management of points locking and unlocking. Its behaviour is represented by a state machine whose main SMD is in Fig. 6.57. This SMD contains the initial pseudo-state of the behaviour (*init*) that brings into the *waiting* state. From this last state, three possible transitions are present. These transitions are *nominal*, *degraded* and *sweepable*, that respectively refer to *Nominal*, *Degraded* and *Sweepable*: the three scenarios of the Points Control EUC (see Subsection 6.3.6). The first two transitions bring to the *CreateRoute* composite state from two different entry points (the *nominal* transition brings to *mainentry* while the *degraded* transition to *forcedentry*).

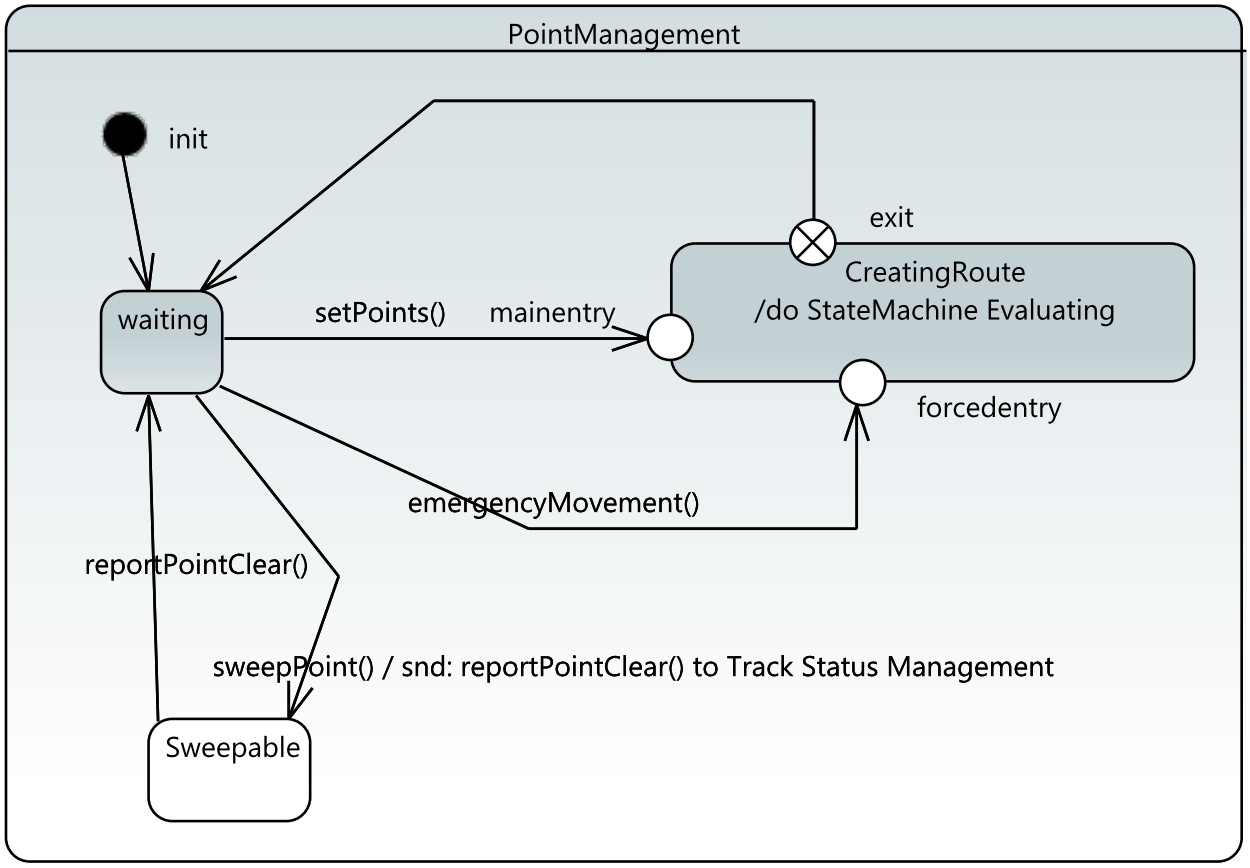


Fig. 6.57. SMD for the Points Management.

The *CreateRoute* composite state is described as follows (see Fig. 6.58).

There are two variables that are local to this state machine: *retval* (containing the status of the entire operation) and the *continueFlag* (representing that there are some further points to explore). This state machine describes the two phases of the creation of a route from the point of view of Points Control. The first phase is related to checking that the involved points in the route to create are in an area that is Occupied/Unknown (Track Status) or is Reserved (Reserved Status). When a point is in one of these conditions, the state machine exits with *retval* equals to *False*. In case the checking phase ends with success, the *UnlockandSet* composite state is reached and, point after point, the route can be created by setting the points in their new position, respectively. The composite state consists of the states: *EndPosition*, *SettingPosition*, *Unlocking*, *routeToSet*, *noError* and *Error* that correspond to moving

the points. A point's position at rest is captured by variable *EndPosition*, which can be "Left" or "Right", whereas variable *PointendPosition* describes the final position of the point after the point has been moved to a new detected position. The transition to state *NoError* is taken as soon as the blades start moving, and the point position is lost (not detected any more). When the position is detected again, once the movement is complete (within a MaxMoving-Time upper bound), the final point position is stored by the *PointendPosition* variable, as a "Left" or "Right" value. In case the maximum prescribed time is exceeded, the *Error* state is entered.

When exiting, a transition from *CreateRoute* to *waiting* is triggered. The state *CreateRoute* is equipped with a shallow history state that allows the state machine to re-enter *CreateRoute* via the last state that was active before leaving the composite state, a useful feature for modelling degraded behaviours, especially when the point position is lost.

An AD has been defined to report the actions related to the sweeping scenario (Fig. 6.59).

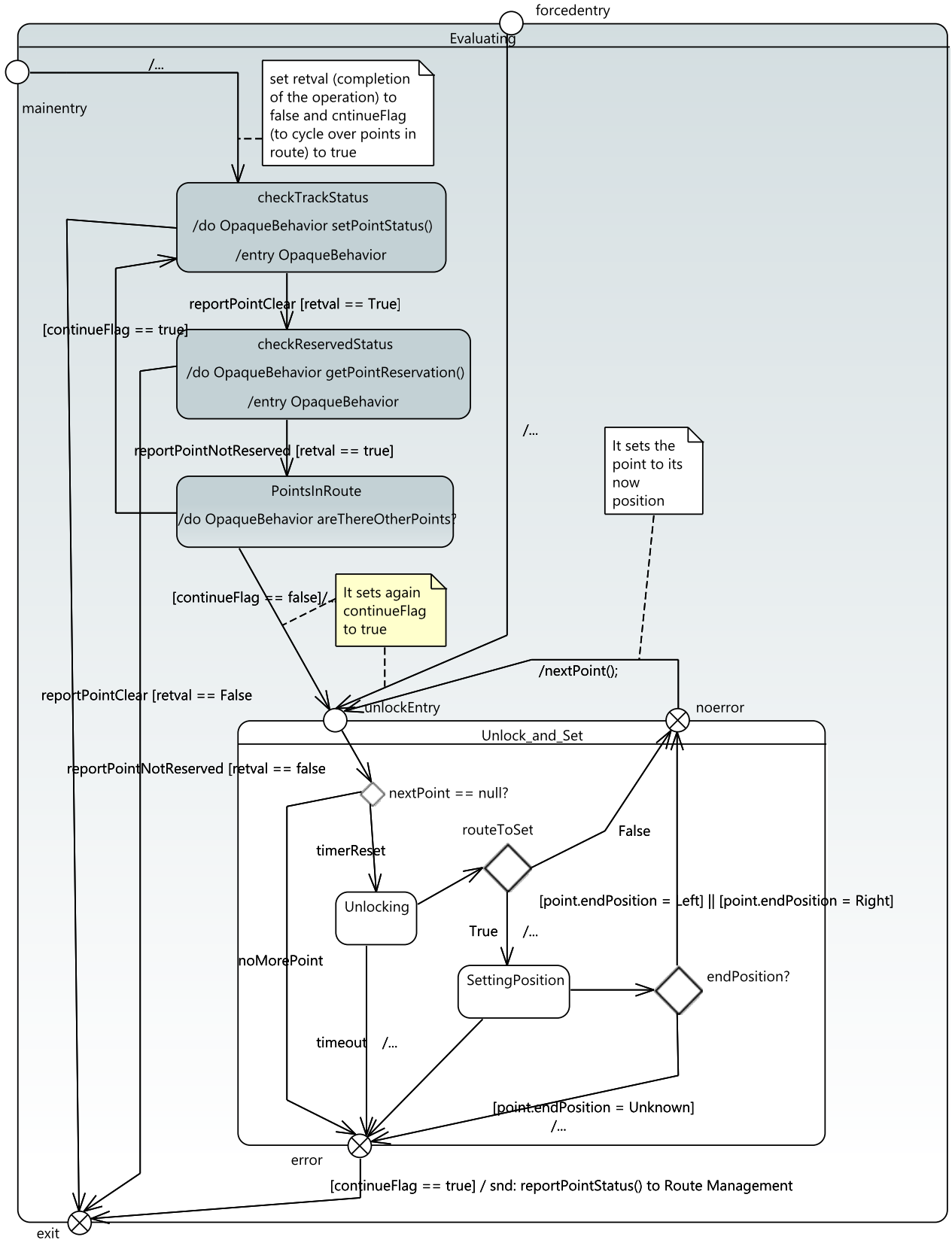


Fig. 6.58. SMD for the Points Management.

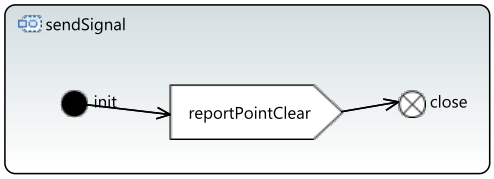


Fig. 6.59. AD for the Points Management.

6.4.9. Communication Management

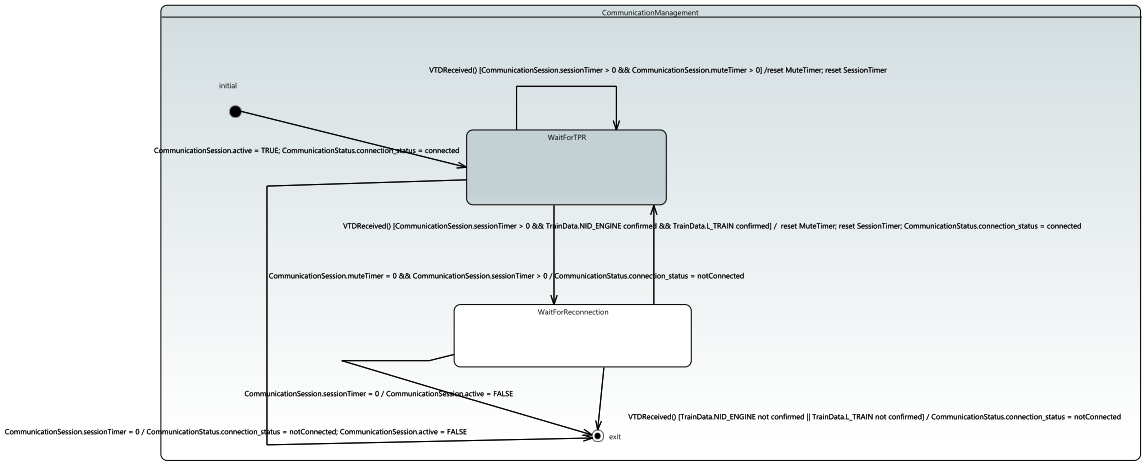


Fig. 6.60. Communication Management State Machine Diagram.

The Communication Management function takes care of managing the communication between the Trackside and a Train. In particular, it is in charge of dealing with possible losses of communication and of managing the conditions under which a lost communication can be recovered. This function assumes that the Train is not currently, and does not enter subsequently, in a RadioHole and has not been sent Reversing Area Information. Loss of communication in these last two situations are dealt with by dedicated Use Cases.

The State Machine modeling this function starts in the **WaitForTPR** state. Receiving a Train Position Report (TPR) while both the SessionTimer and the MuteTimer (if configured) are valid (*i.e.*, not expired) causes the machine to loop back to this state after resetting both the SessionTimer and the MuteTimer (if configured). If the MuteTimer is not configured, the expiration of the SessionTimer while in this state causes the machine to terminate, after setting the Track Status Area (TSA) to `Unknown`. If the MuteTimer is configured, instead, its expiration, while the SessionTimer is still valid, triggers a transition to the **WaitForReconnection** state, while also setting the Track Status Area to `Unknown`. From the **WaitForReconnection** state, should the SessionTimer expire, the machine will simply terminate, the Track Status Area having been already set to `Unknown`. The machine will also move to a session termination state from **WaitForReconnection**, if it receives a TPR before the SessionTimer expires, but the `NID_ENGINE` or the `L_TRAIN` of the communicating train are not confirmed. If, instead, the machine receives a TPR while in **WaitForReconnection** state, both `NID_ENGINE` and `L_TRAIN` are confirmed and the SessionTimer has not expired, it will transition back to **WaitForTPR**, after setting the Track Status Area to `Occupied` and resetting both the SessionTimer and the MuteTimer (if configured).

6.5. The Onboard Behaviour

This subsection presents the SysML sub-models of the ETCS-L3 onboard internal functions, using SMDs.

6.5.1. Train Position Reporting

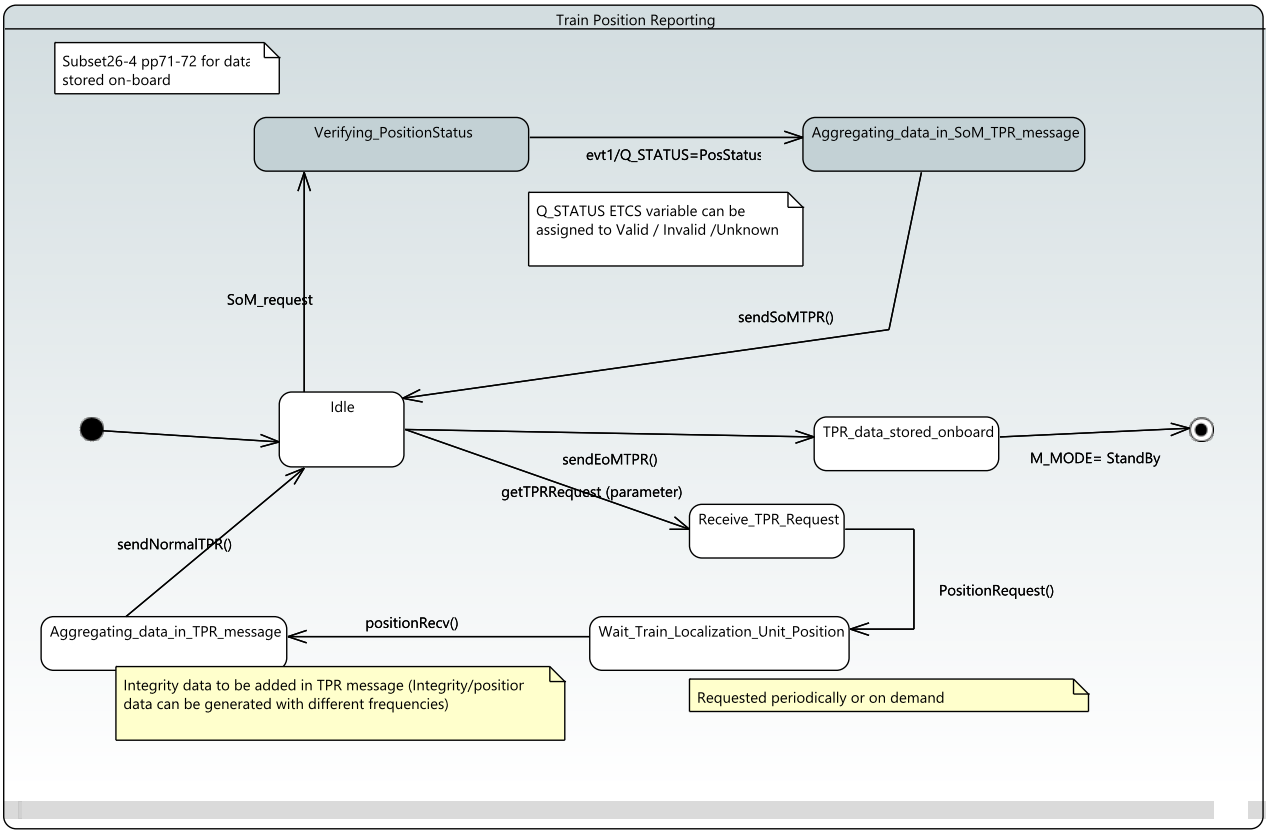


Fig. 6.61. State Machine Diagram for the Train Position Reporting function

The “Train_Position_Reporting” function (TPR) is an on-board function that aggregates two types of information in one TPR message:

- Train Position information,
- Train Integrity information.

The TPR message is sent by radio from the train to the trackside system, mainly using the packet number 0 in Train-to-Track message 136 (Train Position Report), message 157 (Start of Mission TPR) or message 150 (End of Mission). The data in the TPR message does not appear in the State Machine Diagram of Fig. 6.61. Indeed, they are included in the data model Fig. 6.8. The following paragraphs remind the data included in the TPR message and the timed-related conditions for sending it.

DATA INCLUDED IN THE TPR MESSAGE:

The main Train Position (TP) information included in the TPR message is the distance between the estimated train front-end position and a reference location. The reference location is given by a , i.e. a batch of balises (from two to eight) placed on the track

one behind the other on a few meters zone. The BG provides the train with geographic coordinates thanks to the Track-to-Train BG message. This message is obtained from the concatenation of the balise telegrams of the BG, the reference location being the absolute position of the first balise in the BG. When a BG is used as a reference location by the train TPR function, it is called Last Relevant Balise Group (LRBG). Note that the BG telegrams can be read by the train in the nominal or reverse direction.

TP information includes the following main variables:

- NID_LRBG: Identity of last relevant balise group,
- D_LRBG: Distance between the last relevant balise group and the estimated position of the train front-end,
- Q_DIRLRBG: Qualifier for the orientation of the train in relation to the direction of the LRBG,
- Q_DLRBG: Qualifier telling on which side of the LRBG the estimated front end is,
- L_DOUBTOVER: over-reading amount (odometry error + error in detection of BG in rear of the estimated train position) + Q_LOCACC (position error in meter) of the LRBG location,
- L_DOUBTUNDER: under-reading amount (odometry error + error in detection of BG in front of the estimated train position) + Q_LOCACC (position error in meter) of the LRBG location,
- V_TRAIN: Train speed,
- Q_DIRTRAIN: Qualifier for the direction of train movement in relation to the LRBG orientation,
- M_MODE: on-board ETCS operating mode.

In addition, the time at which the TPR message is expected by the Trackside system is associated with conditions described hereunder.

Train Integrity (TI) information in the TPR message includes the following variables:

- Q_LENGTH: Qualifier for train integrity status (0: No TI information available, 1: TI confirmed by TIMS, 2: TI confirmed by the driver, 3: TI lost).
- L_TRAININT: safe train length, i.e. the distance between the “min safe rear end” (at the time the train was last known to be an integer) and “the estimated position of the train Front-End” at the time when the train integrity information is sent to the RBC (remark: Subset 026-§3.6.5.2.4 refers to minSFE and L_TRAIN, but here, the ongoing Change Request CR940 is considered, cf. [4]).

Relation between“TPR information” and “Train Location information”:
L_DOUBTOVER and L_DOUBTUNDER define the “confidence interval” associated to the estimated position of the train front-end (=D_LRBG) and are used to determine the Train Location information (MINSRE, MAXSRE, MINSFE, MAXSFE, ESITMATED FRONT END (ESTIMFE)). L_TRAININT is related to the Trackside CRE (Confirmed train Rear-End) calculated by the “Trains_Management” function. Finally, the correspondence between TPR information and Train Location information is established as follows:

$$MaxSFE = D\_LRBG + L\_DOUBTUNDER$$

$$minSFE = D\_LRBG - L\_DOUBTOVER$$
$$MaxSRE = MaxSFE - L\_TRAIN$$
$$minSRE = minSFE - L\_TRAIN$$
$$EstimFE = D\_LRBG$$

$$\begin{aligned} CRE &= D\_LRBG - L\_TRAININT \\ &= minSRE\_at\_last\_integrity\_confirmed \\ &= D\_LRBG\_at\_last\_integrity\_confirmed - \\ &\quad L\_DOUBTOVER\_at\_last\_integrity\_confirmed - L\_TRAIN \end{aligned}$$

CONDITIONS FOR SENDING THE TPR MESSAGE:

A TPR message is sent when this is requested by ETCS On-board following certain conditions and/or using a periodicity. The conditions for which the On-board system has to request TP and TI information respectively to the Localisation System and the Train Integrity Monitoring System are (cf. Subset 026-§3.6.5.1.4 and §4.5.2):

- when the train reaches a standstill,
- when the ETCS operating mode changes,
- when train integrity is confirmed by the driver (only permitted at standstill),
- when loss of train integrity is detected,
- when the train front/rear passes an RBC/RBC border with MaxSFE/MinSFE,
- when the ETCS level changes,
- when the train establishes a session with the RBC, especially during the Start of Mission procedure,
- when passing a LRBG,
- when requested by RBC (cf.M_LOC and _LOC variables).

A TPR can be sent periodically in space or/and in time with the following parameters given by the RBC:

- T_CYCLOC: time interval between 2 TPRs sent by the train,
- D_CYCLOC: distance between 2 TPRs sent by the train.

Former version of Subset 041 (v2.10) mentioned a maximum value of five seconds between two consecutive TPRs sent by the train. However, today, no frequency value is given in the current specification version (it has to be laid down by railway operators).

The first TPR (message 157-SoM TPR) is sent during the Start of Mission procedure (SoM) after the On-board system establishes a radio session with the Trackside System. The SoM TPR includes the status of the on-board stored position, i.e. Q_STATUS = 0: Invalid, 1: Valid, 2: Unknown. This status depends on whether the train has undergone a cold movement when the On-board system was switched off. The first TPR is depicted on top of Fig. 6.61 and the other sent TPR is depicted on the bottom. During the End of Mission Procedure (EoM), a last TPR is sent (messages 136 and 150) and TPR data are stored on-board (Subset 026-§4.10.1).

$'getTPRRequest(parameter)'$ is a signal triggered when TPR information has to be collected

by ETCS On-board after a request by the ETCS Trackside or On-Board system. In the conditions listed above, some parameters related to a specific location can be used. Namely, when ETCS Trackside (RBC) requests ETCS On-board to report its position at specific train locations (cf. requirement “REQ-TrainLoc-7”), M_LOC and D_LOC are used. These parameters and, also, the cyclic parameters T_CYCLOC and D_CYCLOC are sent using packet number 58 in a Track-to-Train message.

Integrity information can be generated with a different frequency related to the external device output (TIMS) frequency.

Note that, for realizing the TPR function, GNSS (Global Navigation Satellite Systems) are planned to be embedded in the train Localisation Unit in addition to the Odometry System. A GNSS receiver can provide either the reference location to the train (for that, the concept of Virtual Balises is introduced in several GNSS projects in order to replace the reference location provided by BG) or directly the absolute position, depending on the future ETCS-L3 architecture. Today, for delivering TP information, the use of a relative distance from a LRBG is preferred as several BG can be announced in advance using “linking information”. The train has then to pass the expected BG in a calculated expectation window. The linking mechanism is one of the critical ETCS safety functions. Note also that the Speed and Distance Monitoring function refers to distances and speeds counted from a reference location (e.g. the LRBG). For delivering TI information, a second GNSS-based localization system could be placed at the rear of the train.

6.5.2. Integrity Information Management

The main role of the “Integrity_Information_Management” function is to monitor and control the integrity status of the train while considering all the relevant information. The integrity information is, in fact, a crucial input for safe train operation. The event of accidental train separation constitutes a serious hazard for railway operation since it generates an unexpected obstacle on the line for the following train. Hence, it is crucial that such a hazard be promptly reported to the signalling system. Concretely, monitoring the integrity status consists of permanently checking whether the whole train is advancing in a coherent way.

“Integrity_Information_Management” is a key function in the framework of MB operation since movement authority needs to be provided along line parts that are free from any obstacle. As a reminder, implementing the train integrity monitoring onboard trains allows for substantial gains in terms of equipment, maintenance, etc.

Fig. 6.63 shows the behaviour of the “Integrity_Information_Management” function. To do so, various documents, namely the System Requirement Specification (SRS) specifications, MOVINGRAIL deliverables, X2Rail-2 D4.1, are considered. Four states can be enumerated as follows:

- No integrity information,
- Integrity confirmed by driver,
- Integrity confirmed by an external device,
- Integrity lost.

The switching among the various states is described through a transition table given in the European Union Agency for Railways (ERA) change request document CR 940_16042020 (subset-026 v3.6.0), as shown in Figure 6.62.

As seen in the transition table, the changes from one state to another are governed by a number of conditions and priority levels. The list of conditions is given below;

1 : No valid Train data is available

2 : (Train is at standstill) AND (valid Train Data is available and has been acknowledged by the RBC) AND (the train integrity is confirmed by the driver)

3 : (The information ”Train integrity confirmed” is received from an external device) AND (valid Train Data is available and has been acknowledged by the RBC) AND (Train Data regarding train length has not changed since the time the train was last known to be an integer) AND (the train position is valid and is referred to an LRBG) AND (the train position was valid and was referred to an LRBG at the time the train was last known to be an integer) AND (no reverse movement is currently performed nor has been performed since the time the train was last known to be an integer) AND (the distance between the min safe rear end at the time the train was last known to be an integer and the current estimated train position does not exceed the range of the safe train length information)

4 : (The information ”Train integrity lost” is received from an external device) AND (valid Train Data is available since the time the train integrity was last known to be lost)

5 : A position report indicating that the train integrity is confirmed is sent to the RBC

6 : The information ”Train integrity status unknown” is received from an external device

7 : Train Data regarding train length is changed

8 : A reverse movement is performed

9 : The distance between the min safe rear end at the time the train was last known to be an integer and the current estimated train position exceeds the range of the safe train length information

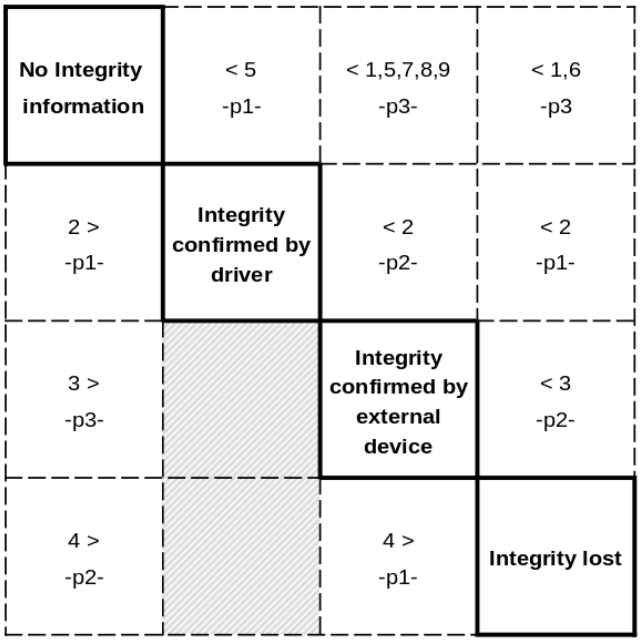


Fig. 6.62. States and transitions in “Integrity_Information_Management”

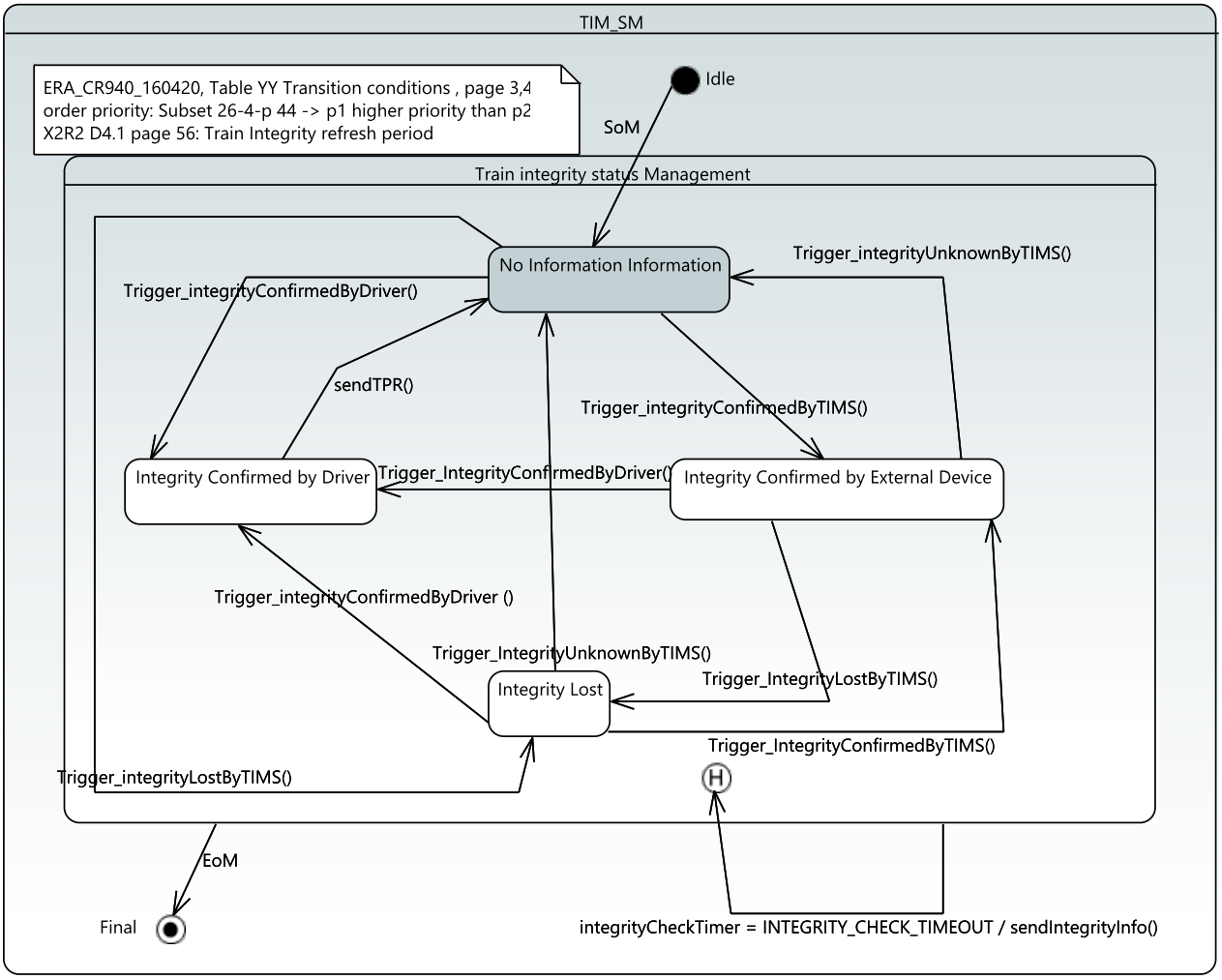


Fig. 6.63. State Machine Diagram for the “Integrity_Information_Management” function

6.5.3. Speed and Distance Supervision

Fig. 6.64 reports the global SMD of this function, which is divided into two regions: one (the left region) managing the sending of VTD message and its acknowledgement, while the second (the right region) related to the entering into the supervision mode when a new braking curve is available.

It is important to stress the separation of duties between this function and “Manage_Dynamic_Speed_Profile”. The second function is in charge of taking into account the data coming from the Trackside — e.g., MAs, SSP — and data coming from the train — e.g., brake information, ALSP. “Manage_Dynamic_Speed_Profile” is then in charge of computing braking curves (see [48]) that “Speed_and_Distance_Supervision” controls.

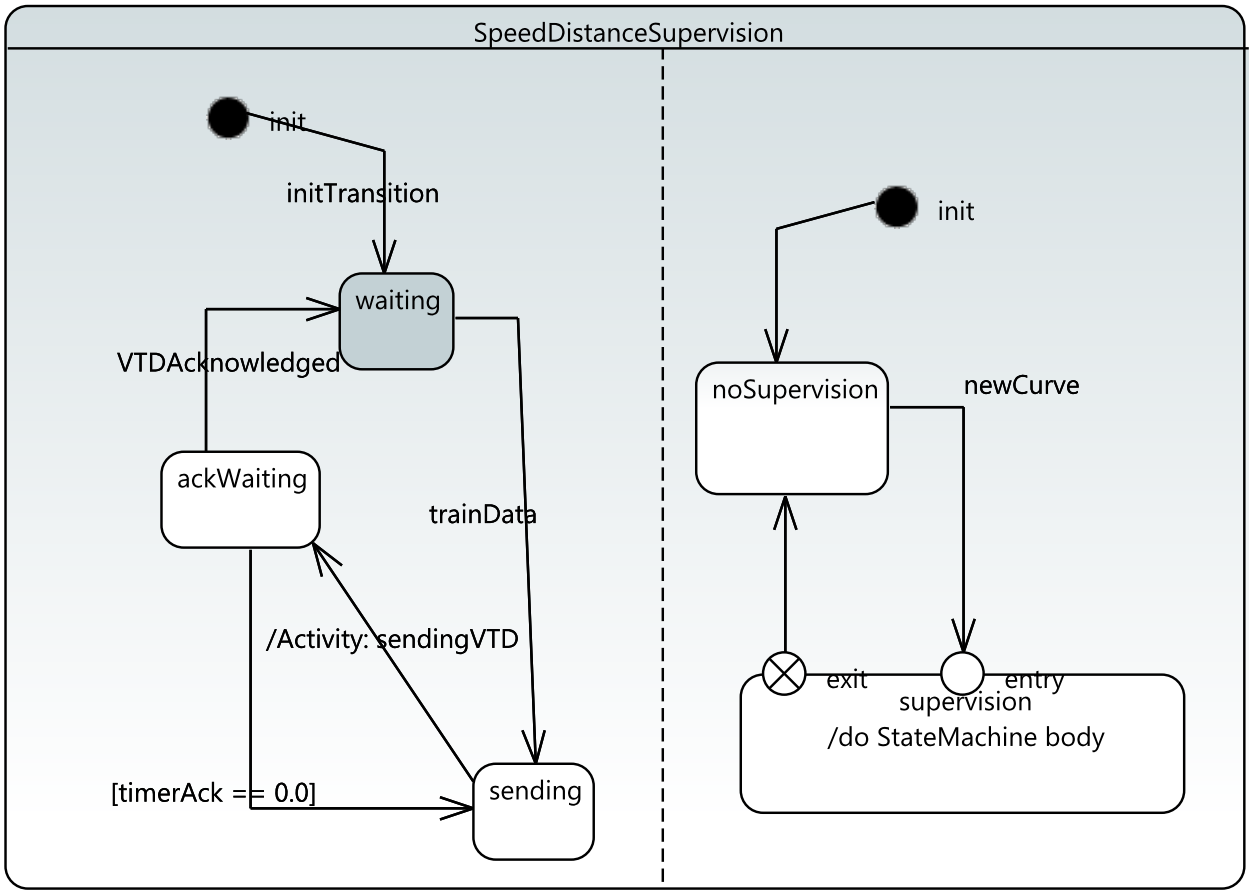


Fig. 6.64. Speed and Distance Supervision SMD

The supervision state is then exploded into another SMD reported in Fig. 6.65.

Some ADs are present to report actions done on some diagram transitions (see Fig. 6.66, Fig. 6.67, Fig. 6.68, Fig. 6.69 and Fig. 6.70).

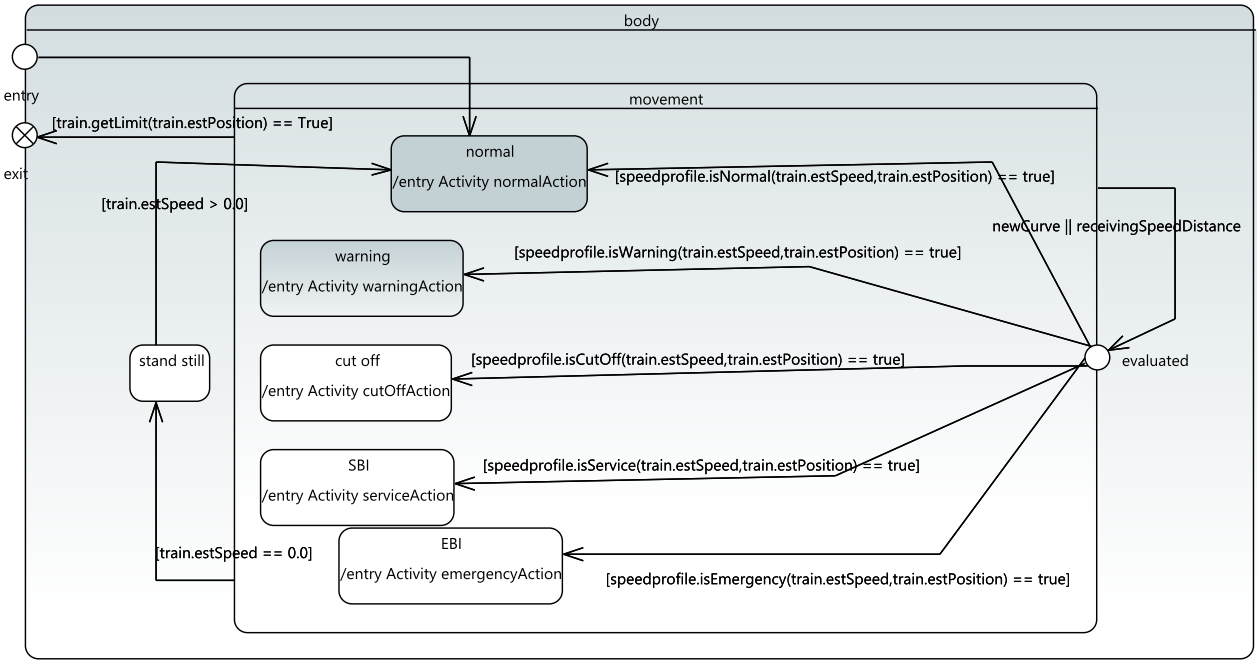


Fig. 6.65. Supervision sub-SMD

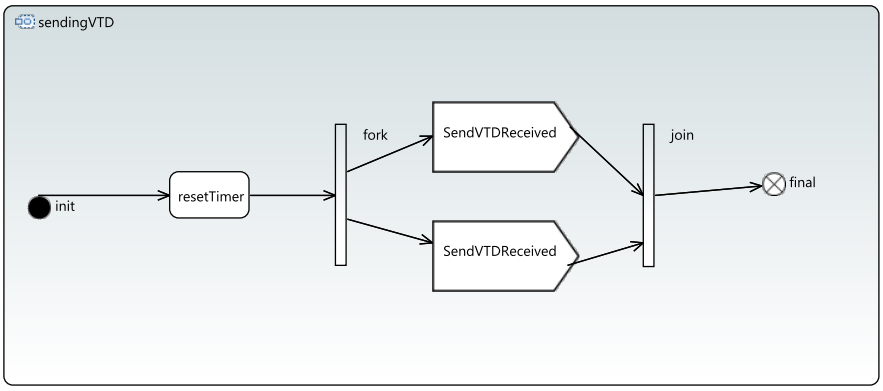


Fig. 6.66. AD of the transition from sending to ackWaiting

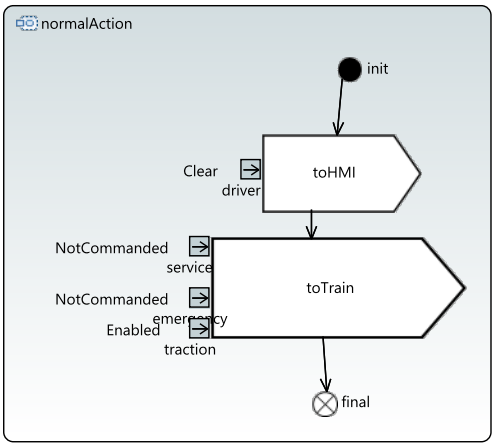


Fig. 6.67. AD of the transition from evaluated to normal

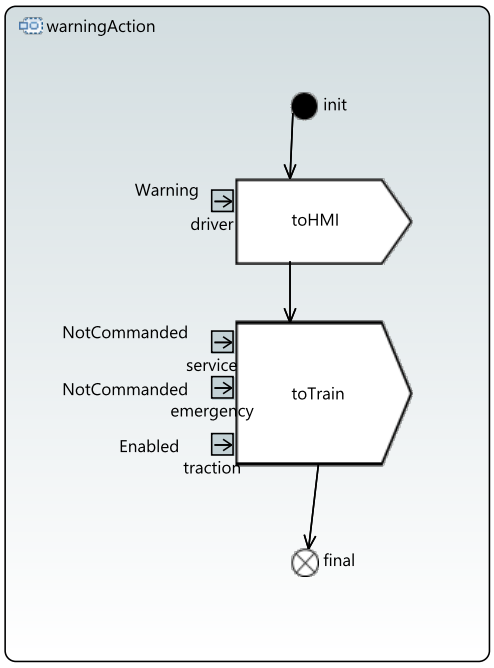


Fig. 6.68. AD of the transition from evaluated to warning

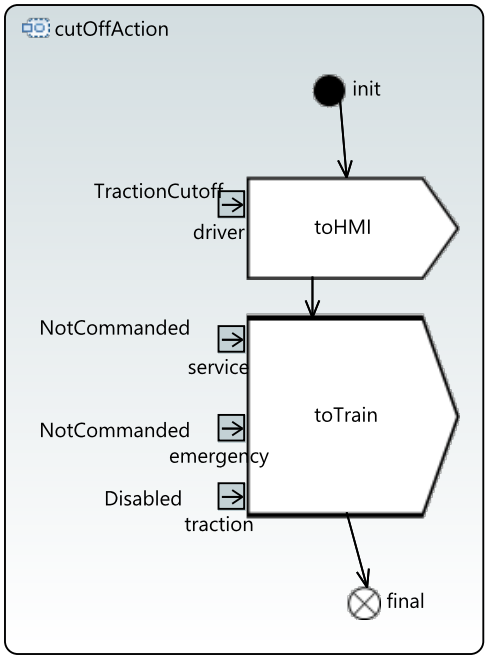


Fig. 6.69. AD of the transition from evaluated to cut off

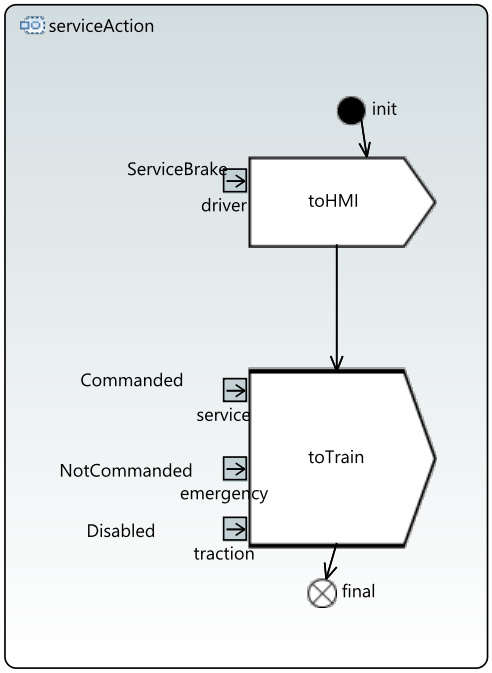


Fig. 6.70. AD of the transition from evaluated to SBI

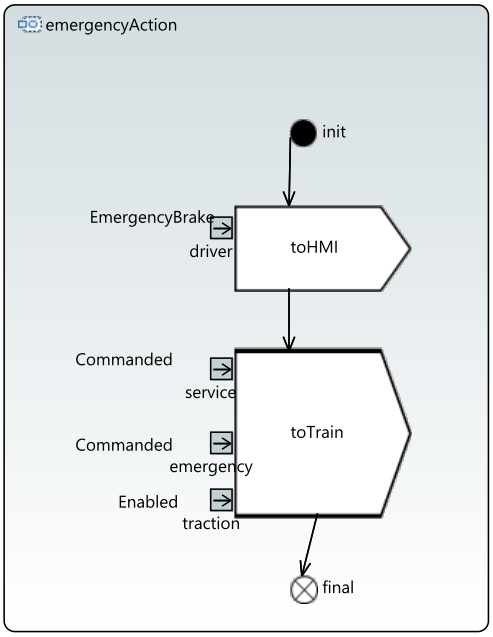


Fig. 6.71. AD of the transition from evaluated to EBI

6.6. The Requirement Allocation Table

This subsection reports the final status of the requirement allocation. It shows how the SysML model satisfies the ETCS-L3 requirements, using Tables 6.5 and 6.6. These tables list the elements of the SysML model that are related to each requirement. There are 87 requirements that are satisfied by some model elements, and they are reported in the table. Other 65 requirements, as they are described in [42], are partially considered. Table 6.7 reports such requirements justifying the reason for not being fully considered in the proposed model[4].

[4] In this table, some rows are grouped for the sake of clarity.

Table 6.5: SysML model RAT

Requirement	Satisfied by
REQ-LossComms-1	MuteTimer expiration
REQ-LossComms-2	MuteTimer expiration
REQ-LossComms-3	SessionTimer expiration (mute) - SessionTimer expiration (no mute) - MuteTimer expiration
REQ-LossComms-4	Trains Management - MuteTimer expiration
REQ-LossComms-5	Trains Management - SessionTimer expiration (mute) - SessionTimer expiration (no mute)
REQ-LossComms-6	Trains Management - SessionTimer expiration (mute) - SessionTimer expiration (no mute)
REQ-LossTI-1	LTI_TIMS_Integrity_SD
REQ-LossTI-2	LTI_TIMS_Integrity_SD
REQ-LossTI-3	LTI_TIMS_Integrity_SD
REQ-LossTI-4	LTI_TIMS_Integrity_SD
REQ-LossTI-5	LTI_TIMS_Integrity_SD
REQ-LossTI-6	LTI_DriverIntegrity_SD
REQ-LossTI-7	LTI_TIMS_Integrity_SD
REQ-LossTI-8	LTI_DriverIntegrity_SD
REQ-MA-3	MA_Management
REQ-MA-4	MA_Management
REQ-MA-5	MA_Management - transition to OS mode
REQ-MA-6	MA_Management - transition to OS mode
REQ-MA-10	MA_Management
REQ-MovSR-1	SR Movement
REQ-MovSR-2	SR Movement
REQ-MovSR-3	SR Movement
REQ-MovSR-4	SR Movement
REQ-MovSR-5	SR Movement
REQ-PTS-1	TrackStatusManagement_global - Nominal_SD
REQ-PTS-2	Nominal_SD
REQ-PTS-3	TrackStatusManagement_global - Degraded_SD
REQ-PTS-4	Sweeping_SD
REQ-RecoveryMgmt-1	MuteTimer expiration - SessionTimer expiration (mute) - SessionTimer expiration (no mute)
REQ-RecoveryMgmt-2	NID_ENGINE and L_TRAIN confirmation - SessionTimer expiration (mute) - SessionTimer expiration (no mute)
REQ-RecoveryMgmt-3	Trains Management - SessionTimer expiration (mute) - SessionTimer expiration (no mute)
REQ-Reserved-1	Reserved Status Management - RouteManagement
REQ-Reserved-2	Reserved Status Management
REQ-Reserved-3	Reserved Status Management
REQ-Reserved-4	Reserved Status Management
REQ-Reserved-5	Reserved Status Management - RouteManagement
REQ-Reserved-6	Reserved Status Management

Table 6.6: SysML model RAT

Requirement	Satisfied by
REQ-SH-3	Sweeping_2_SD
REQ-TTD-1	TTD_Management_SM
REQ-TTD-2	TTD_Management_SM
REQ-TTD-3	TTD_Management_SM
REQ-TTD-4	TTD_Management_SM
REQ-TTD-5	TTD_Management_SM
REQ-TrackInit_1	Trackside Initialisation
REQ-TrackInit_2	Initialise train positions - Trackside Initialisation
REQ-TrackInit_3	Trackside Initialisation
REQ-TrackInit_4	Detect track assets
REQ-TrackInit_5	Trackside Initialisation
REQ-TrackStatus-1	TrackStatusManagement_global
REQ-TrackStatus-2	TrackStatusManagement_global
REQ-TrackStatus-3	TrackStatusManagement
REQ-TrackStatus-4	TrackStatusManagement
REQ-TrackStatus-5	TrackStatusManagement - Sweeping_2_SD
REQ-TrackStatus-6	TrackStatusManagement - Sweeping_1_SD
REQ-TrackStatus-7	TrackStatusManagement - Sweeping_1_SD
REQ-TrackStatus-8	TrackStatusManagement
REQ-TrackStatus-10	TrackStatusManagement
REQ-TrackStatus-12	TrackStatusManagement
REQ-TrackStatus-13	TrackStatusManagement
REQ-TrackStatus-15	TrackStatusManagement - Trains Management
REQ-TrackStatus-16	TrackStatusManagement - Trains Management
REQ-TrackStatus-17	TrackStatusManagement - Trains Management
REQ-TrackStatus-19	TrackStatusManagement - Trains Management
REQ-TrainLoc-1	Trains Management
REQ-TrainLoc-2	Trains Management
REQ-TrainLoc-3	Trains Management
REQ-TrainLoc-4	Trains Management
REQ-TrainLoc-5	Trains Management - OS_SD
REQ-TrainLoc-6	Trains Management - OS_SD
REQ-TrainLoc-7	Trains Management
REQ-TrainLoc-11	Trains Management
REQ-TrainLoc-12	Trains Management
REQ-TrainLoc-13	Trains Management
REQ-TrainLoc-14	Trains Management

Table 6.7: Requirements not included in the RAT

Requirement	Comment
REQ-EoAExclusionArea-(1,2)	This EUC is not included in the chosen OPSs.
REQ-EoM-(1-4)	This EUC is not included in the chosen OPSs.
REQ-FVB-1	This EUC is not included in the chosen OPSs.
REQ-HO-(1-3)	This EUC is not included in the chosen OPSs.
REQ-Join-(1-3)	This EUC is not included in the chosen OPSs.
REQ-LevelTrans-(1,2)	This EUC is not included in the chosen OPSs.
REQ-LossTI-10	Changing train length is mainly related to splitting and joining which are not in the scope of the work.
REQ-LossTI-9	General properties of the function.
REQ-MA-(1,7-9)	General properties of the function.
REQ-MA-2	Position of obstructions have not been stored
REQ-MA-11	Linking Information not considered for the message.
REQ-MA-12	Requires an additional interaction and additional information.
REQ-RadioHole-(1-6)	This EUC is not included in the chosen OPSs.
REQ-Rev-(1-5)	This EUC is not included in the chosen OPSs.
REQ-SH-(1,2,4)	This EUC is not included in the chosen OPSs.
REQ-Split-1	This EUC is not included in the chosen OPSs.
REQ-StartTrain-(1-15)	This EUC is not included in the chosen OPSs.
REQ-TTD-6	Requires an additional interaction and additional information.
REQ-TTD-7	Requires an additional interaction and additional information.
REQ-TrackStatus-9 REQ-TrainLoc-8	The procedure leading the system to a safe state involves ETCS messages and procedures that are not in the scope of the work (e.g., emergency management).
REQ-TrackStatus-11	The overlap of multiple Unknown TSAs is part of an internal evaluation algorithm, not captured by an SM.
REQ-TrackStatus-14	The computation of the extension of a TSA is made by data-oriented operations not covered by SMD's elements.
REQ-TrackStatus-18	The functional architecture does not allow the Track Status Manager to validate the current position of the train.
REQ-TrainLoc-9	The algorithm recognising an unexpected position or a conflict among train movements is not well specified in reference documents.
Req-TrainLoc-10	Non-Leading or Sleeping modes are out of the scope of the modelling activities.

7. The Followed Modelling Approach

This chapter describes the formal modelling approach (Section 7.1) and provides details on the preliminary activities (Section 7.2, Section 7.3 and Section 7.4). The structure and the details of the formal models are presented in Chapter 8.

7.1. The Formal Modelling Process

This subsection describes the formal modelling process. This process can be divided into two main parts. A set of preliminary activities has the objective of describing EUCs/internal functions enumerating, for example, variables, parameters, describing initial conditions.

Fig. 7.1 depicts the detail of the Modelling activity in the AD reported in Fig. 4.2.

In the AD, these activities are depicted in the two loop activity blocks that can be executed in parallel. Preliminary activities are run on both EUCs — the block on the right of the fork bar — and on internal functions — the block on the left.

Preliminary activities are oriented to the definition of a rigorous framework where a formal model of an EUC or of an internal function can be developed. Preliminary activities change in case of EUC or internal function. The details are in the diagram. Some sample activities are the definition of parameters, inputs/outputs, initial configuration, and involved actors. Up to this moment, the specification activities are neutral regarding the formalism, having the side objective to specify behavioural and interaction elements in a formalism-neutral way, opening for possible “implementations” in different formalisms regarding the ones used in this deliverable.

After the preliminary activity phase, there is the formal model construction and analysis phase, which involves the construction of a formal model, according to a chosen formalism, and its analysis, aimed at proving some simple properties.

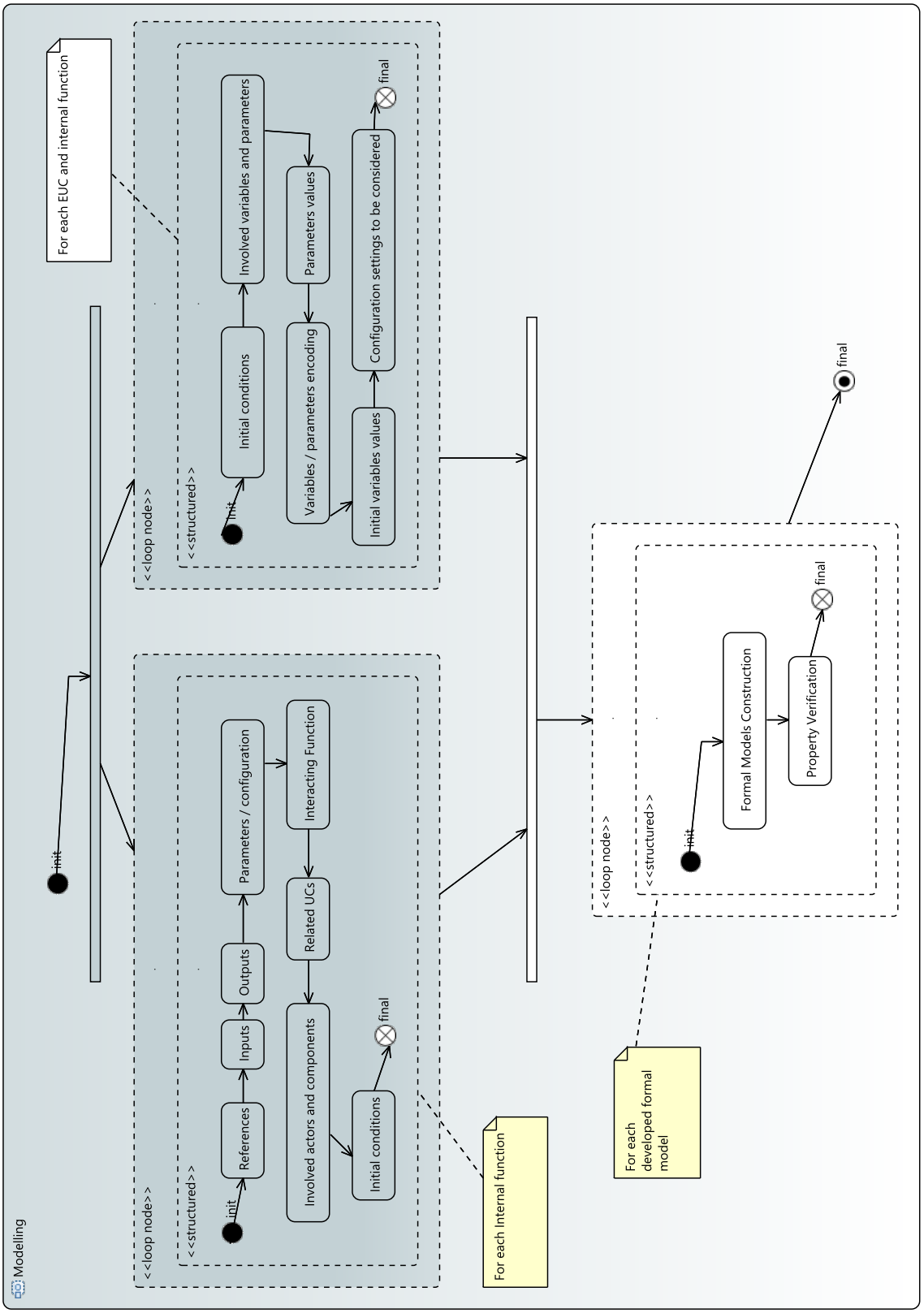


Fig. 7.1. The Formal Modelling Process.

7.2. The Preliminary Activity Template

This section shows the template used to describe the results of the preliminary activities of the formal modelling, according to the process described in this chapter. Table 7.1 and Table 7.2 respectively report the templates for EUCs and functional components. Bold text is fixed, while italics explains the meaning of the table field and changes in the instantiation of such templates. The following rows could be added/deleted according to the needs: *Variable X*, *Parameter X* in Table 7.1 and *Input X*, *Output X*, *Parameter X* in Table 7.2

Table 7.1: Template for EUC's preliminary activity description

Initial Condition	
Position	*defining the initial position*
Operation Mode	*defining the initial operation mode*
Variables	
Variable A	
Variable B	
Parameter	
Parameter A	*value or value range*
Parameter B	*value or value range*
Configuration	
Initial Configuration	*describing the configuration of the system at the beginning of the EUC*
Configuration settings	*to be considered during the EUC*

Table 7.2: Template for functions preliminary activity description

Relevant references	*list of related documents*
Other interacting function	*list of other interacting functions*
Involved actors	*list involved actors*
Involved components	*list involved components*
Related EUCs	*list related EUCs*
Inputs	
Input A	
Input B	
Outputs	
Output A	
Output B	
Parameters	
Parameter A	*value or value range*
Parameter B	*value or value range*
Initial Condition	
Position	*defining the initial position*
Operation Mode	*defining the initial operation mode*
Initial Configuration	*describing the configuration of the system at the beginning of function operation*

7.3. Description of the Preliminary Activities for EUCs

This section reports the instantiation of the template described in the previous section to the 8 considered EUCs.

Table 7.3: Trackside Initialisation preliminary activity description

Initial Condition	
Position	train positions irrelevant; trains are halted before EoA after trackside cuts communication
Operation Mode	*trackside in fail-safe (powerless) mode*
Variables	
Points status	
Signals status	
TTD status	
Configuration	
Initial Configuration	All TTDs occupied, points position unknown, signals closed
Configuration settings	Any TSR are stored in remanent memory

Table 7.4: Normal Train Movement preliminary activity description

Initial Condition	
Position	*precisely known, situated in the middle of the track*
Operation Mode	*Full Supervision*
Variables	
Train Position Report	
Track Status	
Reserved Status	
Points Status	
Movement Authority	
Configuration	
Initial Configuration	*Train integrity is confirmed. Communication timers are not expired.*
Configuration settings	*(1) Presence of TTD, (2) Trackside configured to accept integrity confirmations by driver*

Table 7.5: On Sight Movement preliminary activity description

Initial Condition	
Position	*in the middle of the track*
Operation Mode	*Full supervision*
Variables	
Train position report	
Track status	
Reserved status	
Movement Authority	
Parameter	
CYCLE_TIMEOUT	*[0, 254] s*
L3 margin	$2 \times D\_NVROLL$
Configuration	
Initial Configuration	*Train integrity is confirmed*
Configuration settings	*(1) Whether trackside is configured to accept integrity confirmation by driver, (2) Trackside Reactions if an area of track within a Reserved Status Area becomes Unknown before the L3 Trackside has authorized a train to proceed into that area*

Table 7.6: Loss of Train Integrity preliminary activity description

Initial Condition	
Position	*precisely known, situated in the middle of the track*
Operation Mode	*Full supervision*
Variables	
Train position report	
Track status	
Reserved status	
Movement Authority	
Train Integrity Status	
Train data	
Parameter	
CYCLE_TIMEOUT	*[0, 254] s*
INTEGRITY_CHECK_TIMEOUT	*[0.01:1] s*
L_TRAININT	*[0, 32767] m*
Configuration	
Initial Configuration	*Train Integrity is confirmed*
Configuration settings	*(1) Reaction of trackside in the case of lost integrity (2) Whether the trackside is configured to accept integrity confirmation by driver (3) Whether the trackside configured to authorize a Movement Authority for a train that has lost Integrity*

Table 7.7: Staff Responsible preliminary activity description

Initial Condition	
Position	*in the middle of the track*
Operation Mode	*SR mode*
Train position report	
Track Status	
Movement Authority	
Configuration	
Initial Configuration	*Communication not expired. Train integrity confirmed.*
Configuration settings	*Trackside configured to accept integrity confirmation by driver*

Table 7.8: Points Control preliminary activity description

Initial Condition	
Position	*precisely known where it is situated in the track*
Operation Mode	*full supervision*
Variables	
Point Position	
End Position	
End Position Detected	
Track Status	
Reserve Status	
Parameter	
Maximum moving time	*5sec*
Track	*Configuration of track: Occupied, Unknown, Reserved, Free*
Configuration	
Initial Configuration	*Two trains cross a point consecutively*
Configuration settings	*1. The second train requires the point to move to a different position, 2. The point cannot be moved as long as the first train occupies the associated track area, 3. The point cannot be moved when the point is already reserved for the second train*

Table 7.9: Sweeping preliminary activity description

Initial Condition	
Position	*In the middle of the track*
Operation Mode	*Full supervision*
Variables	
Train position report	
Track status	
TTD Occupancy	
Parameter	
Configurable minimum length of Unknown track status area	*to be defined (D2.3 PERFORMINGRAIL)*
Configuration	
Initial Configuration	*Train Integrity is confirmed*
Configuration settings	*List of Active shunting area*

Table 7.10: Loss of Communication preliminary activity description

Initial Condition	
Position	*The Train must not be located inside a Radio Hole*
Operation Mode	*Upon MuteTimer or SessionTimer expiration, the Trackside shall be notified of the loss of communication*
Variables	
SESSION_EXPIRED_TIMEOUT	
MUTE_EXPIRED_TIMEOUT	
NID_ENGINE	
Parameter	
Configuration	
Initial Configuration	*Train must not have entered an announced Radio Hole and must not have been sent Reversing Area Information*
Configuration settings	*Train must not enter an announced Radio Hole*

7.4. Description of the Preliminary Activities for Internal Functions

This section reports the instantiation of the template described in the section 7.2 to 12 internal functions.

Table 7.11: Trains Manager preliminary activity description

Relevant references	*X2Rail3 D4.2 - part 3, MOVINGRAIL D1.1*
Other interacting function	*Communication Management, Track Status Management, Reserved Status Management, Speed and distance supervision, Train Position Reporting.*
Involved actors	*ETCS On Board, TMS*
Related EUCs	*Normal Train Movement, On-Sight Movement, Loss of Communication, Loss of Train Integrity, Staff Responsible.*
Inputs	
Position Report	
Validated Train Data	
timeoutEvent	
Outputs	
VTDAck	
TSAunknown	
TSArelease	
TSAoccupied	
TrainLocation	
AlertTMS	
Parameters	
L3 margin	$2 \times D\_NVROLL$
Integrity Wait Timeout	*INTEGRITY_WAIT_TIMEOUT*
Initial Condition	
Position	*train in the middle of the track, with integrity confirmed and communication timers not expired*
Operation Mode	*Trackside shall define the procedure to lead the system to a safe state*
Initial Configuration	*TTD is present, the trackside is configured to accept integrity confirmation by drivers*

Table 7.12: MA Manager preliminary activity description

Relevant references	*X2Rail3 D4.2 - part 3, MOVINGRAIL D1.1*
Other interacting functions	*Route Management, Manage Dynamic Speed Profile*
Involved actors	*ETCS on board, TMS*
Related EUCs	*Normal Train Movement, In Sight Movement, Movement in SR mode*
Inputs	
RouteExtension	
RouteRestriction	
Outputs	
updateMA	
receiveMA	
Parameters	
L3 margin	$2 \times D\_NVROLL$
Initial Condition	
Position	*train in the middle of the track*
Operation Mode	*Trackside is in Full Supervision, On Sight, or in Staff Responsible*
Initial Configuration	*TTD presence*

Table 7.13: Route Manager preliminary activity description

Relevant references	*X2Rail3 Deliverable 4.2 - part 3, MOVINGRAIL Deliverable D1.1, Subset 026- part3*
Other interacting functions	*Points Management, Track Status Management, Reserved Status Management, MA management*
Involved actors	*Traffic Management System, Trackside*
Related EUCs	*Normal Train Movement, On Sight, Staff Responsible*
Inputs	
MArequest	
ReportRSA	
ReportPointsStatus	
OccupyTSA	
TSArelease	
Outputs	
RouteExtension	
RouteRestriction	
RSArequest	
RSArelease	
setPoints	
reqPointsStatus	
sweepPoints	
Parameters	
Initial Condition	
Position	*irrelevant train position*
Operation Mode	*irrelevant operation mode*
Initial Configuration	-

Table 7.14: TTD Manager preliminary activity description

Relevant references	*X2Rail-3 System Specifications*
Other interacting function	Trains_Management, Track_Status_Management
Involved actors	*TTD, TMS*
Related EUCs	*LTI EUC, Normal Train Movement*
Inputs	
train information	
TTD_STATUS	
Outputs	
ShortenTSA	
AlertTMS	
Parameters	
Synchronization Timer	*value to be defined (PERFORMINGRAIL T2.5)*
Desynchronization Timer	*value to be defined (PERFORMINGRAIL T2.5)*
Initial Condition	
Position	*train in the middle of the track*
Operation Mode	*Full Supervision*
Initial Configuration	*No faulty TTD*

Table 7.15: TSR Manager preliminary activity description

Relevant references	*X2Rail3 Deliverable 4.2 - part 3, MovingRail Deliverable D1.1, Subset 026- part3*
Other interacting function	*MAmanagement*
Involved actors	*TMS*
Related EUCs	-
Inputs	
TSRcommand	
Outputs	
TSRinfo	
Parameters	
Initial Condition	
Position	*irrelvant train position*
Operation Mode	*irrelvant operation mode*
Initial Configuration	-

Table 7.16: Reserved Status Manager preliminary activity description

Relevant references	*X2Rail3 D4.2 - part 3, MovingRail D1.1*
Other interacting function	*Communication Management, Track Status Management, Speed and distance supervision, Train Position Reporting.*
Involved actors	*ETCS on board, TMS*
Related EUCs	*Trackside Initialisation, Start of Mission, Normal Train Movement, End Of Mission, Loss/Restore Communication, Loss of Train Integrity, Shunting, Joining, Splitting, Reversing, Sweeping, Radio Hole, Points control, Movement in SR Mode.*
Inputs	
RSArequest	
RSArelease	
requestPointNotReserved	
Outputs	
reportRSAs	
reportPointsNotReserved	
Parameters	
Initial Condition	
Position	*irrelevant train position*
Operation Mode	*irrelevant operation mode*
Initial Configuration	-

Table 7.17: Track Status Manager preliminary activity description

Relevant references	*X2Rail3 D4.2 - part 3, MOVINGRAIL D1.1*
Other interacting functions	*Trains Management, Route Management, Points Mangement, TTD management*
Involved actors	*TMS/Responsible person/Dispatcher, external devices (detectors), TTD*
Related EUCs	*Trackside Initialisation, Normal Train Movement, On-Sight Movement, Loss/Restore Communication, Loss of Train Integrity, Points control, Movement in SR Mode, Sweeping*
Inputs	
TSArelease	
TSAunknown	
TSAoccupy	
enable/disable shunting area	
requestPointClear	
ttdStatus	
Outputs	
TSAreport	
reportPointClear	
TSArelease	
occupyTSA	
Parameters	
configurable minimum length	*not specified*
Train Length tolerance	*not specified*
Initial Condition	
Position	*train located in an occupied TSA*
Operation Mode	*Full Supervision, Trackside shall define the procedure to lead the system in a safe state*
Initial Configuration	-

Table 7.18: Points Manager preliminary activity description

Relevant references	*D1.1 MOVINGRAIL, PERFORMINGRAIL D1.1 and D2.1*
Other interacting function	*Points Management*
Involved actors	*TMS, Trackside*
Related EUCs	*Points Control*
Inputs	
Track section containing points occupied/reserved/unknown	
Point is locked initially	
Outputs	
Update positions of the relevant set of points	
Locking status of points	
Parameters	
Safe Point Position	*locked or unlocked*
Initial Condition	
Position	*Moving, locking and releasing of points related to two subsequent trains requesting to pass over different points*
Operation Mode	*Trackside shall prevent movement of points within an unknown or occupied Track Status Area or within Reserved Stratus Area*
Initial Configuration	*The Trackside shall be configured with Release Points to enable Points to be moved when the area of track containing the Points has Consolidated Track Status Clear and does not contain any part of a Reserved Status Area*

Table 7.19: Communication Manager preliminary activity description

Relevant references	*PERFORMINGRAIL D1.1, X2Rail-3 System Specifications*
Other interacting functions	*Track Status Management, Trains Management*
Involved actors	*TMS, TTD*
Related EUCs	*LossOfCommunication*
Inputs	
Train identification	
Outputs	
SessionTimer reset	
MuteTimer reset	
Parameters	
Session expiration	*SESSION_EXPIRED_TIMEOUT*
Mute expiration	*MUTE_EXPIRED_TIMEOUT*
Initial Condition	
Position	*Train must not have entered an announced Radio Hole and must not have been sent Reversing Area Information*
Operation Mode	*The Trackside shall notify the TrainsManagement on the expiration of the mute and/or session timers, in order to update the TrackStatusArea associated with the train*

Table 7.20: TPR Manager preliminary activity description

Relevant references	*Subset 026-part 7, PERFORMINGRAIL D1.1*
Other interacting function	"Integrity_Information_Management", "Communication_Management", "Speed_and_Distance_Supervision, "Trains_Management"
Involved actors	*Train Localization Unit*
Related EUCs	*Start of Mission, Normal Train Movement, End Of Mission, Sweeping, Loss of Train Integrity, On-Sight movement*
Inputs	
Train position	
Integrity information	
Outputs	
Train position Report	
Parameters	
CYCLE_TIMEOUT	*[0, 254] s*
Initial Condition	
Position	*in the middle of the track*
Operation Mode	*FS mode*
Initial Configuration	• 1^{st} on-board position has been reported with a "valid" status to RBC during SoM, • Valid Train Data has been sent to RBC in SoM and acknowledged by RBC to allow the train to run in FS mode, • Train Position information is referred to a LRBG, • The LRBG has transmitted correct data and this data has been correctly decoded on-board

Table 7.21: Integrity Information Manager preliminary activity description

Relevant references	*ERA CR 940_16042020, X2R2 D4.1, PERFORMINGRAIL D1.1*
Other interacting function	"Train_Position_Reporting"
Involved actors	*TIMS, Driver*
Related EUCs	*Loss of Train Integrity UC, Normal Train Movement*
Inputs	
Train DATA	
Train mode	
Train Location	
Train Speed	
Outputs	
Integrity information	
Parameters	
train integrity confirmation by driver	*boolean*
range of safe train length	*to be defined (D2.3 PERFORMINGRAIL)*
INTEGRITY_CHECK_TIMEOUT	*[0.01:1] s*
L_TRAININT	*[0, 32767] m*
Initial Condition	
Position	*In the middle of the track*
Operation Mode	*FS mode*
Initial Configuration	• Valid Train data is always available • Valid Train Data has been acknowledged by the RBC • Train Data regarding train length has not changed since the time the train was last known to be integer • Train position is referred to an LRBG • No reverse movement is currently performed • Distance between the min safe rear end at the time the train was last known to be integer and the current estimated train position does not exceed the range of the safe train length information • Position report indicating that the train integrity is confirmed has just been sent to the RBC

Table 7.22: Speed Distance Supervisor preliminary activity description

Relevant references	[49]
Other interacting functions	Dynamic Speed Profile management, TPR management, "Trains_Management", Communication management
Involved components	Train/Engine
Inputs	
VTD acknowledgement	
Train data	
Braking Curves	
Driver Commands	
Current position and speed of the train	
Outputs	
Information to the driver	
Commands to the train	
Validated Train Data to the trackside	
Initial Condition	
Position	The function operation is independent of the position.
Operation Mode	The train should be connected.

8. Moving Block Formal Models

In this deliverable, six formal models have been defined, capturing the behaviour of a few internal functions each. Fig. 8.1 shows the mapping between functions and model by rearranging and simplifying the SysML's functional architecture presented in Section 6.1. Solid lines represent some communications between the functions.

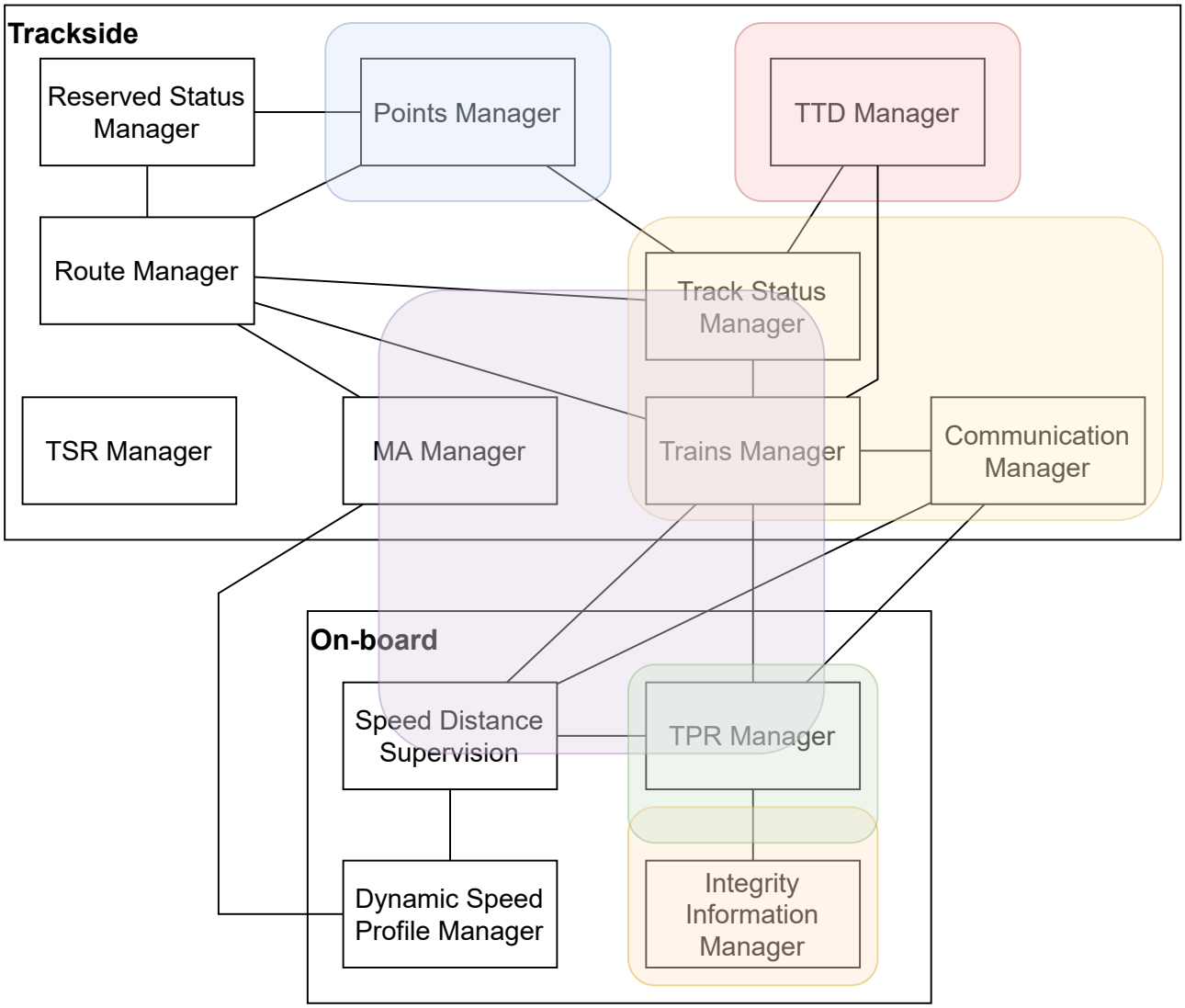


Fig. 8.1. Mapping of formal model to internal functions.

The models are:

- Communication Management & Trains Management UPPAAL model (in yellow), reported in Section 8.1;
- the IIM UPPAAL model (in orange) reported in Section 8.2;
- the TPR UPPAAL model (in green) reported in Section 8.3;
- the TTD model (in red), reported in Section 8.4;

- the Points Management UPPAAL model (in blue), described in Section 8.5;
- the SAN Movement model (in purple), described in Section 8.6, focusing on the interactions among affected functions.

8.1. Communication Management & Trains Management UPPAAL model

This subsection is dedicated to the description of the UPPAAL model of communication management & train management.

The Communication Management function is required to notify the Trains Manager upon expiration of the connection timers, signalling a loss of communication to the Trackside. Due to the nature of the function, the model was developed taking its interactions with the Trackside into consideration. For this reason, five secondary automatons have been included, with particular attention towards those that represent the track status management and, of course, the trains management.

Model Structure The main automaton, CommunicationManagement, is strongly based upon the state machine described in Figure 6.60. The same is true for TrainsManagement and TrackStatusManagement, developed from the state machines depicted in Figure 6.44 and 6.46. The Train automaton is a stub: it emulates a very generic train behaviour by periodically sending TPR messages in order to simulate a Train-Trackside interaction. The Train can also spend an arbitrarily long amount of time idling, to mimic a connection interruption during which the Trackside is not receiving messages from the Train. In addition to these machines, an extremely simple stub of the RouteManagement and TrainIntegrity functions has also been included in order to avoid deadlocks due to the impossibility of receiving messages on synchronization channels.

Model Description This initial phase of the model is intended to have a process instantiation for each train. In other words, each living train will have its own CommunicationManagement, TrainsManagement, etc., each of which will have the train identifier as a parameter. In the future, the model will be refined so as to make it dependent on a single TrainsManagement instance, which will be able to handle multiple trains at once with a buffered message system. TrainsManagement and TrackStatusManagement also require the train length as a parameter, in order to perform comparisons between the received data and the valid data.

- *CommunicationManagement*: The main automaton starts in the WaitForTpr state. While in this state, when receiving a TPR from the Train, the machine resets both timers and loops back into this state. Upon expiration of the mute timer the automaton will go into the WaitForReconnection state, from which there are two possibilities: either the session timer will expire before receiving a message, causing the process to terminate, or the automaton will receive a TPR from the train. In this case TrainsManagement will check the validity of the TPR by comparing the received train id and length with the correct train values, and if the TPR is valid the machine will re-enter the WaitForTPR state, while if the data does not match it will terminate. Termination can also ensue from the WaitForTPR state if the session timer expires, which will happen if the mute timer is not configured.
- *TrainsManagement*: The TrainsManagement automaton handles the messages sent by CommunicationManagement on the connection status of the train. Almost all of

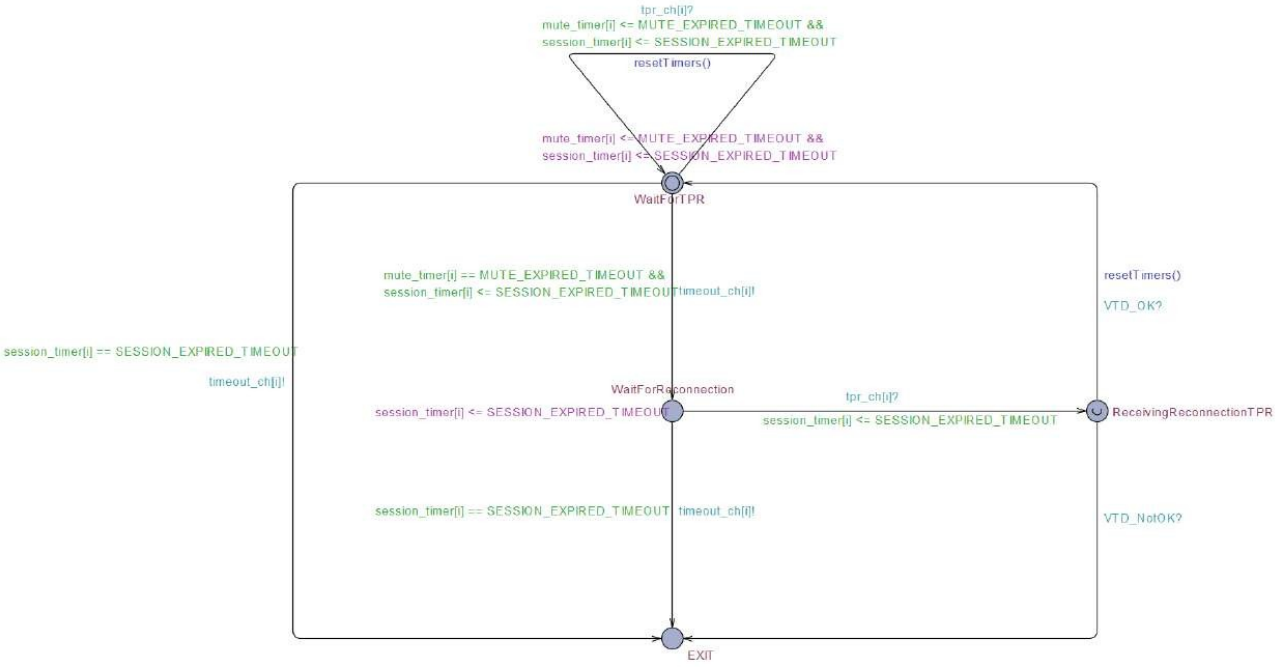


Fig. 8.2. Communication Management UPPAAL model

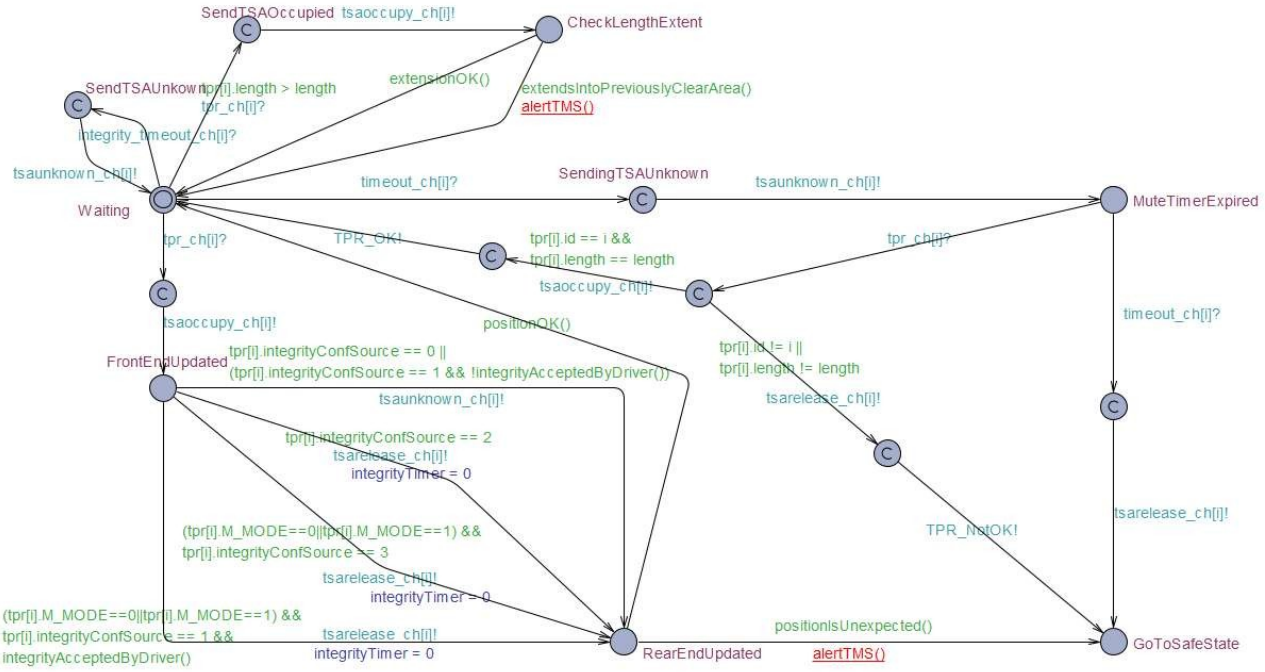


Fig. 8.3. TrainsManagement UPPAAL model

the transitions have been included, albeit with extensive use of boolean abstraction in those that did not explicitly involve interactions with CommunicationManagement. From the initial Waiting state, the automaton will be able to perform a series of actions, the one relevant to the CM function being the one connecting the Waiting state with the committed state SendingTSAUnknown and ending in the MuteTimerExpired state. This

pair of transitions represents the reception of a timeout signal from the CM automaton, which will be the one sent at mute timer expiration. From the MuteTimerExpired state, TM will be waiting for a TPR in order to attempt reconnection. If it receives another timeout signal from CM, that will mean that the session timer has expired, and the automaton will terminate by going into a safe state. If instead the TPR is received, TM will check the validity of the data sent by the train, returning to the Waiting state and notifying CM if the train id and the length are confirmed, and exiting otherwise.

- *TrackStatusManagement* TrackStatusManagement receives the update signals from TrainsManagement. By convention, it starts in the Occupied state, and loops back in it if it receives TSA occupation and release signals by the Trains Management. If, instead, it receives a signal to set the TSA to unknown, then the automaton will transition to either the Removed or Unknown state, depending on whether the train is completely located in an active shunting area or not, respectively. From the Unknown state, returning to the Occupied state is possible upon receiving a TSA occupation signal only if the source of confirmed integrity is the driver and the integrity is accepted by the driver, or if said source is the monitoring device. Receiving the occupation signal without these conditions leads to exiting in a Safe State or looping back into the Unknown state, depending on whether the length communicated by the train is confirmed.

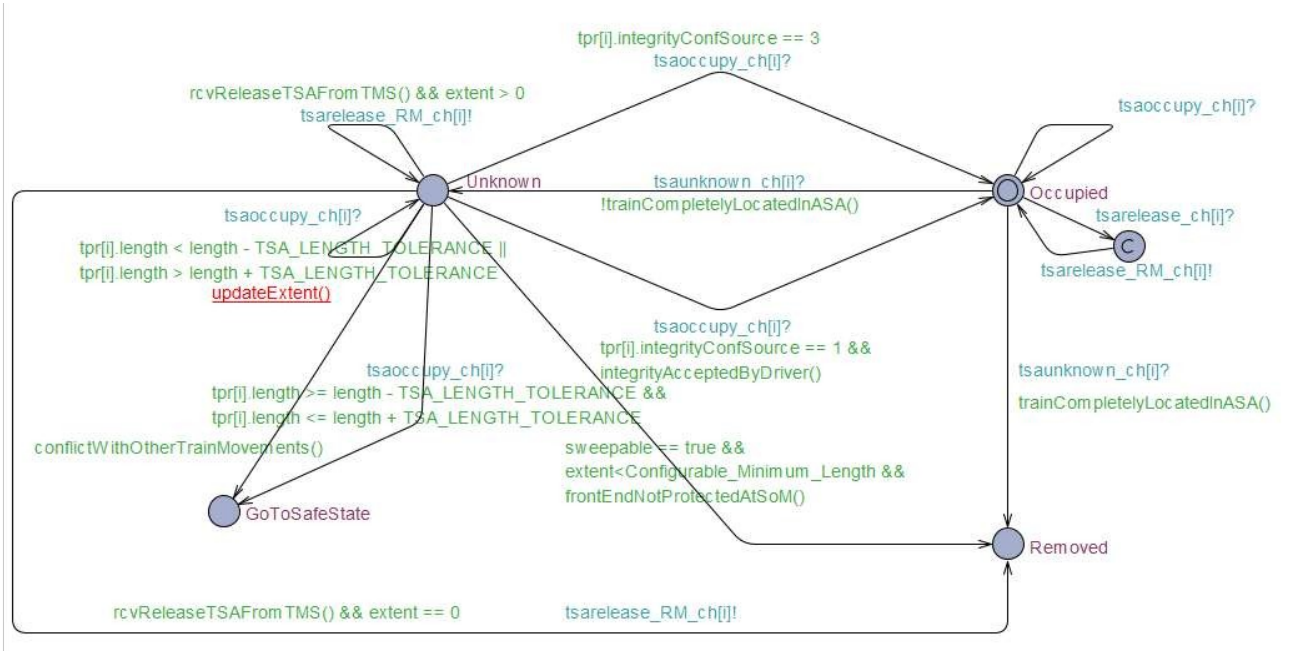


Fig. 8.4. TrackStatusManagement UPPAAL model

- *RouteManagement and TrainIntegrity*: These two automata are completely void of any meaningful transition, and exist for the sole purpose of keeping the other automata alive by receiving and sending necessary messages. TrainIntegrity, in particular, also keeps track of the integrityTimer, sending a signal to TrainsManagement in case of expiration. Due to their triviality, no image has been included in this document.
- *Train*: The Train automaton is a stub, as it also serves the simple purpose of continuously sending TPR messages on a broadcast channel to CommunicationManagement

and TrainsManagement. More specifically, this automaton starts in the WaitingPeriod state and waits a fixed amount of time (TRAIN_MSG_PERIOD) after which it can randomly perform three actions: send a correct TPR, send a TPR with invalid data (incorrect length) or it can simply spend more time doing nothing (IDLE_TIME). This last option is to ensure that the session and mute timers can expire for simulation and verification purposes.

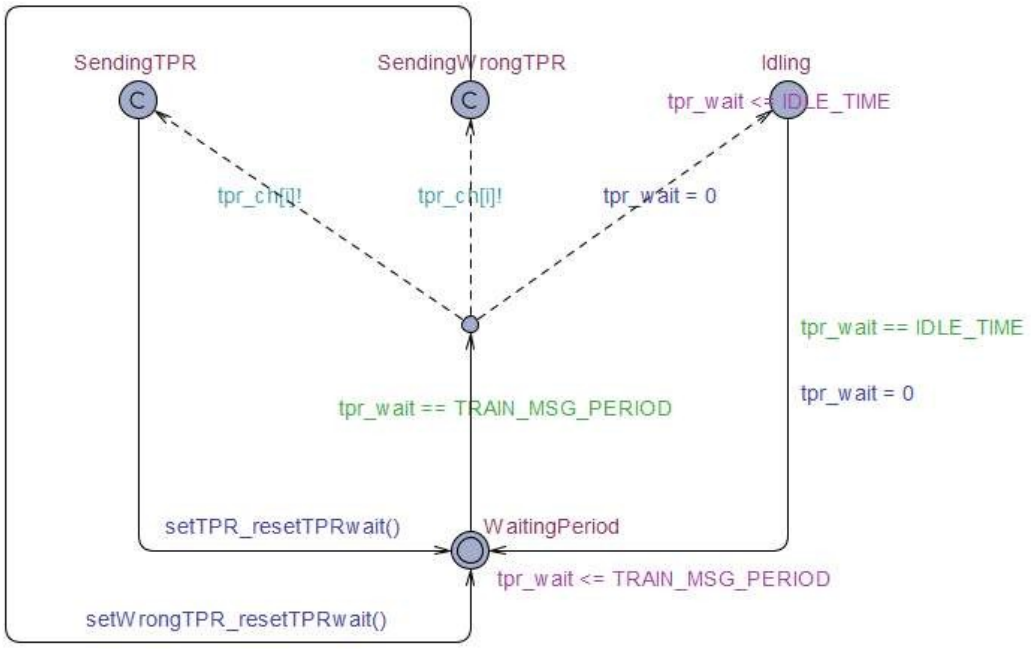


Fig. 8.5. Train stub UPPAAL model

Limitations At the moment, the train movement is not modeled, and most of the transitions in TrainsManagement and TrackStatusManagement depend on position updates sent by it. The problem has been temporarily solved with the aforementioned use of boolean abstraction: every transition guard denoted by a function evaluates to true. Also, some message-sending operations have been modeled with an update, while they should instead be channel synchronizations (e.g., AlertTMS()). This is due to the fact that the automatons with which the communication should happen have not been modeled.

8.2. IIM UPPAAL model

This subsection is dedicated to the description of the UPPAAL model of the *"Integrity_Information_Management"* internal function. The main role of the *"Integrity_Information_Management"* function is to monitor the integrity status of the train by taking into account all the relevant information.

Model Structure The *"Integrity_Information_Management"* (IIM) function receives signals from the *TIMS* and the *Driver* about the train integrity. Then, it computes the train integrity status which can have 4 values: *'Integrity_Confirmed_Driver', 'Integrity_Confirmed_TIMS'*,

'Integrity_Lost' or *'No_Train_Integrity'*. The switching from one status to another is described in the transition table shown in Figure 6.62. The transition conditions are reported in Table 8.1. All these conditions have been implemented in this formal model, except conditions 3 and 9 which require a representation of the train movement. From the conditions table, in order to emulate the behaviour of the IIM function, the following data are considered:

- Train Data sent from *Train* to *"Speed_and_Distance_Supervision"* function;
- the acknowledgement of Train Data sent from *"Trains_Management"* function to the *"Speed_and_Distance_Supervision"* function;
- the Train Position Report sent from *"Train_Position_Reporting"* function to the *"Trains_Management"* function;
- the integrity Status sent from the *"IIM"* function to the *"Train_Position_Reporting"* function.

The structure of the formal models representing the *"IIM"* function is composed of 10 automata:

- "TIMS" automaton which emulates the behaviour of the *TIMS* block by sending 3 signals: $'Train\_Integrity\_Unknown'$, $'Train\_ntegrity\_Confirmed'$ and $'Train\_Integrity\_Lost$'. This automaton is represented in the figure 8.6.

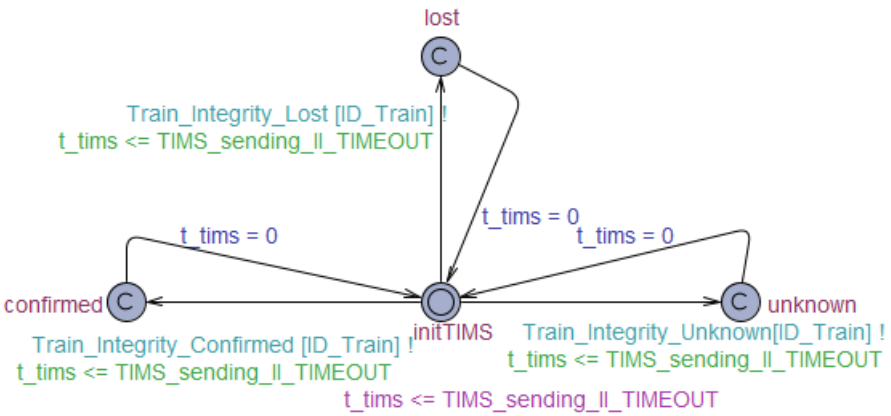


Fig. 8.6. *TIMS* automaton

- "Driver" automaton which emulates the behaviour of the *Driver* block by sending signal $'Integrity\_Confirmed\_By\_Driver$".
- "IIM_StatusManagement" automaton which intercepts *TIMS* and *Driver* signals, and according to the current status of integrity and other conditions described in Table 8.1, performs the switching from one state to another. This automaton is represented in the figure 8.7.

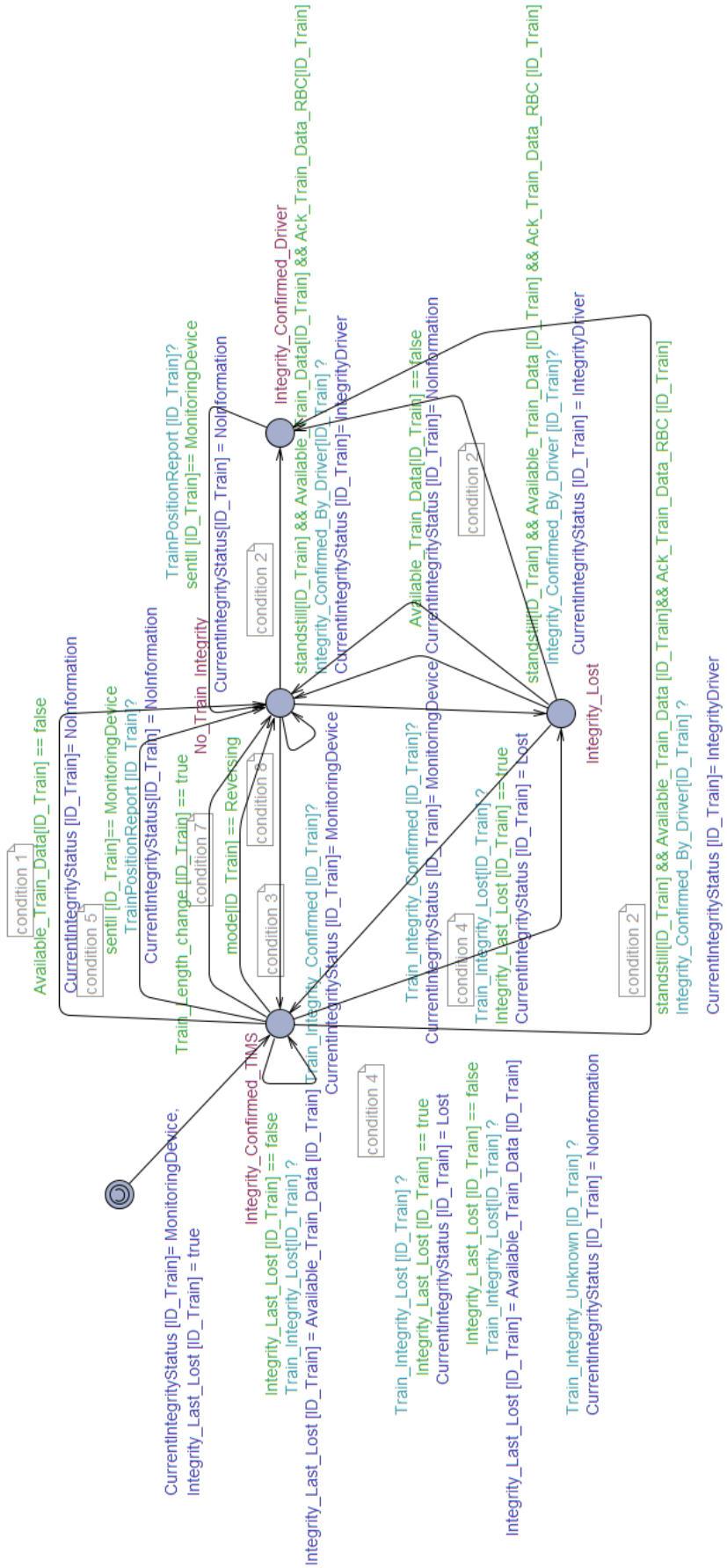


Fig. 8.7. *IIM_StatusManagement* automaton

- "Train data" automaton which sends to the function *"Speed_and_Distance_Supervision"* signals $'VALID\_TRAIN\_DATA'$ and $'INVALID\_TRAIN\_DATA'$. Train data are required to compute integrity status (conditions 1, 2, 3, 4).
- "Speed and Distance supervision" automaton receives from "Train Data" automaton the signals about Train data, sends these information to the *"Trains_Management"* function and waits for acknowledgement from this latter. This automaton also receives the train speed information from "Train speed" automaton and can, therefore, determine whether the train is in standstill or not.
- "Train speed" automaton which emulates the speed of the train by sending the train speed information to the automaton "Speed and Distance supervision".
- "Trains_management" automaton receives Train position report from The "Train_Position_Reporting" automaton, receives Train Data from "Speed and Distance supervision" automaton, and sends an acknowledgement upon reception. This automaton is represented in the figure 8.8.

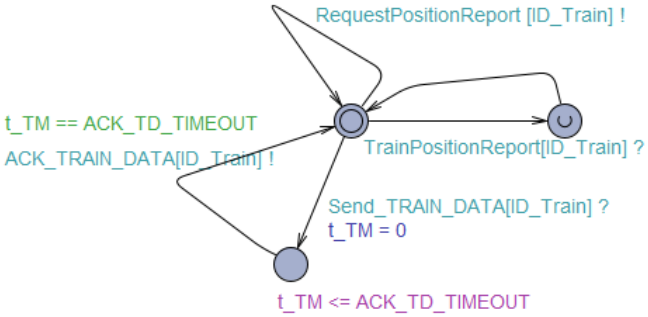


Fig. 8.8. "Trains_management" automaton

- "Train_Position_Reporting" automaton which receives the current integrity status from "IIM updating" automaton and sends the train position report to "Trains_management".
- "IIM updating" automaton which sends each INTEGRITY_CHECK_TIMEOUT the current integrity status to the "Train_Position_Reporting" automaton. It is represented in the figure 8.9.

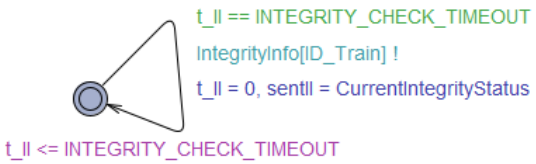


Fig. 8.9. *IIM updating* automaton

- "Train mode" automaton which emulates the switching between the different ETCS operation modes of the train. It is designed particularly in order to consider the condition 8.

Model Description These conditions have been considered in the formal model in the following ways. Condition 1 is considered in the interaction between the "Train

Table 8.1: Switching Conditions

SWITCH_ID	Content of the conditions
[1]	No valid Train data is available
[2]	(Train is at standstill) AND (valid Train Data is available and has been acknowledged by the RBC) AND (the train integrity is confirmed by the driver)
[3]	(The information "Train integrity confirmed" is received from an external device) AND (valid Train Data is available and has been acknowledged by the RBC) AND (Train Data regarding train length has not changed since the time the train was last known to be integer) AND (the train position is valid and is referred to an LRBG) AND (the train position was valid and was referred to an LRBG at the time the train was last known to be integer) AND (no reverse movement is currently performed nor has been performed since the time the train was last known to be integer) AND (the distance between the min safe rear end at the time the train was last known to be integer and the current estimated train position does not exceed the range of the safe train length information)
[4]	(The information "Train integrity lost" is received from an external device) AND (valid Train Data is available since the time the train integrity was last known to be lost)
[5]	A position report indicating that the train integrity is confirmed is sent to the RBC
[6]	The information "Train integrity status unknown" is received from an external device
[7]	Train Data regarding train length is changed
[8]	A reverse movement is performed
[9]	The distance between the min safe rear end at the time the train was last known to be integer and the current estimated train position exceeds the range of the safe train length information

data" automaton and the "speed and distance supervision" automaton. It is represented by the signal $'INVALID\_TRAIN\_DATA'$. Condition 2 is involved in the interaction between "Train speed" automaton, "Speed and Distance supervision" automaton, "Trains management" automaton, "Train data" automaton, "Driver" automaton and "IIM_StatusManadgement" automaton. Condition 4 is involved in the interaction between "TIMS" automaton, "Train data" automaton, "speed and distance supervision" automaton and "IIM_StatusManadgement" automaton. Condition 5 is involved in the interaction between

"Train_Position_Reporting" automaton, "Trains management" automaton, "IIM updating" automaton and "IIM_StatusManadgement" automaton. Condition 6 is involved in the interaction between "TIMS" automaton and "IIM_StatusManadgement" automaton. Condition 7 is involved in the interaction between "Train data" automaton, "Speed and Distance Supervision" automaton and "IIM_StatusManadgement" automaton. Condition 8 is involved in the interaction between "Train mode" automaton and "IIM_StatusManadgement" automaton.

Property specification and verification Some preliminary reachability properties have been verified on this model, to ensure that all the integrity statuses are covered.

Limitations To be able to check the relevant functional and safety properties of *"Integrity_Information_Management"* function, modelling the train movement is required. The model to emulate train dynamics has not been developed so far, but this will be part of the following modelling activities.

8.3. TPR UPPAAL model

This subsection is dedicated to the description of the UPPAAL model of the *"Train_Position_Reporting"* which is an on-board function. This function is in charge of sending to the *"Trains_Management"* function a train position report which includes, mainly, the train position and the train integrity information.

Model Structure The formal model for *"Train_Position_Reporting"* function is composed of four automata:

- "Train position reporting" automaton, which regularly sends requests for location to the "localization unit" automaton. It receives from "IIM updating" automaton the integrity status ($'integrityInfo'$), and from the "localization unit" automaton the train location. It compiles the train position report and sends it to the *"Trains_Management"* function. This automaton is represented in Figure 8.10. In this figure, the $'integrityInfo'$ loops at three states in order to catch at any state the reception of integrity status.

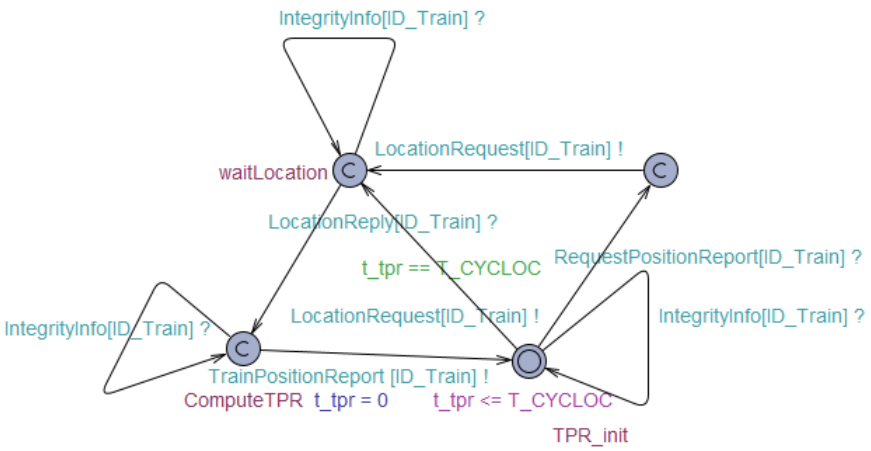


Fig. 8.10. *Train position reporting* automaton

- "Train localization Unit" automaton, which regularly receives from the train position reporting automaton requests for location. It computes the location and sends it back to the "Train position reporting" automaton.
- "IIM updating" automaton, which sends the current integrity status to the "Train position reporting" automaton.
- "Trains_management" automaton, which sends, under some conditions related to train location, a request for train position report to "Train position reporting" automaton and receives train position reports from the "Train_Position_Reporting" automaton. As the train location is not designed, only the request for train position $'RequestPositionReport'$ is modelled.

Limitations The interactions between the train position reporting function and the other related functions and external actors are represented in this model. The main limitation is that the train movement is not emulated yet, thus no precise train location can be computed so far. This issue will be addressed in the following phase of the project.

8.4. Trackside Train Detection UPPAAL model

This subsection is dedicated to the description of the UPPAAL model of the *"TTD_Management"* function which is a trackside function devoted to managing the status of TTDs in the covered area. It is used only for an MB system that uses Trackside Train Detection means (axle counters, track circuits). On the one hand, the function receives and computes a report from the Trackside Train Detection. On the other hand, it receives and computes train location information from *"Trains_Management"* function. Therefore, the *"TTD_Management"* function is responsible for managing the lack of synchronization between TTD occupancy and trains information during normal operation.

Model Structure The formal model for *"TTD_Management"* function is composed of seven automata:

- an automaton "TTD" for each TTD, which emulates the status change of the *TTD* block. This automaton sends the TTD status every desynchronization timeout. This automaton is represented in Figure 8.11.

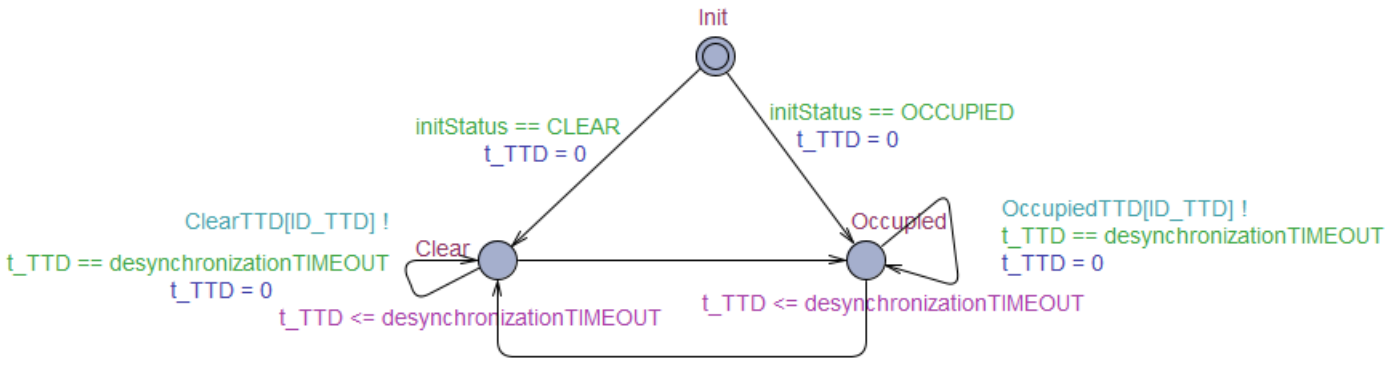


Fig. 8.11. "TTD" automaton

- an automaton “TM_SendTrainInfo” for each train, which emulates the behaviour of the *“Trains_Management”* function. It sends Train information to the automaton “TTDM_ReceiveTrainInfo” upon receiving a train position report.
- an automaton “TTDM_ReceiveTTD”, which represents a part of the behaviour of the *“TTD_Management”* function. It receives the report from each TTD. In the case that a desynchronization timer is assigned to a TTD (due to status inconsistency issue) and if in the next status reporting the status of the TTD is still reported as clear, the *TMS* is alerted about this abnormal situation. This automaton is represented in Figure 8.12.

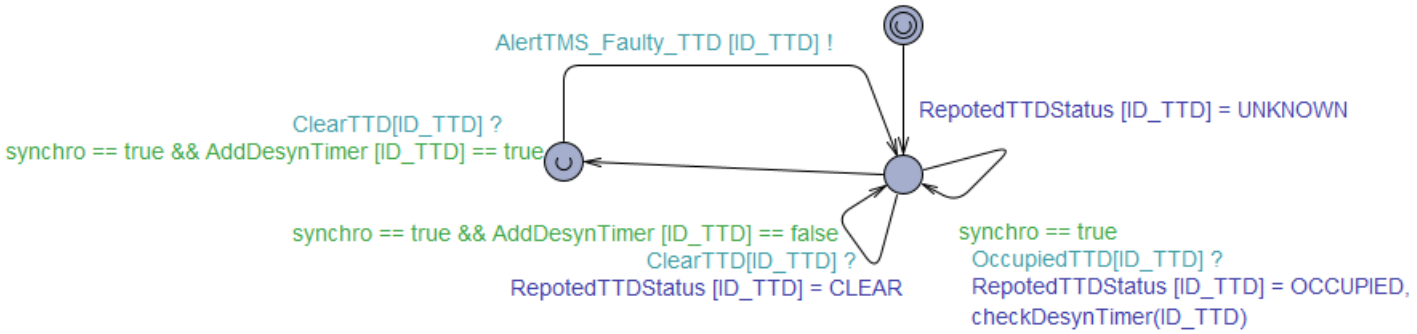


Fig. 8.12. *“TTDM_ReceiveTTD”* automaton

- an automaton “TTDM_ReceiveTrainInfo” for each train, which represents a part of the behaviour of the *“TTD_Management”* function. It receives the train information from “TM_SendTrainInfo” automaton.
- an automaton “TTDManagement” which represents a part of the behaviour of the *“TTD_Management”* function. It performs a synchronization between the received train information and TTD report every synchronization timer using the function $'Synchronise()'$. This function computes for each train, within the area of control of the *“TTD_Management”*, the MaxSFE, the minSFE, the MaxSRE and the minSRE. In the case when the MaxSFE and the minSFE are located in a clear TTD (see Figure 6.51), a desynchronization timer is triggered to this TTD, to monitor whether the TTD is designated as occupied in the next TTD report. If it is not the case, the *TMS* is alerted. In the case that the MaxSFE is located in a clear TTD and the minSFE is located in an occupied TTD (see Figure 6.50), then a “shortening” must be performed ($'checkShorten = true'$). In the case when the MaxSFE is located in an occupied TTD and the minSFE is located in a clear TTD, then the *TMS* shall be alerted. The “TTDManagement” automaton is represented in Figure 8.13.
- an automaton “TMS_TTDM” emulating the behaviour of *TMS* which is alerted either in the case that a TTD is clear while it should be occupied $'(AlertTMS\_Faulty\_TTD[ID\_TTD])'$, or when the train location is incoherent with TTD occupancy ($'AlertTMS\_TTDM'$).
- an automaton “TrackStatus_TTD”, which emulates the behaviour of the *“Track_Status_Management”* function. It receives from “TTDManagement” automaton a release request, and sends a release completed.

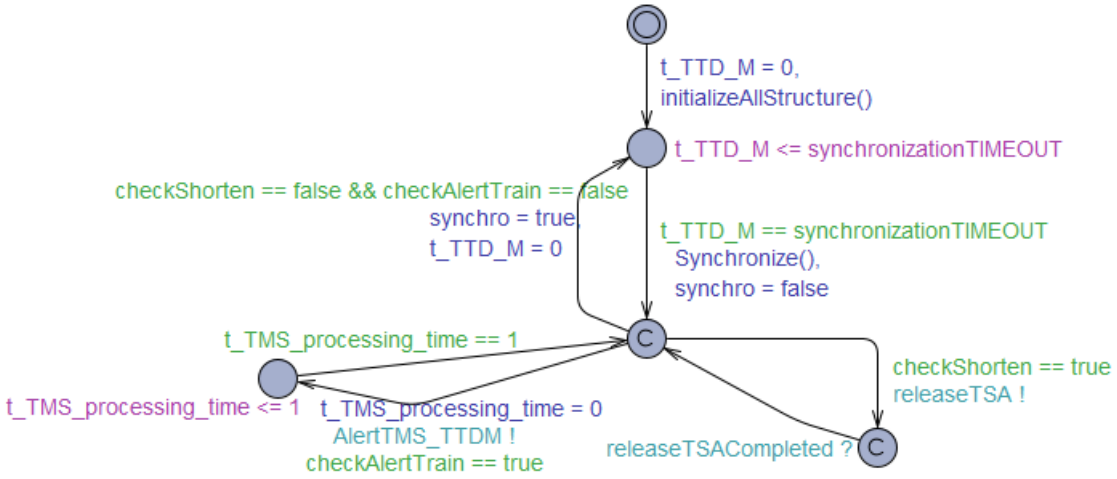


Fig. 8.13. “TTDManagement” automaton

8.5. Points Management UPPAAL model

This subsection is dedicated to the description of the UPPAAL model of the Points Management which is the function of the trackside devoted to the management of points locking and unlocking.

Model Structure The Points Management function has been modelled via a network of four UPPAAL timed automata, as explained below. The Points Management function receives the status of the track section containing the points to be moved, from the *Track/Reserved Status Management* module. Then, the initial pseudo-state of the behaviour ($init$) of *Points Management* brings the latter to the waiting state. From this state, three behaviours, that is, *nominal*, *degraded* and *sweepable* can be achieved via corresponding transitions, as shown in the SMD of Fig. 6.57. The *nominal* and *degraded* transitions bring the diagram to the *CreateRoute* composite state, as seen in Fig. 6.58. The composite state *CreateRoute* models the creation of a route from the point of view of *Points Control*, which involves moving the necessary points to appropriate positions depending on the received *track status*. *CreateRoute* first checks if the involved points in the route to create are in an area that is *Occupied/Unknown* or *Reserved*. In the composite state *SetandLock*, the route is created by setting the points in their new position. A point’s position at rest can be either “Left” or “Right”, and a point is moved to a new detected position that constitutes the final position of the point. The network of timed automata formal models of the *Points Management* function consists of the following four automata:

- *Points Management* main automaton that communicates with the three other timed automata.
- *CreateRoute* timed automaton is responsible for creating the route after checking the received value of the track status that can be *Occupied/Unknown*, *Clear*, or *Reserved*. In the nominal case, that is, if *TrackStatus* is *Clear*, the automaton synchronizes with the *SetandLock* timed automaton, via the synchronization channel *setandlock*.
- *SetandLock* timed automaton is responsible for setting the points in their new positions; for this, the automaton detects the final position of the point (after the point has been

moved to a new detected position), which can be either “Left” or “Right”, and it also models the error location reachable once the prescribed moving time of the blades is exceeded.

- *SweepableandOverride* timed automaton describes the case of a sweeping train that frees some points.

Model Description A detailed description of this UPPAAL formal model follows. In the *CreateRoute* timed automaton, represented in Figure 8.14, the variable *Track Status* encodes the track’s occupancy status, as received from the *Track Status Management* function. The variable ranges over the constant values of “*Clear*”, “*Occupied*”, “*Unknown*”, and “*Reserved*”. These values are used to check the value of *Track Status* variable and to distinguish between the nominal behaviours and exceptional ones.

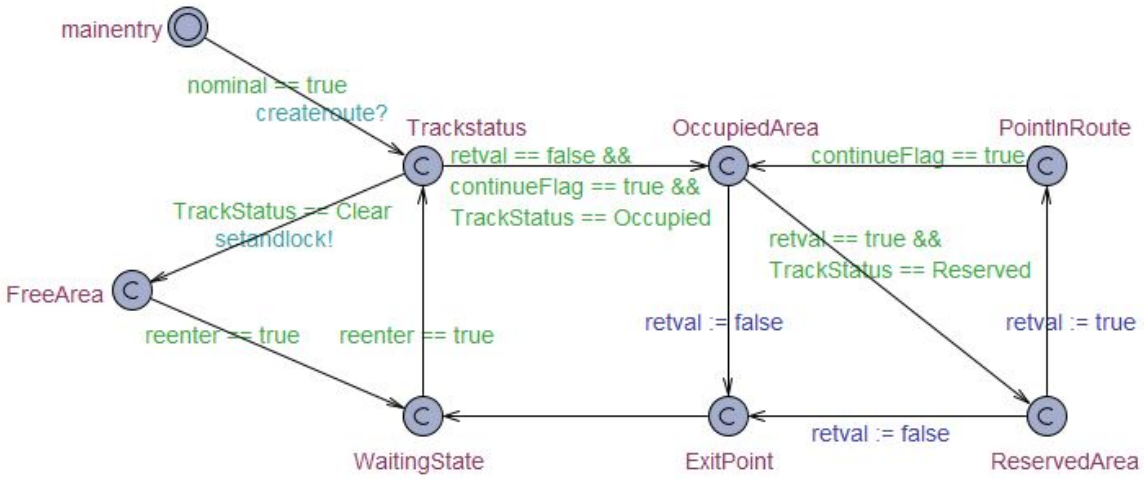


Fig. 8.14. CreateRoute automaton

In the *SetandLock* timed automaton, shown in Figure 8.15, the variable *PointPosition* encodes the position of the point in question, via setting the variable’s value to either“Left” or “Right”. *EndPosition* encodes the point’s position at rest and its initial values can be either “Left” or “Right”. Then, it is checked if *EndPosition* is “Left”, in which case the automaton moves to *PointPositionLeft* and assigns *PointPosition* variable to value “Left”, or if it is “Right”, then the automaton moves to location *PointPositionRight* and executes the assignment *PointPosition:= Right*. The clock variable *time* is used to measure the elapsed time for the moving blades. When the prescribed moving time needed to reach the new point position exceeds its maximum allowed value (modelled by guard $time > maxtime$), the *error* state of the *SetandLock* timed automaton is reached (see Figure 8.15).

Property specification and verification Some preliminary reachability property verification — e.g., to check the sanity of the model or if the *error* location is reachable — are verified on the model. Further invariance and liveness properties will be verified on an extended and improved model of the *Points Management* function.

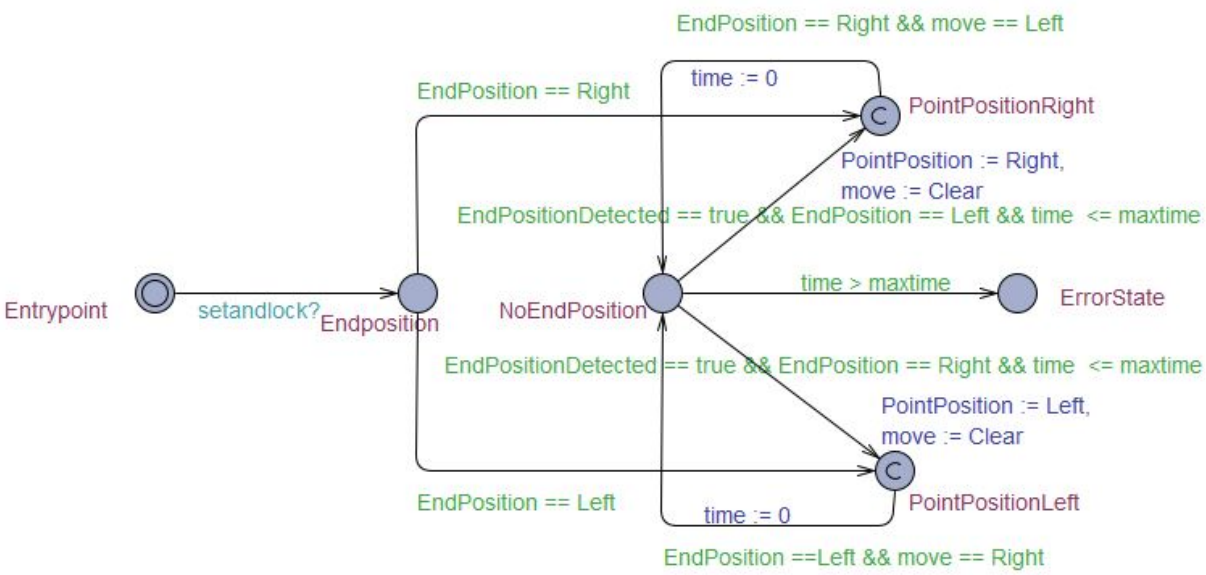


Fig. 8.15. SetandLock automaton

Limitations To be able to verify essential invariance (e.g., safety) properties, as well as liveness properties on this function, the degraded scenarios of overriding in case of unknown track status, as well as lost connection, need to be accounted for in the formal model. Such situations have not been developed yet, future modelling activities will take care of these aspects.

8.6. Movement SAN model

This subsection is dedicated to the description of the SAN model whose objective is to enable the evaluation of performance and performability properties at the system level. These properties are evaluated regarding the movement of a train fleet on a track under the control of a trackside.

Model Structure The SAN model represents a set of trains that, after entering the line, periodically compute their positions and speeds on the basis of the current EoA dynamically assigned by the trackside to it (assuming that the trains run at their maximal speed). Meanwhile, each train periodically sends the Train Position Report to the trackside. On the reception of a Train Position Report from a specific train, the trackside updates the extent of the Track Status Area associated with that train on the basis of the communicated position and integrity status. Then the trackside updates and sends the Movement Authority for that train, up to the ”known” tail of the preceding train (plus a safety margin). When receiving a Movement Authority message from the trackside, the train updates its information regarding the distance it is enabled to run. At last, the model can also consider the effect of possible failures, that are, in the current status, integrity not confirmed by the train and the loss of messages (both Train Position Report and Movement Authority) during the communication.

The model has been developed as a composition of reusable atomic SAN models that are joined by means of place superposition, i.e., by sharing state variables. Namely, the atomic models are sub-models that can be replicated, instantiated, and composed. The structure

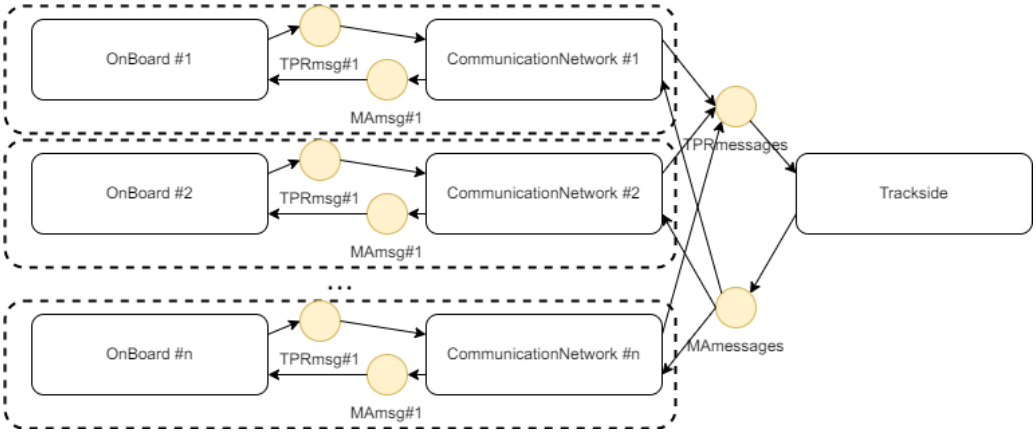


Fig. 8.16. SAN composed model

of the current composite model is shown in Fig. 8.16. It includes n replicas of the on-board and Communication network models, and just one trackside sub-model. The figure also highlights the main superposed places that are extended places in which proper messages (i.e., TPRs and MAs) are stored. The arrows show the logical data flow in the composed model.

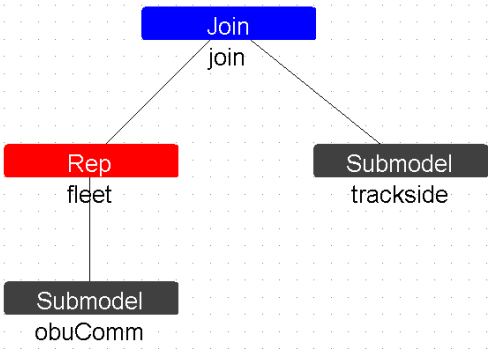


Fig. 8.17. SAN composed model

Möbius is the adopted tool for both SANs modelling and solution. A single atomic model for the onboard and communication is developed. Hence, the joint model, as obtained in Möbius, is depicted in Fig. 8.17, where a *join* operator merges the trackside atomic model with a replica (*rep* operator) of obuComm atomic models.

Model Description The atomic model of the on-board and the communication network is depicted in Figure 8.18. The model contains 10 places, 8 extended places, 5 timed activities, 2 instantaneous activities, 1 input gate, and 9 output gates. For sake of clarity, the different parts of the model are grouped together in the figure by rectangles, described in the following:

- *train scheduling*: the place `arrivalTime` stores the arrival time (in seconds) of each train over an array of short values, i.e., one value for each train. These values initialized through a custom initialization code. This place is shared among the replicas.
- *local info of the train fleet*: the 4 extended places `trainSpeed`, `trainMAs`, `trainHeadPositions`, and `trainTailPositions` store respectively the speed of

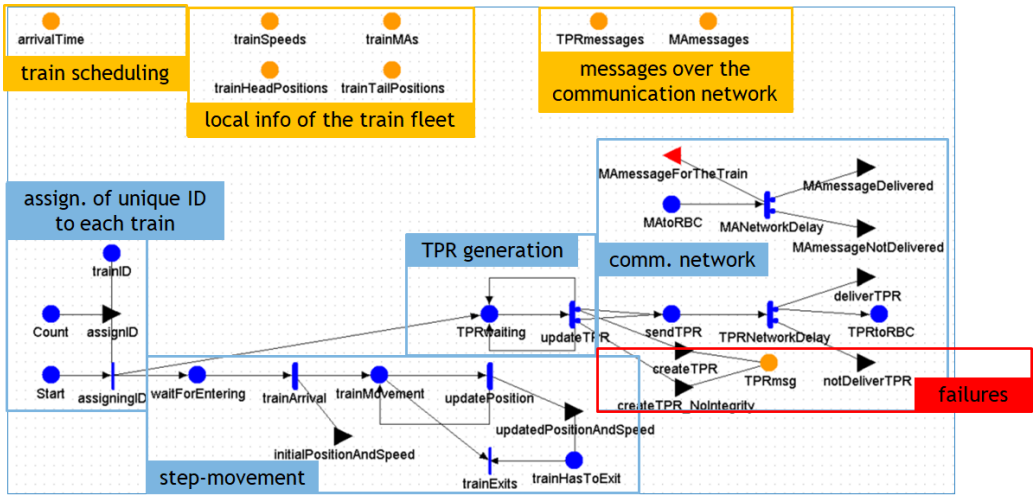


Fig. 8.18. SAN on-board unit + communication atomic model

the trains in meters/second, the end of Movement Authorities, the head and the tail positions of the trains. Each place contains an array of short values, i.e. one value for each train. All these places are shared among the replicas.

- *messages over the communication network*: the two extended places `TPRmessages` and `MAmessages` store the messages over the communication network, and specifically the Train Position Reports and the Movement Authorities. Each place contains an array of data structures, where values in a certain position represent a message from/to a specific train. These places are shared among the replicas and with the trackside model.
- *assign. of unique ID to each train*: this portion has the goal of storing, for each replica, a number of tokens in the place `trainId` from 0 to $n-1$, where n is the number of trains. A token in the place `Start` enables the activity `assigningID`, which fires and enables the execution of the output gate `assignId`. Because of the number of tokens in the place `Count`, shared by all the replicas, the code in the output gate puts a certain number of tokens in the place `trainId` equal to the identifier assigned to this replica.
- *step-movement*: it models the movement of the train as a sequence of steps. A token in the place `waitForEntering` enables the activity `trainArrival` that models the train scheduling. The firing time of this activity depends on the values stored in the place `arrivalTime`. The output gate `initialPositionAndSpeed` computes the initial position and speed. Kindly note that the train with id 0 has a movement authority for the entire track by default. A token in the place `trainMovement` enables the activity `updatePosition` that cyclically fires and enables the execution of the code in the gate `updatePositionAndSpeed`. When the train reaches the end of the modelled line, a token in `trainHasToExit` enables the activity `trainExits` and removes the token in `trainMovement`.
- *TPR generation*: similarly to the previous section, this section has the goal of generating continuously the Train Position Report message. A token in the place `TPRwaiting` periodically activates the activity `updateTPR`, which enables the output gate `createTPR`. The code in this gate calculates the fields of the TPR message based on the current position and speed, and stores the information in the place

TPRmsg.

- *comm. network*: it considers the communication delay on the messages exchanged between each train and the trackside. When a token is present in sendTPR, it enables the activity TPRNetworkDelay that, when fires, updates the messages in the two extended places TPRmessages and MAmessages through the output gate deliverTPR. When a Movement Authority needs to be delivered to the train, the input gate MAmessageForTheTrain and a token in MAtoRBC enable the transition MANetworkDelay, which fires after a certain delay and updates the data stored in trainMAs through the code in MAmessagesDelviered.
- *failures*: this portion models the considered failures. According to certain probabilities, the activities updateTPR, TPRNetworkDelay and MANetworkDelay could activate the different cases, connected to proper output gates, which represent respectively the generation of a TPR with integrity not confirmed, the loss of a TPR message and the loss of an MA in the communication network.

The atomic model of the trackside is depicted in Fig. 8.19. The model contains 4 places, 4 extended places, 1 timed activity, 1 instantaneous activity and 1 output gate. The description of the different sections follows:

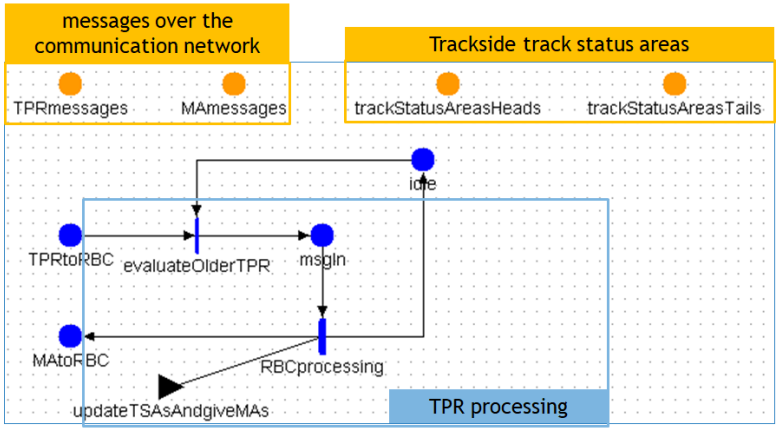


Fig. 8.19. SAN trackside atomic model

- *messages over the communication network*: as described before, the two extended places, shared with the replicas of the trains, represent the messages on the communication network.
- *Trackside track status areas*: the two extended places trackStatusAreasHeads and trackStatusAreasTails models the track status variables of the trackside. Each place contains an array of short values, which represent respectively the front and the back position of a track status area. The values in the position i of the arrays represent the track status area associated with the train with the id i.
- *TPR processing*: this portion models the processing of the TPR messages by the trackside. A token in the place TPRtoRBC means a message ready to be analysed. The trackside has been modelled as a shared resource, able to process a single message at a time. In fact, a single token in the place idle is consumed when the message is processed and regenerated when the processing ends. The activity RBCprocessing models the processing time of the RBC. The output gate updateTSAsAndgiveMAs,

executed when `RBCprocessing` fires, executes the code needed to update the track status areas (and store the values in the proper extended places) and generates the MA for the train on the basis of the "known" position of the preceding train.

Property specification and verification The property specification in SANs is done through the definition of reward variables. A reward variable, computes the number of trains that can cross the line regarding the simulation parameters described in the following. All the arrival times of trains to the value 0. The model parameters are the following:

- Train scheduling: the arrival times (in seconds) of each train is specified through the custom initialization code. If this value is set to 0 for all the trains, the simulation evaluates the maximum number of trains the system can manage in the given time slot.
- Number of trains: dimension of the train fleet, which implies the number of times the atomic model should be replicated.
- Period of updating position: duration (in seconds) of the time period of position update. The shorter is this value, the more accurate is the simulation of the train running.
- Period of sending TPR: duration (in seconds) of the time period of generating a new TPR by the trains.
- Line length: extent (in metres) of the line under the control of the trackside.
- Maximum Train Speed: maximum value (in metres/second) for the train speed (equal to all the trains). For sake of simplicity, this is considered as constant along the track, but with a custom initialization code and an additional extended place it is possible to add a static speed profile.
- RBC processing time: duration (in seconds) of the processing of the TPR and the consequent generation of the MA by the trackside.
- Communication time from RBC to train of MA: delay (in seconds) introduced by the communication network for the delivering of Movement Authorities.
- Communication time from train to RBC of TPR: delay (in seconds) introduced by the communication network for the delivering of Train Position Reports.
- Probability of not confirming integrity: probability of occurrence of a temporary fault preventing the confirmation of integrity by the train.
- Probability of not delivering: probability of occurrence of a temporary fault preventing the delivering of messages by the communication network.

Limitations In the current state, the developed model can consider straight lines with constant static speed profile; the presence of merging and diverging junctions has not been considered. However, this limitation has been considered not relevant for the scope of the analysis carried-out by this model.

9. Discussion

This deliverable is dedicated to the presentation of the results of both specification and modelling activities of the ETCS-L3 system. Although some previous research works have dealt with the formal modelling of this system, according to what we have reported in Chapter 3, the scientific background does not provide any systematic attempt to define both a modelling methodology and a model encompassing the various functionalities of ETCS-L3.

The first valuable result of T2.3 lies in the definition of a SysML model built on the basis of the ETCS-L3 specifications contained in previous S2R projects such as ASTRAIL, MOVINGRAIL, .

It is worth recalling here that the formal models were developed based on the functional architecture that we haveintroduced earlier in this deliverable, as well as on the data model, so as to be automatically derived from the SysML SDs and SMD diagrams. Furthermore, in our work we strive to consider the Eulynx approach to the maximum possible extent. We should also mention that the models are built in different languages (i.e., TA and SAN), that are able to capture the heterogeneity of the signalling system under study. All the developed models are parametric and have been developed following a modular and compositional approach to enable libraries of “building blocks” to be reused, instantiated, and connected.

As a result, a significant effort was made to model MB system behaviour w.r.t. the relevant use-cases and functions selected in D2.1, which went beyond expectations in terms of coverage and details, although final and exhaustive formal modelling of the whole MB system was not in the scope of this project.

It is worthwhile to notice that both the activities of T2.3 and T2.4 required a big effort due to the need of “merging” the knowledge elicited from the different sources, and also due to wide range of considered functionalities. To overcome this challenge, a process inspired by the agile software engineering approaches has been implemented. Such a process allowed an effective distribution of modelling tasks among project partners, effectively lowering the essential uncertainty of the ETCS-L3 specifications.

Finally, we should also mention that the work presented in this deliverable fairly fulfils the requirements stated in the Description of Work (DoW) as detailed in Table 9.1 for T2.3 and in Table 9.2 for T2.4.

The work described in this deliverable is also aligned with the research activities achieved in other related projects, such as ASTRAIL (e.g., use of UPPAAL models) and 4SECURAIL (e.g., use of SysML and model-to-model transformations).

Note that the coverage of both SysML model and formal models with respect to the MB elements is not total. The choice of the part to be modelled has been made according to the main features of ETCS-L3 that have been highlighted in [1] and reported in Section 5.5.

We believe that the work achieved is valuable both for the rail industry and academia, since it provides and demonstrates a viable approach towards formalizing the ETCS-L3 specifications and paves the way to automatizing the generation of formal models to evaluate relevant system properties and performances. One important result of such a formalization is to lower the level of uncertainties in knowledge that may have a detrimental effect in the following development and verification stages.

Table 9.1: Demonstration of T2.3 goal fulfilment.

DoW item	Demonstration
Developing a set of semi-formal and formal specifications based on the deliverables produced by previous S2R projects	The SysML modelling approach of MB signalling systems is based on the requirements defined in [50] and improved in [42], on the functional model in [43] and on the semi-formal languages (i.e., UML/SysML) which are strongly recommended in [1].
Developing a set of semi-formal and formal specifications based on modular engineering approach defined in Task 2.1 - Modelling approach and guidelines (T2.1) and the system characterization defined in Task 2.2 - Moving block system and scenarios characterization (T2.2)	Section 2 reports the modelling approach described in [1] on which the concrete SysML specification process in Section 5.3 is based. On the other hand, the OPS produced by the work in T2.2 and reported in [1] are used in this deliverable to select the functional ETCS-L3 elements — i.e., the EUCs and internal functions — to be modelled.
A high-level standard language based on a UML profile (SysML, UML Real Time), will be used to develop the system specifications.	This has been widely demonstrated in this document. Moreover, the choice of an open modelling environment (i.e., Papyrus) goes in the direction of interoperability.
The high-level description language will be properly extended with formal semantics or integrated with formal notations to model safety requirements and allow the verification of functional and non-functional properties.	Although this has not been extensively discussed, SysML allows the usage of UML profiles such as Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [51] and Modeling and Analysis of Real-Time and Embedded Systems - Dependability Analysis and Modeling (MARTE-DAM)[52]: the first is an official Object Management Group (OMG) profile, while the second has been the subject of an important number of scientific works, demonstrating its appropriateness to specify dependability and safety properties, as well as functional requirements.
In developing these activities, the work conducted within the S2Rproject 4SECURAIL will be considered as well.	The work discussed in this deliverable is fully in line with the 4SECURAIL use of SysML and model-to-model transformations.
The specification phase will integrate the Eulynx modelling methodology as for the specification of the standard interfaces as well as the available results from the S2R project ASTRAIL.	The scopes of Eulynx DP and the model here presented are slight different. Namely, while the first focuses on describing “physical assets” of a railway system, the second is more oriented to the functional and behavioural aspects. In fact, they represent two complementary views whose joint is constituted by the Data Model described in Section 6.2. This data model is inspired by Eulynx DP and further research would reach a practical integration between them.
In particular, the effect on safety of the usage of the European Global Navigation Satellite System (Galileo & EGNOS) (EGNSS) (possibly combined with additional positioning systems) will be considered.	The SysML model considers the TLU components, while taking into account the MOVINGRAIL functional architecture. Integrating this component makes it possible to investigate accuracy and performance features related to the considered GNSS technologies. So far, the TLU is integrated in the SysML model. Furthermore, experiments run in T2.5 will take into account also TLU update mechanisms in the formal modelling.
The system formal and semi-formal models will be updated according to the moving block specifications and architectures which will result from the work in S2R X2Rail-3, including virtual coupling scenarios.	The work reported in this deliverable is already updated with the MB specifications reported in [42]. On the other hand, VC has not been explicitly addressed in this deliverable due to the lack of specifications at the date of this document preparation.

Table 9.2: Demonstration of T2.4 goal fulfilment.

DoW item	Demonstration
According to the approach defined in Task 2.1, the specification models built in Task 2.3 are generic, i.e., they are templates, which are parametric in both a subset of their attributes and/or their structural elements. This allows them to be reused and customized (to some extent) to different signalling systems (based on the deliverables from X2RAIL-1) and diverse market segments.	All the models described in this deliverable are parametric. Relevant ETCS-L3 parameters have been identified as early as from the beginning of the WP2 activities and considered in both SysML and formal models as well. By setting specific parameter values, the different market segments can be modelled and investigated. Track configurations can be modelled in the SysML model.
This task is in charge of defining and implementing proper transformations to generate “deployable” concrete models (from configuration data from Task 2.2) and the input required by the target analysis tools (if needed).	All the formal models have been built in two steps. In the first step, a series of preliminary activities prepared the concrete construction of models by clearly stating each model “interfaces”, i.e., its inputs, outputs, parameters, etc. The second activity concretizes such “module” in a model expressed by means of a concrete formalism. The languages actually chosen — i.e., UPPAAL’s TAs and Möbius’ SANs — present an easy and practical way to accept as input parameter values and further customization. This shall enable straight model-to-model transformation from SysML diagrams into such formal models.
In this way, a possible implementation for the EULYNX use case for formal development is also provided.	See Table 9.1 for further details.
A relevant activity of this task is the development of a model-based proof of concept and analysis of virtual coupling train operations.	VC has not been explicitly addressed in this deliverable due to the lack of specifications at the date of this document preparation.

10. Conclusions

MB, which is the central concept for ETCS-L3, constitutes a substantial breakthrough for railway signalling and command-control systems. Railway operation under MB shall induce substantial gains in terms of infrastructure capacity and cost. Nevertheless, this raises important challenges in terms of safety due to the potential shortened headways between successive trains. In addition, ensuring the functional requirements of the ETCS-L3 system is also a crucial issue due to the numerous intervening components. With this in mind, understanding and analysing the dynamics of the ETCS-L3 system is paramount. This deliverable presents the results of the specification and modelling activities conducted on the ETCS-L3 in the framework of the PERFORMINGRAIL WP2 workpackage. Namely, the current document focuses on the work carried out in tasks T2.3 and T2.4. The overall objective of the conducted activities is to elaborate “systematic” processes that allow for establishing formal behavioural models that can serve as basis for various safety and functional features pertaining to ETCS-L3. The adopted approach is based on a two-staged process, organized according to agile principles. Namely, in the first stage, a SysML model is presented, covering various aspects of the system (requirements, structural, functional and behavioural features). Although the coverage with respect to all the EUC is not total, an important contribution of the work achieved lies in the definition of a modelling approach that can be extensible to all the ETCS functions and procedures, while being customizable regarding different parameter and configuration situations and while being integrable with Eulynx DP. In the second stage, formal models are elaborated covering different parts of the system, and describing one or two ETCS-L3’s internal functions, each. Through these models, a more profound knowledge of the ETCS-L3 signalling system is achieved, and some properties are analysed. The performed modelling activities demonstrate the complexity of the ETCS-L3 system in terms of dynamics, in particular due to the extensive interactions and interdependencies between the involved components.

The work discussed in this deliverable is the middle step in the activities of PERFORMINGRAIL’s WP2. In the third deliverable of this WP, the developed models will be composed and parametrized to address the evaluation of various safety and performance features pertaining to OPSs.

Bibliography

[1] Cristina Seceleanu, et al., “PERFORMINGRAIL D2.1 - Modelling guidelines and Moving Block Use Cases characterization,” Tech. Rep., 2021. [Online]. Available: https://www.performingrail.com/

[2] —, “ASTRAIL Deliverable D2.1 - Modelling of the moving block signalling system,” Tech. Rep., 2019. [Online]. Available: http://www.astrail.eu/Page.aspx?CAT=DELIVERABLES&IdPage=24a285dd-3cfa-42ec-b83b-0f5e0c9db6d6

[3] ——, “4SECURail D2.5 Formal development demonstrator prototype - final release,” Tech. Rep., 2021. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=S2R_4SECURAIL

[4] M. Samra, et al., “PERFORMINGRAIL D1.1 - Baseline system specification and definition for Moving Block Systems,” Tech. Rep., 2021. [Online]. Available: https://www.performingrail.com/

[5] W. H. Sanders and J. F. Meyer, “Stochastic activity networks: Formal definitions and concepts,” in *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science Bergen Dal, The Netherlands, July 3–7, 2000 Revised Lectures*. Springer Berlin Heidelberg, 2001, pp. 315–343.

[6] T. Courtney, S. Gaonkar, K. Keefe, E. W. Rozier, and W. H. Sanders, “Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models,” in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 2009, pp. 353–358.

[7] D. Rizopoulos, N. Olsson, A. Lindahl, and O. Lindfeldt, “Research directions regarding the adoption of formal methods in the railway signaling sector: Determinants and next steps for future-proof railways,” *WIT Transactions on the Built Environment*, vol. 199, pp. 75–86, 2020.

[8] Enhancing railway signalling systems based on train satellite positioning, on-board safe train integrity, formal methods approach and standard interfaces, enhancing Traffic Management System functions. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-2

[9] SAtellite-based Signalling and Automation SysTems on Railways along with Formal Method and Moving Block validation (ASTRail). [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=s2r_astrail

[10] FORMAL METHODS AND CSIRT FOR THE RAILWAY SECTOR (4SECURail). [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=s2r_4securail

[11] Completion of activities for Adaptable Communication, Moving Block, Fail safe Train Localisation (including satellite), Zero on site Testing, Formal Methods and Cyber Security. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-5

[12] PERformance-based Formal modelling and Optimal tRaffic Management for movING-block RAILway signalling. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=S2R_PERFORMINGRAIL

[13] —, “X2Rail-2 D5.1 Formal Methods (taxonomy and survey), Proposed methods and

Applications," Tech. Rep., 2018. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-2

[14] D. Basile, M. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, A. Piattino, D. Trentini, and A. Ferrari, "On the industrial uptake of formal methods in the railway domain: A survey with stakeholders," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11023 LNCS, pp. 20–29, 2018.

[15] A. Ferrari, F. Mazzanti, D. Basile, M. H. ter Beek, and A. Fantechi, "Comparing formal tools for system design: a judgment study," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 62–74.

[16] A. Ferrari, F. Mazzanti, D. Basile, and M. Ter Beek, "Systematic evaluation and usability analysis of formal methods tools for railway signaling system design," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.

[17] D. Basile, M. Ter Beek, and V. Ciancia, "Statistical model checking of a moving block railway signalling scenario with UPPAAL SMC: Experience and outlook," vol. 11245 LNCS, pp. 372–391, 2018.

[18] D. Basile, M. ter Beek, A. Ferrari, and A. Legay, "Modelling and analysing ertms l3 moving block railway signalling with simulink and uppaal smc," vol. 11687 LNCS, pp. 1–21, 2019.

[19] ——, "Exploring the ERTMS/ETCS full moving block specification: an experience with formal methods," *International Journal on Software Tools for Technology Transfer*, vol. 24, pp. 351–370, 2022.

[20] —, "4SECURail D2.1 Specification of formal development demostrator," Tech. Rep., 2020. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=S2R_4SECURAIL

[21] ——, "4SECURail D2.2 Formal development demonstrator prototype 1st release," Tech. Rep., 2020. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=S2R_4SECURAIL

[22] D. Basile, M. ter Beek, and A. Legay, "Strategy synthesis for autonomous driving in a moving block railway system with uppaal stratego," vol. 12136 LNCS, pp. 3–21, 2020.

[23] L. Carnevali, F. Flammini, M. Paolieri, and E. Vicario, "Non-markovian performability evaluation of ERTMS/ETCS level 3," in *Computer Performance Engineering*, vol. 9272, 2015, pp. 47–62.

[24] M. Biagi, L. Carnevali, M. Paolieri, and E. Vicario, "Performability evaluation of the ertms/etcs – level 3," *Transportation Research Part C: Emerging Technologies*, vol. 82, pp. 314–336, 2017.

[25] O. Himrane, J. Beugin, and M. Ghazel, "Towards a Model-Based Safety Assessment of Railway Operation Using GNSS Localization," in *ESREL 2020 PSAM 15, 30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference*, Venice, Italy, Nov. 2020.

[26] O. Himrane, J. Beugin, and M. Ghazel, "Toward Formal Safety and Performance Evaluation of GNSS-based Railway Localisation Function," in *CTS 2021, 16th IFAC Symposium on Control in Transportation Systems*, vol. 54, no. 2, Lille (virtual), France, 2021, pp. 159–166.

[27] A. Cunha and N. Macedo, "Validating the hybrid ERTMS/ETCS level 3 concept with electrum," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 3, pp. 281–296, 2020.

[28] P. Arcaini, J. Kofroň, and P. Ježek, "Validation of the hybrid ertms/etcs level 3 using spin," *International Journal on Software Tools for Technology Transfer*, vol. 22, no. 3, pp. 265–279, 2020.

[29] D. Hansen, M. Leuschel, P. Körner, S. Krings, T. Naulin, N. Nayeri, D. Schneider, and F. Skowron, "Validation and real-life demonstration of ETCS hybrid level 3 principles using a formal b model," *International Journal on Software Tools for Technology Transfer*, vol. 22, no. 3, pp. 315–332, 2020.

[30] A. Mammar, M. Frappier, S. Tueno Fotso, and R. Laleau, "A formal refinement-based analysis of the hybrid ERTMS/ETCS level 3 standard," *International Journal on Software Tools for Technology Transfer*, vol. 22, no. 3, pp. 333–347, 2020.

[31] T. Fotso, S. Jeffrey, M. Frappier, R. Laleau, and A. Mammar, "Modeling the hybrid ertms/etcs level 3 standard using a formal requirements engineering approach," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 3, pp. 349–363, 2020.

[32] J.-R. Abrial, "The abz-2018 case study with event-b," *International Journal on Software Tools for Technology Transfer*, vol. 22, 06 2020.

[33] D. Dghaym, M. Poppleton, and C. Snook, "Diagram-led formal modelling using iuml-b for hybrid ertms level 3," vol. 10817 LNCS, pp. 338–352, 2018.

[34] D. Dghaym, M. Dalvandi, M. Poppleton, and C. Snook, "Formalising the hybrid ERTMS level 3 specification in iUML-B and event-b," *International Journal on Software Tools for Technology Transfer*, vol. 22, no. 3, pp. 297–313, 2020.

[35] M. Butler, D. Dghaym, T. Hoang, T. Omitola, C. Snook, A. Fellner, R. Schlick, T. Tarrach, T. Fischer, and P. Tummeltshammer, "Behaviour-driven formal model development of the ETCS hybrid level 3," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, vol. 2019-November, 2019, pp. 97–106.

[36] P. Gaspari, E. Riccobene, and A. Gargantini, "A formal design of the hybrid european rail traffic management system," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, vol. 2, 2019, pp. 156–164.

[37] A. Ait Wakrime, R. Ben Ayed, S. Collart-Dutilleul, Y. Ledru, and A. Idani, "Formalizing railway signaling system ERTMS/ETCS using UML/Event-B," vol. 11163 LNCS, pp. 321–330, 2018.

[38] M. Bartholomeus, B. Luttik, and T. Willemse, "Modelling and analysing ERTMS hybrid level 3 with the mCRL2 toolset," vol. 11119 LNCS, pp. 98–114, 2018.

[39] F. Flammini, S. Marrone, R. Nardone, and V. Vittorini, "Compositional modeling of railway virtual coupling with stochastic activity networks," *Formal Aspects of Computing*, vol. 33, no. 6, pp. 989–1007, 2021.

[40] Z. Yong and Z. Sirui, "Typical train virtual coupling scenario modeling and analysis of train control system based on vehicle-vehicle communication," in *2020 IEEE 6th International Conference on Control Science and Systems Engineering (ICCSSE)*, 2020, pp. 143–148.

[41] —, "X2Rail-3 Deliverable D4.2 - Moving Block Specifications — Part 2 – System Definition," Tech. Rep., 2018. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-3

[42] ——, "X2Rail-3 Deliverable D4.2 - Moving Block Specifications — Part 3 – System Specification," Tech. Rep., 2018. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-3

[43] R. Goverde, et al., "MOVINGRAIL D1.1 - Report on Moving Block Operational and Engineering Rules," Tech. Rep., 2018. [Online]. Available: https://movingrail.eu/

[44] MOVING BLOCK AND VIRTUAL COUPLING NEXT GENERATIONS OF RAIL SIGNALLING (Movingrail). [Online]. Available: https://movingrail.eu/

[45] M. Ghazel, "Formalizing a subset of ERTMS/ETCS specifications for verification purposes," *Transportation Research Part C: Emerging Technologies*, vol. 42, pp. 60–75, 2014.

[46] —, "X2Rail-3 Deliverable D4.2 - Moving Block Specifications — Part 5 – Engineering Rules," Tech. Rep., 2018. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-3

[47] ——, "X2Rail-2 Deliverable D6.1 - System Requirement Specification (SRS) for the Integration Layer," Tech. Rep., 2018. [Online]. Available: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5cc98373f&appId=PPGMS

[48] L. Ferier, S. Lukicheva, and S. Pinte, "Ertms braking curves modeling using efs," in *9th FORMS/FORMAT 2012 - Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems*, 2012, pp. 99–108.

[49] UNISIG, "ERTMS/ETCS: System Requirements Specification - SUBSET-026, issue 3.6.0," 2016.

[50] —, "X2Rail-1 Deliverable D5.1 - Moving Block System Specification," Tech. Rep., 2016. [Online]. Available: https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-1

[51] ——, "Modeling and analysis of real-time and embedded systems ((ptc/08-06-09)," Tech. Rep., 2018. [Online]. Available: https://www.omg.org/omgmarte/

[52] S. Bernardi, J. Merseguer, and D. Petriu, "A dependability profile within marte," *Software and Systems Modeling*, vol. 10, no. 3, pp. 313–336, 2011.

[53] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

A. Used Formal Languages and Notations

This Appendix recalls the main definitions and concepts of timed automata, networks of extended timed automata, and stochastic activity networks, respectively.

A.1. Timed Automata

Timed automata (TA) is a well-known formalism for modelling and verifying safety-critical systems with timing constraints [53]. TA extend finite state automata with *clocks* (i.e., real-valued variables, all of which evolve linearly at the same rate). The behaviour of a real-time system is modelled by finite graphs augmented with a finite set of clocks. The vertices of the graph are called *locations* and represent the possible control modes of the system, whereas the *edges* that connect locations are called (control) switches, and model discrete changes of control modes. Time can only pass in locations, while switches are instantaneous. Clocks can be compared with rational constants to form *clock constraints*. These constraints are expressed as conjunctions of linear inequalities: for a set X of clocks, the set $\Phi(X)$ of clock constraints ϕ is defined by the grammar:

$$\phi := x \leq c | c \geq x | x < c | c < x | \phi_1 \wedge \phi_2$$

where x is a clock in X and c is a rational constant in $\mathbb{Q}$.

Clock constraints can be used to express enabling conditions for switches, called *guards*, and to specify location *invariants*, namely upper bounds on the time that an automaton can spend in a given location, respectively. Formally, a timed automaton A is a tuple $< L, L_0, \Sigma, X, I, E >$, where:

- L is a finite set of locations,
- $L_0 \subseteq L$ is a set of initial locations,
- Σ is a finite set of input symbols,
- X is a finite set of clocks,
- I is the invariant mapping, associating each location l with a clock constraint in $\Phi(X)$,
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is a set of switches. A switch $< l, a, \phi, \lambda, l' >$ represents an edge from location l to location l' that can be traversed on reading the input symbol a. ϕ is a clock constraint over X that specifies when the switch is enabled, and the set $\lambda \subseteq X$ gives the clocks that must be reset, i.e. set to 0, when the switch is executed.

A.2. Networks of Timed Automata Extended with Variables and Broadcast Synchronization

A system of timed automata with Boolean variables, which can be synchronized using broadcast communication channels, is a tuple $(\mathbb{A}, \mathbb{C}, \mathbb{V})$, in which:

- $\mathbb{A} = \bigcup_{i=1}^{n} \{A_i\}$ is the set of communicating automata, where $A_i = (L_i, \ell_0^i, X_i, T_i, Inv_i, G_i, C_i, R_i, \mathbb{T}_i, \mathbb{F}_i, Syn_i, \Sigma_i)$,
- $\mathbb{C}$ is the set of communication channels,
- $\mathbb{V}$ is a set of Boolean variables.

In this context, an automaton is a tuple $A = (L, \ell_0, X, E, Inv, G, C, R, \mathbb{T}, \mathbb{F}, Sync, \Sigma)$, in which:

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the initial location,
- X is a finite set of clocks,
- $E \subseteq L \times \Sigma \times L$ is the set of edges, $(\ell, a, \ell') \in E$ is written $\ell \xrightarrow{a} \ell'$,
- $Inv : L \to 2^{X \times \{<, \leqslant\} \times \mathbb{R}^+}$ maps a possibly empty *invariant* to each location,
- $G : E \to 2^{X \times \{<, \leqslant, =, \geqslant, >\} \times \mathbb{R}^+}$ maps a possibly empty *clock guard* to each edge,
- $C : E \to 2^{\mathbb{V}}$ maps a possibly empty *Boolean condition* to each edge,
- $R : E \to 2^X$ maps a possibly empty set of clocks to be *reset*, to each edge,
- $\mathbb{T} : E \to 2^{\mathbb{V}}$ maps a possibly empty set of variables that must be set to true, to each edge,
- $\mathbb{F} : E \to 2^{\mathbb{V}}$ maps a possibly empty set of variables that must be set to false, to each edge, with $\mathbb{T} \cap \mathbb{F} = \emptyset$. Intersection is used here since mappings ($\mathbb{T}$ and $\mathbb{F}$ here) are also relations, thereby sets,
- $Syn : E \to (\mathbb{C} \times \{!, ?\}) \cup \{\emptyset\}$ maps a possibly empty synchronisation label to each edge,
- Σ is a finite alphabet of actions.

A clock guard assigned to a clock t is a triple (c, o, v), where c is a clock variable, $o \in \{<, \leq, =, \geq, >\}$ is a comparison operator and v a non-negative real value. For example, the expression "$c \leqslant 0$", stating that the clock c must be lower than or equal to zero, is formally written as the triple $(c, \leqslant, 0)$. This works in the same way for invariants, except that guards are mapped to locations, and use a restricted set of comparison operators (only $<$ and $\leq$).

Mappings C, R, $\mathbb{T}$ and $\mathbb{F}$ all work in the same way: they map to an edge a set of Boolean variables or clocks. For C, the variables associated with an edge e must all be true for e to be enabled. As for R, when e is traversed, every clock in R is reset. Finally, $\mathbb{T}$ and $\mathbb{F}$ specify what Boolean variable must be set to true, and which one to false, respectively.

Synchronization labels are of two sorts. Those ending in *"!"* specify a sending (or *master*) edge that initiates the communication. As it is a broadcast communication, it does not require any listener to be ready to listen, which means that a master edge can always be traversed provided that the other enabling conditions are fulfilled.

An edge holding a synchronization label ending in *"?"* is a receiving (or *slave*) edge. A slave edge is blocked until a master edge sending on the same channel is enabled.

When a master and some slave edges are enabled, they are all traversed synchronously, in a single step (atomically). No enabled slave edge can be left out of the synchronous step if a master edge on the same channel is enabled.

A.3. Stochastic Activity Networks

Stochastic Activity Networks (SANs) are a stochastic extension of , introduced to analyse concurrency, timeliness, fault tolerance, and degradable performance of complex computing systems [5]. A SAN model comprises four primitives that define its structural components, places, activities, input gates, and output gates, whose meaning and roles have been already introduced in Section 2.3.2. Here, a deeper discussion focused on the temporal specification of SANs and how they deal with uncertainty is provided. In addition, the tool and the graphic elements used to model the Moving Block system are introduced by a simple example.

The goal of this sub-section is to provide the necessary basis for understanding the model presented in Section 8.6 to readers not familiar with SANs.

The temporal specification of SANs is stochastic, being defined by associating a time distribution function with each timed activity, and a probability distribution with each set of cases associated with an activity. Cases can be associated with both instantaneous and timed activities. *Case probabilities* associated with instantaneous activities model a non-deterministic choice among alternative activities enabled in a certain state, *Case probabilities* associated with timed activities model the uncertainty about the next state assumed upon completion of the activity. The case probabilities associated to an activity can be marking dependant, and their sum must be equal to one.

A SAN model can only be analysed by simulation.

Figure A.1 shows the structural elements of a SAN. Ordinary places are drawn as blue circles, extended places are orange circles, timed activities are drawn as thick bars, instantaneous activities are represented by thin bars, input gates are depicted as red triangles, whereas output gates are black triangles. Cases are denoted by small circles on one side of the associated activity.

Extended places (e.g., Extended_Place1 in Figure A.1) differ from standard places because they are associated to a variable, i.e., they contain data. Variables can be atomic, data structures or arrays of primitive data types (i.e., short, int, long, float, double, bool and char). Extended places cannot be connected directly to an activity, but only to its input and output gates.

The firing of timed activities is associated with general distributed random variables (e.g., Exponential, Normal, Binomial, etc.) whose parameters can be numeric constant, or marking dependant.

The probability associated with each case (e.g., the cases associated with Instantaneous_Activity1) can be specified as a numerical constant or a function. If no cases are explicitly present, the default is assumed, with a probability equal to one.

Input and output gates can be used to control the enabling condition of an activity, and to change the state of the system when the activity fires. An activity is enabled when the predicates of all input gates connected to the activity are evaluated to true, and each ordinary place connected to the incoming arcs contains at least one token. When an activity fires, the input and the output functions of the input and output gates (respectively) are executed, while tokens of connected ordinary places are updated as in the Petri Net firings. Enabling predicates and functions for gates are typically specified in tabular form.

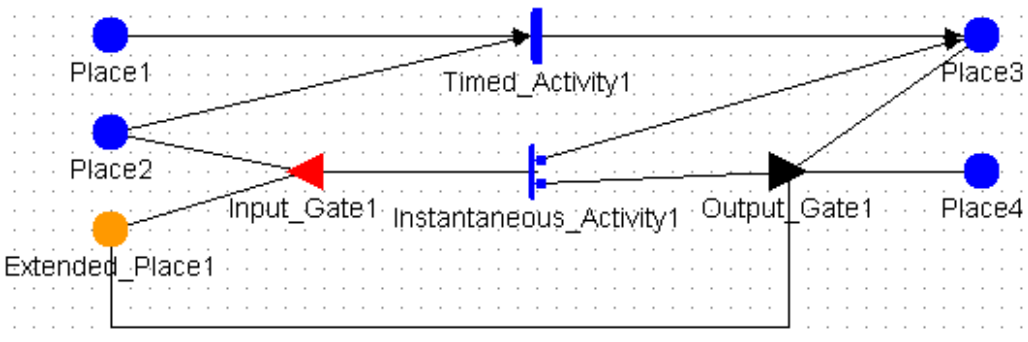


Fig. A.1. A SAN model example

In Figure A.1, the timed activity *Timed_Activity1* is enabled by tokens in places *Place1* and *Place2*. When the activity fires, a new token is added in *Place3*. At the bottom of the figure, the instantaneous activity *Instantaneous_Activity1* is enabled by the predicate of *Input_Gate1*, which, in turn, is evaluated with respect to the marking of *Place2* and *Extended_Place1*. When the activity fires, two cases are possible. If selected, the first case adds a token to *Place3*; alternatively, the second case enables the execution of the output gate *Output_Gate1*, which, in turn, updates the marking of *Place4* and *Extended_Place1*, according to the output function associated with the activity.

SANs are well-supported by Möbius [6], a well-known tool used to edit and analyse SAN models, which also supports the compositional and hierarchical development of models, making SANs well suited to compositional modelling. In Möbius, all enabling predicates, input and output functions, parameters, types, and variables are expressed by C++ statements, thus allowing the introduction of actual code in the model definition.

Composed models are obtained through two compositional operators: *Join* and *Rep*. *Join* composes two or more sub-models (called atomic submodels) by place or activities superposition. *Rep* automatically creates identical copies (replicas) of an atomic sub-model. State variables can be local to an atomic sub-model, or they can be shared among all sub-models. The *Join* operator also supports the sharing of state variables among different subsets of the composed sub-models.