X2Rail-5

Project Title:	Completion of activities for Adaptable Communication, Moving Block, Fail Safe Train Localisation (including satellite), Zero on site Testing, Formal Methods and Cyber Security
Starting date:	01/12/2020
Duration in months:	30
Call (part) identifier:	S2R-CFM-IP2-01-2020
Grant agreement no:	101014520

Deliverable D10.2

Proposed extension of specification approach to meet needs of RCA

Due date of deliverable	Nov 30, 2021
Actual submission date	Jan 13, 2022
Organization name of lead contractor for this deliverable	DB
Dissemination level	PU
Revision	1.0

Deliverable template version: 01 (21/04/2020)

This project has received funding from Shift2Rail Joint Undertaking (JU) under grant agreement 101014520.The JU receives support from the European Union's Horizon 2020 research and innovation programme and the Shift2Rail JU members other than the Union.

Authors & Version Management

Author(s)	**Deutsche Bahn (DB)** Randolf Berglehner Abdul Rasheeq Felix Auris
	Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) Daniel Schwencke
Contributor(s)	**Trafikverket (TRV)** Arne Borälv
Reviewer(s)	**Thales (THA)** Dominik Hansen
	AZD Adam Rychtář

Version Management

Version Number	Modification Date	Description / Modification
0.1	June 30, 2021	First draft.
0.11	Sept 07, 2021	First revision (Meeting Sept 07, 2021)
0.2	Sept 23, 2021	Inputs from DLR, TRV and DB
0.3	Oct 15, 2021	Inputs from meeting Oct 07, 2021
0.4	Nov 24, 2021	Addition of content in chapter 6 and chapter 7
0.41	Nov 26, 2021	Addition of content in chapter 6 and chapter 7
0.42	Dec 07, 2021	Addition of content in chapter 7
0.5	Dec 09, 2021	Addition of chapters 7.1 and 7.2
0.6	Dec 15, 2021	Addition of content in chapters 7.3.1 and 7.3.2
0.7	Dec 16, 2021	Addition of chapters 7.4 and 7.5
0.8	Jan 10, 2022	Inputs from DB, TRV and DLR. References.
0.9	Jan 12, 2022	Minor edits and finalised issue of the deliverable
1.0	Jan 13, 2022	Final revision

1 Executive Summary

This document is deliverable D10.2, describing extensions of the MBSE specification approach to needs of future Functional Railway System Architectures within Task 10.3 of work package *WP10 Formal Methods for Functional Railway System Architecture*, within the X2Rail-5 project.

This deliverable is concerned with a specification approach meeting the needs in ongoing and future developments of ERTMS, and the European initiatives RCA and EULYNX. This is a rather large scope, whose general high-level goal may be formulated as:

Determine a suitable approach to specify, verify, and validate system requirements, that can meet the needs of initiatives and projects RCA and EULYNX that define a future system architecture.

Different specification approaches are currently applied in RCA and EULYNX. RCA (as well as LinX4Rail 5.2) use the ARCADIA method. EULYNX uses an MBSE approach, based on the systems modelling language (SysML) as defined in the EULYNX Modelling Standard. These two approaches complement each other but are not yet “connected” in a stable way. As the function range of system elements in RCA is larger than the one in EULYNX, and there are additional characteristics or kinds of behaviours of functions to be considered, RCA has requested to extend or modify the current EULYNX approach correspondingly. Furthermore, LinX4Rail 5.2 seeks to represent an ARCADIA-based architectural model in SysML, as it is used in EULYNX. The future intention is to merge the two complementary approaches into one closely intertwined approach: The analysis and definition of a modular, standard system architecture shall be carried out following the ARCADIA method, and the corresponding system elements are further specified according to an adjusted EULYNX specification approach.

This deliverable describes proposed extensions of the semi-formal EULYNX MBSE approach to specify different types of behaviour to meet the needs of the RCA initiative, to be used as input for the current EULYNX/RCA modelling standard and other tasks of X2Rail-5 WP10. This deliverable also describes different types of behaviours that must be supported in the specification of system elements and requirements in the RCA approach and their interpretation as mandatory requirements considering the need for executable (testable) specifications that enable formal verification of safety requirements.

2 Table of Contents

3 Abbreviations and acronyms

Abbreviation / Acronyms	Description
AL	Abstraction Levels
AM MBSE	Architecture Model Model-Based Systems Engineering
ARCADIA	ARChitecture Analysis and Design Integrated Approach
ASAL	Atego Structured Action Language
ATO	Automatic Train Operation
BDD	Block Definition Diagram
CBTC	Communications-Based Train Control
CCS	Command Control and Signalling
CENELEC	European Committee for Electrotechnical Standardization
CSP	Crosscutting System Properties
DEDS	Discrete Event Dynamic Systems
EN	European Norm
EPBS	End Product Breakdown Structure
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
FA	Functional Architecture
FE	Functional Entity
FMs	Formal Methods
GUI	Graphical User Interface
IBD	Internal Block Diagram
IM	Infrastructure Manager
ISA	Independent Safety Assessment
LX	Level Crossing
MBSE	Model-Based Systems Engineering
MBSE SF	Model-Based Systems Engineering Specification Framework
OCR	Object Control Requests
PDI	Process Data Interface
PDU	Process Data Unit
RCA	Reference CCS Architecture
ReqIF	Requirements Interchange Format
ROI	Return On Investment
SUC	Sub Use Case
SUS	System Under Specification
SysML	Systems Modelling Language
TCG	Test Case Generation
TD	Technical Demonstrators
TDS	Train Detection System
TFE	Technical Functional Entity
V&V	Validation and Verification
WP10	Work Package *Formal Methods and Standardisation for Smart Signalling Systems*, X2Rail-5

4 Introduction

This document is deliverable D10.2, describing extensions of the EULYNX-RCA MBSE specification approach to needs of future Functional Railway System Architectures within Task 10.3 of work package WP10 Formal Methods for Functional Railway System Architecture, within the X2Rail-5 project [1].

4.1 About this Deliverable

In general, this deliverable is concerned with a specification approach meeting the needs in ongoing and future developments of ERTMS, and the European initiatives RCA and EULYNX.

This is a rather large scope, whose general goal may be formulated as:

- Evaluate the existing specification approaches in EULYNX and RCA.
- Bridge the gap between the specification approaches in EULYNX and RCA.
- Determine a suitable approach to specify, verify, and validate system requirements in a railway system architecture.

This deliverable describes proposed extensions of the semi-formal EULYNX MBSE approach to specifying different types of behaviour to meet the needs of the RCA initiative (see Section 5.1.2), to be used as input for the current EULYNX/RCA modelling standard and other tasks of X2Rail-5 WP10. A future specification approach must consider the impact of RCA in terms of increased range of specification scope, increased function range and additional characteristics of functions that are needed.

This deliverable was created by X2Rail-5 WP10 Task 10.3 (*Extending the MBSE specification approach to needs of future Functional Railway System Architectures*), based on an analysis of different kinds of behaviours that must be supported in the specification of system elements and requirements, how they shall be interpreted as mandatory requirements, and considering the need for executable (testable) specifications that support formal verification of safety requirements.

4.2 Motivation and Purpose

Different specification approaches are currently applied in RCA and EULYNX. Both RCA and LinX4Rail 5.2 [25] use the ARCADIA method as a specification approach. EULYNX uses the EULYNX MBSE approach, as defined in the EULYNX Modelling Standard [6]. Both approaches complement each other but are not yet “connected” in a stable way. For the future, it is intended to merge them into one closely intertwined specification approach:

- The analysis and definition of a modular, standard system architecture shall be carried out following the ARCADIA method, and
- The corresponding system elements are further specified according to an adjusted EULYNX specification approach.

As the function range of the RCA system elements is larger than the one in EULYNX and there are additional characteristics or kinds of behaviours of functions to be considered, it has been requested by the RCA group to supplement or modify the current EULYNX approach correspondingly. Furthermore, in LinX4Rail 5.2 it is increasingly demanded to be able to represent the nascent ARCADIA-based architecture model or artefacts of it based on the systems modelling language (SysML) as used in the EULYNX specification approach.

4.3 Brief Work Plan Overview

This deliverable was created based on the results achieved in X2Rail-5 WP10 Task 10.3. The aim of Task 10.3 was to check whether the currently used EULYNX specification approach meets the requirements for the specification of system elements of a future railway system architecture, such as RCA. In the negative case, necessary additions to the EULYNX modelling approach should be defined. The current specification approaches of the initiatives EULYNX and RCA were used as input for the work. Due to the currently still low progress in RCA, work has had to proceed based on assumptions. It is therefore necessary that results in this deliverable are reviewed again after further progress in RCA and adjusted if needed.

The work was carried out in the following steps:

- Identify kinds of behaviours that must be supported in functional specifications of system elements and interfaces (considering needs in RCA and EULYNX and requirements of Task 10.2).
- Evaluate whether the current EULYNX model-based systems engineering (MBSE) approach fully enables the description of the identified kinds of behaviour.
- Recommend suitable supplements or proposals of modifications to describe kinds of behaviour not yet considered in the current EULYNX MBSE approach. Care should be taken to ensure that the recommendations are adaptable for the automated transformation of specification models into formal models and formal verification of these models
- Define how the described kinds of behaviour shall be expressed as mandatory requirements.
- Evaluate if test case generation (TCG) can be applied to other kinds of CCS system behaviours than those considered in X2Rail-2, how such behaviours can be modelled for the purpose of TCG, and how an MBSE approach can support TCG best.

4.4 Document Overview

The remainder of this document is organized as follows

- Chapter 5 describes the background to this deliverable.
- Chapter 6 describes concepts and principles for the RCA and EULYNX MBSE specification approaches.
- Chapter 7 describes necessary extensions of the EULYNX MBSE approach.
- Chapter 8 describes conclusions.
- Chapter 9 gives the references.

5 Background

This chapter describes the background to this deliverable, its scope and context in relation to ongoing initiatives that define Future Functional Railway System Architectures such as EULYNX and RCA and the purpose and goals of specification approaches in general.

5.1 Scope and Context

The general scope and context for this deliverable D10.2 relate to a specification methodology to define *Future Functional Railway System Architectures*, which in turn relates to ongoing and future developments of ERTMS, and the European initiatives RCA and EULYNX.

The use of formal methods (FMs) in relation to a specification approach for future systems architectures (see Section 5.1.4), and the determination of how the specified requirements of system elements shall be interpreted as mandatory requirements play an important role.

A brief background to this scope and context is described in the remainder of this section 5.1.

5.1.1 ERTMS

The ERTMS 'game changers' that have been identified to have a significant impact on the ERTMS business case are an important part of future architectures. The ERTMS 'game changers' involve both trackside and on-board CCS systems.

Although ERTMS is mature and ready for large scale deployment, some argue [8] it is not possible to reap its full benefits (e.g., lower costs and improved capacity) before legacy systems for Command and Control have been phased out, and before potential 'game changers' identified to have a significant impact on the ERTMS business case [9] have been completed. The game changers include ATO, ERTMS Level 3 with moving block, and future systems for communication and positioning; these technologies are being specified, developed, and demonstrated within Shift2Rail, in different Technical Demonstrators (TDs) work streams.

5.1.2 RCA

There is growing demand from IMs that future architectures shall be standardized and modularized, to ensure interoperability, to increase competition and to enable more reuse. The recent initiative RCA is an example of this demand; RCA aims to create a modular, standard system architecture for the core functionality and interfaces of the trackside safety part of the CCS system, considering the CCS system elements outside the trackside safety part (including onboard systems) as far as relevant, to define interfaces with these system elements.

These quotes from the RCA white paper [8] summarize high-level goals of RCA:

- *Now that ERTMS is mature enough and ready for large scale deployment … it is the right moment to try to define a common, simple reference CCS architecture to support the step from installed base to ERTMS and to increase the capacity of the existing network, improve the deployment speed and reduce life cycle costs for CCS. The Reference CCS Architecture (RCA), developed using*

formalised methods, is the enabler for clear and unambiguous interface definitions. It is aimed to provide generic safety approvals (plug & play), a modular split of work, independent development of components (allowing for technical evolution), an important quality step in the specification of operators' needs towards the supply industry and the strengthening of this supply industry.

- *Cost drivers for CCS are data collection and validation, procurement, design, installation and commissioning, operation, maintenance, and change management. RCA is expected to reduce costs within all these areas.*

The modular, standard system architecture aimed in RCA is currently created using a specification approach based on the ARCADIA method [7].

5.1.3 EULYNX

EULYNX is a European digitalization initiative that aims to define and standardise interlocking interfaces of the future command control and signalling (CCS) system and specify the corresponding system elements. The goal of this is to achieve significantly reduced lifecycle costs for future digital CCS systems by standardising interfaces in the future. According to EULYNX, the definition of standardised interfaces is carried out at the infrastructure manager (IM) side, creating interface specifications that are used in tenders. These include a detailed description of system behaviour visible at interfaces. This entails challenges to ensure correctness and security levels, and to perform safety evaluation of specified behaviour. To meet these challenges, EULYNX uses new approaches with model-based systems engineering (MBSE), to ensure that the created specifications are correct, complete, and consistent. The objective of the EULYNX MBSE approach, defined in the EULYNX Modelling Standard [6], is also to establish a seamless development approach that facilitates reuse, automation, and innovation. To this end, the exchange of specified system elements and interface behaviour as executable models are foreseen, rather than (only) using natural language requirements.

EULYNX was started in 2014, and after seven years, the project has evolved into an organisation for the standardisation of interfaces that have published updated documentation known as the baselines that are regularly published on the EULYNX website [6].

5.1.4 Importance of Formal Methods

TD2.7 recommends [2] using Formal Methods (FMs) for railway control to achieve a sufficient level of trust that critical system properties are satisfied, to raise the quality and verifiability of implementations, and to achieve interoperability goals, because:

- Compared to traditional V&V methods, FMs enable a significantly higher level of trust, and furthermore, FMs can automate tedious V&V tasks.
- Applying FMs provides valuable feedback, insight and helps to detect and correct mistakes.

The railway control domain is very well suited for FMs:

- There are high RAMS demands to be met.
- It is based on well-understood concepts and principles that lend themselves well for FMs.

- Many FMs success stories exist, for instance for interlocking and CBTC.

While railway control has always been complex, future railway architectures are expected to increase software complexity. For this reason, FMs should be considered a must-have technology. Also, the demand to establish a new, modular architecture provides a good (rare?) opportunity to introduce FMs on a larger scale, since the ROI is estimated to be larger:

- FMs for a new, modular architecture can enable cost-efficient reuse of requirements and tools, to off-set initial investments required to introduce FMs into standard processes.
- FMs can help to demonstrate that a novel architecture is as safe as old systems.
- FMs can set the benchmark for future system quality and development cycles.

5.2 Goals for a Future Specification Approach

5.2.1 Purpose and Intended Use

The purpose and intended use of a specification can vary, and any one specification approach may not be able to provide optimal support for all purposes. The following are examples of different purposes/uses of a specification, which may impact what the optimal approach is:

- Define the architecture of a system of interest based on description concepts for abstraction and structure at different levels of granularity facilitating reuse and automation.
- Identify the functions of a system of interest and decompose and allocate them to corresponding system elements.
- Create specifications formalised by a comprehensive modelling theory that provides appropriate models and description techniques in the form of a modelling framework. It is used for modelling the different aspects and artefacts of a system element.
- Create executable specifications and apply V&V to ensure they are coherent, consistent, applicable to existing standards, contain all the features and desired functions and satisfy the intended requirements, for instance verified by:
    - Functional testing based on test suites (manually created or generated).
    - Formal verification of consistency properties.
    - Formal verification of safety properties.
- Provide (release) the specifications as “official (mandatory) system specifications”.
- Use specifications as a basis for tenders.
- Use specifications as a basis for implementation (of software, systems).
- Use specifications as a basis for change request management (change, V&V impact, etc.).
- Use specifications to capture and conserve system knowledge.

5.2.2 Use Case related to Creation of Specification for Tender

In general, this deliverable is concerned with a specification approach to producing good quality specifications for railway CCS system tenders. Figure 1 illustrates the use case for a specification approach that covers the purposes and uses (see Section 5.2.1) related to the creation of system

requirements in tender specifications. This use case is here refined into seven sub use cases (SUCs), as listed below. The numbers depicted in the figure are referenced in the descriptions of the SUCs.

- **SUC1: Create system definition:** the specification process starts with the system definition based on requirements derived from stakeholder needs (1) and regulation-based safety properties (2).
- **SUC2: Derive hazard-based safety properties and adjust system definition:** with the system definition as basis the risk analysis is carried out (3). An analysis of the different types of possible hazards is made and hazard-based safety properties are derived (4). They supplement the pre-existing regulation-based safety properties and are used to adjust the system definition if necessary (3).
- **SUC3: Create tender system requirements specifications:** tender requirements of system elements of modular architectures like RCA and standard interfaces between system elements considering all kinds of behaviours that matter in those architectures are specified. A tender system requirements specification is represented by an executable specification model. Based on results of the system definition phase, including relevant results of the risk analysis, an executable model of the externally observable behaviour is created (5) representing the system requirements.
- **SUC4: Validate system requirements:** the executable specification model is used for validation (6) of system requirements by simulation (virtual prototype).
- **SUC5: Verify safety properties applying formal methods:** safety properties are verified using formal verification and formal proof (7).
- **SUC6: Generate test cases from system requirements:** automated test case generation from the specification model as input for test specifications (8).
- **SUC7: Prove behavioural refinement:** the system requirements specification is given as part of the tenders to the suppliers (9) which respond with the proof that the behaviour of their implemented system is a refinement of the specified one (10).

This deliverable focuses on the sub use cases SUC3, SUC4, SUC5, and SUC6. Though the sub use cases SUC1, SUC2 and SUC7 are also important for a future specification approach, they are not in the scope of this deliverable.

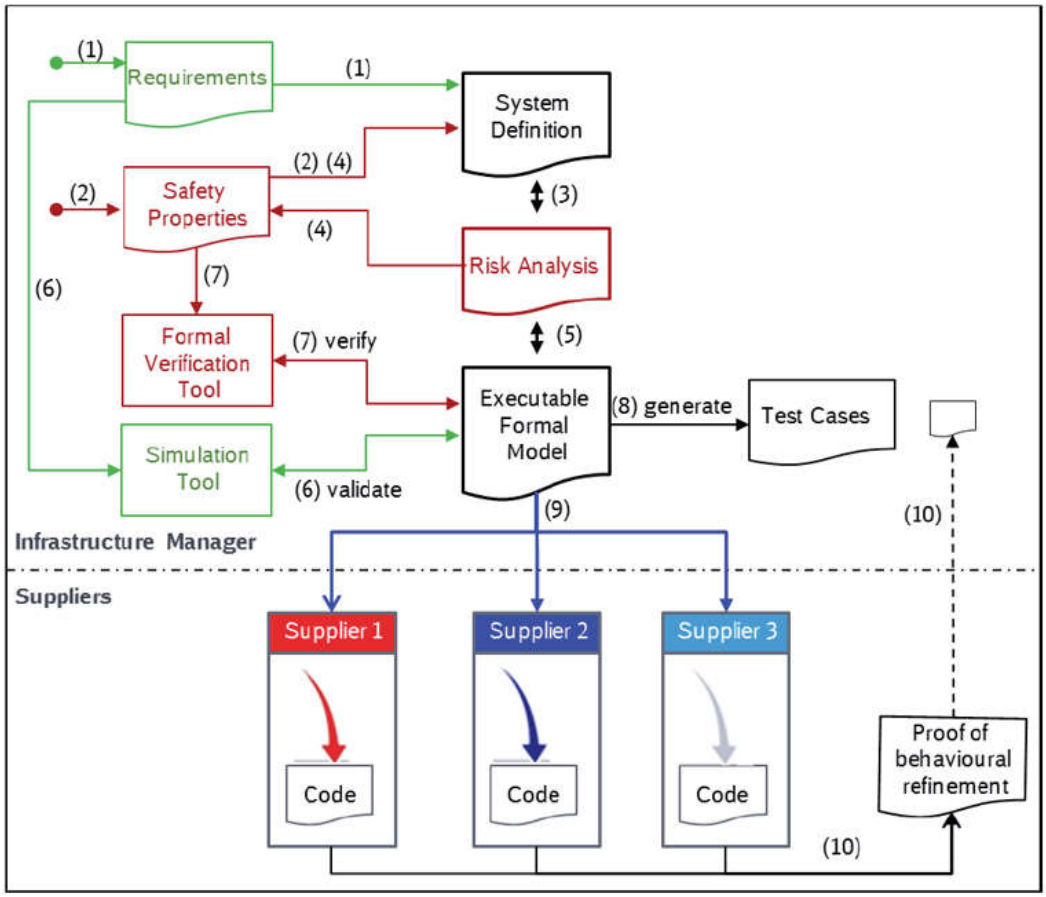

Figure 1 Use case for Development of Systems with Standardised interfaces

5.2.3 Goals related to the Creation of Specifications Used in Tenders

A general goal of a future specification approach is to ensure a good quality of tender requirements. TD2.7 has earlier identified general “requirements on requirements” for tender specifications [3], which are cited below.

R1. Tender requirements should uniquely define, with clear syntax and semantics, all interfaces.

R2. Tender requirements should identify the safety requirements.

R3. Tender requirements should have the following characteristics:

- Correct.
- Necessary.
- Understandable.
- Unambiguous.
- Verifiable (to distinguish a system meeting the requirement from one that does not).
- Clear (concise, terse, simple, precise).

- Feasible (realistic, possible).
- Independent (self-contained).
- Atomic (not expressing several different requirements as one requirement).

Even with tender requirements of good quality, one can expect suppliers to have questions on the tender requirements (related to the interpretation, etc.). For that reason, the owner of tender requirements should provide guidelines on how to interpret their requirements.

Furthermore, the future specification approach may consist of different paradigms, languages or processes that need to be integrated. For example, the RCA specification approach aims for the analysis and definition of modular, standard system architecture and the EULYNX approach focuses on the creation of tender system requirements specifications of the corresponding system elements. Thus, the approach should be supported by a seamless process and toolchain, minimizing the number of different paradigms used to the extent reasonable. It should provide clear decision criteria for which paradigm to use for which portion of the behaviour, avoid confusion of terms across the different kinds of models, and be clear on the integration of different models.

Goals specifically focused on in this deliverable are based on the above-mentioned sub use cases SUC3, SUC4, SUC5, and SUC6. They are defined in the context of those sub use cases below.

5.2.3.1 SUC3: Create system requirements specifications

- **Goal 1:** Enable the creation of system requirements specifications of configurable system elements and standard interfaces between them in the context of a modular standard system architecture.
- **Goal 2:** Enable the definition of all kinds of external observable behaviours of system elements and the relevant layers of interface protocols that matter in modular standard system architectures.
- **Goal 3:** Support the creation of tender system requirements specifications that are understandable to several stakeholders (IMs, suppliers, ISAs, …) on one side and suitable to enable simulation, the application of formal verification and automatic test case generation on the other side.
- **Goal 4:** Enable to express mandatory tender requirements. There is sometimes a need to separate requirements that are mandatory from requirements that are not mandatory. A goal is to ensure that it is clear what is mandatory and what is not.
- **Goal 5:** Enable the definition and use of static configuration data, i.e., define the terminology for writing configuration data for specific system elements (as most system elements are configurable).
- **Goal 6:** Enable the definition of (correctness) requirements for configuration data (data ranges, data consistency, interdependencies).
- **Goal 7:** Enable the definition of safety requirements for specific system elements, i.e. define a process or terminology for writing safety requirements.

- **Goal 8:** Support the definition of concepts and domain knowledge (e.g., based on the model developed in EULYNX Data preparation cluster or Linx4Rail WP3 Conceptual Data Model or RCA domain knowledge) and associate it to tender system requirements.

5.2.3.2 SUC4: Validate system requirements

- **Goal 9:** Enable simulation-based validation (testability) of tender system requirements specifications.
- **Goal 10:** Enable semiautomated/automated static review.

5.2.3.3 SUC5: Verify safety properties applying formal methods

- **Goal 11:** Enable formal verification (formal proof) of properties of the behaviours defined in system requirements specifications.

5.2.3.4 SUC6: Generate test cases from system requirements

- **Goal 12:** Enable semiautomatic/automatic test case generation from the behaviours defined in system requirements specifications.

5.2.3.5 Tradeoffs

Tradeoffs must be considered, as different goals can be conflicting. For instance:

- An executable specification may require a higher degree of detail, whereas a formally verifiable specification could be specified with less detail; on the other hand, the former may be considered less complex to create compared to the latter.
- Style, such as imperative versus declarative specification (see Section 6.2.4).
- Different behaviours may be optimally described using different specification approaches, while a general objective is to use only one (or as few as possible) approaches.

One must consider these tradeoffs, for instance by rating the relative importance placed on various objectives and purposes, to conclude on the modelling paradigms and specification approaches.

5.3 RCA and EULYNX

RCA and EULYNX currently applying different specification approaches: RCA uses the ARCADIA method (Section 6.3.1) and EULYNX uses the EULYNX MBSE approach defined in the EULYNX Modelling Standard. These two approaches may complement each other. However, they have different goals:

- The RCA specification approach aims to define a modular, standard system architecture.
- The EULYNX approach aims to create tender specifications of system elements.

In the future, it is intended to merge these into one closely intertwined specification approach:

- Analysis and definition of a modular, standard system architecture shall be carried out following the ARCADIA method.

- The corresponding system elements shall be further specified according to an adjusted EULYNX specification approach.

The intent is that certain system elements in RCA shall be further specified using the specification approach in EULYNX. As the functional range of RCA system elements is larger than the corresponding range in EULYNX, additional characteristics and kinds of behaviours must be considered. RCA has requested to supplement or modify the current EULYNX approach correspondingly and furthermore, LinX4Rail 5.2 [25] seeks to represent the nascent ARCADIA-based architecture model (or artefacts of it) based on the systems modelling language (SysML) as used in the EULYNX specification approach. Thus, an extended or modified specification approach is needed.

This deliverable is therefore focused on the creation of an extended or modified EULYNX specification approach based on the use case *Semi-formal development of systems with standardised interfaces*, described further in Chapter 6 and Chapter 7.

6 Specification Approaches

This chapter describes how a specification approach can be characterized in terms of relevant properties, methods, paradigms, and principles (see Sections 6.1 and 6.2). For the existing specification approaches considered relevant for this deliverable, RCA and EULYNX, their specification technologies are described, including their intended interaction (see Section 6.3).

6.1 The Concept of Specification Technique

A specification approach for system development can be classified according to three aspects (see the boxes at the bottom of Figure 2):

- Description means: for formulating facts, independent of the problem and its solution.
- Method: to be understood independently of the other two aspects in a process model sense.
- Tool

A description means has concrete characteristics, such as language, notation, and syntax and semantics. The syntax comprises structural properties, which can be determined without reference to the interpretation of the notation. The semantics describes the interpretation independent of application. The tool is (generally) a software program that enables computer-aided use of a description means and analyses of specification models (such as execution-based testing, or analysis of properties of a specification model).

Concrete restrictions to a field of application (for example, command control and signalling systems) and the associated definition of specific tasks can use a description means and a method in a mutually supportive way. A resulting combination of method and description means is referred to as specification technique. The use of a specific tool together with a (specific) specification technique is referred to as specification technology (see Figure 2).

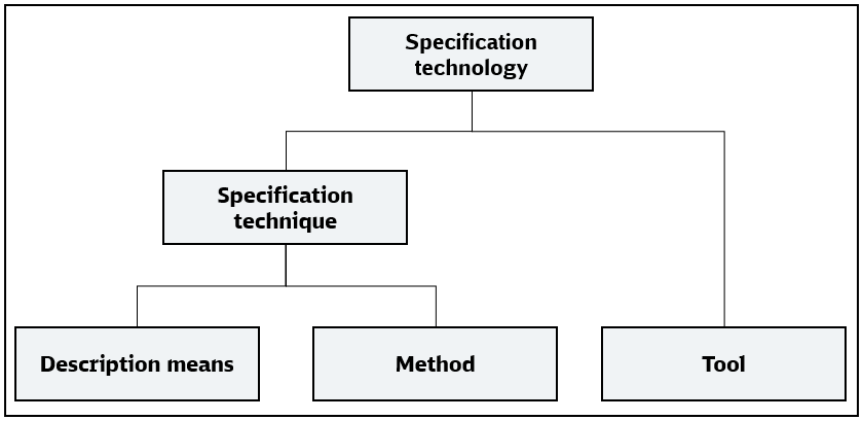

Figure 2 Taxonomy of specification techniques

6.1.1 Informal specification technique

Informal specification techniques use natural language and/or diagrams. The advantages of an informal description are its comprehensibility by a wide range of stakeholders and the comparatively low effort required to create it. However, they leave a lot of room for interpretation due to ambiguity and cannot be processed automatically.

6.1.2 Semi-formal Specification Technique

A semi-formal specification technique uses a semiformal description means. A semi-formal language has formally specified syntax but the task of completing the semantics has been deferred to a later stage, which is then done either by human interpretation or by interpretation through software like code or test case generators. The Systems Modelling Language (SysML) [10] is an example of a semi-formal language. The advantages of using SysML may include that the language is an international standard, that many tool options exist, with many intuitive graphical modelling styles (e.g., use case diagrams, and state machine-based definition of requirements).

6.1.3 Formal Specification Technique

A formal specification technique uses a formal description means, based on a formal language with a formally specified syntax and semantics. The B-method and Event-B languages are examples of formal languages: the semantics of these languages is defined in [23][24]. The advantages of using the B method may include that it provides a refinement-based approach to formal development with high precision and integrated capabilities for formal verification; however, expertise is required for understanding and use of a formal language.

6.1.4 Combination of Semiformal and Formal Specification Techniques

To use an understandable and widely used description means such as the standardised semiformal language SysML on the one hand and to apply formal methods, on the other hand, semiformal and formal specification techniques can be combined. A corresponding approach was evaluated in TD2.7 [5], which aimed to enable formal verification of requirements against semi-formal SysML specification models. This is visualized in principle in the process depicted in Figure 3.

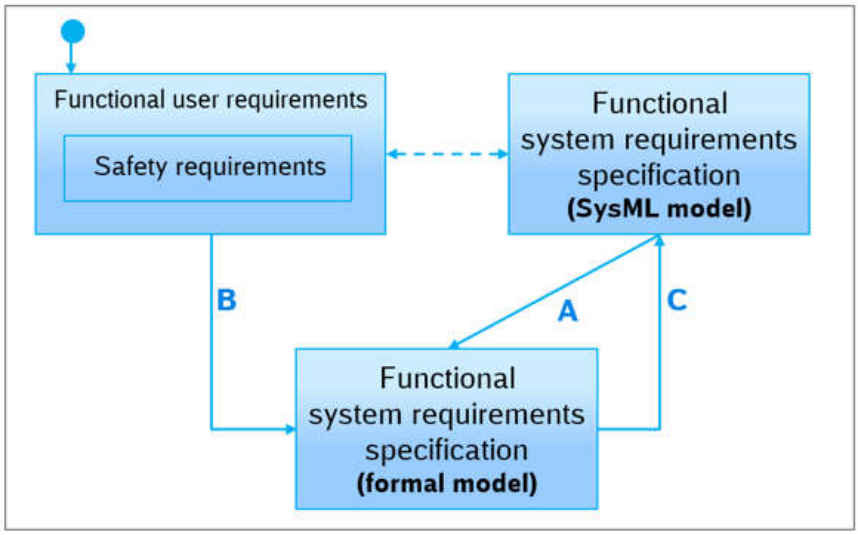

Figure 3 Specification verification approach using formal verification

The semi-formal SysML model (functional system requirements specification) derived from functional user requirements was transformed manually into a formal model in the B language **(A)**. Then safety requirements were proved by applying formal verification **(B)**. If necessary, the SysML model is corrected **(C)** and the process starts again with **(A)**.

If all applicable safety requirements have been formulated and made subject to formal verification using this process, then it has been verified that the model meets the necessary safety requirements.

6.2 Different Specification Principles

Different specification principles exist, and one can compare them using several properties.

6.2.1 Operational Property Specification

An operational specification describes the behaviour of a system using an *abstract machine*. This can be realized using data-flow diagrams that assemble functions connected by data flows. Since data flows may not always be natural for expressing control aspects, finite state machines can be preferred to describe the temporal and behavioural views of a system. Control is specified using states, events, and transitions in response to stimuli. There are many variants of state machine specification languages. A state machine can be executed, to validate the behaviour, and static analyses of the state machine can be performed (including consistency properties, and formal verification of properties).

In general, using an operational specification of behaviour and requirements offers an advantage in that it enables to determine if a specific property holds or not. This can prevent communication issues between different actors (designers, builders, customers, and users) since the operational specification provides a reference model to check the property against.

Whether an operational specification exhibits a specific property may often-case be easy to determine but it may also offer a challenge, for various reasons. To determine if a property holds or not can be non-trivial due to e.g., specification complexity that may prevent inspection alone, state-space explosion impacting the results attainable in automated analysis, and semantics for interpretation that can complicate analyses.

In general, it is desirable to have an implementation-independent operational specification, so that all actors can agree on and use the same specification. The reason for this is to avoid, when the system under specification is delivered, that supplier and customer dispute about whether systems meet the desired properties or not. In general, it is recommended that system specifications are operationalised as much as possible [12].

6.2.2 Stimulus-response Specifications

Stimulus-response specifications are an important class of operational specifications. A stimulus-response specification has the form **S and C ➔ R**, where **S** is a stimulus, **C** is a condition on the system state, and **R** is a response. The design process consists of decisions about R. In a nutshell, whenever a stimulus occurs there will be a corresponding response. The kind of response depends on the condition on the state of the system. ”Do nothing” is also considered a valid response which is usually implicit if no explicit response is specified.

Figure 4 shows stimulus-response specification of system properties using state machines. Condition **C** is represented by states OFF and ON. "button_pressed" represents stimulus **S** and "light_on" the response **R**. If stimulus "button_pressed" occurs when the system is in the state "OFF" (Precondition), the stimulus changes the value from "false" to "true" and this change is triggered by the change event "when(button_pressed)". Therefore, the system changes its state to "ON" (Postcondition) and the response "light_on" is set to "true" (light_on := true) i.e. the system responds with "light_on”. By changing its internal state from OFF to ON, the system updates its value. If the stimulus "button_pressed" occurs when the system is in the state "ON", then the response of the system will be different. It simply will "do nothing".

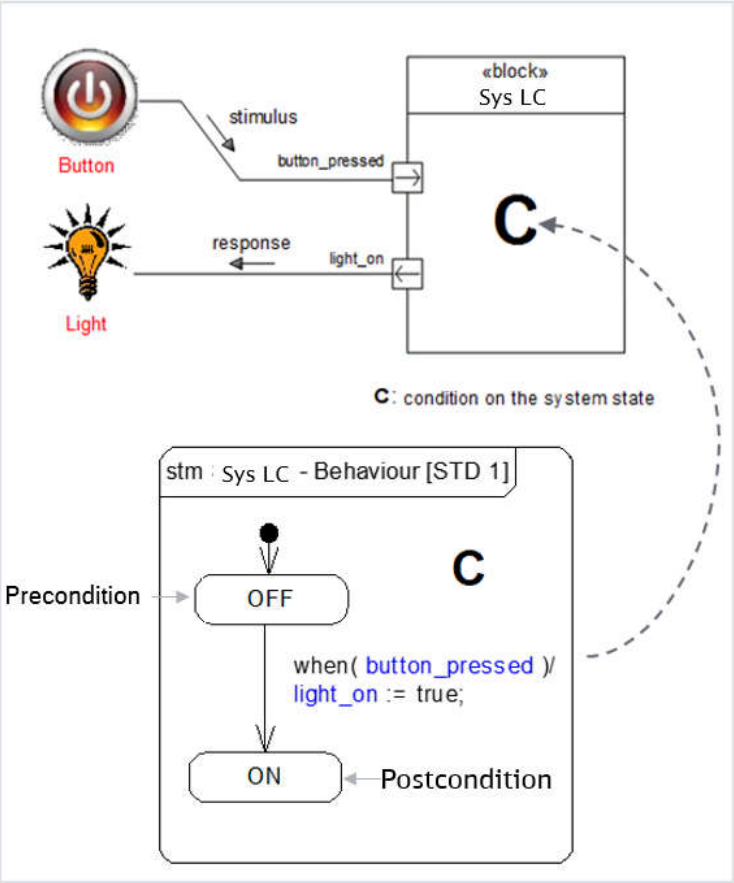

Figure 4 Stimulus-response specification of a functional system property

6.2.3 Interface-centric specification

By an interface centric approach, it is understood that the externally visible stimulus-response behaviour (usage behaviour) of a system is largely described by the behaviours related to its interfaces. These behaviours are linked together and supplemented by behaviour relevant for more than one interface by means of linking behaviour. As depicted in Figure 5, the models of the protocol stacks assigned to the communication interfaces are downscaled to the Process Data Interface protocols (PDI) defining the global PDI behaviours of the application layers. Global behaviour specifies the dependencies between the local PDI behaviours of the communication partners, that is the exchange of Process Data Units (PDU) between them in chronological order. The local PDI behaviours represent the behaviours of the communicating systems related to a certain interface. The relation between local PDI behaviour and global PDI behaviour can be illustrated by a telephone call. The dialing is a local PDI behaviour at the initiator side, the ringing is the associated local PDI behaviour at the partner side. Only the global PDI behaviour defines that the dialing must precede the ringing (i.e., the chronological order).

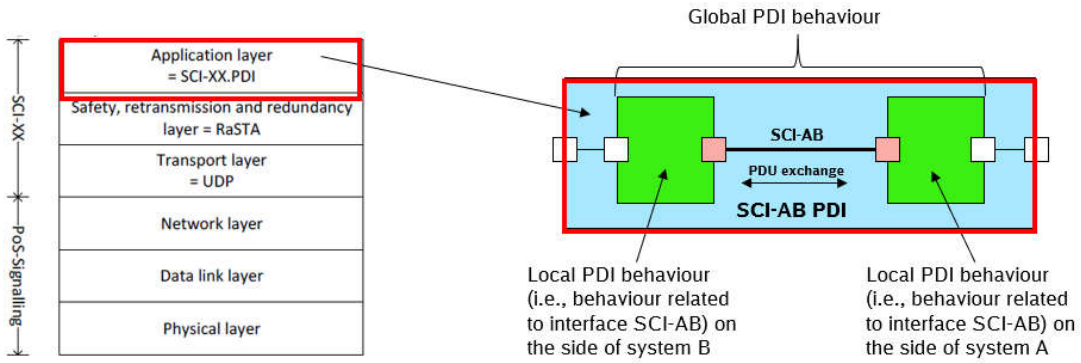

Figure 5 Global and local PDI behaviour

As the local PDI behaviours represent the interface behaviours of the communicating systems they may be specified in the model of the PDI. As depicted in Figure 6, in the model of a system element such as System A, these local PDI behaviours are referenced and linked together (Linking Logic).

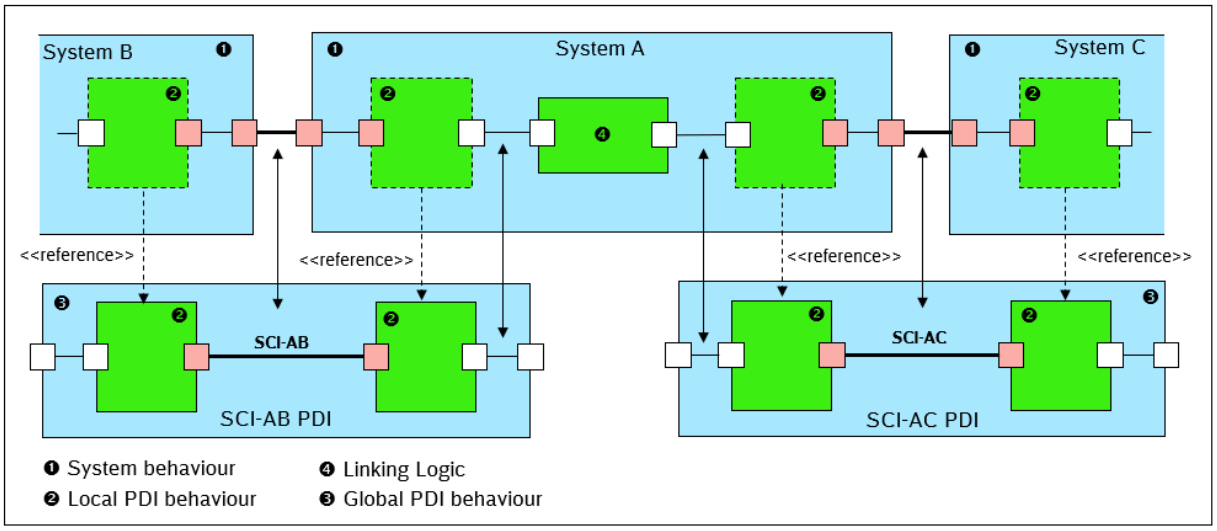

Figure 6 Principle of interface-centric specification

6.2.4 Imperative and declarative specifications

Imperative specification implies to “say how to do something”, whereas declarative specification implies to “say what is required and let the system determine how to achieve what is required”. Imperative process modelling is often referred to as an “inside-to-outside” approach. It mainly specifies the procedure of how work must be done. In general, imperative modelling requires all execution alternatives to be explicitly specified in the model before the execution of the process. Declarative process modelling, by contrast, is referred to as an “outside-to-inside” approach. Compared to imperative specification, declarative specifications do not specify the

procedure/events. Instead of determining how the processes must work exactly, only their necessary characteristics are described in a declarative specification.

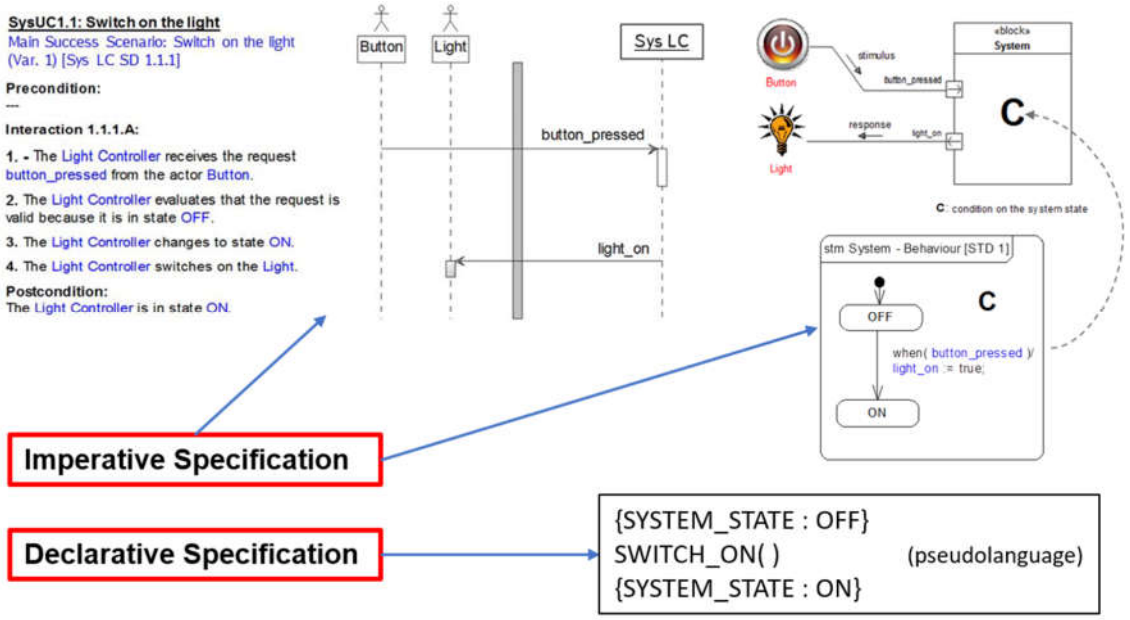

Figure 7 Imperative and declarative specification

As shown in Figure 7, sequence diagrams and state machine diagrams are imperative representations of a system. It specifies the necessary behaviour and interactions to switch ON the light when the system is in the OFF state. It is explicitly specified in the model. The instructions in a pseudo language are a declarative representation of the system. Only the necessary characteristics such as pre and postconditions, functions etc. are specified in a declarative specification.

6.3 Existing specification approaches of interest

As described in the introduction (see Section 4.4), this deliverable aims to define necessary additions to the specification technology currently used in EULYNX in the context of the requirements of future railway system architectures, such as RCA. The remainder of this section describes specification technologies used in RCA and EULYNX followed by the intended interaction of both specification technologies.

6.3.1 RCA specification approach

Figure 8 depicts the specification scope of RCA in the context of ETCS and EULYNX [6]. RCA's architectural principle is based on a layered architecture, where each layer deals with a specific problem. RCA's architecture layers are listed below. Each layer has special types of blocks and may have specific design rules for interfaces (generic for all blocks in the layer) and abstractions used. For example, in the Device Control Layer, a function will know which type of hardware (point,

level crossing, TDS) it controls. In the Safety Control Layer, objects are only known by their abstract, hardware-independent capabilities (e.g., “trafficability on a node-edge-model”).

- **Plan Implementation Layer:** functions of this layer implement the operation plan by issuing single object-control requests (OCR) when conditions for the current operational status are met. These OCRs can, for example, change a switch position or update movement permission.
- **Safety Control Layer:** functions of this layer process requests from upper layers or users. If they lead to a safe state of production, then they are executed. They also check events and the overall status of all objects and invoke emergency reactions for unsafe situations.
- **Object Aggregation Layer:** functions of this layer combine devices for an abstracted object representation. They co-ordinate devices (actors) for the execution of object-control commands.
- **Device Abstraction Layer:** functions of this layer offer abstracted device capabilities (functions and information) and abstracted device access (e.g., topology related).
- **Device Control Layer:** device-control functions of this layer steer and administrate devices. They assure the quality of the device control. They offer easy access to devices via data network for the layer above.
- **Generic Function Layer:** generic functions of this layer interact with every other layer (e.g., diagnostics) or are not part of the main control loops (e.g., data prep).

To each layer, corresponding functions are allocated aggregated in blocks. The colours of the blocks in Figure 8 indicate the relation to the specification scope of RCA. Functions aggregated in orange blocks like “ATO Execution” and “Safety Logic” are fully specified in RCA. Light green blocks like “ATO Transactor” and “ATO Vehicle” contain functions that are specified in RCA with due regard to existing specifications. The same applies to blocks coloured in light blue like “Point” and “Level Crossing”, containing functions specified in EULYNX. The functions of the mobile objects (dark blue blocks) are still to be determined. Violet blocks like “Device and Config. Management” and “Diagnostic and Monitoring” contain functions that are only partially specified in RCA. Devices (device functions) as actors or sensors are not in the scope of RCA and are not depicted in Figure 8. For example, switches, level crossings etc are not in the scope of RCA.

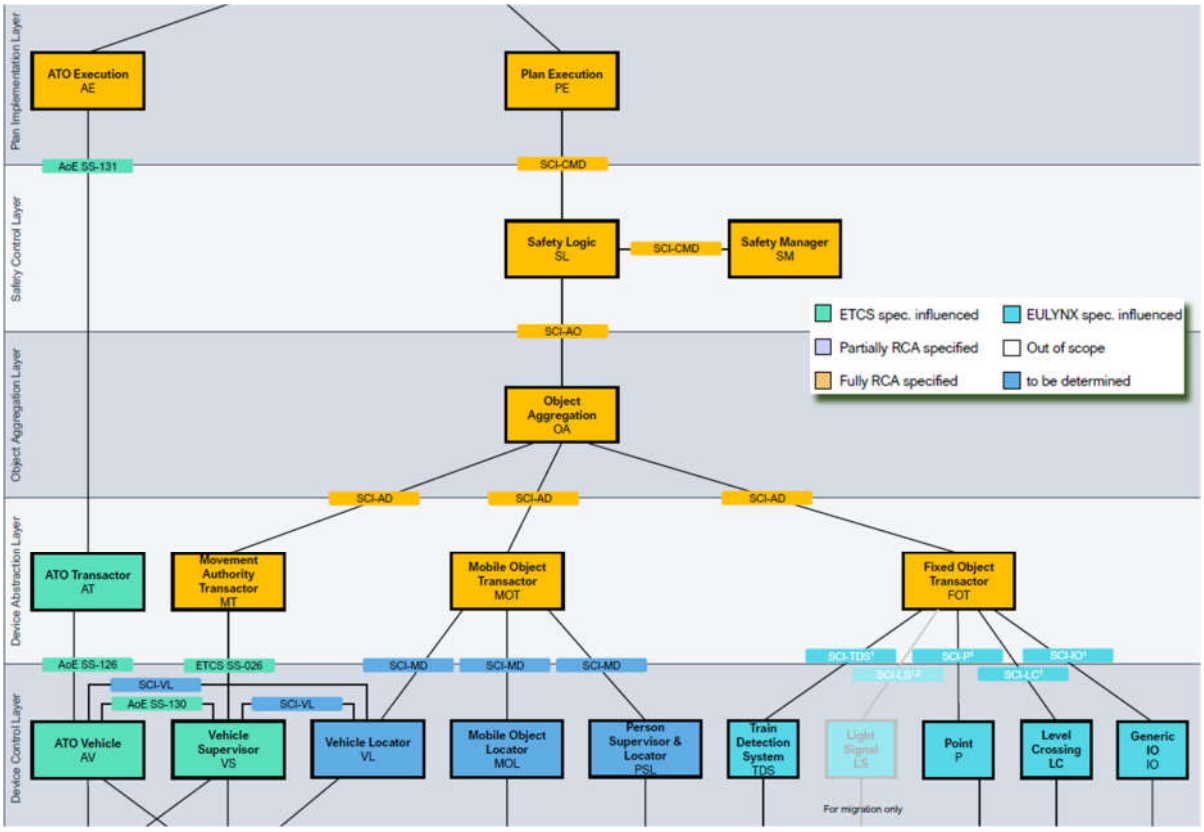

Figure 8 RCA Logical Architecture Overview

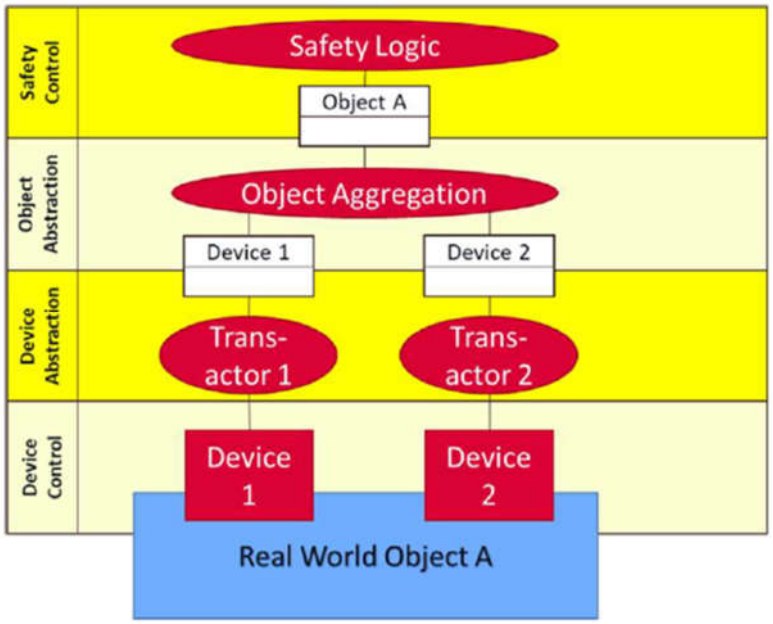

Figure 9 RCA Object aggregation

An important principle of RCA is the aggregation of objects. Object aggregation provides an abstract view of controlled objects to an upper layer. How objects relate to devices is hidden from the upper layers. As depicted in Figure 9, an abstract object monitors and manages one or more devices, and “Object Aggregation” can combine multiple devices into a single object representation of the real-world object.

The specification of RCA has a great impact on current/today’s specifications in terms of architecture and scope, with increased function range and additional characteristics of functions. These aspects must be considered in a future specification approach.

The modular, standard system architecture aimed for in RCA is currently created in the form of an analysis model using the ARCADIA method [7] and the supporting tool Capella. A SysML-like tool-specific semi-formal language is used as description means. ARCADIA stands for ARChitecture Analysis and Design Integrated Approach. It is a structured engineering method aimed at defining and validating the architecture of complex systems. The working levels of ARCADIA are depicted in Figure 10:

- **Operational analysis**
- **System analysis (Functional and Non-Functional Need)**
- **Logical architecture**
- **Physical architecture**
- **EPBS (final product breakdown structure) and integration contracts**

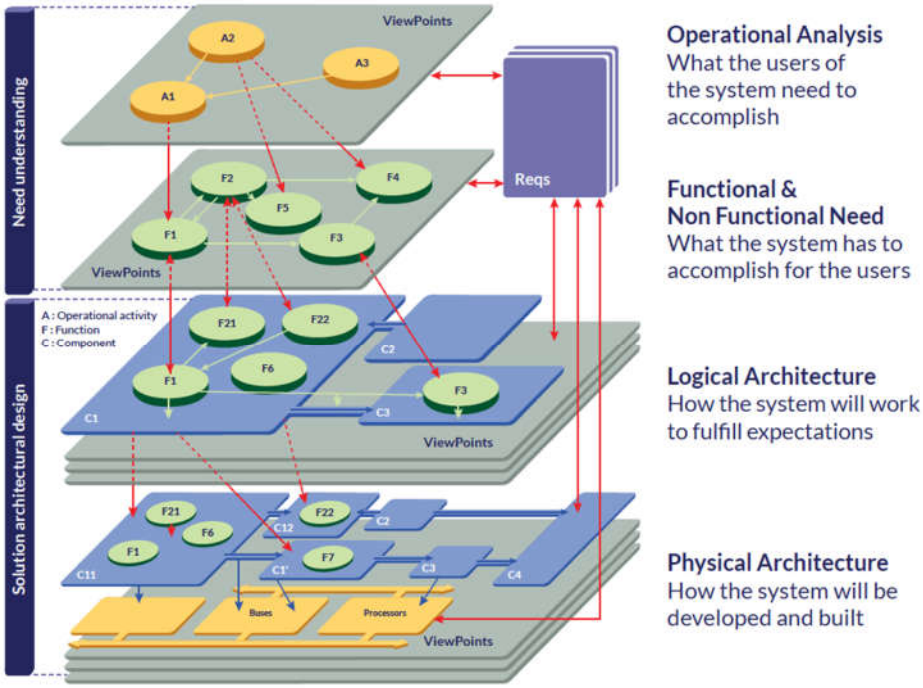

Figure 10 Working levels of ARCADIA method

6.3.1.1 Operational analysis

Operational analysis is the highest level of the Arcadia method with the goal to focus on the identification of the needs and objectives of future users of the system to guarantee the adequacy of the system faced with these operational needs. At this level, the system is not (yet) recognized as a modelling element. It will only be recognized as such from the System Analysis level onward.

6.3.1.2 System analysis

System Analysis involves the identification of the Capabilities and Functions of the system that will satisfy the operational needs ("what the system must accomplish for the users"). The System is identified as a modelling element at this level. It is a "black box" containing no other structural elements, only allocated Functions.

6.3.1.3 Logical architecture

Logical architecture aims to identify logical components inside the System ("how the system will work to fulfil expectations"), their relations and their content, independently of any considerations of technology or implementation. An internal functional analysis of the system must be carried out:

- The subfunctions required to carry out the system functions chosen during the previous phase must be identified,
- Next, a split into logical components to which these internal subfunctions will be allocated must be determined, all the while integrating the non-functional constraints that have been chosen for processing at this level.

6.3.1.4 Physical architecture

The objective of this level is the same as for logical architecture, except that it defines the final architecture of the system, and how it must be carried out ("how the system will be built"). It adds the Functions required for implementation, as well as the technical choices, and highlights two types of physical components:

- Behaviour physical component and
- Node (or Implementation) physical component.

6.3.1.5 EPBS (End Product Breakdown Structure) and integration contracts

This level aims to deduce, from the physical architecture, the conditions that each component must satisfy to comply with the constraints and choice of design of the architecture identified in the previous phases ("what is expected from the provider of each component"). The physical components are often grouped into larger configuration items that are easier to manage in terms of industrial organization and responsibilities.

Further information regarding the ARCADIA method and the supporting tool Capella can be found in [7].

6.3.2 EULYNX specification approach

While the RCA specification approach aims to define a modular, standard system architecture (analysis model) the EULYNX approach focuses on the creation of tender system requirements specifications of system elements (specification model). The behaviour of a system may be described as technology and implementation-independent by system services in the form of use cases. Services are provided by a system to the actors in its environment describing its functionality in terms of how it is used to achieve the goals of the actors. Actors may represent external systems or humans who interact with the system. The execution of a service results in a stimulus-response behaviour visible at the interfaces of the corresponding system. This implies that the service-specific behaviour of a system considering the execution of all its services completely corresponds to the stimulus-response behaviour visible at its interfaces. To guarantee a standardised communication, this behaviour must be completely, consistently, and correctly specified for each interface and linked together to the required overall behaviour of the corresponding system. Due to the necessary high level of detail of the behaviour to be specified this approach requires the creation of understandable high-quality specifications and sophisticated methods to verify and validate them.

6.3.2.1 EULYNX MBSE Specification Framework

To meet the above-mentioned challenges, a model-based systems engineering (MBSE) methodology defined in the EULYNX Modelling Standard [6] has been developed by the initiative EULYNX proposing an MBSE Specification Framework (MBSE SF) as illustrated in Figure 11. MBSE SF facilitates a holistic model-based seamless description of complex CCS systems. To apply MBSE using a formal language that enables formal verification and formal proof may seem to be highly recommendable. However, following the goal to create specifications understandable also for people not familiar with formal languages, the Systems Modelling Language (SysML) [10] has been regarded as a reasonable compromise to be used as the main description means.

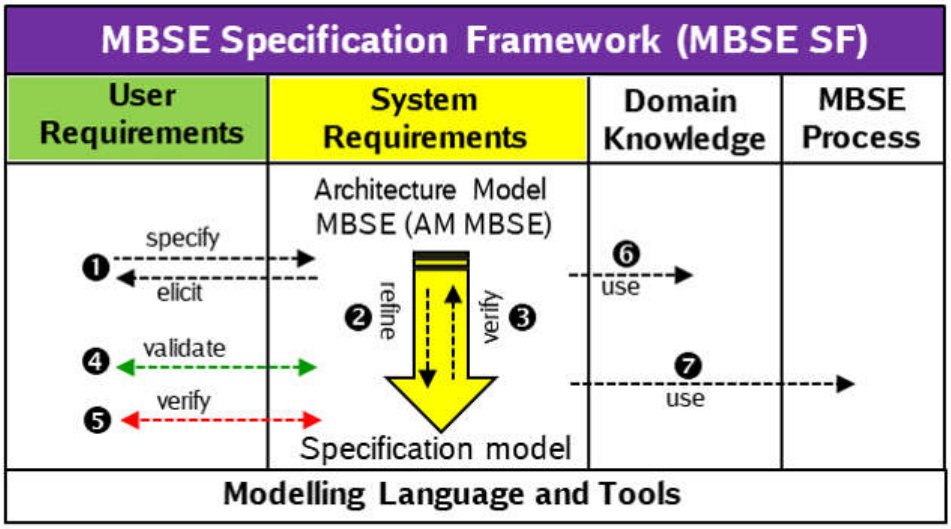

Figure 11 EULYNX MBSE Specification Framework

Guided by an MBSE Process (❼) and using the defined Domain Knowledge (❻) the MBSE SF strictly distinguishes between problem domain (User Requirements) and solution domain (System Requirements). User Requirements represent the model of the problem domain (problem definition). They allow different stakeholders such as infrastructure managers (IM) of different countries to explicitly state what is expected from the future system. User Requirements may also result from system analyses and comprise, as an important part, the safety requirements. Safety requirements, also referred to as safety goals, state safety invariants, i.e. conditions that could lead to hazardous situations if they are not met. They can be split into two categories: safety invariants (what must not happen under any circumstances) and safety overrides (who may do what under which circumstances). User Requirements are the main source for the derivation of design decisions (❶) as the basis for the creation of an abstract system solution (System Requirements). The System Requirements are represented by a specification model. The specification model shall be validated that IM intentions are reflected completely and correctly (❹) and verified (❺) that all defined safety requirements are consistently fulfilled without contradiction.

6.3.2.2 EULYNX Architecture Model MBSE

The creation of the specification model is guided by the structural rules of the Architecture Model MBSE (AM MBSE). The AM MBSE facilitates the specification of a system from different viewpoints and with varying degrees of granularity, that is at different abstraction levels (AL), by means of system views. A system view represents the description of a system from a viewpoint at a specific AL. The basic notion of this approach is to start with rather high-level descriptions of system views. Once these high-level descriptions have been created, they are refined and detailed step by step (❷). Any AL represents design decisions about the refined or decomposed description of its predecessor and the specification of the outcome of these decisions by using appropriate system views. To ensure that the more granular system views resulting from the refinement comply with its predecessors and are traceable to them, the refinement is verified (❸) and correlating links are established according to the EULYNX verification and validation plan. As shown in Figure 12, the abstraction levels AL1, AL2 and AL3 of the AM MBSE strongly reflect the recommendations of CENELEC (EN 50126) [11]. In contrast to the abstraction levels AL1, AL2 and AL3 which provide system-dependent views necessary to specify a concrete system, abstraction level AL0 enables the system-independent definition of the operational specification base (e.g. goals, operational entities, operational process, etc.). Each AL is subdivided into three cross-cutting viewpoints capturing concerns regarding the services a system is expected to provide to its environment and their refinement (Functional Viewpoint), the description of the system from a structural point of view independently of any considerations of technology or implementation (Logical Viewpoint) and the physical architecture of the system (Technical Viewpoint). The corresponding system views are linked together via allocation relations. The system services defined in the Functional Viewpoint are for example allocated to structural elements specified in the Logical Viewpoint.

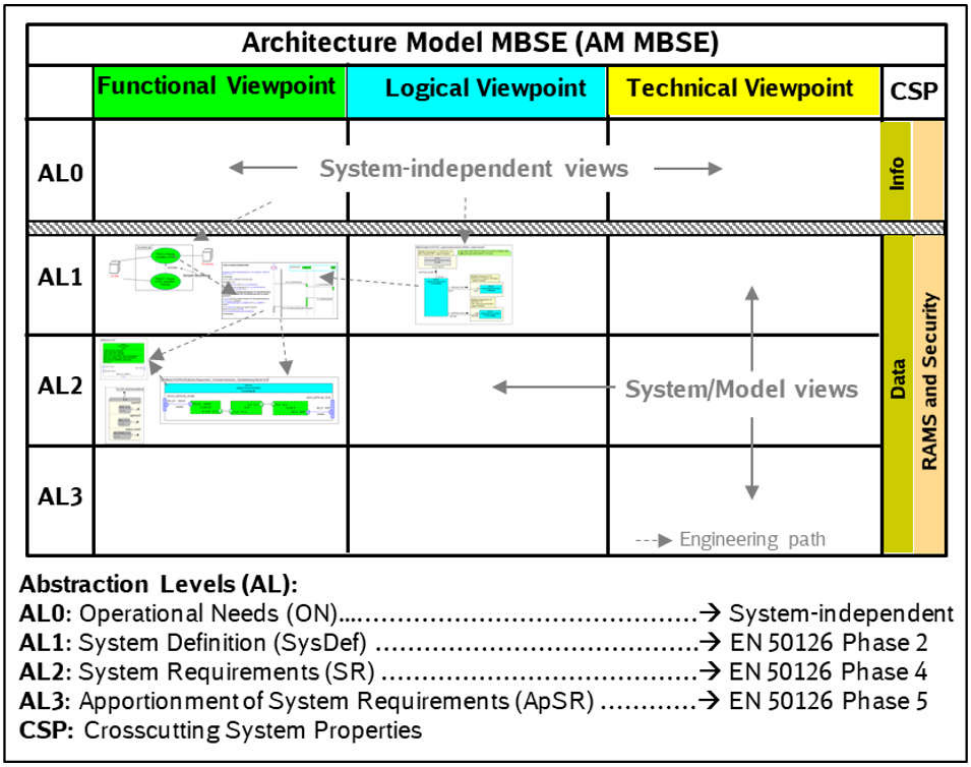

Figure 12 EULYNX Architecture Model MBSE

An important principle of the AM MBSE is the continuous engineering of crosscutting system properties (CSP). Typical crosscutting properties are RAMS [11], security, data used to define information objects to be exchanged and non-functional constraints: they must be considered in any engineering activity and the corresponding system views. Safety, for example, typically defined as freedom from unacceptable risk (of harm), affects almost all process steps in a development lifecycle. For this reason, safety is not represented in a single viewpoint but as a quality aspect of the AM MBSE that has a crosscutting influence and is integrated into several viewpoints. The growing complexity of safety-critical railway systems is leading to increased complexity in safety analysis models. It is therefore not appropriate to develop functionality and consider safety in separate tasks. Safety aspects should be integrated as tightly as possible into the MBSE process and its models.

The AM MBSE is used in two versions:

- **to create specification models of system elements and**
- **to create specification models of interfaces.**

In the next two chapters, the corresponding engineering paths to create specification models of system elements (Section 6.3.2.3) and specification models of interfaces (Section 6.3.2.4) are briefly described. An engineering path (shown as a dotted arrow) summarises the development of views for a system element or interface with a specific degree of granularity.

6.3.2.3 Engineering path to create specification models of system elements

Figure 13 shows the engineering path of the model views used to specify a system element. It describes the context of the model views, with the arrows indicating which model views are developed from which. The specification model of a system element consists of the following model views:

- **Model view "Logical Context":** the model view "Logical Context" describes the system element under specification (SUS), the actors in the environment interacting with the SUS and their quantity structure (multiplicities) at the upper level of abstraction.
- **Model view "Functional Context":** the model view "Functional Context" defines the services to be provided by the SUS in the form of use cases. Relationships are used to represent which actors interact with which SUS use case.
- **Model view "Use case scenario":** the model view "Use case scenario" describes the behaviour of the use cases defined in the model view "Functional Context" by means of use case scenarios.
- **Model view "Functional Architecture":** the model view "Functional Architecture" refines or completes the behaviour of a SUS defined in the model view "Use case scenarios". The behaviour of the SUS is divided into Functional Entities" (FE), which communicate with each other via internal interfaces and with the environment via external interfaces.
- **Model view "Functional Entity":** the model view "Functional Entity" encapsulates a subset of the functional requirements of a SUS in the form of a function module. It delimits the function module from its environment and defines the inputs and outputs. In the discrete case, the behaviour of the function block is described by means of state machines. In this, the binding functional requirements are specified in the form of states and corresponding state transitions. The model view "Functional Entity" is used for the specification of EULYNX system elements as well as for the specification of EULYNX interfaces.

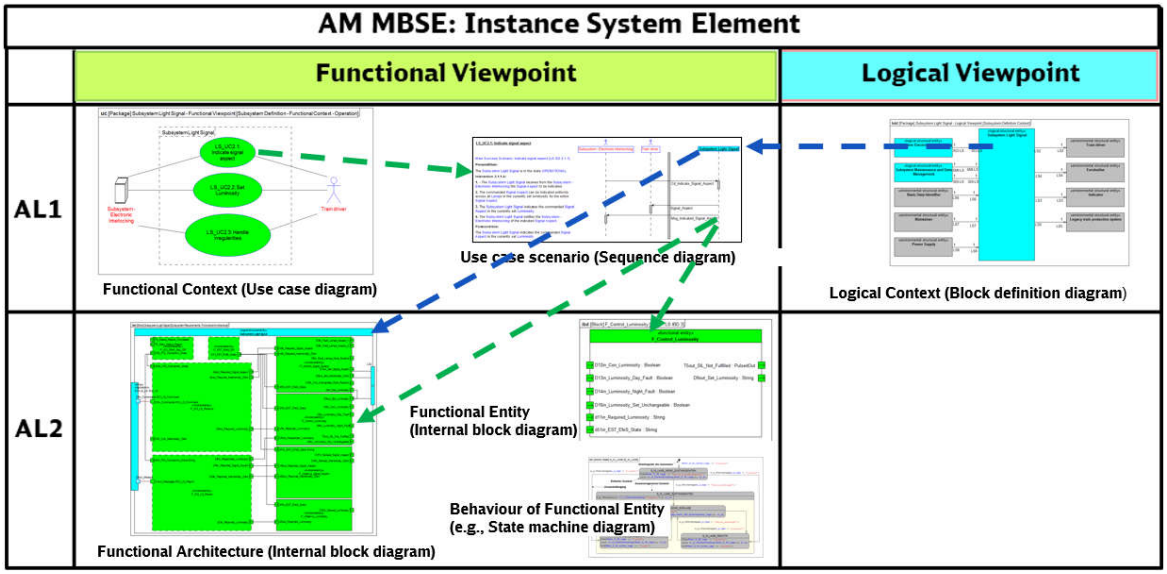

Figure 13 Model views to specify a system element

During the development of the model, the Logical Context (BDD) and the Functional Context (the Use Cases) of the system elements are created. Based on these two model views, the use case scenarios (sequence diagrams) are derived. These sequence diagrams form the basis for the description of the Functional Architecture (IBD) and the behaviour of the Functional Entities (IBD, state machines).

6.3.2.4 Engineering path to create specification models of interfaces

Figure 14 shows the engineering path of the model views used to specify an interface. It describes the context of the model views, with the arrows indicating which model views are developed from which. The specification model of an interface consists of the following model views:

- **Model view "Logical Context":** the model view "Logical Context" describes the logical view of an interface at the upper level of abstraction. An interface is generally defined as a unique connection between two communication participants. From the logical viewpoint at the upper level of abstraction, an interface is represented by a SysML association between them.
- **Model view "Functional Partitioning":** the model view "Functional Partitioning" describes the refinement of the interface defined in model view "Logical Context" using Functional Entities.
- **Model view "Functional Architecture":** the model view "Functional Architecture" defines the global behaviour of the application protocol.
- **Model view "Functional Entity":** the model view “Functional Entity” is described in chapter 6.3.2.4.
- **Model view "Information Flow":** the model view „Information Flow" describes the information objects to be exchanged via an interface which is further refined to telegrams at abstraction level AL3.

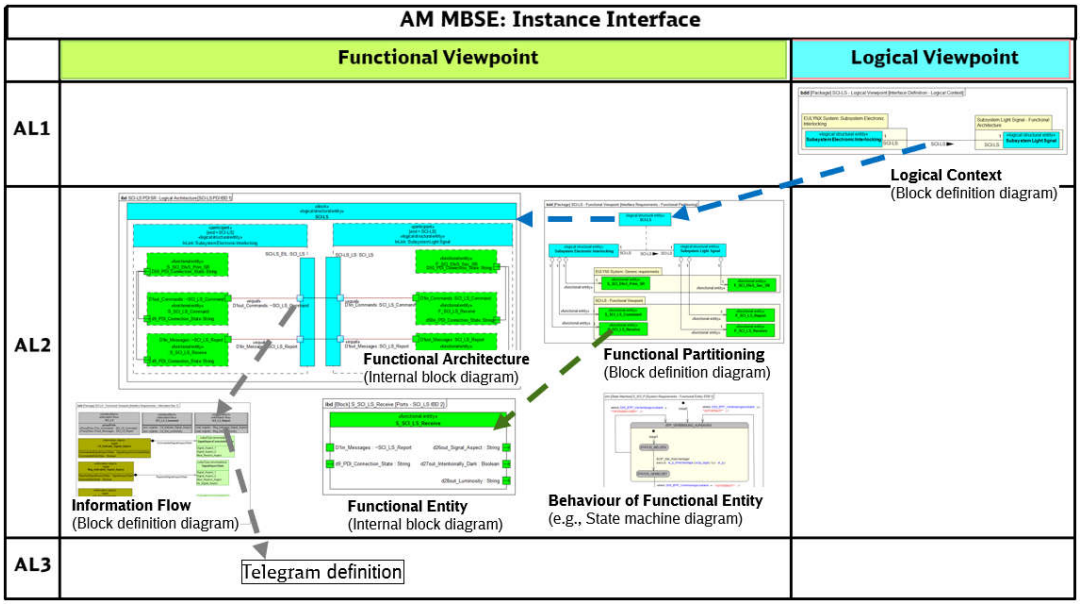

Figure 14 Model views to specify an interface

During the development of the model, the Logical Context (BDD) of the interface is created. In the next steps, the interface represented by an association in the model view "Logical Context" is refined in the model view "Functional Partitioning" by means of Functional Entities and the global behaviour of the application protocol layer is defined in the model view "Functional Architecture". Finally, the behaviour of the functional entities is defined in the model view "Functional Entity" and the information objects to be exchanged in the model view "Information Flow". The information flows are refined by telegrams. This step is currently still carried out using an informal specification technique, i.e. the telegrams are presented in the form of tables.

6.3.2.5 Tool

The EULYNX MBSE Process described in Section 6.3.2.6 is supported by a toolchain as illustrated in Figure 15. It enables the creation of SysML specification models (Windchill Modeler), static checks for completeness, correctness, and consistency (Windchill Reviewer) and simulation-based validation of the models (Windchill Modeler SySim and MS Visual Studio). A connection to IBM Rational DOORS (Windchill Integration for IBM Rational DOORS) enables the representation of specification model elements in the form of atomic requirements in the requirements management tool. They can be transformed into the standardised Requirements Interchange Format (ReqIF) and exchanged with suppliers using Windchill Requirements Connector.

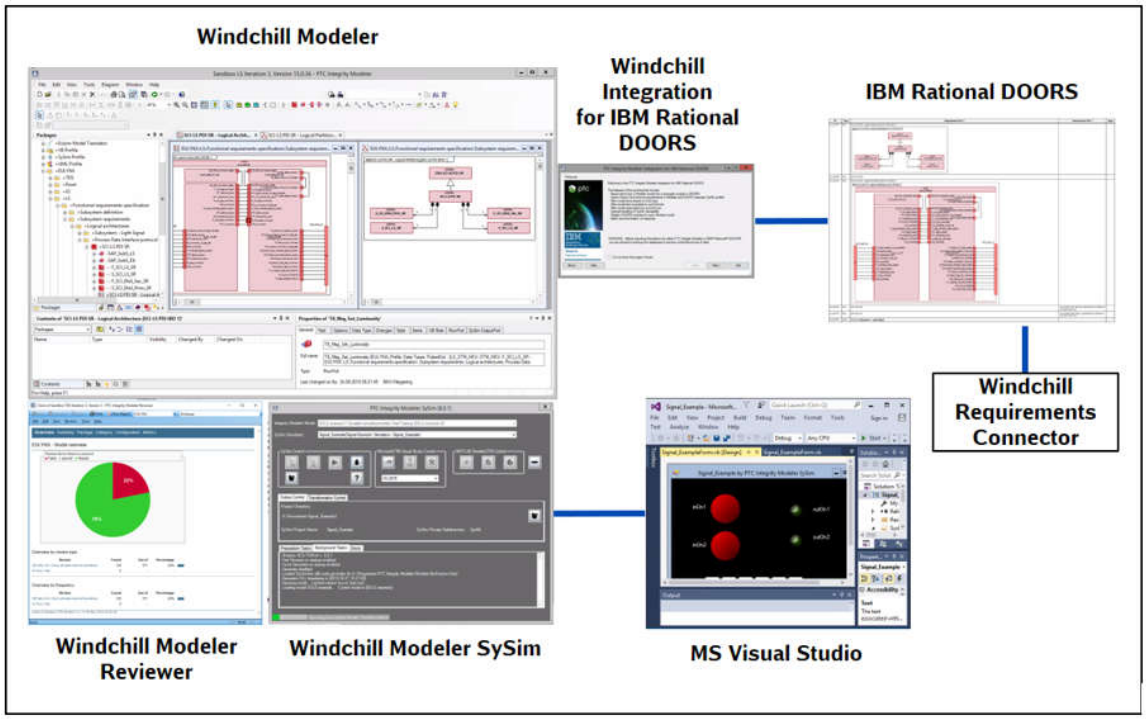

Figure 15 EULYNX tool chain

6.3.2.6 EULYNX MBSE Process

The EULYNX MBSE process is part of the EULYNX systems engineering process with the main process tasks documented in the EULYNX verification and validation plan. The EULYNX systems engineering process is closely oriented on the CENELEC system life cycle defined in EN 50126 and covers the phases listed below:

Phase 1: Concept

Phase 2: System definition

Phase 4: System requirements

Phase 5: Apportionment of system requirements

Phase 10: System acceptance

Phase 11: Operation and maintenance

The CENELEC system life cycle follows the V-model, which highlights verification and validation, especially regarding the fulfilment of safety requirements, as important tasks. Already during the specification phases of the V-model, verification and validation are important activities, applied to assure the quality of the specification itself. This is especially necessary for the context of the EULYNX MBSE approach, where models of the required system behaviour represent abstract reference implementations of the future system (virtual prototypes) and are regarded as mandatory requirements in tender specifications.

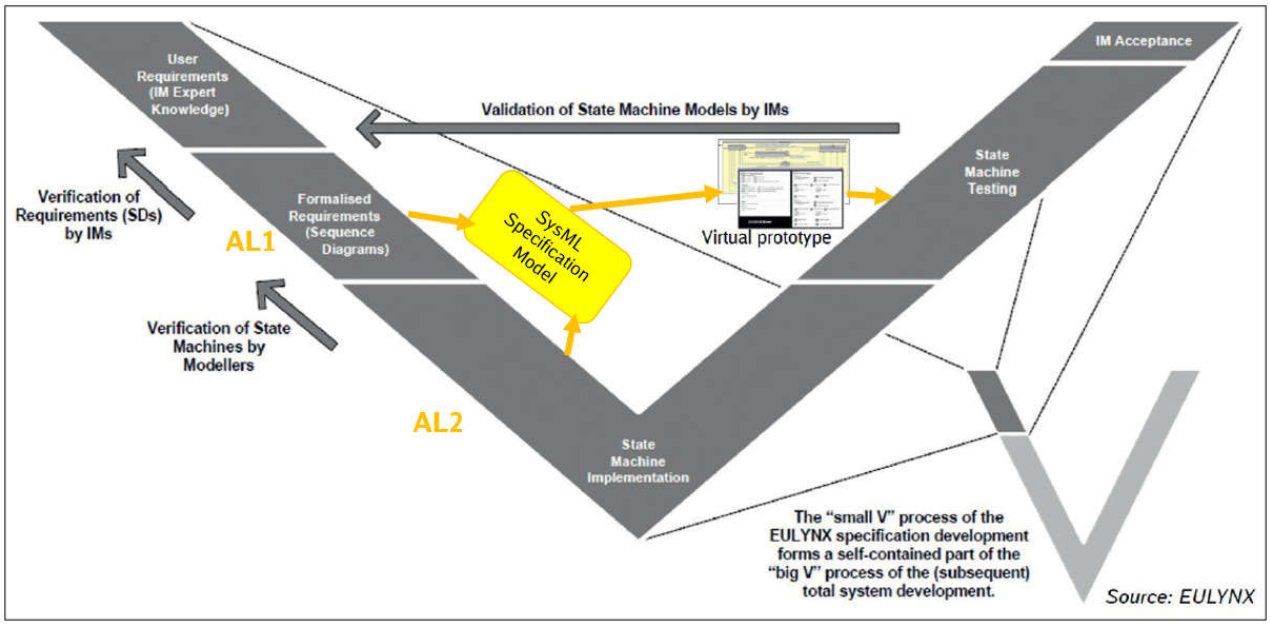

Figure 16 The "small V"-the process of the "big V" - CENELEC process

Following this notion, it is necessary to provide a “small V”-process, guiding the top-down development of those virtual prototypes using executable SysML state machines and their validation and verification within the specification phases of the underlying “big V”-CENELEC process. In Figure 16, the "small V" is highlighted in the "big V" and pictures the relationships of verification and validation as part of the virtual prototype development.

User Requirements derived from IM expert knowledge are represented in IBM Rational DOORS in the form of a "Function List". It lists the required functions used as input information for the creation of the specification model at abstraction level “AL 1 System Definition” of the AM MBSE using the Windchill Modeler. At this point, the system use cases (services) are defined with their stimulus-response behaviour selectively specified by means of use case scenarios using SDs (Formalised Requirements). Subsequently, the conformity of the model to the SysML specification and the modelling rules defined in the EULYNX Modelling Standard is statically checked using the Windchill Modeler Reviewer by a modeler in the role of a model verifier. Additionally, the use case scenarios are validated by means of inspection by the corresponding IMs in the roles of model validators.

In the next step, the system views created at abstraction level “AL 1 System Definition” are refined at abstraction level “AL 2 System Requirements” by means of executable SysML state machines (State Machine Implementation). The conformity of the model to the SysML specification and the EULYNX Modelling Standard is verified using the Windchill Modeler Reviewer and by means of inspection by the model verifier.

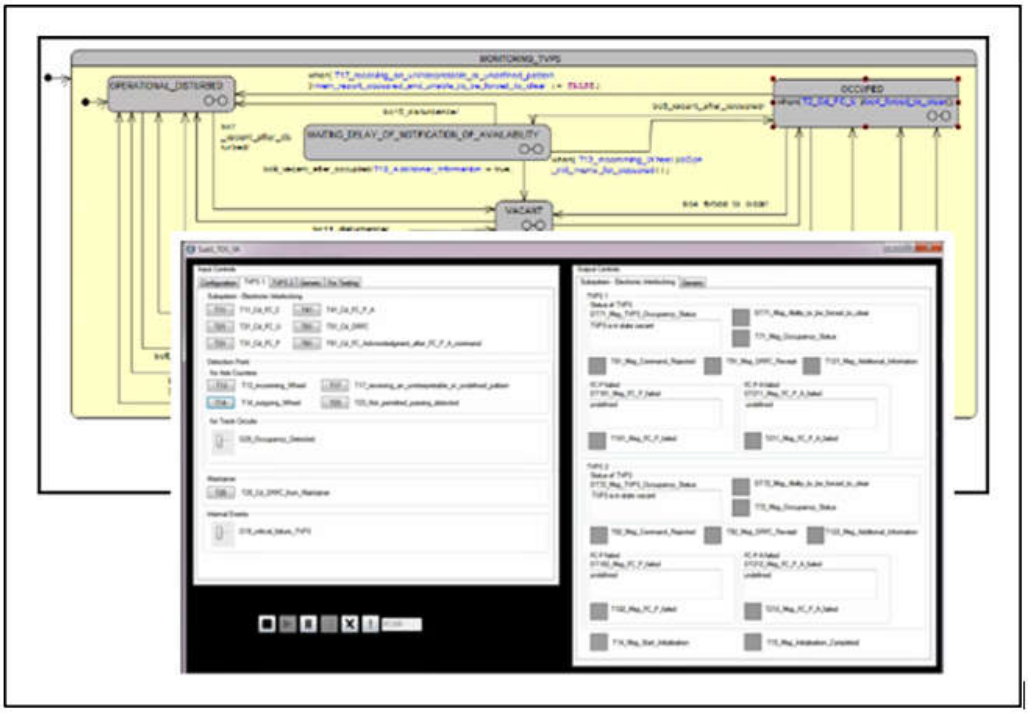

Figure 17 Principle of a virtual prototype

To implement the state machines as a virtual prototype, Visual Basic simulation code is generated using Windchill Modeler SySim. Subsequently, the GUI of the virtual prototype is designed, and an executable is created in MS Visual Studio. The executable representing the virtual prototype enables both the tool-independent standalone simulation of the specified behaviour and when connected to the Windchill Modeler the simulation together with the animation of the corresponding state machines. The virtual prototype enables simulation-based testing of the specified behaviour by injecting stimuli on the GUI and observing the responses optically indicated. The principle of a virtual prototype is depicted in Figure 17.

In the following step (State Machine Testing), the conformity of the behaviour defined by the state machines to the use case scenarios in the overlying abstraction level “AL1 System Definition” is dynamically verified by simulation-based testing of the virtual prototype carried out interactively by the model verifier. For this purpose, the scenarios are used as test cases and in parallel, the animated state machines observed (white box testing of the behaviour). Additionally, the correct creation of the state machines such as freedom of deadlocks is verified by the model verifier using interactive state machine animation based on a dedicated test specification.

The standalone virtual prototype is then handed over to the IMs to validate the behaviour specified by the state machine by means of simulation-based testing (black-box testing of the behaviour). The test approach used (script-based or interactive testing) is left to the IMs. The validation process is finished successfully when all participating IMs provide evidence that their user requirements (including safety requirements) are satisfied by the specified behaviour. The successful validation process leads to the production of a new baseline.

Model tests are carried out according to corresponding model test specifications. These comprise the information suitable to sufficiently test the behaviour of the models and consist of one or more test cases. A test case comprises meta-information (creator, date, subsystem covered, IM applicability) and a test script. The test script contains a list of steps to instruct the model verifier or model validator on how to execute the test case indicating the stimuli to be performed during the test as well as the expected results to be observed. Any test run is documented in a test report that documents the results. The test report comprises status information on every test case included in the model test specification. A more detailed description of the EULYNX model verification and validation approach is given in the EULYNX verification and validation plan available via the EULYNX website [6].

6.3.3 Merging the RCA and EULYNX specification approaches

EULYNX and RCA use different specification approaches. This is because an analysis model of a modular, standard system architecture is to be created according to the RCA specification approach and specification models of the system elements defined in this architecture are to be created according to the EULINX specification approach. As shown in Figure 18, the future joint RCA/EULYNX specification approach will thus involve the creation of an analysis model and a specification model. Both models will be connected via a traceability model.

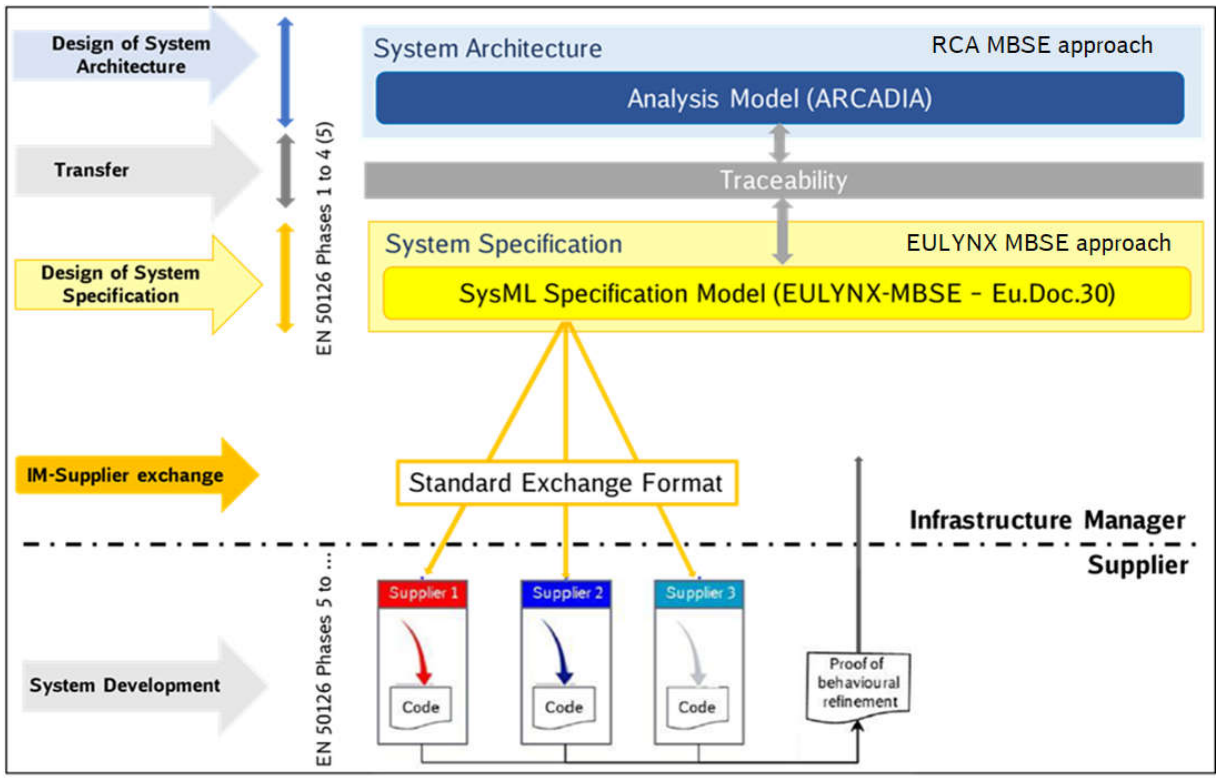

Figure 18 Joint RCA/EULYNX specification approach

7 Necessary extensions of the EULYNX MBSE approach

This chapter describes necessary extensions of the EULYNX MBSE approach.

7.1 Characteristics of a Relevant Railway System

A railway system of interest in the context of a future functional railway system architecture is a non-linear, time-varying, distributed, massively multivariate position control system. Its overall function can be described as monitoring and controlling the positions of trains and moveable infrastructure elements on the network in accordance with multiple types of constraints including scheduled traffic (the timetable). The system has a huge number of diverse sensors and actuators, some of which are inside and some which are outside the system border. Since a railway system of interest has the characteristic of a control system, this term shall be explained first.

7.1.1 Control systems

A system is an arrangement of physical components which act together as a unit to achieve a certain objective. To control means to regulate or direct. Hence a control system is an arrangement of physical components connected in such a manner to direct or regulate itself or another system. If a lamp is switched ON or OFF using a switch, according to the example shown in chapter 6.2.2, the entire system can be called a control system. In short, a control system is in the broadest sense, an interconnection of physical components to provide the desired function, involving controlling action in it. For each control system, there is an input and an output. The input is the stimulus, excitation, or reference value applied to a control system to produce, depending on its internal state, a specific response and the output is the actual response obtained from the control system. The specification of a control system can thus basically be done in stimulus-response form.

7.1.2 Classification of Control systems

7.1.2.1 Time-invariant and time-varying systems

Time-invariant control systems are those in which the system parameters are independent of time (the system behaviour does not change over time). Systems whose parameters are functions of time are called time-varying systems. The behaviour of such systems not only depends on input stimuli but also the time at which the input is applied.

7.1.2.2 Linear and non-linear systems

A linear system obeys the *superposition property*, which states that the net response caused by two or more stimuli is the sum of the responses that would have been caused by each stimulus individually.

7.1.2.3 Continuous time and discrete time system

Mathematical functions are of two basic types, continuous functions, and discrete functions. Continuous-time functions are those functions that are defined for every instant of time. Discrete-time functions, on the other hand, are those functions, whose values are defined only for certain instants of time. In a continuous-time control system, all the system variables are continuous-time functions. In a discrete-time control system at least one of the system variables is a discrete function. Microprocessor and computer-based systems are discrete-time systems. A discrete representation of a continuous-time control system is obtained by sampling continuous variables at discrete time points. Discrete systems can be time-driven or event-driven. Event-driven systems are called discrete event dynamic systems or DEDS for short [13]. DEDS are characterized by a set of states in which the system can be in, and by the set of events that cause the state changes at discrete time points. The events may take place asynchronously as opposed to the synchronous nature in a discrete-time system. The change of states and occurrence of events are the essence of the DEDS dynamic behaviour.

7.1.2.4 Open loop system

In an open loop system (see Figure 19), the control action is independent of the process output. An open loop system cannot correct any errors it makes or correct for outside disturbances.

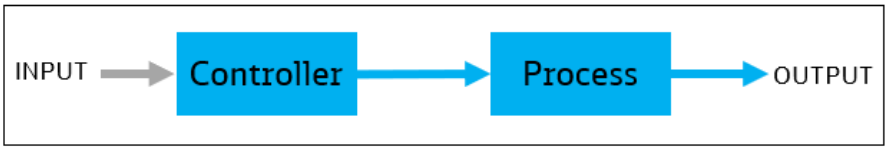

Figure 19 Open loop system

7.1.2.5 Closed loop system

In a closed loop system (see Figure 20), the control action depends on not only the input, but also the output (which is fed back as an input). A closed loop system can issue control actions to handle errors and outside disturbances to give a process output the same as the “reference input” and is amenable to machine learning.

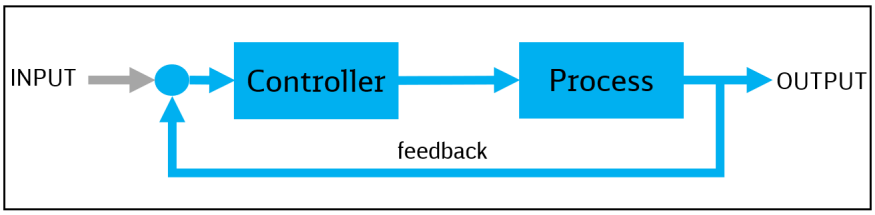

Figure 20 Closed loop system

7.1.2.6 The typical control loop of a railway system of interest

Figure 21 shows a typical control loop of a railway system of interest. The "Plant" is the system being controlled including a collection of moveable assets, train units, people, and everything else

that is controlled by it. However, this does not mean that the whole railway system can be reduced to one item of “Plant”. Rather, there are multiple “Plants” to control with multiple loops (open or closed loops). Hypothetically, the core functions of a railway system in the context of a future functional railway system architecture can be modelled as a combination of many of these control loops, which are interconnected and nested inside each other. Most core railway system of interest functions can be assigned to one of the four categories listed below:

- **Control**: the purpose of a control function is to transform information about a needed change of the plant’s state into instructions or commands for the state of the actuators. Control functions are where all the decisions are made.
- **Actuate**: the purpose of an actuate function is to transform instructions or commands into a physical state that has some effect on the plant’s internal state.
- **Sense:** the purpose of a sense function is to transform a physical external state of the plant into information about the plant’s external state.
- **Observe:** the purpose of an observe function is to transform information about the plant’s external state into an observation about the plant’s internal state. Observe functions are where inferences are made about the state of the plant given incoming data.

Basically, only what can be observed can be controlled. This is not the same as saying that only what can be sensed can be controlled. Sensed data can be used to estimate an internal state that shall be controlled, but an internal state cannot be directly sensed. Only the external states of the plant can be sensed. The speed of a train unit, for example, is an internal state. It can be inferred by sensing certain specific quantities like the doppler shift of passing objects (doppler speed sensors), the rotational speed of individual axles on the train (pulse tachometers or tacho generators), or the change in relative position to a constellation of satellites (GNSS speed sensor). From one or more of these sensed quantities, we can infer the internal state that is the speed of the train unit.

Where a human actor is in a control loop, an additional type of function “Indicate” is used so that information from observed or other controlled states can be used by the human actor to make their control decisions. This is a specialization of “Observe”. Since observed states cannot be actuated, this function category to display an observed state to a human actor is necessary. It is not explicitly shown in Figure 21.

Furthermore, functions associated with non-operational states of a railway system of interest, such as data updates or switching between operating and maintenance states, might not fit these categories because they are not part of the chain of functions directly controlling the railway state.

Figure 21 shows the information flows between the functions [(2), (5), (6)] within the control system and between them and an external reference (1) and the “Plant” [(3), (4)]. The information flows (4), (5) and (6) correspond to the “feedback” of a closed loop control system as described in chapter 7.1.2.5. The information flows are described below:

(1) Required internal state of “Plant”: e.g. the target speed of a train unit,
(2) Required external state of “Plant”: e.g. the required setting of the speed control,

(3) Actual external input state of plant: e.g. the current setting of the speed control,
(4) Actual external output state of plant: e.g. rotational speed of individual axles,
(5) Sensed external output state of plant: e.g. sensed rotational speed of individual axles,
(6) Estimated internal state of plant: e.g. the estimated current speed of a train unit.

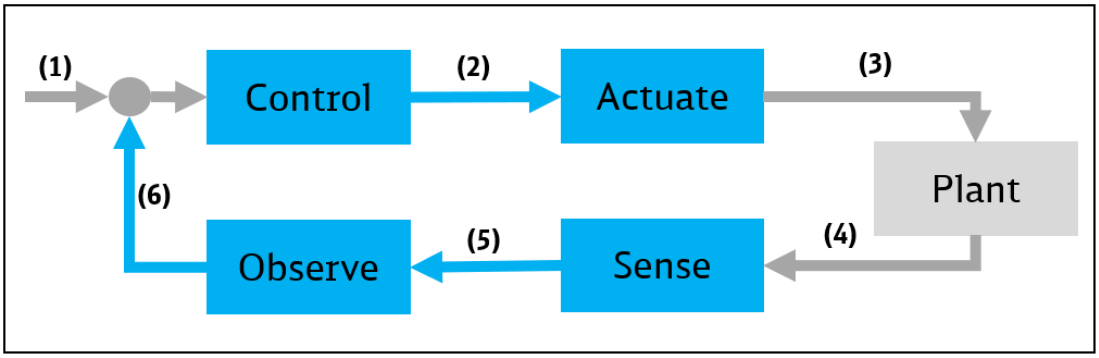

Figure 21 Typical control loop of a railway system of interest

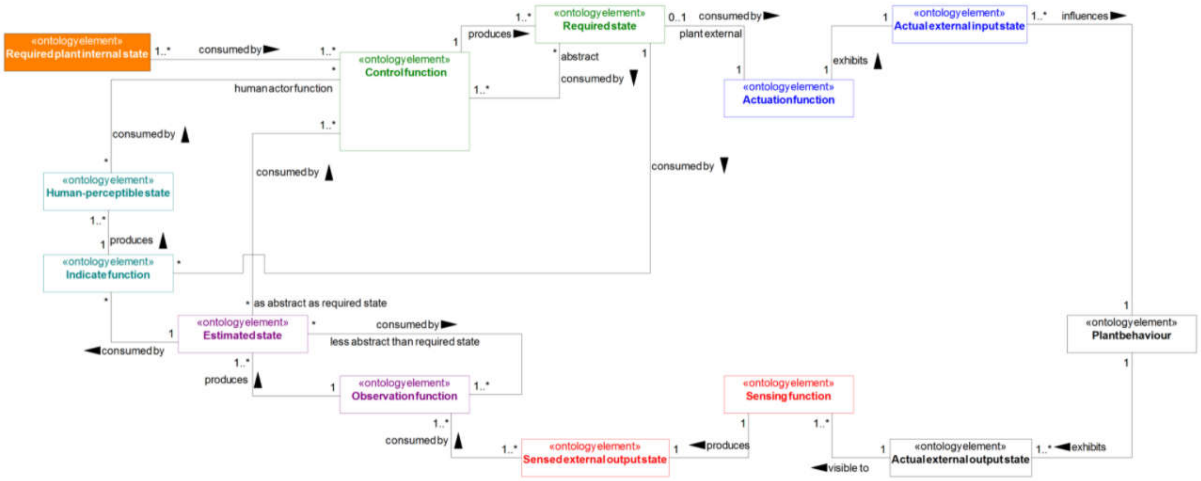

Figure 22 Control loop ontology (source: DB Netz AG, I.NAT 1)

Figure 22 shows the ontology of the control loop typical for a railway system. This ontology allows for estimated states to be exchanged between observation functions so that observations can be made at different levels of abstraction. In other words, an observer does not always have to transform a sensed external state directly into a fully abstract estimated state. Similarly, this ontology allows for control functions to produce a required state that is not the required state of an actuator, but rather the required state of some other abstract concept, that feeds to a further control function. In other words, a control function does not always have to transform a required plant internal state directly into a required actuator state.

7.1.2.7 Example: Control of a railroad turnout

A railroad turnout, switch, or [set of] points is a mechanical installation enabling railway trains to be guided from one track to another, such as at a railway junction or where a spur or siding branches off. A switch (points) consists of a pair of linked tapering rails, known as points (switch rails or point blades), lying between the diverging outer rails (the stock rails). These points can be moved laterally into one of two positions to direct a train coming from the point blades toward the straight path or the diverging path. A mechanism is provided to move the points from one position to the other (change the points). This is done by a remotely controlled electric motor called a point machine.

If we take a simple turnout like the one in Figure 23, and draw out the mechanical parts, we can immediately draw a couple of very important conclusions:

- We do not directly control the position of the point blades; we control the state of the actuator (usually, this means voltages on the terminals of the point machine motor).
- We assume, based on a model of the mechanics of the switch, that the state of the actuator influences the position of the point blades.
- We do not directly observe the position of the point blades.
- We infer the position of the point blades from the state of position sensors (point machine switching contacts).

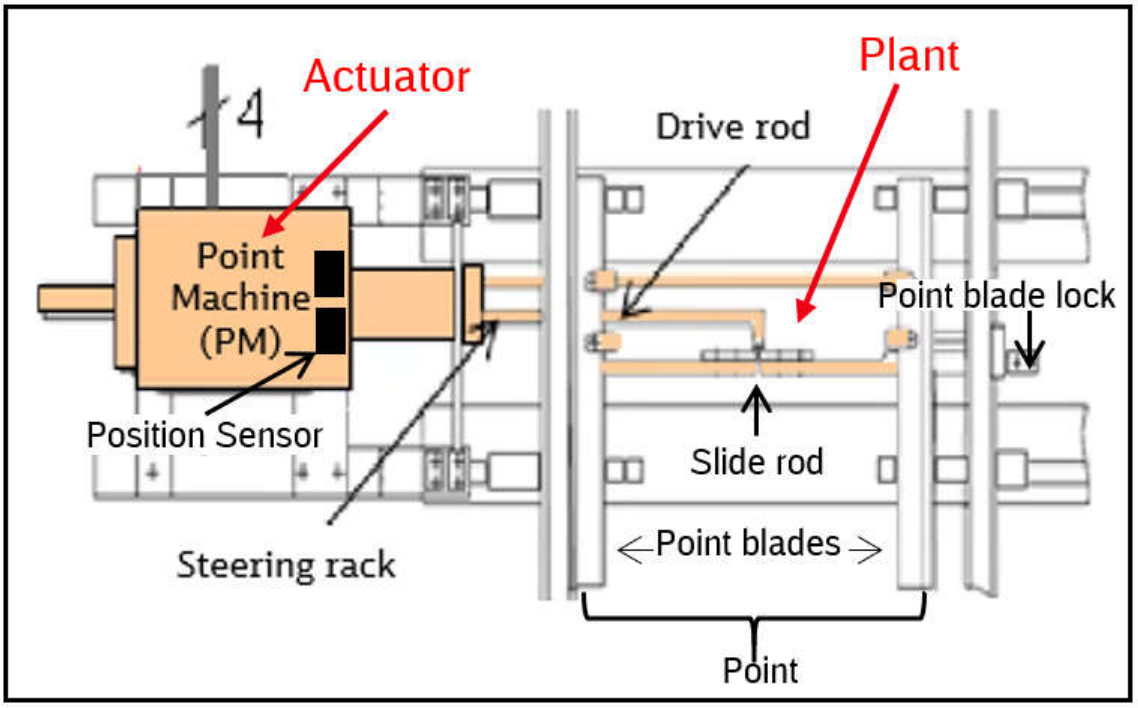

Figure 23 Simple railway turnout

The bottom line is that in the control loop of a railroad turnout all four of the basic control system functions “control”, “actuate”, “sense” and “observe” are present. The corresponding control loop is depicted in Figure 24. The information flows are described below:

(1) Required internal state of “Plant”: required point state “LEFT”,
(2) Required external state of “Plant”: required PM state “DRIVE LEFT”,
(3) Actual external input state of plant: Connecting voltage for moving the PM to the left,
(4) Actual external output state of plant: current flow via the PM position sensor contacts,
(5) Sensed external output state of plant: sensed state “UNLOCKED”,
(6) Estimated internal state of plant: estimated point state “RIGHT” or “TRANSITION”.

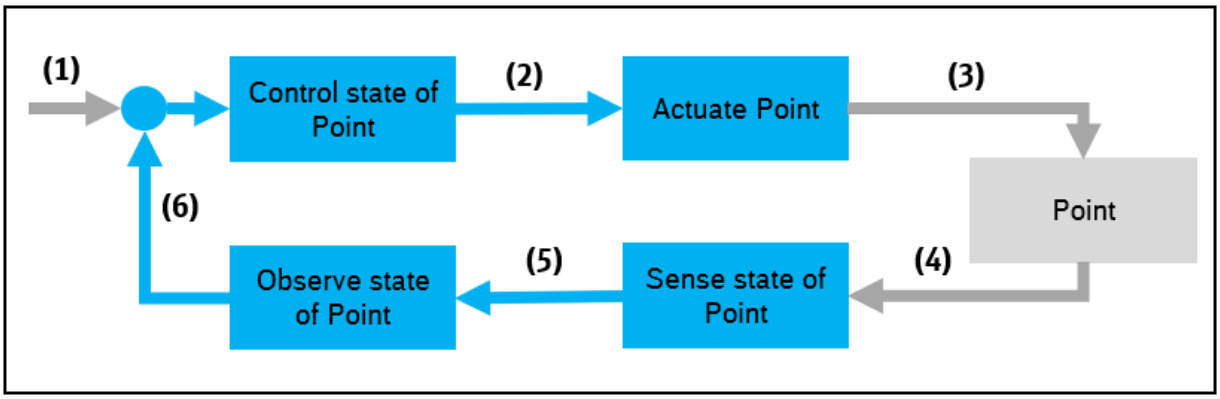

Figure 24 Control loop of a point

7.1.3 Interpretation of the concept of “function”

As explained in chapter 6.3.1, the Arcadia method [7] forms the basis for the creation of the RCA analysis model. The elements of ARCADIA’s “modelling language” (known as ArcML, which is not a modelling language, but rather an ontology) are defined in an experimental French standard XP Z 67-140 [14]. This standard defines a function as action, operation or service fulfilled by the system, or by an actor interacting with the system. When modelled in ontological terms, as depicted in Figure 25, this results in the following visualisation, immediately highlighting the broad nature of the definition.

The key difference between actions/operations and services (in common interpretation) is that operations have a finite execution cycle (they are called, executed, and return a value) whereas services are persistent (they are available over a longer-term timescale and can be seen as being continuously available). According to the RCA specification approach (see chapter 6.3.1) a function is interpreted in a persistent sense. It is available in a range of system states and is a container for a mathematical behavioural specification. The behavioural specification describes a mathematical transformation of inputs into outputs and holds for all possible values of input and output parameters defined in the information objects.

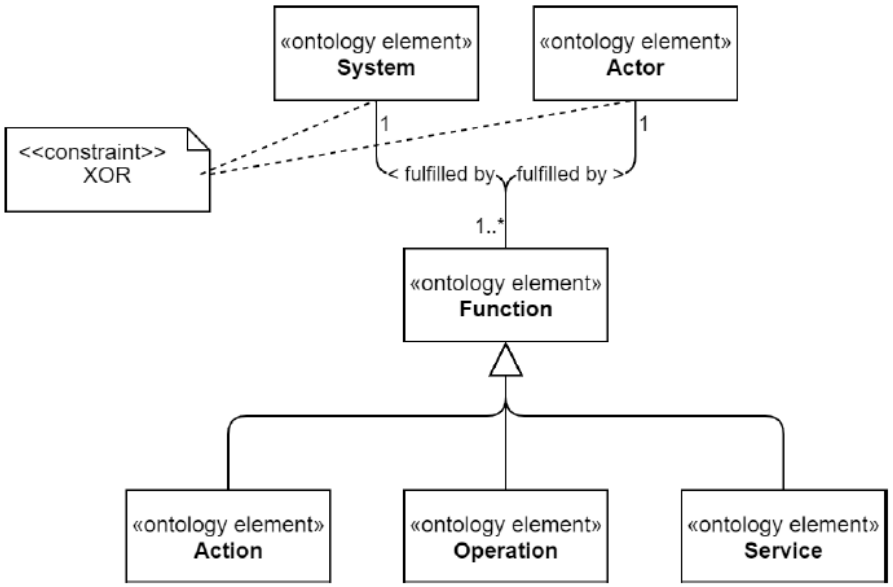

Figure 25 Ontology model of a function according to ACARDIA (source: DB Netz AG, I.NAT 1)

According to the EULYNX specification approach (see chapter 6.3.2), a function is represented by a Functional Entity (FE). A FE encapsulates a subset of the functional requirements of a EULYNX system element in the form of a function module. It delimits the function module from its environment and defines the inputs and outputs. FEs are used for the specification of EULYNX system elements as well as for the specification of EULYNX interfaces. The behaviour of FEs is currently defined in EULYNX by SysML state machines.

The principal structure of a FE is shown in Figure 26. Apart from state machines, FEs may have:

- SysML block properties (3).
- SysML block operations (2).
- SysML proxy ports used as atomic "in ports" and "out ports" (5) or typed with an interface block in which the information objects to be exchanged via the port are defined (4).
- SysML flow ports used as atomic "in ports" and "out ports".

The description of a FE (1) contains the stereotypes <<block>> and <<functional entity>> as well as the FE name (e.g., S_W).

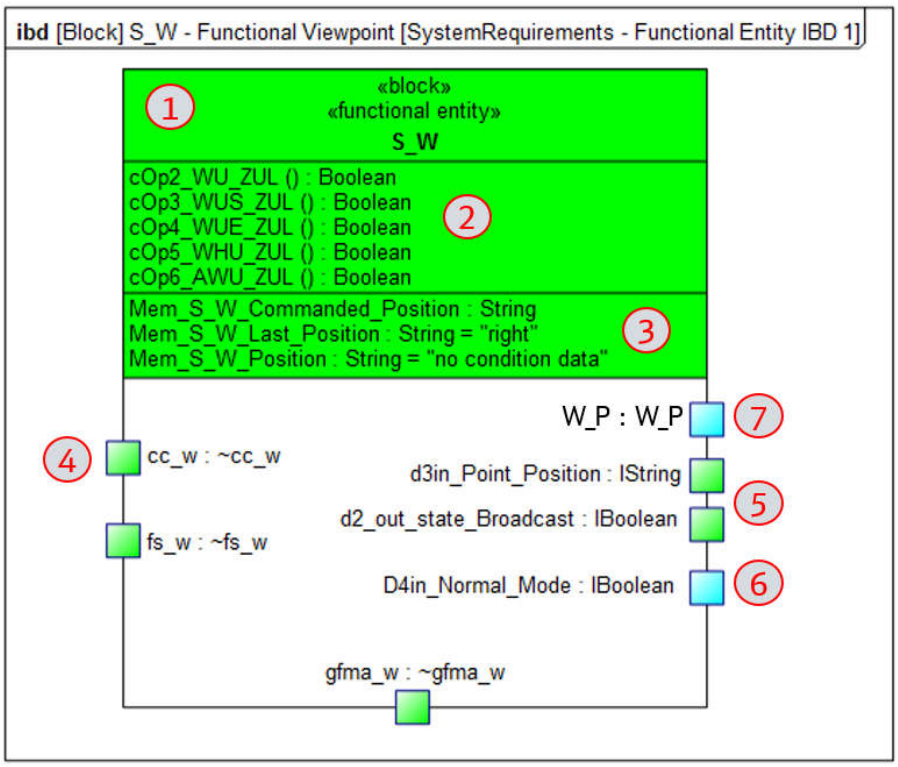

Figure 26 Principal structure of a Functional Entity

Block properties (3) are to be interpreted in the sense of variables or constants that store values. They are prefixed with "Mem". Block operations (2) are used to specify internal broadcast events or algorithms of data transformations (call behaviour). Call behaviour is invoked on demand, executed and terminated after execution. It is supposed to define event-driven data transformations. The algorithm of the data transformations is described in the body of the corresponding block operation using the Atego Structured Action Language [15].

A FE has interfaces that define continuous in-flow of information consumed by the assigned state machine, represented by data in ports, and continuous out-flow of information generated by the assigned state machine, represented by data out ports. Data in ports and data out ports (5, 6) are specified as SysML proxy ports or SysML flow ports of the SysML block representing the FE depicted in an internal block diagram (ibd). Data ports start with a capital letter if they are part of an external connection between a FE and the system environment (system interface) or if it is an open port (such as D4in_Normal_Mode). In this case, they have the colour blue (6). Data ports are especially suited to indicate permanently available information. The value of a D-port only changes if it is explicitly changed. Data in ports are used as arguments of Boolean expressions in change events or transition guards. They may represent arguments in data transformations or other data, that need to be permanently reachable by the behaviour of a FE. Their values can be permanently regarded as valid. Data out ports are used to provide continuous data created within a FE for its environment (e.g. to be available for adjacent FEs, reachable via their data in ports).

In addition to the continuous interfaces defined by data ports, a FE features interfaces that define event-driven in-flow of information consumed by the assigned state machine and event-driven out-flow of information generated by the assigned state machine. The information flows are represented by SysML proxy ports typed with SysML interface blocks (4, 7). The information objects to be exchanged are represented by signals. The interface blocks define the receptions for these signals. When a signal is received, a signal event is triggered by the corresponding reception, which is then used as a trigger for a state transition, for example. Ports and their interface blocks are written in small letters (such as cc_w:~cc_w) if they are part of an internal connection between two Functional Entities. In this case, they have the colour green like the corresponding Functional Entity (4). Ports and their interface blocks are written in capital letters if they are part of an external connection (system interface) between a functional entity and the system environment (such as W_P:W_P). In this case, they have the colour blue (7). An information object defined as outgoing in the interface block (port type) becomes an incoming information object through conjugation. This conjugation is indicated by the character "~" preceding the corresponding interface block (example: cc_w : ~cc_w).

To describe the externally visible behaviour of a system element, the FEs are interconnected in the form of a Functional Architecture (FA). The principal structure of a FA is shown in Figure 27.

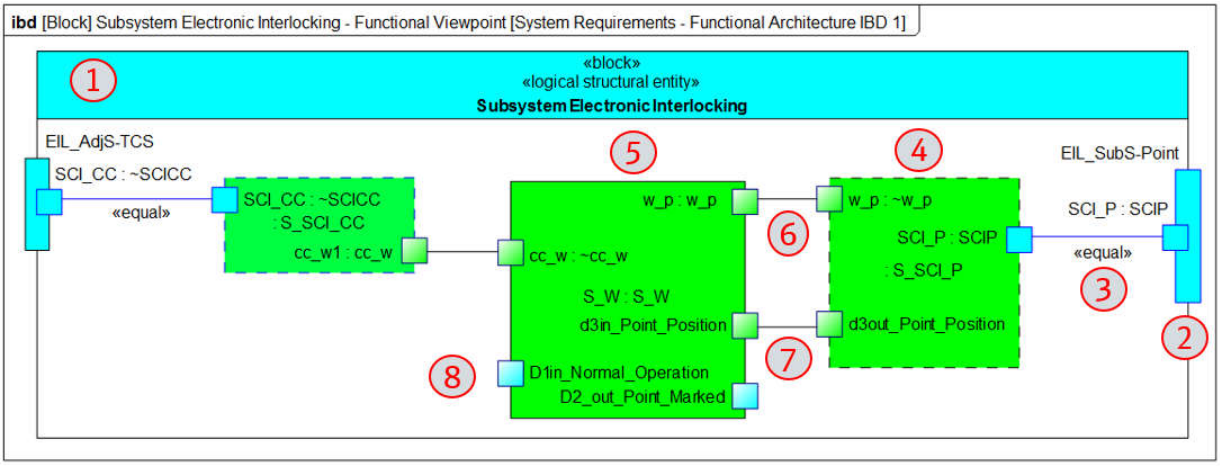

Figure 27 Principal structure of a Functional Architecture

There are two different representations of the FEs used: FEs with a solid border (5) and FEs with a dashed border (4). Following the interface centric specification paradigm (see Section 6.2.3), a solid-bordered FE represents the directly specified behaviour of the system element that is the "linking behaviour" (e.g., S_W). It is an inseparable part of the behavioural model of the system element. FEs with dashed borders, on the other hand, are references (reference properties) to the interface protocols specified in the models of the application levels. These local behaviours are linked to the overall behaviour of the system element by the directly specified linking behaviour. In Figure 27, for example, the functional entity "S_SCI_P" (4) is shown as a dashed block. This means that it is the local behaviour of the SCI-P protocol at the application level, which is defined in the SCI-P specification.

FEs can be connected in three different ways: by internal FE coupling, by external FE coupling or via open ports to behavioural parts that are not explicitly specified. The information flows defined in the internal FE-couplings or the couplings themselves are to be interpreted as descriptive elements of the behaviour and are only binding in the context of the overall behaviour. That means that an information flow defined in an internal FE coupling only becomes a mandatory requirement in the context of its active use, e.g., in a transition. Internal FE-couplings are implemented in two types. In the first type (6), communication between two FEs takes place by means of signals and in the second type (7), permanent information is transmitted. An internal FE coupling according to variant 1 defines an event-driven flow. It consists of two SysML proxy ports with the same name that are connected via a connector (SysML Connector). The connector represents the communication channel over which the information objects defined in the port type (SysML interface block) can be exchanged. The information objects are represented by SysML signals. The port type is used conjugated on one side (e.g., ~w_p). This means that an information object defined as outgoing in the interface block (port type) becomes an incoming information object through conjugation. An internal FE-coupling according to variant 2 defines a continuous flow. It consists of two SysML proxy ports or alternatively SysML flow ports with the same name that are connected via a connector (SysML Connector). The continuity of the information transmission is indicated by the abbreviation "D = Data" at the beginning of the names of the ports involved. The overall behaviour to be implemented by the manufacturers is connected to the interfaces of the system element via external FE-couplings. In contrast to the internal FE-couplings, the information objects defined in the information flows or the couplings themselves represent binding requirements (mandatory requirements). An external FE coupling consists of a proxy port representing an interface (2), located at the outer boundary of the system element, and labelled with the name of the interface concerned (e.g., EIL_SubS-Point). The port delegated from the FE relevant to the interface (e.g., SCI_P: ~SCIP) is embedded in it. The delegated port and the original port are linked (3) via a delegation relationship (stereotype <<equal>>). In other words, the port at the FE is moved to the outer boundary of the system element. The information flows and their direction remain unchanged. The name of the delimited port designates the kind of interface (e.g., SCI_P). The port type (e.g., SCIP) defines the information objects that must be exchanged via the respective interface. Open ports represent requirements and define the interface to specification parts not contained in the model, i.e., expected behaviour in the environment of the FEs. This behaviour can be implemented proprietarily by each manufacturer if the information expected at the ports is provided or the information delivered via the ports is processed accordingly. Open ports are also used to configure the specified behaviour.

7.1.4 Classification of systems according to their behaviour

The control system functions "control", "actuate", "sense" and "observe" introduced in Section 7.1.2.6 as well as the plant in the control loop have a behaviour that can vary depending on the type of system. These possible system types are listed below and briefly described according to the properties "dynamics", "driving type", "randomness", "type of variables and linearity" and "time" [16].

Dynamics:

- **Static system**: A static system is a system in which the output depends only on the present value of the input[1].
- **Dynamic system**: A dynamic system is a system in which the output depends on the past or future values of the input[2]. It has an internal state that evolves over time and that determines the output.

Driving type:

- **Time-varying system**: A time-varying system is a system in which the output is dependent on time (see also Section 7.1.2.1).
- **Time-invariant system**: A time-invariant system is a system in which the output is independent of time (see also Section 7.1.2.1). It is common to assume that a dynamic system is time-invariant [16].
- **Time-driven system**: A time-driven system is a system in which the state changes depending on a uniformly progressing physical time variable (see also Section 7.1.2.3).
- **Event-driven system**: (Discrete Event Dynamic Systems or DEDS) An event-driven system is a system in which the state changes depending on asynchronous discrete events. The state cannot change between two events (see also Section 7.1.2.3).
- **Fully driven system**: A fully driven system might contain elements of time-invariant, time-variant, time-driven and event-driven systems.

Randomness:

- **Deterministic system**: A system is deterministic if its output variables are all completely determined by the input and system state. It is a system in which no randomness is involved in the development of future system states[3].
- **Stochastic system**: A stochastic system is a system in which randomness (probability distributions) is involved in the development of future system states[4].

Type of variables and linearity:

- **Finite**: The variables of a finite system are bound by a lower and upper limit.
- **Discrete**: In a discrete system, values of variables or the time might be discrete.
- **Continuous linear**: A continuous linear system is a system in which the state variables are real or complex and the transition function is linear (see also Section 7.1.2.2).

[1] https://electricalworkbook.com/static-and-dynamic-systems-theory-solved-examples/

[2] https://electricalworkbook.com/static-and-dynamic-systems-theory-solved-examples/

[3] https://en.wikipedia.org/wiki/Deterministic_system

[4] https://de.wikipedia.org/wiki/Stochastisches_System

- **Continuous nonlinear**: A continuous nonlinear system is a system in which the state variables are real or complex and the transition function is nonlinear (see also Section 7.1.2.2).
- **Hybrid**: A hybrid system might contain elements of discrete, continuous linear and continuous nonlinear systems.

Time:

- **Continuous-time system**: In a continuous-time system the domain time is considered continuous (see also Section 7.1.2.3).
- **Discrete-time system**: In a discrete-time system the domain time is only present at discrete times (see also Section 7.1.2.3).

In Table 1 System types and their probability of occurrence in control system functions, the system types, i.e., the corresponding typical behaviour, are assigned to the control system functions considering the probability of their occurrence. The probability is expressed by the white to completely black filled circles. The more a circle is filled with black, the higher the probability. For example, a circle half-filled with black symbolises a probability of 50 per cent and a filled circle of 100 per cent. The allocation of probabilities is based on the following assumptions:

(1) Railway systems of interest can function as transformation systems. Transformation systems, such as decision-making systems (artificial intelligence), have the characteristics of static systems. Furthermore, in many cases the function "indicate", a specialisation of the function "actuate" (see Section 7.1.2.6), is a transformation function with the character of a static function.
(2) Railway systems of interest have in most cases the characteristics of dynamic systems. However, the functions "sense" and "observe" often have a transformational character. They represent a transfer function from the outer state to the inner state of the plant without having a history.
(3) The function “control” might be time-varying, e.g., day and night switching of the luminosity of a light signal. The functions “actuate”, “sense” and “observe” are not assumed to have a high probability of being time-varying (see also (4)).
(4) According to [16] it is common to assume that a dynamic system is time invariant.
(5) Current railway systems of interest (e.g., EULYNX systems) are considered event-driven, not time driven. Nevertheless, future systems may also exhibit time-driven behaviour. Since no concrete examples are available yet, this is only assumed for the time being.
(6) Current railway systems of interest (e.g., EULYNX systems) are considered event-driven. It is assumed that this will also apply to most future systems. Since no concrete examples are available yet, this is only assumed for the time being.
(7) It is assumed that future railway systems of interest will predominantly show event-driven behaviour but may also partly (especially in the "control" function) show time-driven behaviour concurrently, i.e., they will be fully driven. Since no concrete examples are available yet, this is only assumed for the time being.
(8) Railway systems of interest are safety-relevant systems and therefore behave deterministically.
(9) Sensors might include jitter or drift, random errors, but the rest of the system behaviour should be deterministic (see 8).

(10)Current railway systems of interest (e.g., EULYNX systems) are considered finite. It is assumed that this will also apply to future systems.

(11)Current railway systems of interest (e.g., EULYNX systems) are considered discrete. It is assumed that this will also apply to a large extent to future systems.

(12)It is assumed that future railway systems of interest will also partly exhibit continuous linear behaviour. Concrete examples are currently not available.

(13)It is assumed that future railway systems of interest will also partly exhibit continuous nonlinear behaviour. Concrete examples are currently not available.

(14)It is assumed that future railway systems of interest will also have partially simultaneous continuous linear, continuous nonlinear as well as discrete behaviour. Concrete examples are currently not available.

(15)Future railway systems of interest are predominantly computer-based. Computer-based systems have discrete-time behaviour. Theoretically, however, they can also consist partly or entirely of electrical components. It is therefore assumed that to a certain extent the behaviour of continuous-time systems must also be considered.

(16)Future railway systems of interest are predominantly computer-based. Computer-based systems have discrete-time behaviour.

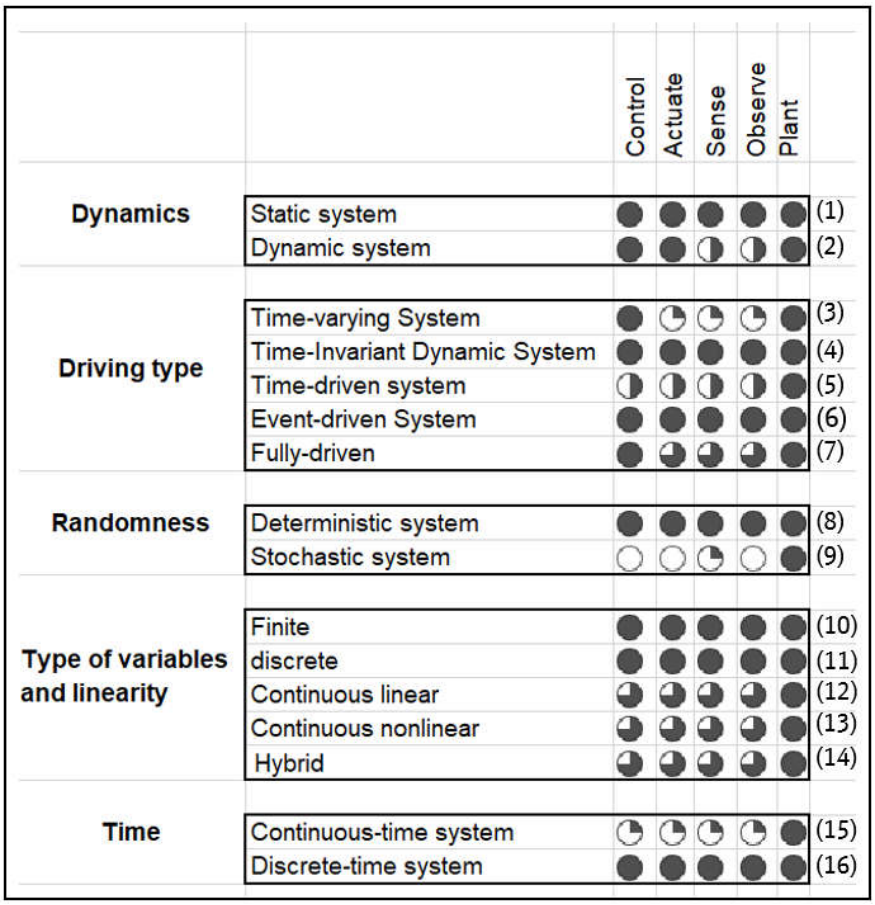

Table 1 System types and their probability of occurrence in control system functions

To get a complete description of the system, not only the system element of interest needs to be modeled, but also the environment it interacts with. In control theory, this is referred to as the plant model. It is assumed that the plant can exhibit all the types of behaviour listed [17].

7.2 Evaluation of the current EULYNX specification approach

In the following sub-chapters, requirements for an extended EULYNX specification approach are derived from the analysis results achieved in Section 7.1. Furthermore, it is evaluated to what extent and how these requirements are already fulfilled by the existing EULYNX specification approach.

7.2.1 Analysis results and resulting requirements for a specification approach

According to the results of the analysis carried out in Section 7.1 regarding the characteristics of railway systems in the context of a future railway system architecture, such as RCA, their system elements have the characteristics of control systems. Since the analysis model of RCA is currently being developed and there is not yet any in-depth knowledge regarding the behaviour of the corresponding system elements, the classification of the control systems is largely based on assumptions. As can be seen in Table 1 System types and their probability of occurrence in control system functions, the system elements predominantly have the characteristics of discrete, event-driven, deterministic finite dynamic systems. Besides event-driven characteristics, discrete-time behaviour and continuous time behaviour will also occur to a certain extent and must therefore be considered in an enhanced EULYNX specification approach. Likewise, the different behaviours can occur in mixed form in one system element (fully driven, hybrid). To what extent these behaviours really play a role can only be determined when the RCA analysis model is closer to completion. The situation is similar regarding time-varying and time invariant behaviour. Here, a dominance of the time invariant behaviour is assumed. According to [16] it is common to assume that a dynamic system is time invariant. However, it must be possible to represent time-variant behaviour in the specification model. Besides the characteristics of dynamic systems, those of static systems play an important role. Static system elements or functions are needed for transformations. In contrast, stochastic and infinite aspects play a subordinate role. A further developed EULYNX specification approach shall therefore support the semiformal/formal description of the following behaviour:

- Finite discrete event dynamic behaviour.
- Discrete time behaviour.
- Continuous time behaviour (linear, nonlinear and hybrid).
- Combination of continuous time behaviour, discrete time behaviour and finite discrete event dynamic behaviour.
- Static behaviour (logic and algebraic data transformation).

It must be possible to describe all behaviours in an executable way and the corresponding behaviour model must allow a transformation into a formal model according to the approach introduced in [5] and as further developed in WP10's ongoing Task 10.8. The knowledge gained

and documented in [5] regarding the improvements of the models created in the EULYNX initiative shall be considered. Furthermore, the specification approach must make it possible to describe the functionality of system elements in the form of control loops as shown in chapter 7.1.1. It must also be possible to describe the control system functions in different configurations, as is inevitable, for example, in the specification of a system element representing an electronic interlocking.

7.2.2 Assessment of the existing EULYNX specification approach

7.2.2.1 Functional control system architecture

The existing EULYNX specification approach already allows the description of functional control system architectures and their governing control loops, as introduced in Section 7.1, through the "Functional Architecture" model view of AM MBSE (see Section 6.3.2.2). The control system functions are represented by interconnected functional entities. The interconnection of the functional entities is done considering the interface-centric specification approach presented in Section 6.2.3. The model view “Functional Architecture” also makes it possible to link infrastructure-related data with the control system functions. For this purpose, the infrastructure elements are abstracted as functional entities, mapped in a functional architecture, and interconnected in a topology-compliant manner. The principle is shown in Figure 28. However, it is difficult to describe different configurations of functional architectures. This is possible with a small defined number of configurations by corresponding functional architectures. Problems arise with an arbitrary number of configurations. For example, routes consist of a varying number of different route elements. It is a challenge to describe all possible configurations generically using functional architectures.

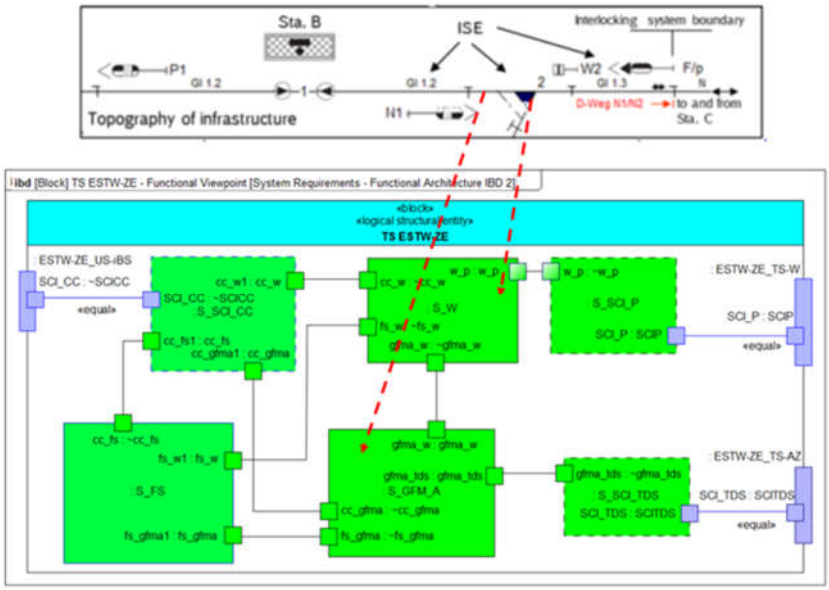

Figure 28 Topological abstraction of infrastructure in the form of a functional architecture

The control system functions "actuate/indicate" and "sense" are in some cases technology related. The project is required to include these technical aspects in the specification. The functional entities are therefore to be supplemented by technical functional entities (TFE). A technical functional entity (yellow-coloured SysML block stereotyped with <<technical functional entity>>) represents a certain piece of technology-dependent behaviour based on technical requirements. It is part of a technical functional architecture supplementing the technology-independent behaviour defined by functional entities. The definition of technical functional architectures is provided in the EULYNX specification approach in the technical viewpoint of the AM MBSE. However, the model views of the technical viewpoint are not yet fully defined in the current version of the AM MBSE. For this reason, the description of technical functional architectures and thus the integration of technical functional entities are not yet possible. Another important point to note is that the description of the environment of a system element of interest, i.e., the plant is not considered in EULYNX according to the current specification approach. However, to perform a complete description of a system, it is also necessary to consider the environment of a system element of interest with which it interacts in the specification [17].

7.2.2.2 Behaviour

According to the EULYNX specification approach, a control system function may be represented by a Functional Entity. Functional entities have SysML state machines and SysML block operations to describe behaviour (see Section 7.1.3). SysML state machines enable the specification of finite discrete event dynamic behaviour. SysML block operations are used to perform logical or algebraic transformations. The corresponding algorithms are defined in the operation bodies using the action language ASAL [15]. Block operations are currently used as call operations. This means that they have a finite execution cycle (they are called, for example during state transitions, executed, and return a value). The description of further types of behaviour is currently not possible. The EULYNX specification approach shall be extended to include the types of behaviour defined in Section 7.2.1.

7.2.2.3 Data types

The current EULYNX specification approach uses the following data types: Boolean, DateTime, Decimal, Double, Integer, Long, Single, String and Enumeration. The data types are to be supplemented according to the requirements of the behavioural extensions.

7.2.2.4 Validation of system requirements

As shown in Section 6.3.2.6, the EULYNX specifications are created executable, i.e. in the form of a virtual prototype that can be validated by simulation (simulation-based testing).

7.2.2.5 Verification of safety properties applying formal methods

The EULYNX initiative specifies system elements with standardised interfaces. The functional requirements, i.e., the behaviour of the respective system element is described by the stimuli-response behaviour visible at the interfaces. Since this behaviour is standardised and must be implemented by the manufacturers exactly as defined, it must be ensured that all safety

requirements are considered in the specification model. This is done according to the current EULYNX MBSE process by simulation-based testing of the specification model. Since freedom from errors cannot be established with 100% certainty through testing, it is advisable to formally prove this freedom from errors using formal methods. The applicability of formal methods for EULYNX specifications was evaluated in [5]. The principle is briefly explained in chapter 6.1.4. The approach developed in [5], in which the SysML specification model is manually converted into a formal model, will become automated in WP10's Task 10.8.

7.2.2.6 Automated test case generation from system requirements

In the current EULYNX specification approach, test cases are derived manually from the system requirements. The aim is to generate test cases automatically from the specification model. Approaches to this have been evaluated in TD2.7. They are documented in [3] and [5] and are currently under development (see Section 7.5).

7.3 Proposed additions to the EULYNX specification approach

In X2Rail-2 Deliverable D5.5 [5] several suggestions for improvements to the existing EULYNX specification approach have already been made (which are not repeated in this deliverable). These improvements have been forwarded to the EULYNX initiative where they will be incorporated into the EULYNX modelling standard (Eu.Doc.30) [6]. The enhancements proposed in this document relate mainly to the extension of the finite discrete event dynamic behaviour of functional entities by the types of behaviour listed in Section 7.1.1. The necessary improvements mentioned in the previous chapters, such as the configuration of functional architectures or the addition of model views of the technical viewpoint, were communicated to the EULYNX initiative and worked on further there.

7.3.1 Modelling of continuous-time and discrete-time dynamic behaviour

When modelling a dynamic function, it is possible at any time to create a representation in the generalised state space form. This form of representation is based on three basic equations:

1. Continuous dynamic states: The first derivative of the state vector is a function of state, inputs, and time.
2. Discrete dynamic states: The next value of the state is based on the current values for state, inputs, and time.
3. Outputs: The output value is a function of states, inputs, and time. No dynamic developments should be included in this equation.

Figure 29 shows a general representation of a dynamic function (or system). In the EULYNX specification approach, this construct is represented by a functional entity.

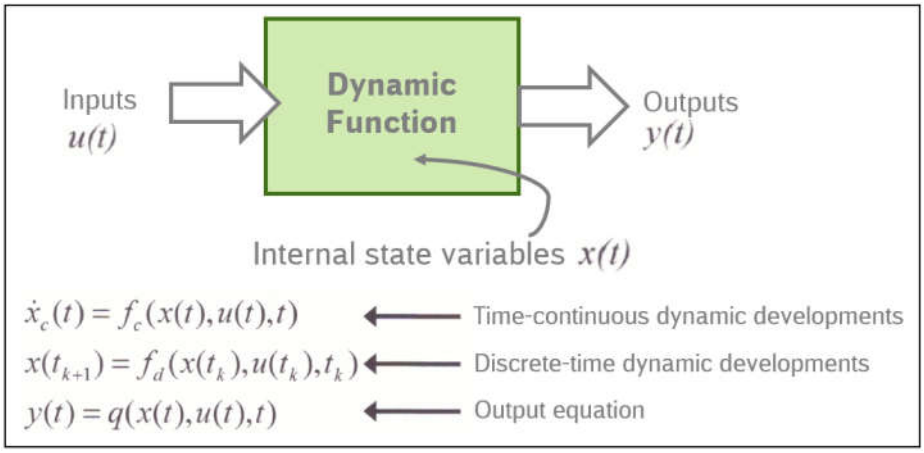

Figure 29 General dynamic function

A system, system function or plant model can be described at different levels of abstraction, see Figure 30. These abstraction levels are adapted from [18]:

- **Acausal models.** At the highest level of abstraction, there are acausal models, where models are described by one or several differential or algebraic equations, possibly combined with state machines to model hybrid systems. This approach is referred to as first principles modelling.
- **Causal models.** At the causal level of abstraction, it is defined what is input, and what is output inside the system or function, and between components in a composed system. Typical causal models include block diagrams, like continuous Simulink models [20]. The model view “Functional Architecture” of the EULYNX AM MBSE can also be arranged on the abstraction level "causal models".
- **Time-discretised models.** To solve a differential equation numerically it is typically discretised in time. A discretised model is an algorithmic representation in the sense that it generates a defined output for a certain input and internal state. A model can be discretised in different ways, e.g., using forward/backward Euler.
- **Simulation behaviour.** To perform the calculations of the discretised models, a solver and a scheduler are needed as part of the simulation engine. The simulation engine can decide the time-step, execution order, triggering, communication, etc. of the model. A typical numerical tool to solve differential equations is Simulink.

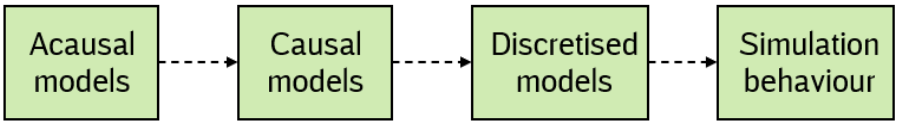

Figure 30 Different levels of abstractions of system or plant models

Acausal models are generally more flexible and reusable than models at a lower abstraction level [19]. For this reason, acausal models are well suited for the representation of binding requirements. Acausal models can be expressed in SysML by parametric diagrams. Parametric diagrams

describe constraints between variables, like equations, and how they are related to each other. The constraints are acausal, and by combining many functional entities within a functional architecture, acausal relationships for a large system element can be achieved. In addition, finite discrete event dynamic behaviour represented by state machines of connected functional entities can tell which equations to be used in the parametric diagrams, to be able to describe hybrid system elements. Figure 31 shows an example of the application of the SysML parametric diagram that defines the addition of two real numbers. This is a very simple example, but this works also for more complex ones, of course. A detailed description of the use of SysML Parametrics and more complex examples can be found in the SysML specification [10].

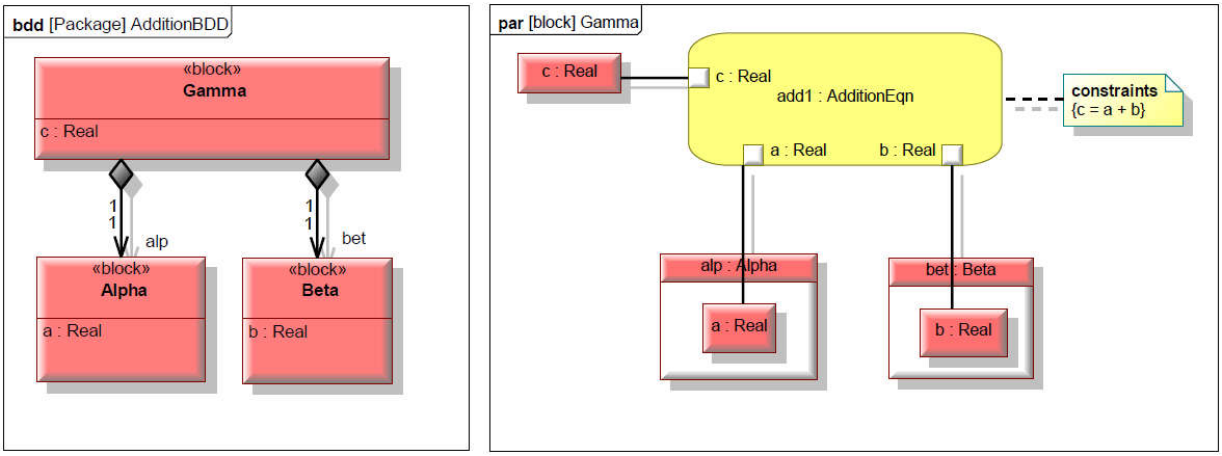

Figure 31 Simple example of the application of SysML Parametrics

Proceeding to lower abstraction levels means that the model gets more sophisticated, in the sense that the model can produce simulation results. The integration for MATLAB Simulink of the modelling and simulation tool "Windchill Modeler" used in EULYNX (see Section 6.3.2.5) enables the creation of a causal model in the form of a Simulink block diagram from an acausal Windchill Modeler parametric diagram. Through Simulink, the mathematical model can be developed further to model the constraints so that the performance, reliability, and correctness of the algorithm can be tested by simulation. After making changes to the Simulink model, its associated parametric diagram can be updated with those changes. Integration for MATLAB Simulink also enables the creation of a Windchill Modeler parametric diagram from a Simulink block diagram. It can export the following items from a parametric diagram - constraint properties, constraint parameters, value properties and connectors. In this way, the functional architecture of a system element can be extended to include continuous-time and discrete-time dynamic behaviour. It is proposed to extend the current EULYNX specification approach accordingly.

7.3.2 Modelling of static behaviour

Static behaviour is characterised by the fact that the output only depends on the value of the current input and does not require any history. In many cases, decision processes are described, and data transformations are performed based on a corresponding algorithm. Both tasks can be fulfilled by the means available in the existing EULYNX specification approach. Decision

processes can, for example, be modelled by flowcharts or defined in persistent operations (time advance operations). Time advance operations are assigned to a functional entity and permanently generate an output depending on the current input. They can also be used for data transformations. A flowchart is a chain of logical patterns used to implement a sequence of decision-making processes. It is used to implement sequential, nested, and iterative processes. In the current EULYNX specification approach, flowcharts are modelled by pseudo-states and transitions on SysML state charts. In future, it should also be possible to define complex algorithms using parametric diagrams and then transfer them to Simulink block diagrams as described in Section 7.3.1. It is also proposed to extend the EULYNX specification approach to include truth tables. Truth tables map actions to the possible Boolean combinations that can be derived from a set of conditions. In this way, the description of complex logical processes is facilitated, and readability is increased.

7.4 Expression of behaviour as mandatory requirements

7.4.1 Current approach

According to the EULYNY specification approach the SysML specification model is stored in the repository of the modelling tool windchill modeler. Specification-relevant model elements are mapped in the requirements management tool IBM DOORS as DOORS objects, which represent atomically referenceable system element requirements or interface requirements. The following types of requirements are defined:

- "Req": denotes a mandatory requirement.
- "Info": denotes additional information to help understand the specification and does not specify any additional requirements.
- "Head": denotes chapter headings.

A requirement consists of the respective SysML model element, e.g., a SysML diagram, and, if applicable, an additional extension of the same. For this reason, requirements have two attributes "Requirement Part 1" and "Requirement Part 2", which are shown in adjacent columns (see Figure 32). In "Requirement Part 1" the respective SysML model element is listed and in "Requirement Part 2" the corresponding extension is shown. Column 'Type' defines the type of the requirement and applies normally both to "Requirement Part 1" and "Requirement Part 2". In the case of requirements with a type "Req", in which the "Requirement Part 2" is provided with the heading "Info", the defined type "Req" only applies to "Requirement Part 1".

ID	Type	Requirement Part 1	Requirement Part 2
Eu.LS.4687	Req	Cd_Indicate_Signal_Aspect	Command (Cd) from the Subsystem - Electronic Interlocking to the Subsystem - Light Signal to indicate the transmitted Signal Aspect.

Figure 32 Structure of a requirement

Functional requirements are represented in the EULYNX specification approach by finite discrete event dynamic behaviour defined in the form of SysML state machines. The behaviour of a system element or the process data interface protocol (PDI) of a communication interface specified in the functional entities and interconnected in a functional architecture must be implemented in its entirety by the manufacturers. Of course, the specification model can be transformed into a manufacturer-specific model (model-to-model transformation). However, the respective manufacturer must prove the semantic equivalence of its model to the specification model. Since such a procedure is currently only rarely used, documentation of the behaviour in the form of referenceable atomic requirements is expected.

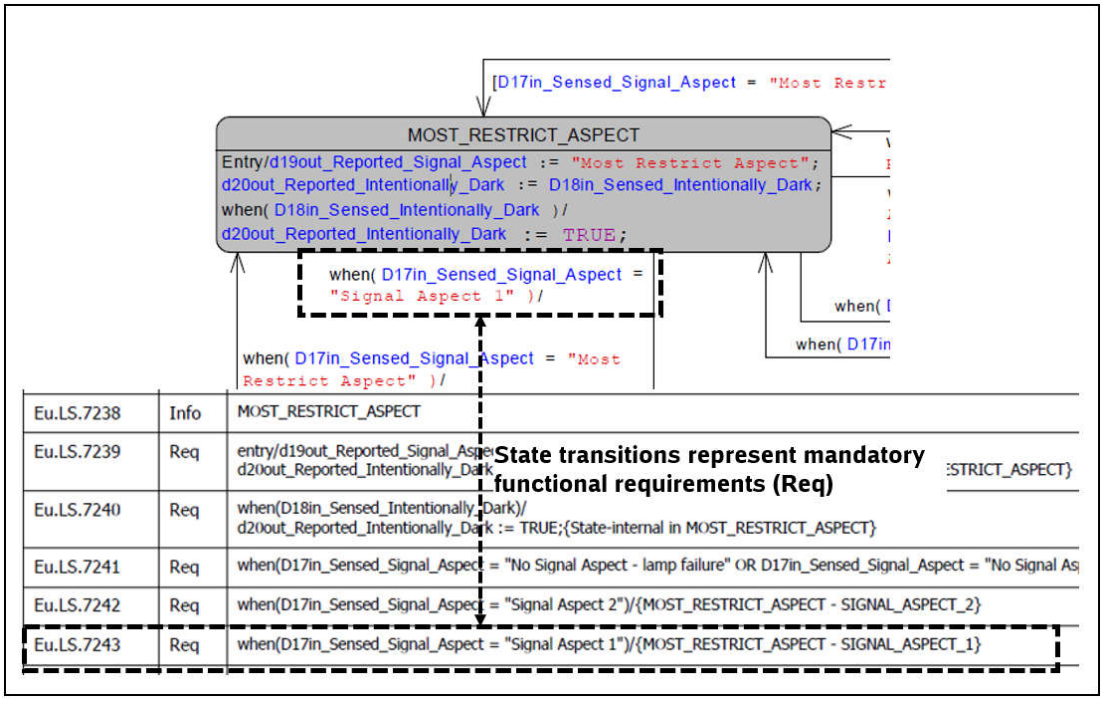

Figure 33 Representation of atomic functional requirements

According to the EULYNX specification approach, functional requirements with the degree of bindingness "Req" are represented by the state transitions of the behaviour model. As shown in Figure 33, they are listed in the specification document in addition to the state machine mapping in atomic referenceable form. For example, the state transition "when (D17in_Sensed_Signal_Aspect = "Signal Aspect 1")/" is formulated as an atomic functional requirement in the form "when (D17in_Sensed_Signal_Aspect = "Signal Aspect 1")/ {MOST_RESTRICT_ASPECT-SIGNAL_ASPECT_1}".

7.4.2 Advanced approach

Algorithms are defined in call operations, which are then used in state transitions. The same applies to attributes represented by SysML block properties. These are first assigned a default value before they are further processed in state transitions or call operations. At present, these,

and other model elements (such as ports) are marked with the requirement type "Req". As a result, manufacturers must separately prove that these model elements are fulfilled by their implementation, although proof is only necessary when they are used, for example in a state transition. Since these model elements are basically definitions, it is recommended to introduce the requirement type "Def" and to carry out the labelling accordingly in the future. In contrast, persistent operations (time advance operations), which are assigned to a functional entity and describe, for example, decision-making processes or data transformations (static behaviour), are to be marked with the requirement type "Req". In the case of behaviour described in the form of differential equations or algebraic equations using parametric diagrams (continuous-time and discrete-time dynamic behaviour), these equations represent the binding functional requirements, which are therefore to be marked with the requirement type ”Req".

7.5 Supporting automated test case generation

7.5.1 Motivation

Model-based testing in relation to other verification means

Ideally, all intended properties of a signalling system would be verified exhaustively using formal methods. However, there exist several reasons why this is usually not possible in reality: properties may not be defined explicitly, not lend themselves well to formalization, be hard to prove or very different in nature (requiring different formalism). Testing is not exhaustive, but very flexibly applicable; moreover, it involves an actual environment in which the system is executed, which adds some implicit system validation to the original verification task. But testing can consume many resources, even if test execution is largely automated, and classical testing requires the product to exist and thus find errors at a late development stage.

Model-based testing can be considered “in between” formal methods and traditional testing: it is still non-exhaustive but helps towards more efficient test design and early testing, and formal methods are used by advanced test case generation (TCG) tools, guaranteeing that test cases cover the behaviour of the model they were generated from. In particular, model-based testing has proven useful for functional testing at higher test levels (integration, system, acceptance), but is not limited to that. Still, there will be tests (e.g. usability, smoke) and reviews that need to be designed and/or executed manually. But to reduce them to a minimum and be as efficient as possible, the general ideas behind this Section (7.5) are:

- To summarize why automatic test case generation should be used for signalling systems (remainder of Section 7.5.1);
- To explore the scope of test case generation in terms of different kinds of behaviour present in signalling systems (Section 7.5.2), in order to test “as much model-based as possible” using “as few different models as reasonable”;
- To provide recommendations for system specifications (in Section 7.5.3) on how to support TCG best.

General advantages of model-based testing

Automation and reuse/maintainability. Manual design of high-quality test suites takes time – on average it is responsible for more than half of all testing costs. In model-based testing, only the model needs to be created manually, afterwards test cases are generated automatically from the model. While coming up with a good model initially may require some time as well, the approach becomes much more efficient than manual test design when changes need to be made for maintenance or reuse of the test suite: usually changing the model and regenerating the test cases is easy to do.

High test suite quality and structured test design process. The process of model creation and test case generation can be structured very well into smaller steps (e.g., structural, and behavioural modelling, system and environment modelling). Also, many – partly automated – V&V steps can be used during the process (e.g., automated checks of the model, interactive model simulation, coverage analysis of generated test cases). This usually leads to a high-quality model, from which a high-quality consistent test suite is created by using fully automated, mature, and often formal methods-based generation.

Comprehensible and flexible test selection. TCG tool usually offers several settings that allow realizing different test selection criteria (e.g. coverage goals, search strategies, input interface constraints). This way, the generation can be flexibly influenced to balance the number and size of generated test cases as well as the generation time. Furthermore, the tools usually measure how much of the coverage goals are reached and allow tracing between model (including attached requirements) and test cases, making transparent and comprehensible what has been achieved by the TCG.

Support of early testing. Model-based test design can begin early during system development and it is possible to start with partial models and generate test suites from them. This in turn allows early V&V of the test suite. Reuse of models or model parts can accelerate test design very much. Another aspect is that the modelling and TCG process often leads to the early detection of specification issues. All in all, this leads to less cost and risks in projects.

Advantages of using models. It is commonly known that the use of models usually fosters precision and completeness, can serve to capture knowledge compactly, and can support communication (especially for visual models). Generating many different test cases from a model, the model also acts as a “single source of truth” in test design.

Fit with new signalling system specification approach

The EULYNX MBSE approach provides state machines that capture the possible behaviour of a signalling interface and/or subsystem (see Section 6.3.2.3). Their semantics is defined by the SysML standard which is textual and not complete as it intentionally leaves open how e.g., scheduling of event dispatching or choice among several enabled transitions will be implemented. Apart from that, SysML provides a quite precise semantics which allows tools – dealing with the open issues – to execute state machines. In this sense, the state machines can form the basis of a test model, from which test cases can be generated. As many TCG tools are based on state

machines, for them – besides automated generation of test cases from the model – also the model creation becomes very efficient.

Moreover, requirements coverage becomes particularly easy, as in the EULYNX specifications the statechart transitions are considered requirements. The TCG can produce test cases according to the goal of transition coverage, and tracing requirements forth and back between specification and test cases becomes very easy because of (a) the similarity between the state machine-based specification and test models and (b) the coverage and traceability relation between test model and test cases established by the TCG.

Finally, the EULYNX specifications do not include the desired system properties explicitly as required for formal verification. Testing does not require that; it can be based on an operational ("imperative", see Section 6.2.4) description of system behaviour. This can be an advantage because IMs traditionally think in operational procedures, which are often historically grown and not all their rationales may be known any longer. And even if the desired system properties are explicitly available and complete, IMs may decide not to make them publicly available.

7.5.2 TCG for different kinds of behaviour

Basic stimulus-response behaviour

Rail control systems usually are reactive software systems, sometimes as embedded systems such as field element controllers, sometimes running on computers such as in an electronic interlocking. As such, they are constantly running, waiting for discrete events to occur at their external interface, and processing them, which can lead to output at their external interface and/or to change of their internal state. Such basic stimulus-response behaviour can be modelled as state machines (see Section 6.2.2); and the state machines together with test selection criteria, e.g. transition coverage, can be used by a TCG tool to automatically produce test cases which e.g. cover all the transitions. This way, one can obtain effective tests for stimulus-response behaviour of rail control systems; this has been demonstrated in WP5 of the preceding X2Rail-2 project for a level crossing (LX) controller specified by the Swedish IM Trafikverket (see Chapters 7 and 10 of [3]).

Timed systems

In the simplest case, the stimuli for the behaviour just come from the system's environment such as adjacent systems, sensors, or human operators. But it is also common that signalling systems have timers, whose expiry serves as a stimulus. The above-mentioned LX controller is such a system, as the closing and opening procedures of the LX are very much time-dependent. The same holds for its environment, as e.g., raising and lowering of the controlled barriers takes time. This can be dealt with by state machine transitions in the system and environment model, respectively, that are triggered by timeout-events according to a global clock (i.e., timed automata), and by adding the possibility to advance that global clock for the TCG tool. It leads to test cases that respect the modelled time dependencies at least in the existence and order of test steps. This has been demonstrated in the LX controller example as well; however, no actual time intervals were generated into the test cases, although possible in principle as demonstrated as part of TCG

for the EULYNX point subsystem (see Chapter 7.5 of [5], although the time aspect of the TCG is not much discussed there).

Regarding time, in addition, one may wish to generate system performance test cases, to verify that the system will show the desired reaction within a limited time. More generally, it can make sense to define time limits as part of tests such as load or stress tests. The best way to generate such information into test cases depends on the TCG tool (test case elements that can be generated) as well as on the test environment (capabilities to interpret that information for test execution), but basically, the use of TCG does not limit testing of time-related properties. A corresponding case study is currently underway in WP10.

Non-determinism

Another kind of behaviour that may require additional care is non-determinism. Non-determinism in the system environment is quite common and is dealt with by the TCG tools that decide which inputs to choose to generate – usually deterministic – test cases. If the generation itself needs to be reproducible, techniques like pseudo-random numbers are used in generation tools and environment models to mimic non-determinism. Non-determinism of the system under test is often restricted to uncertainties of the output values or their timing; this can be accounted for with the definition of appropriate ranges to be used during the building of the test verdict. More seldom does one encounter genuine non-determinism of the system, which then needs to be considered by the test model and the test cases (which in general become trees instead of sequences as mentioned in [21]).

Complex algorithms

More centralized signalling systems tend to include also complex calculations or algorithms (that run without consumption of further stimuli). The application of model-based testing is clearly possible for such behaviour, but often is not in the focus: doing a model of the behaviour using classical model-based testing formalisms like state machines may only be worthwhile if a certain level of abstraction is reached, otherwise one may end up with an (expensive) second implementation. Instead, a detailed analysis of the behaviour may be done using specialized mathematical models or classical testing techniques for lower levels. Nevertheless, it should be considered that test models for a generation of higher-level tests may require some abstract model of algorithmic behaviour.

Communication

Advanced signalling systems with centralized control have always been distributed, rising the need for communication between an interlocking and its controlled field elements. For new digital signalling systems, this communication is based on computer network technology and is an essential part of standardized interface specifications, but also more complex than traditional communication means. To generate tests for the specified interaction on the application level, it is often necessary to model communication-related behaviour such as establishing and releasing a connection, sending, and receiving messages, and important properties of the communication protocols and channel. Classical modelling approaches used for TCG such as multi-statechart or communicating processes support asynchronous communication appropriate for use between

distributed systems, but also synchronous communication appropriate between logical components of the same system. This makes it possible for the TCG tools to generate test cases that are correct w.r.t. communication behaviour. In practice, it is advisable to establish a good understanding of the semantics of communication-related model constructs (e.g., circular synchronous calls between components) and how the test generator deals with asynchronous communication.

Data processing

Subsystems in modern signalling architectures like ETCS or EULYNX communicate by exchange of messages which contain data of interest; they also usually require configuration data and may need to store processed data. Such data needs to be decoded and encoded, aggregated and disaggregated according to the system's needs and standardized formats, i.e. read from memory, validated, possibly converted and saved to memory. That kind of behaviour can constitute a considerable percentage of the overall system behaviour but is rather simple in nature. Tests would need to check detailed rules describing relations between original and transformed data. This low degree of abstraction and often declarative specification seems less suited for extensive modelling in an – often executable – test generation model. Instead, it is usually a good idea to abstract from original data formats and simplify data exchanged at external and internal interfaces. Due to usually quite precise and standardized specifications, the use of formal means to check more detailed data processing rules appears well-suited. Nevertheless, TCG for data conversion behaviour is clearly possible and can make sense to some extent, e.g., if dedicated components for the aggregation/disaggregation of data (like in the RCA Object Aggregation layer, cf. Section 6.3.1) need to be modelled anyway. Finally, TCG models can contain some internal data processing for which no direct test cases are generated but which is part of some other kind of behaviour; this is normal and should not be confused with explicit testing of data processing.

Continuous systems

Embedded systems often exhibit real-time continuous behaviour due to physical processes (mechanical, electrical/electronic). Clearly, such systems are part of railway signalling but discretizing their state space for use in central control has a long tradition and is applied even earlier in the processing chains in modern systems as the software parts increase. If needed, continuous system dynamics can be modelled instead of discrete behaviour or in addition to it (hybrid systems), e.g., using hybrid automata. However, test generation involving continuous behaviour is hard and approaches are rare (see e.g., [22]).

7.5.3 Model-based system specification facilitating TCG

In general, TCG does not require a certain style of system specification; it is the task of the tester to analyse the specification, determine what tests need to be conducted and come up with a suitable test model. However, the specification style and quality may greatly influence how difficult and thus how costly and time-consuming that task will be. For this reason, in the sequel, several aspects that facilitate efficient TCG are discussed, and recommendations are provided. It is assumed that a system model based TCG approach is followed, which not only seems to be most

beneficial for testing CCS systems, but also is affected most by how the system is specified. This approach requires a behavioural system specification as input.

Conciseness

It is in general beneficial for verification to have a high degree of precision and completeness in the specification because what is not specified will not be verified. This does not mean that every system detail needs to be specified – there are good reasons for specifications to leave details open for choice during design and implementation – but rather that all relevant information should be available and should be unambiguous. Being precise and complete can also mean to include "non-behaviour" which can be turned into negative tests. Model-based testing increases the need for precision and completeness because modelling tends to enforce these properties. If the specification does not meet them, this will not only lead to questions to the specifier, but a missing understanding of the system can also lead to bad design decisions for the test model.

It is good practice for specifications to limit their variety: the number of terms, language constructs and diagram types used in the specification should not be unnecessarily big and similar facts should be expressed using similar ways. This serves in general the understandability of the specification; it is even more important for model-based testing, as it simplifies the mapping of the specification facts to appropriate model elements in the test model.

Closeness to test model paradigm

Often test models need to be executable for test case generation, so an imperative specification style is closer to what is needed than a declarative style. Most system model-based test generation tools use state-based models, especially UML/SysML state machines, to specify behaviour. Preferably, the specification already includes such models; this can simplify the translation to the test model behaviour considerably. Moreover, if the specification already links requirements to (specification) model elements (such as states or transitions of a statechart), it will be particularly easy to make sure that test cases for those requirements are generated and to trace them back to the specification.

Explicitness

Often, there exist implicit assumptions and implicit knowledge behind specifications. For different types of those, it is considered beneficial for model-based testing to make them explicit:

1. Environment assumptions. Knowledge about the system environment can be used to complement the system model by an environment model. The TCG can exploit this model to generate less and more realistic test cases and to become more efficient (i.e. require less generation time). In addition, the system model may become simpler if environment behaviour is restricted.
2. Intended basic system properties. The creation of a specification is often guided by general principles. It is helpful to include them in the specification although they may not constitute mandatory requirements: besides better understanding of the specification, they may include basic system properties, which can be used for testing in general (e.g. for test case validation, or by implementing observers for those properties for use during test execution), but they also can guide test model design and may be turned into static checks of the model.

3. Configuration data. Defining possible configurations or even suggesting configurations (standard configurations, configurations that include special behavioural features) is very helpful to provide an idea to the tester of how the generic system is going to be used, which will guide the test selection. In the best case, defined configuration data can be directly used to configure a generic test model so that test cases can be generated for that system configuration.
4. Allowed execution modes. To define system behaviour, it is usually not sufficient to specify a number of interconnected functions; it also depends on how communication between system components and with the environment works (synchronous vs. asynchronous, channel properties), whether system parts can be parallelized (e.g., multi-threading), and which rules are applied for event/task scheduling in the system. Knowing about the allowed execution modes is important to model the behaviour correctly, but also to judge to which extent the TCG tool can generate the behaviours that may occur for a real system.

Classification of behaviour

It is good practice to distinguish between functional and non-functional requirements in specifications. This – and further subdivision of non-functional requirements – can support model-based testing, because (a) functional testing is the main use of TCG, (b) also non-functional tests like performance or load tests may be generated, and (c) TCG is less suited for other non-functional test like stress or usability tests. Further classifications of requirements according to their importance (mandatory vs. optional, core vs. peripheral functions, safety relevance, normal vs. degraded mode, etc.) may be relevant as well. Such information can be used in the test model to support flexible test selection (before test generation) and filtering (after test generation, based on importance of information generated into the test cases).

It can also be useful to mark nondeterministic system behaviour if that is part of the specification. Being explicit about any kind of nondeterminism is helpful for testing, but it should be noted that genuine nondeterminism of the system (not just decisions left for the implementer, or limited deviations of output timing or values) is not supported by many TCG tools. For this reason, requirements that explicitly introduce nondeterministic behaviour should be avoided or be clearly recognizable, ideally also stating which parts of the system are influenced by that nondeterminism. The model-based tester then knows in advance that he needs to choose an appropriate tool, exclude the requirement from test case generation, or needs to cope with specific nondeterministic behaviour in the test environment.

8 Conclusion

This deliverable has evaluated the existing specification approaches used by the EULYNX and RCA initiatives, considering the intent to merge them into one closely intertwined approach in the future. Both complement each other, as RCA with its broader scope focuses on architectural design and EULYNX with its narrower scope focuses on behavioural specification. However, the approaches are not well connected yet and only partly cover specification needs, since further types of behaviour are part of the RCA scope. This deliverable has described proposed extensions of the semi-formal EULYNX MBSE approach to specify different types of behaviour that are needed to meet the needs of RCA. The proposed extensions of the EULYNX MBSE approach will bridge the gap between EULYNX and RCA.

More generally, the extensions described in this deliverable are:

- Specifying the characteristics of a functional railway system (system of systems).
- Specifying behaviour as mandatory requirements.
- Improving the existing EULYNX specification approach (according to evaluation results from previous work in X2Rail-2 WP5[5]).
- Additions to the existing EULYNX specification approach regarding further types of system behaviour.
- Enhancing support for automated test case generation.

Importantly, the extensions described in this deliverable aim to specify, verify, and validate system requirements efficiently within the scope of RCA, while being compatible with the automated transformation of specification models into formal models, for formal verification of these models (X2Rail-5 Task 10.8). This builds upon previous work in X2Rail-2 for the application of formal methods for verification of EULYNX specifications as described in [5].

Due to the currently still low progress in RCA, work has had to proceed based on assumptions. It is therefore necessary that results in this deliverable are reviewed again after further progress in RCA and adjusted if needed. This includes the use of appropriate specification approaches that also provide adequate support for rigorous verification of signalling systems and of the specification itself.

9 References

[1] X2Rail-5: Grant Agreement 101014520, call: H2020-S2RJU-2020, topic: S2R-CFM-IP2-01-2020

[2] X2Rail-2 Deliverable D5.1, Formal Methods (Taxonomy and Survey), Proposed Methods and Applications, May 16, 2018.

[3] X2Rail-2 Deliverable D5.2, Formal Methods Application, Revision 1.6, November 3, 2020.

[4] X2Rail-2 Deliverable D5.3, Business case, Revision 1.4, November 20, 2020.

[5] X2Rail-2 Deliverable D5.5, Application on Standard Interface(s), Revision 2.25, November 17, 2020.

[6] https://www.eulynx.eu/.

[7] https://en.wikipedia.org/wiki/Arcadia_(engineering).

[8] Reference CCS Architecture Based on ERTMS. White paper, 12-07-2018. PDF.

[9] ERTMS Longer Term Perspective, Final Report. 18/12/2015. European Railway Agency. PDF.

[10] https://sysml.org/sysml-specs/

[11] CENELEC (EN 50126)

[12] R. J. Wieringa, Design methods for reactive systems, Elsevier Science (USA), 2003

[13] B. Hruz and M.C. Zhou, Modelling and Control of Discrete-event Dynamic Systems, Springer-Verlag Limited, 2007

[14] https://www.boutique.afnor.org/fr-fr/norme/xp-z67140/technologies-de-linformation-arcadia-methode-pour-lingenierie-des-systemes-/fa192970/1723#

[15] Atego Structurer Action Language (ASAL)

[16] Mieczyslaw M. Kokar, Kenneth Baclawski, Modeling Combined Time- and Event-Driven Dynamic Systems, Northeastern University Boston, MA 02115, September 17, 2000

[17] Carl-Johan Sjöstedt, De-Jiu Chen, Phillipe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, David Servat, Martin Törngren, Developing Dependable Automotive Embedded Systems using the EAST-ADL; representing continuous time systems in SysML

[18] Sen, S., Vangheluwe, H., Multi-Domain Physical System Modeling and Control Based on Meta-Modeling and Graph Rewriting, proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, Munich, Germany (2006)

[19] Fritzson, P., Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, IEEE Press, Wiley-Interscience (2004)

[20] https://www.mathworks.com/products/simulink.html

[21] M. Utting, A. Pretschner, B. Legeard (2012): A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab., vol. 22. https://doi.org/10.1002/stvr.456

[22] K. Berkenkötter, R. Kirner (2005): Real-time and hybrid systems testing. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (eds.): Model-based Testing of Reactive Systems (Lecture Notes in Computer Science, vol. 3472), Springer Berlin, 355–387

[23] Jean-Raymond Abrial. The B-book. Assigning programs to meanings. Forewords by Professor A. Hoare and Pierre Chapron.

[24] Rodin User's Handbook v.2.8. https://www3.hhu.de/stups/handbook

[25] LinX4Rail. https://projects.shift2rail.org/s2r_ipx_n.aspx?p=LINX4RAIL

Appendix A: Ownership of results

The following Table 2 lists the ownership of results for this deliverable.

Ownership of results			
Company	**Percentage**	**Short Description of share/ of delivered input**	**Concrete Result (where applicable)**
Deutsche Bahn (DB)			
Trafikverket (TRV)			
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)			

Table 2: Ownership of results

This deliverable is jointly owned by the organisations listed above. The last three columns in the table are intentionally left empty.